1

BİLGİSAYARA GİRİŞ, INTERNET VE WWW

AMAÇLAR

 Temel bilgisayar kavramlarını anlamak.

 Değişik tipte programlama dillerini tanıdık hale gelmek.

 C programa dilinin tarihçesini tanıdık hale gelmek.

 C standart kütüphanesini hakkında bilgi edinmek.

 Tipik bir C programı geliştirme ortamının elemanlarını anlamak.

 Programlama öğrenmeye C ile başlamanın niçin uygun olduğunu anlamak.

 C öğrenmenin genel anlamda programlama dillerini ve kısmen C++ ve Java'yı

öğrenmedeki faydalarını anlamak.

BAŞLIKLAR

1.1 Giriş

1.2 Bilgisayar nedir?

1.3 Bilgisayar organizasyonu

1.4 İşletim sistemlerinde evrim

1.5 Kişisel kullanım, çoklu kullanım, istemci/sunucu kullanımı

1.6 Makine dilleri, assembly dilleri ve yüksek seviyeli diller

1.7 C 'nin tarihi

1.8 Standart C kütüphanesi

1.9 Yazılımda kilit nokta: nesne teknolojisi

1.10 C++ ve C++ ile programlama

1.11 Diğer yüksek seviyeli diller

1.12 Yapısal programlama

1.13 C programı geliştirme ortamının temelleri

1.14 Donanım eğilimleri

1.15 Internet'in tarihi

1.16 WWW'in tarihi

1.17 C ve bu kitap hakkında genel notlar

Özet \* Terimler \* İyi programlama alıştırmaları \* Taşınırlık ipuçları \* Performans

ipuçları \* Cevaplı alıştırmalar \* Cevaplar \* Alıştırmalar \*

1.1 GİRİŞ

C ve C++ 'a hoş geldiniz. Sizin için öğretici ve eğlenceli bir deneyim olacağına inandığımız

bu kitabı yazmak için oldukça sıkı çalıştık. Bu kitap, diğer kitaplar arasında aşağıda

sayacağımız özelliklerden dolayı tektir.

 Bu kitap, daha önceden hiç programlama tecrübesi olmayan ya da çok az tecrübesi

olan kişiler için uygundur.

 Ayrıca, daha tecrübeli programcılar için de detaya inme fırsatı sunmaktadır.

2

Peki nasıl oluyor da bu kitap iki gruba da hitap edebiliyor? Cevap olarak, bu kitabın

iskeletinin daha önceden ispatlanmış yapısal programlama tekniklerini programlara açık bir

dille uygulamasını verebiliriz. Daha önceden programlamayla ilgilenmemiş kişiler,

programlamayı doğru bir şekilde öğreneceklerdir. Kitabı açık ve düzgün bir tarzda yazdık. Bu

kitap, mümkün olabildiğince bol şekillendirilmiştir ve bundan daha önemlisi, kitapta yüzlerce

çalışan C program örneği ve bu programların çıktıları sunulmuştur. Buna, gerçek kod

yaklaşımı diyoruz. Tüm bu örnek programları www.deitel.com adresinden indirebilirsiniz.

İlk dört ünitede hesaplamanın temelleri, bilgisayar programlama ve C bilgisayar dilinin

temelleri anlatılmıştır. Kurslarımıza ilk kez katılanlar elinizdeki kitabın, ilk dört ünitesinin beş

ile on dördüncü üniteler arasındaki konuların daha iyi anlaşılabilmesi için taban

oluşturduğunu söylerler. Daha tecrübeli programcılar, ilk dört üniteyi hızlıca gözden geçirip,

daha zorlayıcı olan beş ile on dördüncü üniteler arasında çalışırlar ve özellikle göstericiler,

stringler, dosya ve veri yapıları konusundaki detayları takdir ederler.

Çoğu tecrübeli programcılar, yapısal programlama hakkındaki kısmı da takdir etmektedirler.

Aslında, Pascal gibi yapısal programlama dilleriyle uğraşmalarına rağmen, yapısal

programlama hakkında gerçek bir eğitim almadıklarından, asla mümkün olan en iyi kodu

yazamamışlardır. C 'yi bu kitapla öğrenirken, programlama tarzlarını da ilerletme imkanına

kavuşurlar. Özet olarak ister deneyimli, ister deneyimsiz olun, bu kitap sizi bilgilendirecek,

eğlendirecek ve zorlayacak.

Bir çok kişi, bilgisayarların yaptığı heyecan verici işlere aşinadır. Bu kitabı kullanarak

bilgisayarlara bu işleri nasıl yaptıracağınızı öğreneceksiniz. Bilgisayara işlemler

yaptırabilmek ve karar verdirtebilmek için yazılan kalıplara yazılım denir. Yazılım, genel

olarak donanım olarak adlandırılan kısımları kontrol eder. Bu kitap, C programlama dilinin,

1989 yılında Amerika'da American National Standards Institute(ANSI) ve dünyada,

International Standartds Organization(ISO) tarafından standartlaştırılmış sürümünü

anlatmaktadır.

1999 yılında ISO, C' nin yeni bir sürümünü (C99) tanıtmıştır. Ancak bu kitap yazılırken hala

C99 derleyicileri piyasaya sürülmemişti. Bu yüzdende, gerçek kod yaklaşımı kullanarak

yazdığımız örnekler, C99 derleyicileriyle uyumlu olmayabilir. C99 derleyicileri piyasaya

sürüldükten sonra bu kitaptaki bütün programları C99 derleyicileriyle derleyeceğiz. Eğer

herhangi bir programda değişiklik yapılması gerekirse, değişiklik yapılan programı

www.deitel.com adresindeki sitemizde yayınlayacağız.

Ekler B kısmında, C99 derleyicilerine Internet kullanarak ulaşabileceğiniz kaynakların

listesini bulabilirsiniz. Okuyucularımıza C99 standardındaki gelişmeleri takip etmelerini

şiddetle öneriyoruz.

Bilgisayar kullanımı her alanda büyük bir artış göstermektedir. Fiyatların hızlı bir artış

gösterdiği çağımızda bilgisayar fiyatları, donanım ve yazılım teknolojisindeki hızlı gelişimden

dolayı düşüş göstermektedir. 25-30 yıl önce büyük odaları dolduran ve milyonlarca dolara

mal olan bilgisayarlar, bugün her biri bir kaç dolara mal olan ve tırnağımızdan daha küçük

silikon çipler sayesinde üretilmektedir. Silikon, toprağı oluşturan elementlerden biridir ve

dünya üzerinde bol miktarda bulunmaktadır. Silikon çip teknolojisi, ucuzluğu dolayısıyla

milyonlarca genel amaçlı bilgisayarda kullanılmaktadır ve bu sayı bir kaç yıl içinde ikiye

katlanacaktır.

3

Nesneye yönelik programlama dilleri olan C++ ve Java, C temeline dayanmaktadır ve

günümüzde oldukça büyük bir ilgi toplamaktadır. Bu sebepten dolayı, on beş ile yirmi üçüncü

üniteler arasında nesneye yönelik programlama ve C++ hakkında gerekli bilgiyi kitabımıza

ekledik. Programlama dilleri pazarında önde gelen üreticiler, C ve C++'ı ayrı ayrı sunmak

yerine birlikte sunmayı tercih etmektedirler. Bu sayede, kullanıcı, C ile programlamaya

devam etmekte ve hazır olduğunda da C++'a geçiş yapmaktadır.

Şu anda, oldukça zorlayıcı ama karşılığını mutlaka en iyi şekilde alacağınız bir yola girmiş

bulunuyorsunuz. Eğer herhangi bir anda, bizimle haberleşmek isterseniz bize

deitel@deitel.com e-mail adresinden ulaşabilirsiniz ya da www.deitel.com adresindeki

sitemizi ziyaret edebilirsiniz. Mümkün olduğu kadar hızlı cevap alacaksınız.

1.2 BİLGİSAYAR NEDİR ?

Bilgisayarlar, hesaplamaları ve mantıksal kararlar vermeyi insanlardan milyonlarca hatta

milyarlarca kez hızlı yapabilme yeteneğine sahip cihazlardır. Örneğin, bugünkü çoğu kişisel

bilgisayar, saniyede milyonlarca toplama işlemini gerçekleştirebilir. Aynı işlemi, hesap

makinesiyle yapmak yıllar alabilir. ( Dikkat edilmesi gereken bir başka nokta da, kişinin

sayıları hesap makinesine hatasız giremeyeceği gerçeğidir. ) Bugünkü en hızlı bilgisayarlar

ise saniyede milyarlarca toplama işlemini gerçekleştirebilir. Bunların daha da ötesinde bazı

araştırma laboratuarlarında trilyonlarca toplama işleminin yapılabildiğinide söylemeden

geçemeyeceğiz.

Bilgisayarlar, bilgisayar programı adı verilen bir dizi komutla verileri işlerler. Bilgisayar

programlarcıları tarafından yazılan bu programlar, bilgisayarın işlemler yapabilmesini

sağlatır.

Bir bilgisayar, donanım olarak adlandırılan bir çok parçadan ( klavye, ekran, fare, CD-ROM,

hafıza, sabit diskler ve işlemciler ) oluşur. Bilgisayarda kullandığımız programlara yazılım

denir. Donanım maliyetleri son yıllarda büyük bir düşüş gösterirken, yazılım maliyetleri,

programcılar daha güçlü ve karmaşık işlemleri gerçekleştirebilen programlar yazdıkça

artmaktadır. Bu kitap boyunca yazılım maliyetlerini azalttığı kanıtlanmış yazılım geliştirme

yöntemlerini öğreneceksiniz. Bu yöntemler yapısal programlama, yukarıdan aşağıya adımsal

iyileştirme, fonksiyonellik, nesne tabanlı programlama, nesneye yönelik programlama, event

driven ve generic programlamadır.

1.3 BİLGİSAYAR ORGANİZASYONU

Fiziksel görünümlerindeki farklılıklara rağmen genellikle bilgisayarlar altı mantıksal kısma

ayrılabilir.

Bunlar aşağıda özetlenmiştir:

1-Giriş ünitesi. Bu kısım, bilgisayarın dış dünyayla haberleştiği kısımdır. Giriş ünitesi, çeşitli

giriş cihazlarından veri ve bilgisayar programlarını alır ve diğer birimlerin kullanımına sunar.

Böylece bilgi işlenmeye hazır hale gelir. Bugünlerde bilgisayarlara bilgiler, klavyeler ve

fareler sayesinde yüklenmektedir. Ayrıca bilgiler konuşarak ya da görüntülerin taranmasıyla

da bilgisayarlara girilebilir.

4

2-Çıkış Ünitesi. Çıkış ünitesi, bilgisayardan bilgilerin alındığı kısımdır. Bu ünite, işlenmiş

bilgiyi bilgisayardan alır ve kullanıcıya uygun bir hale getirerek çıkış cihazlarına gönderir.

Bugünlerde çıktı, ekranda gösterilerek, kağıtlara yazdırılarak ya da diğer cihazları kontrol

etmek için kullanılarak alınır.

3-Hafıza Ünitesi. Bu kısım, bilgisayarın düşük kapasiteli ancak hızlı erişimli depolama

kısmıdır. Giriş birimlerinden girilen bilgiyi saklayarak, işleme sırasında ihtiyaç duyulduğunda

bu bilgilerin hızlı bir şekilde kullanıma hazırlanmasını sağlar.

Hafıza ünitesi ayrıca, işlenmiş bilgiyi çıkış cihazlarına gönderene kadar saklamakla

görevlidir. Hafıza kısmı genellikle hafıza ya da birincil hafıza olarak adlandırılır.

4-Aritmetik Mantık Ünitesi(ALU). Bu kısım bilgisayarın üretim kısmıdır. Toplama, çıkartma

çarpma ve bölme gibi işlemlerin yapılmasından sorumludur. Hafıza ünitesinden iki elemanı

karşılaştırma ve bu elemanların eşit olup olmadıklarına karar verme gibi görevleri de

yapmakla sorumludur.

5-Merkezi İşleme Ünitesi(CPU). Bilgisayarın yönetici kısmıdır. Bilgisayarın

koordinasyonundan ve diğer kısımlarının çalışmasının denetlenmesinden sorumludur. CPU,

giriş ünitesine bilgilerin hafızaya aktarılacağını, ALU 'ya hafızadaki bilgilerin işlemlerde

kullanılacağını, çıkış ünitesine hafızadaki bilgilerin belirli çıkış cihazlarına gönderileceğini

söyler.

6-İkincil Depolama Ünitesi. Bu kısım yüksek kapasiteli depolama kısmıdır ve bilgiler burada

daha uzun süreyle tutulurlar.

Programlar ya da diğer üniteler tarafından faal olarak kullanılmayan veriler, normal olarak

ikincil depolama cihazlarında (diskler ) saklanırlar. Böylece bilgiler saatler, günler, aylar ya

da yıllar sonra yeniden ihtiyaç duyulduklarında kullanılabilirler. İkincil depolama ünitesindeki

bilgilere ulaşma zamanı, birincil depolama ünitesindeki bilgilere ulaşma zamanından daha

fazladır. İkincil depolama ünitelerinin birim maliyeti, birincil depolama ünitelerinin birim

maliyetinden daha ucuzdur.

1.4 İŞLETİM SİSTEMLERİNDE EVRİM

Önceleri, bilgisayarlar belli bir anda yalnızca bir işi ya da görevi yapabiliyorlardı. Bu formda

bir bilgisayar işletimi genellikle, tek kullanıcılı yığın işleme(batch processing) olarak

adlandırılır. Bu formdaki bilgisayarlar, bir anda tek bir program çalıştırırken, verileri gruplar

ya da yığınlar halinde kullanırdı. O zamanki sistemlerde kullanıcılar, işlerini bilgisayara

delikli kartlar sayesinde yaptırırlardı. Kullanıcıların, işlemlerin sonucunu alabilmeleri saatler

ya da günler sürebilirdi.

İşletim sistemi olarak adlandırılan yazılımlar, bilgisayarları daha rahat kullanabilmek için

geliştirilmiştir. İlk işletim sistemleri, işler arasındaki geçişleri yönetebiliyordu. Bu,

kullanıcıların işler arasında geçişler yapabilme zamanını azaltarak, yapılacak iş sayısını yani

throughputu arttırmıştı.

Bilgisayarlar daha güçlü hale geldiklerinde, tek kullanıcılı yığın işlemenin bilgisayarın

kaynaklarını verimli kullanamadığı anlaşıldı. Çünkü, zamanın çoğu yavaş çalışan giriş/çıkış

cihazlarının görevlerini tamamlamalarını beklemekle geçiyordu. Bunun yerine, işlerin ya da

görevlerin bilgisayarın kaynaklarını paylaşabilecekleri düşünüldü. Bu, çoklu programlama

( multiprogramming ) olarak adlandırılır. Çoklu programlama, bilgisayarda birden çok işin eş

5

zamanlı olarak yapılmasını sağlar. İlk çoklu programlama işletim sistemlerinde, kullanıcılar

hala delikli kartlar kullanıyordu ve işlemlerin sonuçlanması saatler ya da günler alıyordu.

1960'larda sanayide ve üniversitelerde çeşitli gruplar, zaman paylaşımlı işletim sistemlerine

öncülük ettiler. Zaman paylaşımı, kullanıcıların bilgisayara klavye ve ekran gibi cihazlardan

oluşan terminallerden erişmelerini sağlayan, bir çoklu programlama biçimidir. Tipik bir

zaman paylaşımlı bilgisayar sisteminde onlarca ya da yüzlerce kullanıcı, bilgisayarı

kullanabilir. Aslında bilgisayar, bütün kullanıcıların isteklerini aynı anda yapmaz. Bunun

yerine, bir kullanıcının işinin bir kısmını yapar ve diğer kullanıcın işine geçer. Bilgisayar, bu

işlemi o kadar hızlı yapar ki her kullanıcının işlemine bir saniye içinde birkaç kez servis

sağlayabilir. Bu yüzden, kullanıcıların programları eş zamanlı çalışıyormuş gibi görünür.

Zaman paylaşımının avantajı, kullanıcıların isteklerine neredeyse anında cevap almalarıdır.

Bu, eski yöntemlerle daha uzun bir zaman gerektirirdi.

1.5 KİŞİSEL KULLANIM, ÇOKLU KULLANIM, İSTEMCİ/SUNUCU

KULLANIMI

1977 senesinde APPLE COMPUTER, kişisel kullanım kavramını yaygınlaştırdı. Bu ilk başta,

bilgisayarla hobi olarak uğraşan birinin hayali gibi görünüyordu. Bilgisayarlar, insanların

kişisel ya da işleriyle ilgili kullanımları amacıyla satın alabilecekleri kadar ekonomik bir hale

geldi. 1981'de dünyanın en büyük bilgisayar üreticisi IBM, IBM kişisel bilgisayarı tanıttı.

( IBM-PC )

Fakat bu bilgisayarlar standalone birimlerdi ve insanlar işlerini yaptıktan sonra bilgileri

paylaşmak için disklerini taşıyorlardı. ( Buna genelde sneakernet denir.) Bu ilk kişisel

bilgisayarlar, birden fazla kullanıcının zaman paylaşımı yapması için yeterince güçlü

olmamasına rağmen, bilgisayar ağları oluşturmak amacıyla birbirlerine bağlanıyorlardı.

Böylece telefon ağları kullanarak yerel ağ ( LAN ) oluşturuluyordu. Bu, çoklu kullanım

( distibuted computing ) kavramını ortaya çıkarttı. Çoklu kullanımda, bir organizasyonun

hesaplamaları için merkezi bir bilgisayar yerine, ağ üzerinden bu organizasyonla ilgili

kişilere iş dağıtılıyordu. Kişisel bilgisayarlar her kullanıcının kendi hesaplamalarını

yapabilecek kadar güçlü olmanın yanında, bilgileri elektronik olarak göndermek ve almak

gibi temel haberleşme görevlerini de yapabiliyorlardı.

Bugünün en güçlü kişisel bilgisayarları, on ya da yirmi yıl öncenin milyon dolarlık

bilgisayarları kadar güçlüdür. En güçlü masa üstü makineler ( iş istasyonları ), kullanıcılara

muazzam yetenekler sunmaktadır. Ağ üzerinde bulunan istemciler tarafından kullanılması

muhtemel program ve verilere, bilgisayar ağları üzerinden dosya sunucularına (file server)

erişilerek ulaşılır. Buna, istemci/sunucu (client/server) kullanımı denir. C ve C++ işletim

sistemleri, bilgisayar ağları ve çoklu kullanım istemci/sunucu uygulamaları için en uygun

seçimdir. Bugünkü en popüler işletim sistemleri: UNIX, LINUX ve Microsoft'un Windows

tabanlı sistemleri bu kısımda anlatılan yeteneklere sahiptirler.

1.6 MAKİNE DİLLERİ, ASSEMBLY DİLLERİ VE YÜKSEK SEVİYELİ

DİLLER

Programcılar değişik programlama dillerini kullanarak komutlar yazarlar. Bunlardan bazıları

bilgisayar tarafından doğrudan anlaşılabilirken, bazıları ise çevirme (translation) işlemlerine

tabi tutulmak zorundadır. Günümüzde yüzlerce bilgisayar dili vardır. Bunlar üç genel tipe

ayrılabilirler.

6

1-Makine dilleri

2-Assembly dilleri

3-Yüksek seviyeli diller

Herhangi bir bilgisayar, doğrudan yalnızca kendi makine dilini anlayabilir. Makine dili

bilgisayarın doğal dilidir ve o bilgisayarın donanım tasarımına bağlıdır. Makine dilleri, belirli

sayıların özel dizilimler ile bilgisayarın temel işlevlerini yaptırtmalarını sağlarlar (bu sayılar

genellikle 1 ve 0'lara indirgenirler.) Makine dilleri her makinede farklılık gösterebilir. Bu

yüzdende, makine bağımlı(machine dependent) olarak adlandırılırlar. Makine dilleri insanlar

için oldukça zordur. Aşağıda, makine diliyle yazılmış bir programı görebilirsiniz. Bu program

parçacığında iki sayı toplanıp, sonuç hafızada saklanmıştır.

+130042774

+1400593419

+1200274027

Bilgisayarlar popüler hale geldikçe, makine dilleriyle programlamanın oldukça yavaş,

zahmetli ve hata yapma oranının yüksek olduğu görüldü. Bilgisayarların doğrudan

anlayabileceği belli sayı dizilişleri kullanmak yerine, programcılar İngilizce’ye yakın

kısaltmalar kullanmaya başladılar. Bu kısaltmalar, assembly dillerinin temelini oluşturur.

Assembler olarak adlandırılan çevirici programlar, assembly dilinde yazılmış programları

makine diline çevirmek için geliştirilmiştir. Şimdi, yukarıda makine diliyle yazdığımız

programın assembly diliyle yazılmış halini göreceksiniz.

LOAD SAYI1

ADD SAYI2

STORE TOPLAM

Bu kod, insanlara oldukça yakın gelse de makine diline çevirmediği sürece bilgisayarlara bir

şey ifade etmez.

Bilgisayar kullanımı, assembly dillerinin ortaya çıkmasıyla hızlıca artmıştı fakat hala bazı

basit görevleri yapmak için birçok kod yazmak zorunda kalınıyordu. Programlama sürecini

hızlandırabilmek amacıyla yüksek seviyeli diller geliştirildi. Bu dillerde, tek bir ifadeyle

birden çok görevi yerine getirmek mümkün oluyordu. Derleyici ( compiler ) olarak

adlandırılan çevirici programlar, yüksek seviyeli dilleri makine dillerine çevirirler. Yüksek

seviyeli diller, programcılara günlük İngilizce’ye oldukça yakın kodlar yazma imkanı sunar.

Ayrıca, genellikle kullanılan matematik ifadeleri de yüksek seviyeli dillerde kullanılabilir.

Şimdi de yukarıda önce makine diliyle daha sonra assembly dilleriyle yazılmış program

parçacığını, yüksek seviyeli dillerle yazalım.

toplam = sayi1 + sayi2

Açıkça görüleceği üzere, yüksek seviyeli diller makine dilleri ya da assembly dillerine göre

programcılar tarafından daha çok tercih edilirler. C ve C++ , yüksek seviyeli diller arasında en

güçlü ve en çok kullanılanlarıdır. Yüksek seviyeli dillerle yazılmış bir programın makine

diline çevrilme süreci, bilgisayarda bir müddet süre alabilir. Bu problem, yüksek seviyeli

7

dillerle yazılmış programları, makine diline derleme ihtiyacı duymadan çalıştırabilen

yorumlayıcı ( interpreter ) programlar sayesinde çözülmüştür. Derlenmiş programlar,

yorumlanmış programlara göre daha hızlı çalışmalarına rağmen, program geliştirme

ortamlarında programlara yeni özellikler eklenirken ve hatalar düzeltilirken yorumlayıcılar

daha yaygın bir şekilde kullanılmaktadır. Bir program geliştirildikten sonra bu programın

derlenmiş versiyonu daha verimli çalışabilir.

1.7 C TARİHÇESİ

C, temelde iki eski dile dayanır : BCPL ve B. BCPL, 1967 yılında Martin Richards tarafından

işletim sistemleri ve derleyiciler yazmak için geliştirilmiştir. Ken Thompson, BCPL

çalışmalarının ardından kendi yarattığı dil olan B'yi geliştirmiştir ve B ile UNIX'in ilk

versiyonları üzerinde, Bell Laboratuarlarında, DEC PDP-7 bilgisayarı ile çalışmıştır. Bu iki

dilde de, her veri hafızada bir “word” ( 16 bit ) alan kaplamaktaydı ve değişkenlerin yazımı

programcıya ağır bir yük getiriyordu.

C dili, 1972'de bu çalışmaların izinde yine Bell Laboratuarlarında Dennis Ritchie tarafından

DEC PDP-11 bilgisayarlarında geliştirilmiştir. C, BCPL ve B dillerinin önemli bir çok

kavramını kullanırken, veri yazımı ve daha bir çok güçlü özellikleri de içerir. C, genel

anlamda bir işletim sistemi olan UNIX' in geliştirilmesinde kullanılmasıyla ün kazanmıştır.

Bugün, bütün yeni işletim sistemleri C ve/veya C++ ile yazılmaktadır. Geçen yirmi yıl içinde

C, bütün bilgisayarlar için uygun hale getirilmiştir. C, donanımdan bağımsızdır. Bu yüzden

C'de dikkatli bir biçimde yazılmış bir program her bilgisayara taşınabilir.

1970'lerin sonunda C, şu anda geleneksel C olarak bilinen haline geldi. 1978 yılında

Kernighan ve Ritchie tarafından yazılan, The C Programming Language adlı kitabın

yayınlanmasından sonra, C'ye olan ilgi artmıştır. Bu yayın, bütün zamanların en iyi bilgisayar

kitaplarından biridir. C'in değişik tipte bilgisayarlarda ( donanım platformlarında ) yayılması,

birbirine benzer ama genellikle uyumsuz bir çok çeşidinin ortaya çıkmasına sebep olmuştur.

Bu, değişik platformlarda çalışacak kodlar yazan program geliştiricileri için ciddi bir problem

haline gelmişti. Bu sebeplerden dolayı, C'nin standart bir versiyonuna ihtiyaç duyulduğu

anlaşıldı. 1983 yılında, American National Standards Committee'nin bilgisayar ve bilgi işlem

komitesinde ( X3 ), X3J11 adı altında teknik bir komite oluşturuldu ve C'nin sistem bağımsız

bir tanımı yaptırıldı. 1989 yılında bu standart onaylandı ve 1999 yılında da tekrar gözden

geçirildi. Bu standart, ISO/IEC 9899:1999 olarak adlandırıldı ve ISO tarafından onaylandı. Bu

standardın kopyaları bu kitabın giriş kısmında adresi verilen American Standards Instute'tan

bulunabilir.

Taşınabilirlik İpuçları 1.1

C, donanıma bağımlı olmadığından, C'de yazılacak bir program değişiklik yapılmadan ya da

çok az değişiklikle bir çok bilgisayarda sorunsuz olarak çalışabilir.

[Not: Bu kitapta, sizlere, değişiklik yapmadan ya da çok az değişiklik yaparak, bir çok

bilgisayarda çalışacak programlar yazmanızı sağlatacak teknikleri taşınırlık ipuçları kısmında

anlatacağız. Ayrıca daha açık, daha anlaşılabilir, incelenmesi test edilmesi ve hataları

ayıklanabilmesi kolay programlar yazabilmeniz için iyi programlama alıştırmalarını

bulacaksınız. Genel programlama hataları kısmında, aynı hataları programlarınızda

yapmamanız için genellikle karşılaşılan hataları, performans ipuçlarında ise daha az hafıza

kaplayan ve daha hızlı çalışan programlar yazabilme tekniklerini, test ve hata ayıklama

ipuçlarında programlarınızdan hata ayıklamayı ve daha önemlisi ilk seferinde hatasız

8

programlar yazmanızı sağlatacak teknikleri ve yazılım mühendisliği gözlemleri kısmında

büyük yazılım sistemlerinin bütün yapısında geliştirmeler yapmanızı sağlatacak kavramları

bulacaksınız. Bu tekniklerin ve alıştırmaların büyük kısmı sizlere rehber olmak amacıyla

konulmuştur. Elbetteki kendi programlama tarzınızı geliştireceğiniz kesindir.]

1.8 STANDART C KÜTÜPHANESİ

5. Ünitede göreceğimiz gibi, C programları fonksiyon adı verilen parçalardan ya da

modüllerden oluşur. Elbette kendi fonksiyonlarınızı yazmanız mümkündür ama çoğu

programcı, C standart kütüphanesindeki hazır fonksiyonları kullanır. Bu sebepten, C

öğrenmede aslında iki kısım vardır. İlk kısım, C dilinin kendisini ikinci kısım ise C standart

kütüphanesindeki fonksiyonlarının nasıl kullanılacağını öğrenmekten oluşur. Bu kitapta, bu

fonksiyonların büyük bir kısmını anlatacağız. Kütüphane fonksiyonları hakkında daha detaylı

bir araştırma yaparak bu fonksiyonların nasıl kullanılacağını öğrenmek ve bu fonksiyonları

kullanarak taşınabilir kodlar yazmak isteyen okuyucuların P.J.Plauger'ın The Standard C

Library adlı kitabını incelemeleri gerekmektedir.

Bu kitap, blokları yerleştirme yaklaşımıyla program yazmayı hedeflemektedir. Tekerleği

yeniden icat etmekten kaçının. Daha önceden hazırlanmış parçaları kullanarak, 15. üniteden

kitabın sonuna kadar anlatacağımız nesneye yönelik programlama alanlarında kilit bir rol

oynayan yazılımın yeniden kullanılabilirliğine şimdiden alışın. C ile çalışırken genellikle

aşağıdaki blokları kullanacaksınız.

 C standart kütüphane fonksiyonları.

 Kendi yazdığınız fonksiyonlar.

 Diğer programcıların yazdığı fonksiyonlar.

Kendi fonksiyonlarınızı yazmanın avantajı, bu fonksiyonların nasıl çalışacaklarını tam olarak

bilmenizdir. Böylece, bu C kodlarını kolaylıkla inceleyebilirsiniz. Olumsuz taraf ise yeni

fonksiyonlar yazmanın ve geliştirmenin oldukça zaman almasıdır.

Eğer daha önceden yazılmış fonksiyonları kullanırsanız, böylelikle tekerleği yeniden icat

etmekten kurtulabilirsiniz. ANSI standart fonksiyonlarını kullanırken, bunların oldukça özenli

bir biçimde yazıldığını ve ANSI C koşullarına uyan tüm sistemlerde çalışabileceğine emin

olabilirsiniz. Böylelikle programlarınız daha taşınabilir olacaktır.

Performans İpuçları 1.1

Kendi yazdığınız fonksiyonlar yerine, ANSI standart kütüphane fonksiyonlarını kullanmak

programın performansını artırır. Çünkü bu fonksiyonlar verimli çalışmaları için özenle

yazılmıştır.

Taşınırlık İpuçları 1.2

Kendi yazdığınız fonksiyonlar yerine ANSI standart kütüphane fonksiyonlarını kullanmak

taşınırlığı artırır. Çünkü bu fonksiyonlar bütün ANSI C koşullarına uyarlar.

9

1.9 YAZILIMDA KİLİT EĞİLİM: NESNE TEKNOLOJİSİ

Yazarlardan Harvey M. Deitel, büyük ölçekli projeler geliştiren yazılım geliştirme

organizasyonlarında, 1960'larda hissedilen büyük hayal kırıklığını hatırlıyor. Üniversite

yıllarının yaz aylarında yazar, önde gelen bilgisayar şirketlerinin birinde zaman paylaşımı ve

sanal hafıza sistemleri geliştirme takımlarında çalışma fırsatı bulmuştur. Ancak 1967 yazında

gerçek, şirketin, yüzlerce kişinin yıllardır üzerinde uğraştığı ürünü pazarlama düşüncesinden

vazgeçmesiyle ortaya çıkmıştır. Bu yazılımın gerçekleştirilmesi mümkün değildi. Yazılım

karmaşık bir işti.

Donanım maliyetleri son yıllarda, kişisel bilgisayarlar bir eşya haline gelene kadar hızlı bir

düşüş gösterdi. Fakat yazılım geliştirme maliyetleri, programcılar yazılım geliştirme

teknolojilerinin temellerinde önemli bir iyileştirme yapmadan daha güçlü ve karmaşık

uygulamalar gerçekleştirdikçe, sabit bir şekilde arttı. Bu kitapta, yazılım geliştirme

maliyetlerini azaltacak bir çok yöntem öğreneceksiniz.

Yazılım toplumunda bir devrim başlıyor. Yeni ve daha güçlü yazılımların yerinde saydığı bir

anda hızlı, ekonomik, doğru yazılımı geliştirmek hala ulaşılamaz bir hedef olarak duruyor.

Nesneler, gerçek hayattaki parçaları modelleyen ve yazılım içinde tekrar tekrar kullanabilen

parçalardır. Yazılım geliştirenler ; modüler, nesneye yönelik tasarımlar ve yerine koyma

yaklaşımıyla, yazılım geliştirme gruplarının eski programlama tekniklerini kullanarak

yazabilecekleri programlardan daha yaratıcı olabileceklerini keşfetmeye başladılar. Nesne

yönelimli programlar genellikle daha kolay anlaşılır, düzeltilir ve değiştirilir.

Yazılım teknolojisindeki değişim, 1970'lerde, yapısal programlamanın (ve yapısal sistem

analizi ve tasarımı ile konuların) yararlarının fark edilmesiyle başlamıştır. Ancak nesneye

yönelik programlamanın, 1980'lerde yaygınlaşmaya başlaması ve 1990'larda iyice

yaygınlaşmasıyla, yazılım geliştirenler yazılım geliştirme sürecinin iyileştirilmesinde büyük

adımlar atmak için gerekli tüm malzemeyi bulduklarını düşündüler.

Aslında nesne teknolojisinin geçmişi 1960'ların ortalarına rastlar. C++ programlama dili

1980'lerde AT&T'de Bjarne Stroustrup tarafından geliştirildi. C++ aslında iki dile dayanır.

AT&T tarafından UNIX işletim sistemini geliştirmek için kullanılan C ve 1967'de Avrupa’da

geliştirilen Simula67. C++, C'nin tüm özelliklerini alıp Simula'nın nesne yaratma ve işletme

özelliklerini bünyesine eklemiştir. C ya da C++ 'nın AT&T laboratuarları dışında kullanılması

düşünülmemişse de ikisinin de yayılması çok hızlı olmuştur.

Peki nesne nedir ve nesneler neden özeldir? Nesne teknolojisi, belirli uygulama alanlarında

büyük ve odaklanmış anlamlı yazılım birimleri oluşturmamıza yardımcı olan paketleme

şemasıdır. Her isim, bir nesne olarak gösterilebilir. Örneğin, tarih nesneleri, zaman nesneleri,

ses nesneleri, video nesneleri, dosya nesneleri, kayıt nesneleri.

Nesnelerle dolu bir dünyada yaşıyoruz. Etrafınıza bir bakın bir çok nesne göreceksiniz ;

arabalar, uçaklar, insanlar, hayvanlar, binalar, trafik ışıkları, anahtarlar ve benzerleri. Nesne

yönelimli dillerden önce, programlama dilleri ( FORTRAN, PASCAL, BASIC ve C gibi )

nesneler yerine eylemlere odaklanmıştı. Nesnelerle dolu bir dünyada yaşayan programcılar,

bilgisayarların başına geçince eylemlerle uğraşıyorlardı. Bu numune değişimi, program

yazmayı hantal hale getiriyordu. Şimdi ise Java, C++ ve bir çok diğer nesneye yönelik

programlama dilleri sayesinde programcılar, normal yaşamlarında olduğu gibi

10

bilgisayarlarının karşısında da nesnelerle uğraşmaya devam ederler. Bu, onların programlarını

dünyayı gördükleri biçimde yazmaları anlamına gelir. Bu, Procedural programlamadan daha

doğal bir yoldur ve verimliliğin artmasına önemli katkılarda bulunmuştur.

Procedural programlamadaki önemli sorunlardan biri de programcıların yarattığı program

birimlerinin, gerçek dünyayı tam olarak yansıtamamasıdır. Bu sebepten, bu birimler yeniden

kullanılabilir değildir. Programcıların, baştan başladıktan sonra diğer kodlara yakın kodlar

yazmaları oldukça sık görülen bir durumdur. Bu durum, zaman ve yatırım maliyetlerini artırır.

Çünkü her seferinde tekerlek yeniden icat edilir. Nesne teknolojisiyle yazılım girdileri

(nesneler) oluşturulur ve eğer bunlar iyi tasarlanmış olursa ilerleyen projelerde yeniden

kullanılmaları mümkün hale gelir. MFC( Microsoft Foundation Classes) ya da Roque Wave

ve diğer yazılım geliştirme organizasyonlarının kütüphanelerindeki yeniden kullanılabilir

parçaları kullanmak, belli sistemleri oluşturmakta harcanarak gücü önemli derecede azaltır.

Bazı organizasyonlar, nesneye yönelik programlamada esas faydanın yeniden kullanım

olmadığını belirttiler. Bunun yerine, nesneye dayalı programlamanın daha anlaşılır ve daha iyi

organize olmasının, incelenmesinin, değiştirilmesinin ve hatalarının ayıklanmasının daha

kolay olmasının bu teknolojinin en önemli özellikleri olduğunu söylediler.Bunlar, gerçektende

önemlidir. Çünkü yazılım maliyetlerinin %80'nin programlarının geliştirilmesinde ve

iyileştirmelerin yapılmasında harcandığı belirlenmiştir.

Nesneye yönelik programlamanın faydaları ne olursa olsun, nesneye yönelik programlamanın

önümüzdeki yıllarda programlama yöntemlerinde kilit nokta olacağı kesindir.

Kendi kodlarınızı yazmanın avantajı, bu kodun nasıl çalışacağını bilmenizdir ve

inceleyebilmenizdir. Olumsuz tarafı ise zaman alması ve yeni bir kod tasarlamak ve yazmanın

oldukça fazla uğraş gerektirmesidir.

Yazılım Mühendisliği Gözlemleri 1.1

Yeniden kullanılabilir yazılım parçalarından oluşan genel kütüphaneler Internet üzerinde

bulunabilir. Bunlardan bazıları ücretsizdir.

1.10 C++ ve C++ ile PROGRAMLAMA

C++ programlama dili, C'nin, BELL laboratuarlarında Bjarne Stroustrup tarafından

geliştirilmiş halidir. C++ , C'yi daha güçlü hale getiren bir çok özellik sunar ve daha önemlisi

C++, nesneye yönelik programlama yeteneklerine sahiptir.

Nesneler, gerçek hayattaki araçları modelleyen ve yeniden kullanılabilir yazılım parçalarıdır.

Yazılım toplumlarında bir devrim gerçekleşmektedir. Yeni ve daha güçlü yazılımların yerinde

saydığı bir zamanda hızlı, ekonomik ve doğru yazılımı gerçekleştirmek hala ulaşılamaz bir

hedef olarak duruyor.

Yazılım geliştirenler modüler, nesneye dayalı tasarımlar ve yerine koyma yaklaşımıyla,

yazılım geliştirme gruplarının, eski programlama teknikleriyle mümkün olandan on ile yüz

kat arasında daha yaratıcı olduklarını keşfettiler.

C++, hem endüstride hem de üniversitelerde en önemli dil haline geldi. C++ How to

Program (yazarın bu isimde bir kitabı bulunmaktadır) kitabını yazdığımızda hedefimiz açıktı.

Hiç programlama deneyimi olmayan ya da oldukça az programlama deneyimi olan üniversite

11

öğrencilerine, bilgisayar programlamaya giriş derslerinde yardımcı olacak ve daha üst

düzeyde C++ kursları için teorilerin daha derin ve daha ayrıntılı bir biçimde anlatılmasıyla,

pratiğe dökülmesi amacımızdı. Bu hedeflere ulaşabilmemiz için, diğer C++ kitaplarından çok

daha büyük bir kaynak oluşturduk. Ayrıca kitabımız Procedural programa, nesneye yönelik

programlama ve Generic Programlama hakkındaki yöntemleri öğretmektedir. Yüzbinlerce

insan bu kaynağı akademik kurslarda ve dünya çapındaki profesyonel seminerlerde

çalışmışlardır.

Çoğu insan, en iyi eğitim taktiğinin önce C'yi tam anlamıyla öğrenip daha sonra C++'ı

çalışmanın olduğunu düşünmektedir. Bu yüzden, bu kitabın 15 ile 23. üniteleri arasında "C++

how to program" adlı kitabımızdan özenle seçtiğimiz bölümleri ekledik. Bunun sizi, bu kitabı

bitirdikten sonra C++ hakkında daha detaylı bir çalışmaya yönlendireceğini umuyoruz.

1.11 DİĞER YÜKSEK SEVİYELİ DİLLER

Bugüne kadar yüzlerce yüksek seviyeli dil geliştirilmiştir ancak bunlardan yalnızca birkaçı şu

an kullanımdadır. FORTRAN ( Formula translator ), IBM tarafından 1954 ile 1957 yılları

arasında bilimsel uygulamalarda ve mühendislik uygulamalarında kullanılan matematik

hesaplamalarını yapmak için geliştirilmiştir. Fortran, özellikle mühendislik uygulamalarında

hala yaygın bir biçimde kullanılmaktadır.

COBOL ( Common Bussiness Oriented Language ) 1959'da, bilgisayar üreticileri, devlet ve

endüstriyel bilgisayar kullanıcıları tarafından geliştirilmiştir. COBOL, büyük verilerin

kullanılmasını gerektiren ticari uygulamalarda kullanılmaktadır. Bugün iş yazılımlarının

yarıdan fazlası COBOL ile programlanmaktadır.

Pascal ise C ile aynı zamanlarda, Profesör Niklaus Wirth tarafından akademik kullanım

amacıyla geliştirilmiştir. Bir sonraki kısımda Pascal hakkında daha fazla bilgi vereceğiz.

1.12 YAPISAL PROGRAMLAMA

1960'larda yazılım geliştirme çabaları ciddi zorluklarla karşılaştı. Yazılım takvimleri

genellikle geç kalıyordu, maliyetler yatırımları aşıyordu ve bitirilmiş projeler yeterince iyi

değildi. İnsanlar, yazılım geliştirmenin düşünülenden çok daha zor bir iş olduğunun farkına

vardılar. 1960'lardaki araştırma çalışmaları, yapısal programlamanın ortaya çıkmasına sebep

olmuştu. Yapısal programlama ile programları daha açık, daha doğru ve değiştirilmesi daha

kolay yazabilmek için bir disiplin oluşturulmuştu. 3. ve 4. üniteler yapısal programlamanın

temellerinin bir özetidir. Kitabın geri kalanında ise yapısal C programları geliştirmek

anlatılacaktır.

Bu araştırmanın en önemli sonuçlarından birisi, PASCAL programlama dilinin 1971 yılında

Profesör Niklaus Wirth tarafından geliştirilmesi olmuştur. PASCAL ismi 17. yüzyılın

matematikçi ve filozoflarından Blais PASCAL'dan geliyordu ve yapısal programlamayı

akademik çevrelerde öğretmek amacıyla tasarlanmıştı. Pascal, hızlı bir biçimde çoğu

üniversitelerde programlama dillerine girişte tercih edilir hale gelmişti fakat bu dil ticaret ve

endüstriyel uygulamalar ile hükümetin istediği uygulamaları gerçekleştirmek için çok önemli

bazı özelliklerden yoksundu. Bu yüzden de bu çevrelerde çok geniş bir kullanım alanı

bulamadı.

12

Ada programlama dili, United States Department of Defence ( DOD ) tarafından finanse

edilmişti ve 1970'lerle 1980'lerin başı arasında geliştirilmişti. Yüzlerce ayrı dil DOD'un ağır

emir ve yazılım kontrol sistemlerini oluşturmak amacıyla kullanılıyordu. DOD, bütün bu

işlemleri tek bir dilin yapmasını istiyordu. Öncelikle PASCAL temel alındı fakat en sonunda

kabul edilen Ada dili Pascal' dan oldukça farklıydı. Bu dil adını, şair Lord Byron'ın kızı Lady

Ada Lovelace'tan alıyordu. Lady Lovelace, tüm dünyada ilk bilgisayar programını yazan kişi

olarak bilinir (Charles Babbage'ın mekanik analitik hesap makinesi için program yazmıştır)

Ada’nın en önemli özelliklerinden birisi çoklu görevdir ( multitasking ) : Bu, programcıya

birden fazla işi paralel bir biçimde yapma fırsatı sunar. Diğer yüksek seviyeli dillerde (C ve

C++ dahil olmak üzere) programcı bir anda yalnızca bir işi yapabilir.

1.13 C PROGRAMI GELİŞTİRME ORTAMININ TEMELLERİ

C sistemleri genellikle bir kaç kısımdan oluşur : Program geliştirme ortamı, dilin kendisi, C

standart kütüphanesi. Aşağıdaki kısımda Şekil 1.1'de gösterilen C geliştirme ortamı

açıklanmıştır.

Tipik olarak bir C programı çalışmadan önce altı safhadan geçer ( Şekil 1.1 ). Bunlar : yazım

(edit), önişleme ( preprocess ), derleme ( compile ) , bağlama ( link ), yükleme ( load ),

çalıştırma (execute) olarak bilinir. Bu kitap, herhangi bir işletim sistemi detaylarından

bağımsız olarak yazılmasına rağmen bu kısımda UNIX tabanlı bir C sistemi incelenmiştir.

[Not : Bu kitaptaki programlar, çok az bir değişiklikle ya da hiç değişiklik yapmadan çoğu C

sisteminde (Microsoft Windows tabanlı sistemlerde dahil olmak üzere) çalışır.] Eğer bir

UNIX sistemi kullanmıyorsanız, yukarıda saydığımız görevleri nasıl gerçekleştireceğinizi

öğrenmek için kendi sisteminizin çalışma prensiplerini inceleyiniz.

İlk safha yazım aşamasıdır. Bu işi yapmak için bir editör program kullanılır. UNIX

sistemlerinde genellikle kullanılan editörler vi ve emacs dir. Borland C++ ve Microsoft

Visual C++ gibi birleştirilmiş program geliştirme ortamları ( IDE ) yazılım paketlerinde,

editörler paketin içindedir. Okuyucunun bir programı yazabileceğini kabul ediyoruz.

Programcı bir C programını editörle yazar ve eğer gerekiyorsa düzeltmeleri yaptıktan sonra

programını disk gibi ikincil depolama cihazlarından birine kaydeder. C programlarının

uzantıları .c ile bitmelidir.

Daha sonra kullanıcı, programa derle ( compile ) komutunu verir. Derleyici (compiler) , C

programını makine diline ( nesne kodlarına ) çevirir. C sistemlerinde önişlemci, derleyicinin

çevrim safhası başlamadan otomatik olarak çalışır. C önişlemcisi, önişlemci komutları

(preprocessor directives) denilen özel komutlara uyar. Bu komutlar, program derlenmeden

önce program üzerinde çeşitli işlemlerin yapılmasını sağlar. Bu işlemler genelde, derlenecek

dosyanın içine başka dosyalar katmak ve özel sembolleri program metninin içine yerleştirmek

için yapılır. 13. ünitede bu konu detaylı olarak anlatılacaktır.

Daha sonraki safha, bağlama ( link ) safhasıdır. C programları genelde kütüphanelerde ya da

belirli bir proje üzerinde çalışan programcıların oluşturduğu özel kütüphanelerde tanımlanmış

fonksiyonlar içerebilir. C derleyicisi tarafından oluşturulan makine kodunda bu kısımlar boş

olarak bırakılır. Bağlayıcı (linker), makine koduyla kütüphanelerde tanımlanmış fonksiyonları

birleştirme işlemini yaparak, tamamlanmış çalıştırılabilir programı oluşturur. UNIX tabanlı

bir sistemde bir programı derlemek ve bağlamak için cc komutu kullanılır. merhaba.c adında

bir programı derlemek ve bağlamak için UNIX sistemlerinde

cc merhaba.c

13

yazılır ve giriş ( enter ) tuşuna basılır (Not: UNIX komutları büyük harf ve küçük harf

duyarlıdır. Bu yüzden cc yazarken küçük harf kullanmanız gerekir). Eğer program doğru bir

biçimde derlenir ve bağlanırsa a.out adında bir dosya oluşturulur. Bu, merhaba.c

programımızın çalıştırılabilir biçimidir.

Editör

Ön İşlemci

Derleyici

Bağlayıcı

Yükleyici

Disk

CPU

Disk

Disk

Disk

Disk

Program editörde

yaratılır ve diske

kaydedilir

Ön işlemci

kodları işler

Derleyici nesne

kodlarını üretir ve

diske kaydeder

Bağlayıcı nesne

kodunu

kütüphanelerle

birleştirir ve a.out

yaratarak diske

kaydeder

Yükleyici

programı

hafızaya koyar

CPU her

komutu alır

ve çalıştırır.

Program

çalışırken

muhtemelen

yeni veri

değerlerini

kaydeder

Birincil Hafıza

Birincil Hafıza

Şekil 1.1 Tipik bir C ortamı

14

Daha sonraki safha, yükleme ( loading ) safhasıdır. Bir programın çalıştırılabilmesi için

hafızaya yerleştirilmesi gerekir. Bu iş, yükleyici ( loader ) tarafından yapılır. Yükleyici,

programın çalıştırılabilir biçimini diskten alıp hafızaya yerleştirir. Programı destekleyen bazı

özel bileşenler, ortak kütüphanelerden alınarak programla birlikte yüklenir.

Son olarak bilgisayar, CPU kontrolü altında her komutu teker teker çalıştırır. Bir UNIX

sisteminde bir programı yükleyip çalıştırmak için a.out yazılıp giriş tuşuna basılır.

Programlar, her zaman ilk denemede çalıştırılamayabilir. Az önce saydığımız safhalardan

herhangi birinde çeşitli hatalar oluşmuş olabilir. Örneğin, çalıştırılan bir program bir sayıyı 0'a

bölmeye çalışabilir. ( Bu işlem bilgisayarlarda yasaktır ) Bu, bilgisayarın bir hata mesajı

yazdırmasına sebep olur. Programcı, yazım aşamasına geri dönüp gereken düzeltmeleri yapıp,

geri kalan safhaları yeniden gerçekleştirip programını yeniden çalıştırmalıdır.

Genel Programlama Hataları 1.1

0'a bölme gibi hatalar programın çalışma anında ortaya çıkar. Bu yüzden, bu hatalara

çalışma zamanı hataları ( RUNTIME/EXECUTION TIME ERROR ) denir. 0'a bölmek

ölümcül bir hatadır. Ölümcül hatalar, programın başarılı bir biçimde tamamlanamadan

sonlanması anlamına gelir. Ölümcül olmayan hatalar ise programın yanlış sonuçlar verecek

biçimde çalışmasına sebep olur. ( Not:Bazı sistemlerde 0'a bölmek ölümcül bir hata değildir.)

Çoğu C programı, veri girişi ya da çıkışı yapar. Belirli C fonksiyonları, verileri stdin

(standard input stream ) adı verilen standart giriş birimlerinden alır. Standart giriş birimi

genellikle klavyedir. Fakat stdin başka bir birime de bağlanabilir. Veriler genellikle, stdout

(standard output stream ) adı verilen standart çıkış birimlerinden alınır. stdout genellikle

bilgisayar ekranıdır fakat farklı birimlerde kullanılabilir. Veriler, disk ya da yazıcı gibi diğer

birimlere de verilebilir. Ayrıca, stderr ( standard error stream) adı verilen standart hata birimi

vardır. stderr, birimleri genellikle ekrana bağlıdır ve hata mesajlarını göstermekte kullanılır.

Çıkış verileri genellikle, stderr ekranla ilişkili iken ekrandan başka bir cihaza yönlendirilir.

Böylelikle, çıkış verilerinde hata kontrolü yapılmış olur ve kullanıcı hatalardan anında

haberdar olur.

1.14 DONANIM EĞİLİMLERİ

Programlama, donanım, yazılım ve haberleşme teknolojilerindeki inanılmaz gelişime

sayesinde başarılı olmuştur. Her sene insanlar, ürünler ve servisler için daha fazla para

ödemeyi beklerler. Bunun tam tersi, bilgisayar ve haberleşme alanlarında yaşanmıştır. Bu

zıtlığın sebebi, bu teknolojileri destekleyen donanımların maliyetlerinin hızlı bir biçimde

düşmesidir. Yıllardır, donanım maliyetleri sürekli olarak düşmüştür. Her iki yılda bir,

bilgisayarların kapasiteleri özellikle de programların çalıştırıldığı hafıza miktarı, verilerin

uzun süre için tutulduğu ikincil depolama alanları ( diskler ), programların çalıştırılmasındaki

hızları belirleyen işlemci hızlarının ikiye katlanması muhtemeldir. Aynı gelişme, haberleşme

alanında da yaşanmıştır. Fiyatların cazip hale gelişi ve son yıllarda haberleşme bant

genişliğine olan talebin artışı büyük bir rekabeti doğurmuştur. Teknolojinin başka hiçbir

alanında fiyatlar bu kadar hızlı düşmemiş ve teknoloji bu kadar hızlı ilerlememiştir.

60'larda ve 70'lerde bilgisayar kullanımı yaygınlaştığında, bilgisayarların ve haberleşmenin

insan yaratıcılığını çok büyük bir şekilde geliştireceği konuşulmuştu. Fakat bu gelişmeler

gerçekleşmedi. Organizasyonlar, bilgisayarları çok fazla kullanmalarına rağmen beklenen

verimi alamadılar. Mikroçip teknolojisinin 1980'lerin başında keşfedilmesiyle 1990'larda

15

yaşanacak olan verim artışı başlamış oldu. ARPA finanse ettiği öğrencileri, Illinois

Üniversitesinde bir araya getirerek, öğrencilerin fikirlerini ve çalışmalarını paylaşacakları bir

konferans düzenledi. Bu konferans boyunca ARPA, finanse ettiği öğrenciler ve araştırma

enstitüleri arasında bir ağın tasarılarını açıkladı. Bu ağ, insanlar telefon hatlarıyla

bilgisayarlara saniyede 110 bit ile ulaşırken, saniyede 56000 bitlik bir haberleşme hattıyla

bağlanacaktı.

1.15 INTERNET'İN TARİHİ

1960'larda yazarlardan Harvey M.Deitel MIT'de öğrenciydi. Yazarın MIT'deki araştırması

olan Mac projesi, Amerikan Savunma Bakanlığına bağlı Advanced Research Projects Agency

(ARPA) tarafından finanse ediliyordu. Massachusets Harvard'taki araştırmacılar,

UNIVAC1108 süper bilgisayarları sayesinde, UTAH üniversitesindeki bilgisayarlarla

haberleşmeyi düşünmüşlerdi. Bu sayede, UTAH'taki bilgisayarlarda grafik üzerine yapılan

araştırmalarda gerekli olan hesaplar yapılabilecekti. Bunun gibi bir çok heyecan veren olasılık

daha ortaya çıktı. Akademik çalışmalar dev bir adım atmak üzereydi. Bu konferansın hemen

ardından ARPA, bugünkü Internet’in temeli olan ARPANET haline geldi.

İşler ilk başta planlanandan daha farklı gelişti. ARPANET araştırmacılara birbirlerinin

bilgisayarlarını kullanma fırsatı sunarken ; esas faydası elektronik posta ( e-mail ) olarak

bilinen hızlı ve basit haberleşmeyi sağlamasıydı. Elektronik posta, bugünde Internet üzerinde

milyonlarca insanın birbirleriyle iletişim kurmalarını sağlamaktadır.

ARPA'nın bu ağ için en önemli amaçlarından birisi, kullanıcıların aynı haberleşme yolu

üzerinden ( telefon hattı gibi ) aynı anda bilgileri gönderebilmeleri ve alabilmelerini

sağlamaktı. Packet switching tekniği ile işletilen ağda, dijital veri, küçük paketler halinde

gönderilir. Bu paketler veriyi, adresi, hata kontrol bilgisini ve dizi bilgisini içerir. Adres

bilgisi, paketleri varacağı yere kadar yönlendir. Dizi bilgisi, paketlerin birleştirilmesini ve bu

sayede orijinal biçimine geri dönüşümü sağlar. Bir çok kişinin paketleri aynı hat içinde karışık

halde bulunur. Packet switching tekniği, haberleşme maliyetlerini bilgisayarlar arasında ayrı

hatların kullanımına göre azaltır.

Ağ, herhangi bir merkezi kontrol olmadan çalışabilecek biçimde tasarlanmıştır. Bu olay, ağın

herhangi bir kısmı çökse bile çalışan kısımların paketleri alternatif yollardan göndermesi

anlamına gelir. ARPANET'in haberleşmesini sağlayan protokol, TCP ( Transmission Control

Protocol ) olarak bilinir. TCP, göndericiden alıcıya kadar mesajların doğru bir şekilde

yönlendirilmesini garanti eder.

Internet'in ilk kullanımlarına paralel olarak, dünyadaki organizasyonlar kendi ağlarını

oluşturmaya başladılar. Bu ağlar, şirket içinde intra-organizasyon, şirketler arasında interorganizasyon olarak adlandırılır. Ağlarla ilgili bir çok donanım ve yazılımın ortaya çıkması

da bu zamanlara denk gelir. Bir zorluk ise donanım ve yazılımı birbiriyle birleştirmektir.

ARPA bunu, IP ( Internetworking Protocol ) geliştirerek yapmıştır ve böylece ağların ağını

oluşturmuştur. Bu sistemde, bugünkü Internet’in temelini oluşturmuştur. İki protokolün

birleşimi olan TCP/IP günümüzde geniş bir kullanıma sahiptir.

İlk başlarda Internet, üniversiteler ve araştırma enstitüleri arasında sınırlıydı. Daha sonra ordu,

büyük bir kullanıcı haline geldi. Son olarak da hükümet, Internet’in ticari amaçlar için

kullanımına izin verdi. İlk başlarda, araştırma grupları ve askeriyede, ağın çok fazla kullanıcı

tarafından kullanılmasından dolayı cevap alma süresinin düşeceği gibi bir düşünce vardı.

16

Gerçekte bunun tam tersi oldu. Şirketler Internet’i verimli bir hale getirerek, müşterilerine

daha iyi ve yeni servisler sunabileceklerinin farkına vardılar ve Internet’i geliştirmek ve

genişletmek için büyük yatırımlar yaptılar. Bu, donanım ve yazılım sağlayanlar arasında

büyük bir rekabet oluşturdu. Sonuç olarak, Internet’te bant genişliği ( haberleşme hatlarının

bilgi taşıma kapasitesi ) hızlı bir biçimde artarken, fiyatlar düştü. Amerika Birleşik Devletleri

ve diğer endüstriyel devletlerin ekonomisinde Internet’in büyük bir rol oynadığı düşünülür.

1.16 WWW' in TARİHİ

World Wide Web, bilgisayar kullanıcılarının Internet üzerinden multimedya tabanlı belgelere

(grafik, animasyon, ses ve videoları içeren belgeler ) ulaşmasını ve görüntülemesini sağlar.

Internet neredeyse otuz yıl önce geliştirilmesine rağmen, www henüz oldukça yenidir. 1990

yılında CERN(Parçacık Fiziği Avrupa Laboratuarı) www'i ve haberleşme protokollerini

geliştirdi.

Internet ve www, insanoğlunun yaptığı en önemli gelişmeler arasında gösterilir. Geçmişte

bilgisayar uygulamaları standalone bilgisayarlarda çalışırdı. Bugünkü uygulamalar ise dünya

üzerinde milyonlarca bilgisayar arasında haberleşmeyi sağlayacak şekilde yazılabilir. Internet,

hesaplama ve haberleşme teknolojilerini birleştirir ve işimizi kolaylaştırır. Bilgilerin, güvenilir

ve hızlı bir biçimde dünya çapından ulaşılabilirliğini sağlar. Kişilerin ve küçük işletmelerin

dünyaya açılmalarını sağlar ve bu sebeplerden bütün iş yaşantısını değiştirmiştir. İnsanlar bir

ürünün ya da servisin mümkün olan en iyi fiyatını araştırabilirler. Belli bir konu üstünde

gruplar birbirleriyle haberleşebilirler. Araştırmacılar dünya çapındaki en son gelişmeleri

anında takip edebilirler.

Akademik kullanım için Internet ve web programcılığı hakkında temel prensipleri

anlattığımız iki kitabımız bulunmaktadır :

Internet & World Wide Web How to Program ve e-bussiness and e-commerce how to

program.

1.17 C ve BU KİTAP HAKKINDA GENEL NOTLAR

Bazı tecrübeli C programcıları garip, içice geçmiş ve farklı biçimlerde program yazmakla

övünürler. Bu, aslında oldukça zayıf bir programcılık örneğidir. Programların daha zor

okunmasına, daha garip davranmasına, daha zor test edilmesine ve hatalarının daha zor

ayıklanmasına, ayrıca değişen durumlara daha zor adapte olmasına sebep olur. Bu kitap,

programlamaya yeni başlayanlar için oluşturulmuştur. Bu yüzden, programlarda açıklık temel

hedefimizdir. Aşağıda ilk iyi programlama alıştırma tavsiyenizi bulacaksınız.

İyi Programlama Alıştırmaları 1.1

C programlarınızı basitçe ve doğrudan yazın. Programlarınızı gereksiz kullanımları

deneyerek uzatmayın.

C'nin taşınılabilir bir dil olduğunu ve C'de yazılan bir programın farklı bilgisayarlarda

çalışabileceğini biliyorsunuz. Taşınırlık, en önemli hedeflerden biridir. ANSI C standart

belgeleri içinde, taşınırlık konuları hakkında oldukça fazla bilgi bulunur ve yalnızca taşınırlığı

anlatan kitaplarda vardır.

17

Taşınırlık İpuçları 1.3

Taşınılabilir programlar yazmak mümkün olsa da bazen farklı C derleyicileri ve farklı

bilgisayarlar taşınırlığı oldukça zor bir hale getirebilirler. C 'de sadece programlar yazmak

taşınırlığı garanti etmez. Programcı çoğu zaman, değişik bilgisayar sistemleriyle uğraşmak

zorunda kalabilir.

C standart dokümanları üzerinde oldukça titiz bir çalışma yapmamıza rağmen kimi zaman

kitabın çalışılabilirliğini artırmak maksadıyla bazı noktaları atlamak zorunda kaldık. C, zengin

bir dil olduğu için dilde bazı alt konuların ve bazı ileri başlıkların bu kitapta bulunmaması

normaldir. Eğer C hakkında daha fazla teknik detaya ihtiyaç duyarsanız, C standart

dokümanlarının kendisini ve Kernighan ve Ritchie tarafından yazılmış kitabı okumanızı

tavsiye ediyoruz.

Bu kitapta anlattıklarımızı ANSI/ISO C ile sınırlı tuttuk. Bu sürümün bazı özellikleri eski C

sürümleriyle uyumlu olmayabilir. Bu yüzdende, bu kitaptaki programları eski derleyicilerle

derlediğinizde bazı sorunlarla karşılaşabilirsiniz.

İyi Programlama Alıştırmaları 1.2

Kullandığınız C versiyonunun talimatlarının okuyunuz. Böylece size sunduğu imkanları doğru

bir biçimde kullanarak daha iyi programlar yazabilirsiniz.

İyi programlama Alıştırmaları 1.3

Bilgisayarınız ve derleyiciniz iyi birer öğretmendir. Eğer C' de bir özelliğin nasıl çalıştığına

emin olamazsanız bu özelliği içeren küçük bir program yazın ve çalıştırın. Hatalarınızı

öğretmeniniz söyleyecektir.

ÖZET

 Yazılım (bilgisayarın işlemler yapması ve karar vermesi için yazılan emirler),

bilgisayarı kontrol eder (genellikle donanım olarak adlandırılır).

 ANSI C, C programlama dilinin, 1989 yılında Amerika'da American National

Standards Instute(ANSI) ve tüm dünyada Internationals Standards Orgranization(ISO)

tarafından standart hale getirilmiş sürümüdür.

 C'in yeni bir sürümü(C99) geliştirilmiştir ancak henüz bu yeni sürümün derleyicileri

bulunmamaktadır.

 25 yıl önce büyük odaları dolduran ve milyonlarca dolara mal olan bilgisayarlar bugün

tırnağımızdan daha küçük silikon çipler üzerinde yalnızca bir kaç dolara mal

edilmektedir.

 Dünya çapında, insanlara iş, endüstri, hükümet ve kendi kişisel yaşamlarında yardımcı

olan genel amaçlı 150 milyon bilgisayar bulunmaktadır. Bir kaç yıl içinde bu sayı

ikiye katlanacaktır.

 Bilgisayar, insandan milyonlarca ve hatta milyarlarca kez hızlı bir biçimde karar

verme ve işlem yapma yeteneğine sahip cihazlardır.

 Bilgisayarlar, verileri bilgisayar programları kontrolünde işlerler.

 Donanım olarak bilinen çeşitli cihazlar (klavye, ekran, disk, hafıza, işlemci üniteleri)

bir araya gelerek bir bilgisayar sistemi oluştururlar.

 Bilgisayarda çalışan programlar yazılım olarak adlandırılır.

18

 Giriş ünitesi, bilgisayarın bilgileri aldığı kısımdır. Bugünkü bilgisayarlarda bilgi

genellikle klavye sayesinde girilir.

 Çıkış ünitesi, bilgisayarın bilgi çıkışının alındığı kısmıdır. Günümüzde bilgiler

genellikle ekranda ya da kağıt üzerinde alınır.

 Hafıza, bilgisayarın verileri depoladığı kısımdır ve genellikle hafıza yada birincil

hafıza olarak adlandırılır.

 Aritmetik Mantık Ünitesi (ALU), işlemler yapar ve kararlar verir.

 Merkezi İşlemci Ünitesi (CPU), bilgisayarın koordinasyonundan ve diğer kısımları

kontrolünden sorumludur.

 Diğer üniteler tarafından kullanılmayan programlar ya da veriler yeniden kullanılana

kadar genellikle ikincil hafıza araçlarına (disk) kaydedilir.

 Tek kullanıcılı yığın işlemede, bilgisayar, verileri gruplar yada yığınlar halinde

işlerken yalnızca bir program çalıştırır.

 İşletim sistemleri, bilgisayarlardan en iyi performansı almak ve bilgisayarları en uygun

biçimde kullanabilmeyi sağlamak için yazılmış yazılım sistemleridir.

 Çoklu programlama kullanım sistemleri, bilgisayarda birden fazla işi eş zamanlı olarak

yapabilmemizi sağlar. (Bilgisayar, kaynaklarını işler arasında paylaştırır.)

 Zaman paylaşımı, çoklu programlamanın kullanıcının bilgisayara terminaller

sayesinde ulaştığı özel bir halidir. Kullanıcılar eş zamanlı olarak çalışıyormuş gibi

görünür.

 Çoklu kullanımda bir organizasyonun bütün hesap işleri, işlemlerin yapıldığı sitelere

ağ sayesinde dağıtılır.

 Sunucular, çoklu kullanım sırasında istemcilerin ihtiyaç duyabileceği program ve

verileri depolar. Buna, istemci/sunucu sistemler denir

 Bir bilgisayar yalnızca kendine ait makine dilini anlayabilir. Makine dilleri genellikle

bilgisayara işlemler yaptırabilmek için sayıların belirli dizilerini içerir. (Çoğunlukla bu

sayılar 1 ve 0'lara indirgenir.) Makine dilleri her makinede farklılık gösterebilir.

 İngilizce’ye yakın kısaltmalar assembly dillerinin temelini oluşturmuştur. Assembler

adı verilen programlar bu dilde yazılmış programları makine diline çevirir.

 Derleyiciler, yüksek seviyeli dillerle yazılmış programları makine diline çevirir.

Yüksek seviyeli diller İngilizce kelimeler ve geleneksel matematik gösterimlerini

içerir.

 Yorumlayıcı programlar, yüksek seviyeli dillerle yazılmış programları derlemeden

çalıştırabilir.

 Derlenmiş programlar, yorumlanmış programlardan daha hızlı çalışmalarına rağmen,

yeni özelliklerin ekleneceği ve hataların düzeltileceği program geliştirme ortamlarında

yorumlayıcılar daha çok kullanılırlar. Bir program geliştirildikten sonra derlenmiş

sürümü oluşturularak daha verimli çalışması sağlanır.

 C, UNIX işletim sisteminin geliştirme dili olarak bilinir.

 C ile çoğu bilgisayarda çalışabilecek programlar yazmak mümkündür.

 ANSI C standardı, 1989 yılında onaylanmış, 1999 yılında gözden geçirilmiştir.

 FORTRAN matematik işlemleri için, COBOL ticari uygulamalar için kullanılır.

 Yapısal programlama, daha açık, test etmesi, hata ayıklaması ve değiştirilmesi daha

kolay programlar yazmak için geliştirilmiş bir yaklaşımdır. PASCAL, yapısal

programlamayı akademik çevrelerde öğretmek için geliştirilmiştir.

 ADA, PASCAL temel alınarak, United States Department of Defence (DOD)

tarafından finanse edilmiştir.

 Çoklu görev, programcıların paralel işler yapabilmesini sağlar.

19

 Bütün C sistemleri üç kısım içerir: Ortam, dil ve standart kütüphaneler. Kütüphane

fonksiyonları C dilinin kendisinin parçası değildir. Bu fonksiyonlar, giriş/çıkış ve

matematik hesaplamaları gibi işlemleri gerçekleştirir.

 C programları çalıştırılana kadar altı safhadan geçer: Yazım, önişleme, derleme,

bağlama, yükleme ve çalıştırma.

 Bir programcı, programını ve gerekli düzeltmeleri editör adı verilen programlarla

yapar. C dosya isimleri, UNIX tabanlı sistemlerde .c uzantısı ile biter.

 Derleyici, C programını makine dili koduna (nesne kodu) çevirir.

 C önişlemcisi, derlenecek dosyanın içine bazı özel işaretler yerleştirmek ve dosyalar

eklemek gibi önişlemci komutlarını yerine getirir.

 Bağlayıcı, çalıştırılabilir bir programı oluşturmak için nesne kodu ile fonksiyonları

birleştirir. Tipik bir UNIX tabanlı sistemde, C++ programını derlemek ve çalıştırmak

için cc komutu kullanılır. Eğer program düzgün bir biçimde derlenir ve bağlanırsa

a.out asında bir dosya oluşturulur. Bu, programın çalıştırılabilir halidir.

 Yükleyici, çalıştırılabilir programı diskten alıp hafızaya gönderir.

 Sıfıra bölme gibi hatalar program çalışırken ortaya çıkar. Bu sebepten, çalışma zamanı

hataları olarak adlandırılır.

 Sıfıra bölmek genellikle ölümcül bir hatadır. Ölümcül hatalar programın, hiçbir çıktı

üretmeden aniden sonlanmasına sebep olur. Ölümcül olmayan hatalar, programın

yanlış çıktı oluşturmasına sebep olur.

 Bir bilgisayar, CPU kontrolü altında programın her emrini bir anda çalıştırır.

 Belli C fonksiyonları (scanf gibi) bilgilerini stdin’den alır. stdin genellikle klavyedir.

Verinin, stdout sayesinde çıktısı alınır. stdout genellikle ekrandır.

 stderr hata mesajlarını göstermek için kullanılır.

 Değişik C sistemleri ve değişik bilgisayarlarda bir çok farklılık ortaya çıkabilir. Bu

yüzden, taşınırlık program yazarken büyük önem kazanır.

 C++ nesneye yönelik programlama yeteneklerini içerir.

 Nesneler, gerçek dünyadaki araçları modelleyen yeniden kullanılabilir yazılım

parçalarıdır.

ÇEVRİLEN TERİMLER

.c extension.................................................................c uzantısı

arithmetic and logic unit (ALU )................................aritmetik ve mantık ünitesi

batch processing.........................................................yığın işleme

building block approach.............................................blokları yerleştirme yaklaşımı

C preprocessor............................................................C önişlemcisi

C standart library........................................................C standart kütüphanesi

Central processing unit (CPU )..................................merkezi işleme ünitesi

clarity.........................................................................açıklık

client...........................................................................istemci

client/server computing..............................................istemci/sunucu kullanımı

compiler......................................................................derleyici

data.............................................................................veri

distributed computing................................................çoklu kullanım

editör..........................................................................editör

environment...............................................................ortam

execute a program......................................................bir programı çalıştırmak

file server...................................................................dosya sunucusu

20

function....................................................................fonksiyon

functionalization......................................................fonksiyonellik

hardware..................................................................donanım

hardware platform...................................................donanım platformu

high-level language.................................................yüksek seviyeli dil

input unit.................................................................giriş ünitesi

input/output.............................................................giriş/çıkış

linker........................................................................bağlayıcı

loader.......................................................................yükleyici

logical units.............................................................mantıksal birimler

machine dependent..................................................makine bağımlı

mechine independent...............................................makine bağımsız

memory....................................................................hafıza

memory unit.............................................................hafıza birimi

multiprogramming...................................................çoklu programlama

multitasking.............................................................çoklu görev

object.......................................................................nesne

object code...............................................................nesne kodu

object-oriented programming..................................nesneye yönelik programlama

output device............................................................çıkış cihazı

personal computer....................................................kişisel bilgisayar

portability.................................................................taşınırlık/taşınılabilirlik

primary memory.......................................................birincil hafıza

run a program...........................................................bir programı çalıştırmak

screen........................................................................ekran

software.....................................................................yazılım

software reusability...................................................yazılımın yeniden kullanılabilirliği

standart error ( stderr )...............................................standart hata

standart input ( stdin )................................................standart giriş

standart output ( stdout )............................................standart çıkış

stored program...........................................................depolanmış program

structured programming............................................yapısal programlama

supercomputer...........................................................süper bilgisayar

task.............................................................................görev

timesharing................................................................zaman paylaşımı

top-down, stepwise refinement.................................yukarıdan aşağıya,adımsal iyileştirme

translator program.....................................................çevirici program

workstation................................................................iş istasyonu

ÖZEL İSİM VE KISALTMALAR

Ada COBOL

ALU CPU

ANSI C FORTRAN

C Pascal

C++ UNIX

21

GENEL PROGRAMLAMA HATALARI

1.1 Sıfıra bölme gibi hatalar programın çalışma anında ortaya çıkar. Bu yüzden, bu hatalara

çalışma zamanı hataları (RUNTIME/EXECUTION TIME ERROR) denir. 0'a bölmek

ölümcül bir hatadır. Ölümcül hatalar, programın başarılı bir biçimde tamamlanamadan

sonlanması anlamına gelir. Ölümcül olmayan hatalar ise programın yanlış sonuçlar

verecek biçimde çalışmasına sebep olur.(Not:Bazı sistemlerde 0'a bölmek ölümcül bir

hata değildir.)

İYİ PROGRAMLAMA ALIŞTIRMALARI

1.1 C programlarınızı basitçe ve doğrudan yazın. Programlarınızı gereksiz kullanımları

deneyerek uzatmayın.

1.2 Kullandığınız C versiyonunun talimatlarının okuyunuz. Böylece size sunduğu imkanları

doğru bir biçimde kullanarak daha iyi programlar yazabilirsiniz.

1.3 Bilgisayarınız ve derleyiciniz iyi birer öğretmendir. Eğer C'de bir özelliğin nasıl

çalıştığına emin olamazsanız bu özelliği içeren küçük bir program yazın ve çalıştırın.

Hatalarınızı öğretmeniniz söyleyecektir.

PERFORMANS İPUÇLARI

1.1 Kendi yazdığınız fonksiyonlar yerine, ANSI standart kütüphane fonksiyonlarını

kullanmak programın performansını artırır. Çünkü bu fonksiyonlar verimli çalışmaları

için özenle yazılmıştır.

TAŞINABİLİRLİK İPUÇLARI

1.1 C donanıma bağımlı olmadığından, C'de yazılacak bir program değişiklik yapılmadan ya

da çok az değişiklikle bir çok bilgisayarda sorunsuz olarak çalışabilir.

1.2 Kendi yazdığınız fonksiyonlar yerine, ANSI standart kütüphane fonksiyonlarını

kullanmak taşınırlığı artırır. Çünkü bu fonksiyonlar bütün ANSI C koşullarına uyarlar.

1.3 Taşınılabilir programlar yazmak mümkün olsa da bazen farklı C derleyicileri ve farklı

bilgisayarlar taşınırlığı oldukça zor bir hale getirebilirler. C'de sadece programlar yazmak

taşınırlığı garanti etmez. Programcı çoğu zaman, değişik bilgisayar sistemleriyle uğraşmak

zorunda kalabilir.

YAZILIM MÜHENDİSLİĞİ GÖZLEMİ

1.1 Yeniden kullanılabilir yazılım parçalarından oluşan genel kütüphaneler Internet üzerinde

bulunabilir. Bunlardan bazıları ücretsizdir.

ÇÖZÜMLÜ ALIŞTIRMALAR

1.1 Aşağıdaki boşlukları doldurunuz.

22

a) Kişisel bilgisayar kavramını ortaya çıkartan şirket \_\_\_\_\_\_\_\_ dır.

b) Kişisel kullanımı endüstriyel kullanıma uygun hale getiren \_\_\_\_\_\_\_\_ dır.

c) Bilgisayarlar, bilgisayar \_\_\_\_\_\_\_\_ adı verilen bir takım emirler sayesinde veri işlerler.

d) Bilgisayarın mantıksal olarak altı birimi \_\_\_\_\_\_\_\_ ,\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_,

\_\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_\_ dır.

e) \_\_\_\_\_\_\_\_ çoklu programlamanın, kullanıcıların bilgisayara terminal adı verilen özel

cihazlarla ulaştığı özel bir halidir.

f) Bu ünitede bahsedilen dillerin sınıfları \_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_\_ dır.

g) Yüksek seviyeli dilleri, makine dillerine çeviren programlara \_\_\_\_\_\_\_\_ denir.

h) C dili \_\_\_\_\_\_\_\_ işletim sisteminin geliştirilmesinde kullanılmıştır.

i) Bu kitap C'nin American National Standards Institute tarafından standart hale

getirilmiş \_\_\_\_\_\_\_\_ versiyonunu kullanmaktadır.

j) \_\_\_\_\_\_\_\_ dili Wirth tarafından, üniversitelerde yapısal programlamayı öğretmek

amacıyla geliştirilmiştir.

k) Department of Defence, Ada dilini programcıların bir çok işi paralel bir biçimde

yapabilmelerini sağlayan \_\_\_\_\_\_\_ yeteneği ile geliştirmiştir.

1.2 Aşağıdaki C ortamı hakkında yazılmış cümlelerin içinde ki boşluları doldurunuz.

a) C programları bilgisayarlara genellikle \_\_\_\_\_\_\_\_ programları kullanılarak yazılır.

b) Bir C sisteminde \_\_\_\_\_\_\_\_ programı çevirim süreci başlamadan önce otomatik olarak

çalışır.

c) Ön işlemci komutlarının en yaygın kullanımları \_\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_\_ yapmaktır.

d) \_\_\_\_\_\_\_\_ programı derleyicinin çıktısıyla çeşitli kütüphane fonksiyonlarını

birleştirerek çalıştırılabilir programı oluşturur.

e) \_\_\_\_\_\_\_\_ programı çalıştırılabilir programı diskten hafızaya aktarır.

f) UNIX sistemlerinde derlenmiş bir programı yükleyip çalıştırmak için \_\_\_\_\_\_\_\_

yazılır.

ÇÖZÜMLER

1.1 a) Apple. b) IBM Personal Computer. c) programlar. d) giriş birimi, çıkış birimi, hafıza

birimi, aritmetik mantık ünitesi(ALU), merkezi işleme ünitesi, ikincil depolama ünitesi. e)

23

zaman paylaşımı. f) makine dilleri, assembly dilleri, yüksek seviyeli diller. g) derleyiciler. h)

UNIX. i) ANSI. j) Pascal. k) çoklu görev

1.2 a) editör. b) ön işlemci. c) derlenecek dosyanın içindeki diğer dosyaları içermek,özel

karakterler yerine program metnini yazmak. d) bağlayıcı. f) yükleyici. e) a.out

ALIŞTIRMALAR

1.3 Aşağıdakileri yazılım veya donamım olarak sınıflandırınız.

a) CPU

b) C derleyicisi

c) ALU

d) C ön işlemcisi

e) Giriş birimi

f) Kelime işleme programı

1.4 Programınızı neden makine bağımlı bir dil yerine makine bağımsız bir dille yazmayı

tercih dersiniz? Makine bağımlı diller neden bazı tipte programları yazmak için uygun

olabilir?

1.5 Çevirici programlar, örneğin assembly çeviricileri ve derleyiciler, programları bir dilden

diğer bir dile (kaynak dil esas alınarak) çevirirler. Aşağıdakilerden hangileri doğru

hangileri yanlıştır?

a) Derleyici, yüksek seviyeli dillerin programlarını makine diline çevirir.

b) Assembler, kaynak dil programlarını makine diline çevirir.

c) Derleyici, kaynak dil programlarını makine diline çevirir.

d) Yüksek seviyeli diller makine bağımlıdır.

e) Makine dilindeki bir program çalışmadan önce başka bir dile çevrilmeye ihtiyaç

duyar.

1.6 Aşağıdaki boşlukları doldurunuz.

a) Kullanıcıların zaman paylaşımlı bilgisayar sistemlerine erişmek için kullandıkları

araçlara genellikle \_\_\_\_\_\_\_\_ denir.

b) Assembly dilinin programlarını makine dili programlarına çeviren programlara

\_\_\_\_\_\_\_\_ denir.

c) Bilgisayarın, dışarıdan bilgi alan mantık birimine \_\_\_\_\_\_\_\_ denir.

d) Bilgisayarın belli problemleri çözmesi için bilgisayarın programlama sürecine

\_\_\_\_\_\_\_\_ denir.

e) Hangi tür bilgisayar dili, makine dili komutları için İngilizce kısaltmalar

kullanır?\_\_\_\_\_\_\_\_\_\_\_\_

f) Bilgisayarın altı mantık birimini sayınız.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

g) Bilgisayarın hangi mantıksal birimi bilgisayar tarafından işlenmiş verilerin

bilgisayarın dışında kullanılabilmesi için çeşitli cihazlara gönderir? \_\_\_\_\_\_\_\_\_

h) Belli bir programlama dilinde yazılmış olan programı makine diline çeviren

programların genel adı \_\_\_\_\_\_\_\_ dır.

i) Bilgisayarın hangi mantık birimi bilgiyi saklar ? \_\_\_\_\_\_\_\_.

24

j) Bilgisayarın hangi mantık birimi hesaplamalar yapar ? \_\_\_\_\_\_\_\_.

k) Bilgisayarın hangi mantık birimi mantık kararları verir ? \_\_\_\_\_\_\_\_.

l) Genellikle bilgisayarın kontrol biriminin kısaltması \_\_\_\_\_\_\_\_ dır.

m) Programcının hızlı ve kolay bir şekilde programlama yapmasına uygun olan

programlama dili seviyesi \_\_\_\_\_\_\_\_ dir.

n) Bugünlerde en çok kullanılan işe dayalı dil \_\_\_\_\_\_\_\_ dir.

o) Bilgisayarın doğrudan anlayabileceği tek dil bilgisayarın \_\_\_\_\_\_\_\_ dilidir.

p) Bilgisayarın hangi mantık birimi bütün diğer mantık birimlerinin çalışmalarını

düzenler?\_\_\_\_\_\_\_\_\_\_\_\_\_

1.7 Aşağıdakilerden hangilerinin doğru hangilerinin yanlış olduğuna karar veriniz. Yanlış

olanları açıklayınız.

a) Makine dilleri makine bağımlıdır.

b) Zaman paylaşımı bir çok kullanıcının eş zamanlı olarak bir bilgisayarı kullanması

demektir.

c) Diğer yüksek seviyeli diller gibi C dili de genellikle makine bağımlı olarak kabul

edilir.

1.8 Aşağıdakilerin tanımlarını yapınız.

a) stdin

b) stdout

c) stderr

1.9 Bugünlerde neden nesneye yönelik programlamanın, özellikle C++ ‘ın bu kadar ön planda

olduğunu açıklayın.

25

C İLE PROGRAMLAMAYA GİRİŞ

AMAÇLAR

 C ile basit programlar yazabilmek

 Basit giriş/çıkış ifadelerini kullanabilmek

 Temel veri tiplerini tanımak

 Bilgisayar hafızasını kullanmayı anlamak

 Aritmetik operatörleri kullanabilmek

 Aritmetik operatörlerin önceliklerini anlamak

 Basit karar verme ifadeleri yazabilmek

BAŞLIKLAR

2.1 GİRİŞ

2.2 BASİT C PROGRAMLARI-BİR METNİ YAZDIRMAK

2.3 BASİT C PROGRAMLARI-İKİ TAM SAYIYI TOPLATMAK

2.4 HAFIZA KONULARI

2.5 C'DE ARİTMETİK

2.6 KARAR VERME:EŞİTLİK VE KARŞILAŞTIRMA OPERATÖRLERİ

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Taşınırlık İpuçları\*

Çözümlü Alıştırmalar\*Cevaplar\*Alıştırmalar

2.1 GİRİŞ

C, yapısal ve disiplinli bir bilgisayar programı yazmak için ideal bir dildir. Bu ünitede, C ile

programlama nasıl yapılır konusunu tanıtacağız ve C'de oldukça büyük önem taşıyan

özelliklerin kullanıldığı örnek programlar göstereceğiz. 3. ve 4. ünitede ise yapısal

programlamada detaya gireceğiz. Kitabın geri kalan kısmında yapısal programlama

yaklaşımını kullanacağız.

2.2 BASİT C PROGRAMLARI - BİR METNİ YAZDIRMAK

C'deki bazı özel yazım biçimleri eğer daha önceden C ile programlama yapmadıysanız size

garip gelebilir. Ama zamanla bu özel yazım biçimlerine alışacaksınız. İsterseniz basit bir

programla başlayalım. İlk örneğimiz, bir satırlık bir metni bilgisayarda yazdırmak ile ilgilidir.

Program ve programın bilgisayardaki çıktısı aşağıdaki şekilde ( Şekil 2.1 ) gösterilmiştir.

26

1 /\* Şekil 2.1:fig02\_01.c

2 C ile ilk program \*/

3 #include<stdio.h>

4

5 int main ( )

6 {

7 printf ( "C'ye hoş geldiniz!\n" ) ;

8

9 return 0;

10 }

C'ye hoş geldiniz!

Şekil 2.1 Metin yazdırma programı

Her ne kadar basit bir program olarak gözükse de C'nin çok önemli bir kaç özelliğini bu

sayede tanımış oluyoruz. Şimdi programı satır satır, daha detaylı bir biçimde inceleyelim.1 ve

2 numaralı satırlar /\* ile başlayıp \*/ ile bitmektedir. Bu işaretler arasına yorumlar yazılır.

Yorumlar yazmak, okunurluğu artırmak amacıyla özellikle uzun programlarda kullandığımız

bir özelliktir. Yorum satırlarında bilgisayar hiçbir işlem yapmaz çünkü C derleyicileri bu

satırları atlar. Dolayısıyla, yorum satırları için makine diline çevrilmiş kodlar oluşturulmaz.

Programımızdaki yorum satırı ise şekil numarasını, dosya adını ve programın amacını

açıklamaktadır.Yorumlar, diğer kişilerin programınızı anlamasında yardımcı olur ancak çok

fazla yorum programın okunurluğunu azaltır.

Genel Programlama Hataları 2.1

Yorum satırının sonuna \*/ işaretini koymayı unutmak

Genel programlama hataları 2.2

Yorum satırına \*/ ile başlamak ve /veya yorum satırını /\* ile bitirmek.

3. satırda karşılaştığımız

#include <stdio.h>

C önişlemcisine bir emir göndermektedir. # işaretiyle başlayan satırlar, program derlenmeden

önce önişlemci tarafından işlenirler. Bu satır, önişlemciye standart giriş/çıkış öncü dosyasının

(stdio.h) içeriğini programa eklemesini söyler. Bu öncü dosya, derleyicinin printf gibi

standart giriş/çıkış fonksiyonlarını derlerken kullanacağı bilgi ve bildirimleri içerir. Öncü

dosya ayrıca, derleyicinin kütüphane fonksiyonu çağrılarının doğru yapılıp yapılmadığını

anlamasında yardımcı olan bilgiler içerir. Öncü dosyalar hakkında daha detaylı bilgiyi

5.ünitede vereceğiz.

27

İyi Programlama Alıştırmaları 2.1

<stdio.h> öncü dosyasının eklenmesi tercihe bağlıdır fakat standart giriş/çıkış

fonksiyonlarının kullanıldığı programlara eklenmelidir. Bu sayede, derleyici, hataları

derleme anında bulabilecektir. Aksi takdirde, hatalar programın çalıştırıldığı anda ortaya

çıkar. Bu tür hataların düzeltilmesi oldukça güç olur.

5.satırdaki

int main( )

her C programının bir parçasıdır. main kelimesinden sonraki parantezler main'in fonksiyon

adı verilen program oluşturma bloklarından biri olduğunu gösterir. C programları bir veya

birden fazla fonksiyon içerebilir ancak bunlardan biri mutlaka main olmalıdır. C'de her

program main fonksiyonunu çalıştırarak başlar.

İyi Programlama Alıştırmaları 2.2

Her fonksiyondan sonra fonksiyonu anlatan bir yorum satırı yazılmalıdır.

Küme parantezi, { , her fonksiyonun gövdesinin başına yazılır. Diğer küme parantezi , } , ise

her fonksiyonun sonuna yazılmalıdır. Bu iki parantez arasında kalan program parçacığına blok

denir. Bloklar C'de önemli program birimleridir.

7.satırdaki

printf ( "C'ye hoş geldiniz!\n" ) ;

bilgisayara bir iş yaptırır. Yaptırdığı iş, iki tırnak işareti arasındaki karakterleri ekrana

yazdırmaktır. Yazdırılacak karakterlerin tümüne karakter dizesi ( string ), mesaj ya da hazır

bilgi ( literal ) denir. printf, parantezler içindeki bağımsız değişkenler (argument) ve noktalı

virgülden oluşan bu satıra ifade denir. Her ifade noktalı virgül ile bitmelidir.(Noktalı virgüle

ifade sonlandırıcı da denir) Az önceki printf ifadesi çalıştırıldığında ekrana, C'ye hoş

geldiniz! yazdırır. printf ifadesindeki tırnak işaretleri arasındaki karakterler aynen ekrana

yazdırılır. Ancak \n karakterlerinin yazdırılmadığına dikkat ediniz. Ters eğik çizgi ( \ ), çıkış

karakteri olarak adlandırılır ve printf'in farklı bir iş yapması gerektiğini belirtir. printf, ters

çizgi işaretiyle karşılaştığında, bu işaretten sonraki karaktere bakar ve bu karaktere göre bazı

özel işler yapar.Ters çizgi işareti ( \ ) ve bu işaretten sonra gelen karaktere çıkış sırası denir.

\n çıkış sırası, yeni satır anlamına gelir ve imlecin yeni satıra geçmesine sebep olur.Diğer

çıkış sıraları Şekil 2.2'de gösterilmiştir. Şekil 2.2'deki bazı çıkış sıraları garip gözükebilir.

printf, ters çizgi işaretini ( \ ) çıkış karakteri olarak algıladığından printf ile ters çizgi işareti

yazdırmak istediğimizde iki tane ters çizgi işaretini ( \\ ) birlikte kullanmalıyız. printf ile

tırnak işaretini yazdırmak da bir sorun gibi gözükmektedir çünkü tırnak işareti printf ile

kullanıldığında, yazdırılacak karakterlerin sınırlarını belirler. printf ile tırnak işareti

yazdırmak istersek, \" çıkış sırasını kullanmalıyız.

Küme parantezi , } , main fonksiyonunun sonuna ulaşıldığını gösterir.

28

Genel Programlama Hataları 2.3

printf yerine print yazmak hatadır.

printf’in bilgisayara bir iş yaptırdığını söyledik. Her program çalıştırıldığında bir çok çeşitli

işlem yapar ve kararlar verir. Bu ünite sonunda karar verme üzerinde duracağız ve 3. ünitede

karar verdirme modellerini daha ayrıntılı açıklayacağız.

printf ve scanf gibi standart kütüphane fonksiyonlarının C programlama dilinin bir parçası

olmadığını bilmek oldukça önemlidir. Bu yüzden, derleyici printf ve scanf yanlış yazılırsa

hata bulamaz. Derleyici, printf gibi bir kütüphane fonksiyonu gördüğünde makine diline

çevrilmiş programda boşluk bırakır ve bu boşluğa kütüphaneye gidileceğini belirten bir işaret

koyar. Çünkü, derleyici kütüphanenin nerede olduğunu bilemez. Fakat bağlayıcı bilir.

Böylelikle, bağlayıcı çalıştığında makine diline çevrilmiş programdaki boşluklara uygun

kütüphane fonksiyonlarının kodlarını yerleştirir ve makine diline çevrilmiş programı

tamamlar. Artık program çalıştırılmaya hazırdır. Bağlanmış programlara çalıştırılabilir

(executable) program denir.Eğer fonksiyon ismi yanlış yazılırsa hatayı bağlayıcı bulabilir.

Çünkü programda yazılan fonksiyon ismiyle kütüphane fonksiyonunun ismini eşleyemez.

Çıkış Sırası Tanım

\n Yeni satır.İmleci yeni satırın başına geçirir.

\t Yatay tab.İmleci bir sonraki tab başlangıcına taşır.

\a Alarm.Sistemdeki zili çalar.

\\ Ters çizgi. printf içinde ters çizgi karakterini yazdırır.

\" Tırnak . printf içinde tırnak karakterini yazdırır.

Şekil 2.2 Bazı çıkış sıraları

İyi Programlama Alıştırmaları 2.3

Yazdırma işlemi yapan bir fonksiyon tarafından yazdırılan son karakter ( \n )

olmalıdır.Bu sayede fonksiyonun, ekran imlecini yeni satırın başlangıcına götürmesi

sağlanır. Bu tarz bize, yazılım geliştirme ortamlarında temel amaç olan yazılımın

yeniden kullanılabilirliğini artırma fırsatı verir.

İyi Programlama Alıştırmaları 2.4

Fonksiyon bloklarının içini yazmaya, küme parantezlerinden daha içerde başlamak (3

boşluk bırakarak) fonksiyonun gövdesini daha belirgin hale getirir. Bu sayede,

programımız daha okunur hale gelecektir.

İyi Programlama Alıştırmaları 2.5

Kendinize göre bir girinti miktarı belirleyin ve gerekli tüm yerlerde bu girinti miktarını

kullanın. Girintiler yaratmak için tab tuşu kullanılabilir ancak kimi zaman tab başlangıçları

sorun yaratabilir. Bu sebepten, en iyisi 3 boşluk bırakarak girintiler oluşturmaktır.

printf fonksiyonu kullanılarak, C'ye hoş geldiniz! mesajı farklı biçimlerde yazdırılabilir.

Örneğin, Şekil 2.3'teki program, Şekil 2.1'deki programla aynı çıktıyı üretir. Çünkü her

printf, kendi mesajını yazdırmaya, eğer bazı özel çıkış dizileri kullanılmamışsa, diğer

printf'in kaldığı yerden başlar. 7. satırdaki ilk printf , C'ye kısmını ve bir adet boşluk

29

karakterini yazdırır. 8.satırdaki printf yazdırmaya bu boşluktan sonra başlar. Tek bir printf

ile Şekil 2.4'te olduğu gibi birden fazla satır yazdırmak mümkündür. Her seferinde \n çıkış

sırası imleci yeni satıra geçirir ve yazım işleminin yeni satırdan devam etmesini sağlar.

1 /\*Şekil 2.3:prog02\_03.c

2 Tek bir satıra iki ayrı printf ifadesiyle yazdırma yapmak\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 printf( "C'ye " );

8 printf( "hoş geldiniz!\n" );

9

10 return 0;

11 }

C' ye hoş geldiniz!

Şekil 2.3 Tek bir satıra ayrı printf ifadeleriyle yazdırma yapmak.

1 /\*Şekil 2.4:prog02\_04.c

2 Tek bir printf ile birden fazla satırda yazdırma işlemi yapmak\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 printf("C'ye\nhoş geldiniz!\n");

8

9 return 0;

10 }

C'ye

hoş geldiniz!

Şekil 2.4 Tek bir printf ile birden fazla satırda yazdırma yapmak

2.3 BASİT C PROGRAMLARI-İKİ TAM SAYIYI TOPLATMAK

Bu programımızda scanf fonksiyonunu kullanarak, kullanıcıların klavyeden gireceği iki

tamsayıyı tespit edip, bunları toplatacağız. Toplamı ise printf fonksiyonu sayesinde ekranda

göstereceğiz. Program ve programın örnek bir çıktısı Şekil 2.5'te gösterilmiştir.

30

/\*Toplama programı\*/ yorumu, programın amacını belirmektedir. Daha önceden

belirttiğimiz gibi, her program main çalıştırılarak başlar. Küme parantezleri main

fonksiyonunun gövdesinin başlangıcını ve bitişini belirtirler.

1 /\*Şekil 2.5:prog02\_05.c

2 Toplama programı\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int tamsayi1,tamsayi2,toplam; /\*bildirim\*/

8

9 printf ( "İlk tamsayıyı giriniz\n" ); /\*mesaj yazdırma\*/

10 scanf ( "%d",&tamsayi1); /\*ilk tamsayının okunması\*/

11 printf ( "İkinci tamsayıyı giriniz\n" ); /\*mesaj yazdırma\*/

12 scanf ( "%d",&tamsayi2); /\*ikinci tamsayının okunması\*/

13 toplam = tamsayi1+tamsayi2; /\*toplamın atanması\*/

14 printf ( "Toplam %d dir\n",toplam ); /\*toplamın yazdırılması\*/

15

16 return 0; /\*programın başarıyla sona erdiğini belirtmek\*/

17 }

İlk tamsayıyı giriniz

45

İkinci tamsayıyı giriniz

72

Toplam 117 dir

Şekil 2.5 Toplama programı

7.satırdaki

int tamsayi1, tamsayi2, toplam;

bir bildirimdir. tamsayi1, tamsayi2 ve toplam, değişkenlerin isimleridir. Değişkenler,

programın kullanabileceği bir değerin saklanacağı hafıza konumlarıdır. Bu bildirim,

tamsayi1, tamsayi2 ve toplam değişkenlerinin int tipinde olduklarını yani bu değişkenlerin

7, -11, 0, 31914 gibi tamsayı değerlerini tutacağını belirtir. Değişkenlerin programda

kullanılabilmesi için değişkenler bir isim ve değişken tipiyle, main fonksiyonunun başladığını

belirten küme işaretinden , { , hemen sonra bildirilmelidir. C'de int tipinden başka veri tipleri

de vardır. Aynı tipte değişkenler tek bir bildirim sayesinde bildirilebilirler. Her değişken için

3 ayrı bildirim yazabilirdik ancak az önce yaptığımız bildirim daha uygundur.

İyi Programlama Alıştırmaları 2.6

Virgülden hemen sonra bir boşluk bırakmak programın okunurluğunu arttırır.

C'de değişken isimleri geçerli tanıtıcılardan (identifier) biri olabilir. Tanıtıcılar, harf, rakam

ve alt çizgi karakterlerinin dizisidir. Ancak bu diziler rakamla başlayamaz. Bir tanıtıcı, istenen

31

uzunlukta olabilir ancak ANSI C standardındaki derleyiciler için yalnızca ilk 31 karakter

önemlidir. C, harf duyarlıdır. C'de büyük harf küçük harf ayrımı yapılır. Bu sebepten, a1 ve

A1 farklı tanıtıcılardır.

Genel Programlama Hataları 2.4

Küçük harf kullanılması gereken bir yerde büyük harf kullanılması hatadır. Örneğin,

main yerine Main yazmak hatadır.

Taşınırlık İpuçları 2.1

Tanıtıcılarınızı 31 karakterden kısa tutmaya çalışın. Bu, taşınırlığı artıracaktır.

İyi Programlama Alıştırmaları 2.7

Anlamlı değişken isimleri kullanmak programda daha az yorum satırı yazmamız demektir.

İyi Programlama Alıştırmaları 2.8

Basit bir değişken olarak kullanılacak tanıtıcılar küçük harfle başlamalıdır. İleride, büyük harfle

başlayacak ya da tüm harfleri büyük harf olan, özel öneme sahip değişkenlerden bahsedeceğiz.

İyi Programlama Alıştırmaları 2.9

Bir çok kelimeden oluşan değişken isimleri, programı daha okunabilir yapar. Ancak

kelimeleri birleşik yazmaktan kaçının. Bunun yerine, kelimelerin arasında alt çizgi kullanın.

Eğer kelimeleri birleşik yazmak istiyorsanız, ikinci kelimeden sonrasını büyük harfle başlatın.

toplamkomisyon yerine toplam\_komisyon ya da toplamKomisyon yazın.

Bildirimleri, bir fonksiyonun gövdesini başlatan küme parantezinden hemen sonra ve

çalıştırılabilir ifadelerden önce yazın. Örneğin, Şekil 2.5'teki örnekte bildirimleri ilk printf'ten

sonra yapmak bir yazım hatası ( syntax error ) olacaktı. Yazım hataları, derleyici bir ifadeyi

tanıyamadığında gerçekleşir. Derleyici, bir hata mesajı oluşturarak programcıya yanlış

ifadeleri düzeltmesi gerektiğini söyler. Yazım hataları dilin yanlış kullanılması yüzünden

oluşur.Yazım hatalarına derleme hataları ya da derleme zamanı hataları da denir.

Genel Programlama Hataları 2.5

Değişken bildirimlerini çalıştırılabilir ifadelerden sonra yapmak hatadır.

İyi Programlama Alıştırmaları 2.10

Bildirimlerle, çalıştırılabilir ifadeler arasında boş bir satır bırakmak bildirimlerin sona

erdiğini vurgulamaya yarar.

9.satırdaki

printf ( "İlk tamsayı giriniz\n" );

ifadesi, İlk tamsayıyı giriniz mesajını ekrana yazdırır ve imleci yeni satıra geçirir.Bu mesaj,

kullanıcıya bir işlem yapması gerektiğini söyler.

Diğer ifade olan

scanf ( "%d", &tamsayi1 );

32

kullanıcının gireceği değeri almak için kullanılır. scanf fonksiyonu, giriş değerini standart

girişten alır. Standart giriş genellikle klavyedir.

scanf fonksiyonunda iki argüman ( bağımsız değişken ) görüyoruz ; “%d” ve &tamsayi1. İlk

argüman, biçim kontrol dizesi olarak adlandırılır ve kullanıcı tarafından girilmesi beklenen

verinin tipini belirtir. %d dönüşüm belirteci, verinin tamsayı olması gerektiğini belirtir. (d

harfi İngilizce decimal integer teriminin kısaltmasıdır.) % karakterine, scanf tarafından (ve

ileride göreceğimiz gibi printf tarafından) ters çizgi ( \ ) gibi bir çıkış karakteri biçiminde ve

%d'ye ise çıkış dizisi biçiminde davranılır. scanf'in 2.argümanı & karakteriyle başlar ve bir

değişken ismiyle devam eder. & karakterine adres operatörü denir. Değişken ismiyle birlikte

kullanıldığında & karakteri, scanf fonksiyonuna tamsayi1 değişkeninin hangi hafıza

konumuna yerleştirileceğini söyler. Bilgisayar, tamsayi1 değişkeninin değerini o konuma

yerleştirir. & operatörünün kullanımı, yeni programcılara ya da başka programlama dillerinde

böyle bir operatöre ihtiyaç duymayan programcılara garip gelebilir. Şimdilik, scanf içinde her

değişken isminden önce & operatörünü kullanın. Bu kuralın istisnalarını 6. ve 7.ünitelerde

anlatacağız. & operatörün gerçek anlamı, 7.ünitede göstericiler konusunu anlattığımızda

anlaşılacaktır.

Bilgisayar, scanf'i çalıştırdığında kullanıcının tamsayi1 değişkeni için bir değer girmesini

bekler. Kullanıcı, bir tamsayı yazarak ve ardından da giriş tuşuna basarak sayıyı bilgisayara

gönderir. Bilgisayar bu sayıyı, ya da bu değeri, tamsayi1 değişkenine atar. Programın

devamında tamsayi1 kullanılacağında, her sferinde girilen bu değer kullanılır. printf ve scanf

fonksiyonları, kullanıcıyla bilgisayar arasında iletişimi sağlar. Bu iletişim, bir diyalog

sayesinde gerçekleştiği için buna interaktif kullanım da denir.

printf ( "İkinci tam sayıyı giriniz\n");

ifadesi, İkinci tam sayıyı giriniz mesajını ekrana yazdırır ve imleci yeni satırın başlangıcına

taşır. Bu printf, kullanıcıya bir işlem yapması gerektiğini belirtir.

scanf ( "%d", &tamsayi2 );

ifadesi, kullanıcıdan tamsayi2 değişkeni için bir değer elde eder. 13. satırdaki

toplam = tamsayi1 + tamsayi2;

ataması tamsayı1 ve tamsayı2 değişkenlerinin toplamını hesaplayarak sonucu, atama

operatörünü ( = ) kullanarak, toplam değişkenine atar. Bu ifade, "toplam, tamsayi1 +

tamsayi2 değerini alır" şeklinde okunur. + operatörü, tamsayi1 ve tamsayi2 olmak üzere,

iki operand kullanmıştır. = operatörü ise toplam ve tamsayi1 + tamsayi2 ifadesinin sonucu

olmak üzere, yine iki operand kullanmıştır.

İyi Programlama Alıştırmaları 2.11

Operatörün her iki tarafına da bir boşluk bırakılmalıdır. Bu sayede program daha

okunabilir olur.

Genel Programlama Hataları 2.6

Atama ifadelerinde hesaplama, = operatörünün sağ tarafında bulunmalıdır. Hesaplama

operatörün solunda yapılırsa yazım hatası ortaya çıkar.

14. satırdaki

33

printf ( "Toplam %d dir\n", toplam );

ifadesi, printf fonksiyonunu çağırıp Toplam bilgisinden sonra, toplam değişkeninin sayısal

değerini yazdırır. printf'in iki argümanı vardır; "Toplam %d dir\n" ve "toplam". İlk

argüman, biçim kontrol dizesi olarak adlandırılır. Bu argüman, yazdırılacak karakterlerle, bir

tamsayının yazdırılacağını gösteren %d dönüşüm belirtecini içerir. İkinci argüman

yazdırılacak değeri belirler. Bir tamsayı için dönüşüm belirtecinin printf ve scanf'de aynı

olduğuna dikkat ediniz. Bu durum, C de çoğu veri tipi için geçerlidir.

Hesaplamalar, printf ifadelerinin içinde de gerçekleştirilebilir. Daha önceden yazdığımız iki

ifadeyi tek bir ifade biçiminde

printf ( "Toplam %d dir\n", tamsayi1 + tamsayi2 );

şeklinde de yazabilirdik.

16. satırdaki

return 0;

ifadesi 0 değerini, programın çalıştırıldığı işletim sistemi ortamına gönderir ve böylelikle

işletim sistemine programın başarılı bir şekilde çalıştırıldığını belirtir. Bir programdaki her

hangi bir hatayı işletim sistemine nasıl belirteceğinizi öğrenmek için işletim sisteminizin

kılavuzunu inceleyiniz.

Küme parantezi, } , main fonksiyonunun sonuna ulaşıldığını gösterir.

Genel Programlama Hataları 2.7

printf ya da scanf içindeki biçim kontrol dizesinde, tırnak karakterlerinden birini ya da ikisini

birden unutmak.

Genel Programlama Hataları 2.8

printf ya da scanf içindeki biçim kontrol dizesinde, % dönüşüm belirleme karakterini

unutmak.

Genel Programlama Hataları 2.9

printf ya da scanf içinde ki \n çıkış sırasını, biçim kontrol dizesi dışına yerleştirmek.

Genel Programlama Hataları 2.10

Dönüşüm belirteçleri içeren bir printf ifadesi içinde, değerleri yazdırılacak deyimleri

dahil etmeyi unutmak.

Genel Programlama Hataları 2.11

Bir deyim yazdırılacağında printf biçim kontrol dizesi içinde bir dönüşüm belirteci

yazmamak.

Genel Programlama Hataları 2.12

34

Biçim kontrol dizesini yazdırılacak deyimlerden ayırmak için kullanılması gereken

virgül ( ,) karakterini biçim kontrol dizesi içine yazmak.

Genel Programlama Hataları 2.13

scanf ifadesi içinde, bir değişkenin başına & konması gerekirken bu karakteri unutmak.

Çoğu sistemde, çalışma zamanlı bu hataya erişim hatası denir. Böyle bir hata, programcı

erişim haklarına sahip olmayan bir hafıza alanına erişmek istediği zaman oluşur. Bu hatanın

sebebi hakkında detaylı bilgiyi 7. ünitede açıklayacağız.

Genel Programlama Hataları 2.14

printf ifadesi içinde, bir değişkenin başına & operatörü yazmak.

7. ünitede göstericileri çalıştığımızda, bir değişkenin adresini yazdıracağımızda bu değişken

ismi ile birlikte & operatörünü kullandığımızı göreceksiniz. Ancak ilerleyen bir kaç ünite

boyunca printf ifadeleri içinde & operatörünü kullanmayacağız.

2.4 HAFIZA KONULARI

tamsayi1, tamsayi2 ve toplam gibi değişken isimleri gerçekte bilgisayar hafızasında

konumlar belirtir. Her değişkenin ismi, tipi ve değeri vardır.

Şekil 2.5’deki toplama programında

scanf ( "%d", &tamsayi1 );

ifadesi çalıştırıldığında, kullanıcı tarafından girilen değer tamsayi1 isminin atandığı hafıza

konumuna yerleştirilir. Kullanıcının, tamsayi1 değişkenin değeri olarak 45 girdiğini

varsayalım. Bilgisayar 45'i tamsayi1 konumuna Şekil 2.6' da gösterildiği gibi yerleştirir.

45

Şekil 2.6 Bir değişkenin ismini ve değerini gösteren hafıza konumu

Bir değer hafıza konumuna yerleştirildiğinde, daha önceden o hafıza konumunda bulunan

değerin üzerine yazılır. Bu konumda daha önceden bulunan değer silindiğinden hafızaya

yazma işlemi destructive read-in olarak adlandırılır.

Toplama programımıza geri döndüğümüzde

scanf ( "%d", &tamsayi2 );

ifadesi çalıştırıldığında, kullanıcının 72 değerini girdiğini varsayalım. Bu değer, tamsayi2

konumuna yerleştirilir ve hafıza Şekil 2.7’de görüldüğü gibi olur. Bu konumların hafızada ard

arda gelmeleri zorunluluğu yoktur.

Program tamsayi1 ve tamsayi2 değerlerini aldıktan sonra bu değerleri toplar ve toplamı,

toplam değişkenine yerleştirir.

tamsayi1

35

toplam = tamsayi1 + tamsayi2;

ifadesi, toplama yapmanın yanında toplam değişkeninin eski değerini silerek yerine yenisini

yazar. Toplam hesaplandıktan sonra hafıza Şekil 2.8’deki gibi görünür.

tamsayi1 ve tamsayi2 değişkenlerinin değerlerinin, toplama işlemine girmeden önceki

değerlerini koruduğuna dikkat ediniz. Bu değerler, toplama işleminde kullanılmışlardır fakat

değiştirilmemişlerdir. Bu sebepten, bir hafıza konumundan okuma işlemi yapıldığında buna

non-destructive read-out denir.

45

72

Şekil 2.7 Değişkenler girildikten sonra hafıza konumları

45

72

117

Şekil 2.8 Hesaplamadan sonra hafıza konumları

2.5 C' DE ARİTMETİK

Çoğu C programında aritmetik işlemler yapılır. Şekil 2.9'da aritmetik operatörlerin bir özetini

bulacaksınız. Matematikte kullanılmayan bazı özel sembollerin kullanıldığında dikkat ediniz.

\* ( yıldız işareti ) çarpım için, % (yüzde işareti) mod almak için kullanılır. Matematikte a ile

b'yi çarpmak istediğimizde, tek harften oluşan bu değişken isimlerini, yan yana ab biçiminde

yazarız. C’de bunu yapmaya kalktığımızda, ab iki harften oluşan tek bir tanıtıcı olarak

algılanacaktır. Bu sebepten, C’de (ve genel olarak diğer programlama dillerinde) çarpım, \*

operatörü kullanılarak a\*b şeklinde gösterilmelidir.

Bütün aritmetik operatörler, ikili operatörlerdir. Örneğin, 3 + 7 deyimi, + ikili operatörünü ve

3 ile 7 operandlarını içerir.

tamsayi1

tamsayi2

tamsayi1

tamsayi2

toplam

36

Tamsayılar kullanıldığında bölüm bir tamsayı değeri verir. Örneğin, 7/4 işlemi 1 sonucunu,

17/5 işlemi 3 sonucunu verir. C'de bölümden kalan sayıyı bulmak için mod operatörü, %,

kullanılır. Mod operatörü yalnızca tamsayılarla kullanılabilir. x % y, x'in y'ye bölümünden

kalan sayıyı hesaplar. 7 % 4 hesaplaması 3 sonucunu ve 17 % 5 hesaplaması 2 sonucunu

verir. Mod operatörünün oldukça ilginç uygulamalarını ileride tartışacağız.

Genel Programlama Hataları 2.15

Bir sayıyı 0'a bölmek bilgisayar sistemlerinde tanımlı değildir ve genellikle ölümcül bir

hatadır. Ölümcül hatalar, programın çalışmasının aniden durmasına ve başarılı bir

sonuç vermemesine sebep olur. Ölümcül olmayan hatalar, programın yanlış sonuçlar

vermesine sebep olur.

Aritmetik işlemler C'de satırlara yazılmalıdır. Yani a / b yazmak yerine

a

-

b

yazarsak, derleyici bu gösterimi kabul etmeyebilir (bazı özel amaçlı yazılım paketlerinde,

karmaşık matematik deyimlerini alışık olduğumuz biçimde yazma fırsatı bulunabilir.)

C işlemi aritmetik operatör matematiksel deyim C deyimi

toplama + f + 7 f + 7

çıkarma - p - c p - c

çarpma \* bm b \* m

bölme / x / y yada x ÷ y x / y

mod alma % r mod s r % s

Şekil 2.9 Aritmetik operatörler

C'de parantez kullanımı, cebirsel deyimlerde kullandığımız biçimdedir. Örneğin, b + c

işleminin sonucu ile a sayısını çarpmak için

a \* ( b + c )

yazarız.

C, aritmetik deyimleri, operatör önceliği kuralları adı verilen bir dizi kurala uyarak hesaplar.

Bu kurallar genellikle matematikteki kurallarla aynıdır.

1. Parantezler içindeki deyimler ilk önce hesaplanır. Bu sebepten, parantezler, programcı

tarafından hesaplama sırasının istenilen biçimde gerçekleşmesini sağlatır. Parantezlerin en

yüksek öncelik sırasına sahip olduğu söylenir. Yuvalı ya da iç içe geçmiş parantezlerde en

içteki parantez çiftinin içindeki deyim ilk önce hesaplanır.

2. Çarpma, bölme ve mod işlemleri daha sonra hesaplanır. Eğer bir işlemde birden fazla

çarpma , bölme ve mod alma işlemleri yapılacaksa hesaplama soldan sağa doğru yapılır.

Çapma, bölme ve mod almanın öncelik seviyeleri eşittir.

37

3. Toplama ve çıkartma işlemleri en son yapılır. Eğer bir deyimde birden fazla toplama ve

çıkartma işlemi varsa hesaplama soldan sağa doğru yapılır. Toplama ve çıkartmanın öncelik

sırası birbirine eşittir.

Operatör öncelikleri, C'nin deyimleri doğru bir biçimde hesaplamasını sağlatır. İşlemlerin

soldan sağa doğru ilerlediğini söylediğimizde, operatörler öncelikle soldaki operandı işleme

sokarlar. İleride göreceğimiz bazı operatörler ise sağdan sola doğru çalışırlar yani öncelikle

sağdaki operandı işleme sokarlar. Şekil 2.10, operatör öncelikleriyle ilgili kuralları

özetlemektedir.

Şimdi, operatör önceliği kurallarının kullanıldığı birkaç deyimden bahsedelim. Her örnek bir

matematik ifadeyi ve bu deyimin C'de yazılış biçimini göstermektedir.

Aşağıdaki örnek, 5 terimin aritmetik ortalamasını hesaplamaktadır.

Matematik gösterimi a + b + c + d + e

m =

5

C : m = (a + b + c + d + e) / 5;

Operatör İşlem Öncelik Sırası

( ) parantez İlk önce hesaplanır.Eğer parantezler içice yazılmışsa, en içteki

parantez ilk önce hesaplanır. Eğer bir satırda birden fazla

parantez varsa (iç içe değillerse), bunlar soldan sağa doğru

hesaplanırlar.

\*, / ,% çarpım İkinci olarak hesaplanırlar.Eğer birden fazla varsa, soldan sağa

bölüm doğru hesaplanırlar.

mod alma

+, - toplama En son hesaplanırlar. Eğer birden fazla varsa, soldan sağa doğru

çıkartma hesaplanırlar.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Şekil 2.10 Aritmetik operatörlerin öncelikleri

Parantezler bu işlemde gereklidir çünkü bölme, toplamaya göre daha önceliklidir. Parantez

koyarak (a + b + c + d + e) deyiminin 5'e bölünmesini sağlatmış oluruz. Eğer parantez

koymasaydık ve a + b + c + d + e / 5 olsaydı, e'nin 5'te birinin diğerleriyle toplanmasını

sağlardık.

Aşağıdaki örnek, bir doğrunun denklemidir.

Matematik gösterimi y = mx + b

C: y = m \* x + b;

38

parantezlere gerek duyulmaz çünkü çarpma işlemi toplama işlemine göre önceliklidir.

Aşağıdaki örnek, mod, çarpım, bölüm, toplama ve çıkartma işlemleri içermektedir.

Matematik gösterimi z = pr % q + w / x - y

C: z = p \* r % q + w / x - y;

6 1 2 4 3 5

Operatörlerin altındaki sayılar, C'nin bu deyimi hesaplarken hangi sırayı takip ettiğini

göstermektedir. Çarpım, mod alma ve bölüm ilk önce hesaplanır çünkü toplama ve çarpmaya

göre daha önceliklidir. Çarpma, bölme ve mod alma kendi aralarında ise soldan sağa doğru

işlem görürler. Toplama ve çıkartma daha sonra hesaplanır. Toplama ve çıkartma da kendi

aralarında soldan sağa doğru hesaplanır.

Birden fazla parantez çifti içeren deyimlerde, parantezler yuvalı olmayabilir. Örneğin

a \* ( b + c ) + c \* ( d + e )

deyiminde parantezler yuvalı değildir. Bunun yerine, parantezlerin aynı seviyede öncelik

sırasına sahip olduğu söylenir. Bu durumda C , parantez içindeki deyimleri soldan sağa doğru

hesaplar.

Operatör önceliği kurallarını daha iyi anlamak için, ikinci dereceden bir polinomun C ile nasıl

hesaplanacağını görelim

y = a \* x \* x + b \* x + c;

6 1 2 4 3 5

Operatörlerin altındaki sayılar, C'nin bu deyimi hesaplarken hangi sırayı takip ettiğini

göstermektedir. C'de üs gösterimi olmadığı için, x² yerine x \* x yazmak zorundayız. C

standart kütüphanesi, pow ( power ) fonksiyonu ile üslü işlemleri gerçekleştirir. pow

fonksiyonunu kullanmak için değişik veri tipleri gerektiğinden, pow fonksiyonu 4. ünitede

daha detaylı bir biçimde anlatacağız.

Yukarıdaki örneğimiz için a = 2, b = 3, c = 7 ve x = 5 olsun. Şekil 2.11, ikinci dereceden bir

polinomun nasıl hesaplandığını açıklamaktadır.

1.Adım y = 2 \* 5 \* 5 + 3 \* 5 + 7;

2 \* 5 = 10 (en soldaki çarpım)

2.Adım y = 10 \* 5 + 3 \* 5 + 7;

10 \* 5 = 50 (en soldaki çarpım)

3.Adım y = 50 + 3 \* 5 + 7;

39

3 \* 5 = 15 (toplamadan önceki çarpım)

4.Adım y = 50 + 15 + 7;

50 + 15 = 65 (en soldaki toplam)

5.Adım y = 65 + 7;

65 + 7 = 72 (son toplam)

6.Adım y = 72; (son operasyon-atama)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Şekil 2.11 İkinci dereden bir polinomun hesaplanmasını

2.6 KARAR VERME: EŞİTLİK ve KARŞILAŞTIRMA OPERATÖRLERİ

Çalıştırılabilir C ifadeleri ya bir işlem gerçekleştirir ( girilen verilerin toplanması gibi ) ya da

kararlar verir. (bunun örneklerini ileride göreceğiz) Programda bir karar verebiliriz. Örneğin,

bir kişinin bir sınavdan aldığı not 60'tan büyükse ya da 60'a eşitse "tebrikler geçtiniz"

yazdırabiliriz. Bu kısım, bir koşulun doğruluk ya da yanlışlığına göre karar veren, C'nin if

kontrol yapısının basit bir biçimini tanıtmaktadır. Eğer koşul doğru ise, if yapısının

gövdesindeki ifade çalıştırılır. Eğer koşul yanlış ise, if yapısının gövdesindeki ifadeler

çalıştırılmaz. if yapısının gövdesi çalıştırılsa da çalıştırılmasa da, çalıştırma süreci if

yapısından hemen sonraki ifadeyle devam eder.

if yapısı içindeki koşullar, Şekil 2.12'de özetlenen eşitlik operatörleri ve karşılaştırma

operatörleriyle sağlanır. Karşılaştırma operatörleri aynı seviyede önceliğe sahiptir ve soldan

sağa doğru işlerler. Eşitlik operatörü, karşılaştırma operatörlerinden daha düşük önceliğe

sahiptir ve soldan sağa doğru işler. [Not: C'de bir koşul 0(yanlış) ya da 0'dan farklı (doğru) bir

değer üreten herhangi bir deyim olabilir. Bu konuyla ilgili bir çok uygulamayı kitap boyunca

göreceğiz.

Operatörler C'deki karşılığı C'de örneği C'de anlamı

eşitlik operatörleri

= == x = = y x eşittir y

= != x != y x eşit değildir y

karşılaştırma operatörleri

> > x > y x büyüktür y

< < x < y x küçüktür y

≥ >= x >= y x büyüktür ya da eşittir y

≤ <= x <= y x küçüktür ya da eşittir y

Şekil 2.12 Eşitlik ve karşılaştırma operatörleri

Genel Programlama Hataları 2.16

==, !=, >= ve <= operatörlerinin arasında boşluk kullanılması yazım hatasına sebep

olur.

Genel Programlama Hataları 2.17

!=, >=, <= operatörlerinin ters çevrilerek =!, =>, =< şeklinde kullanılması yazım hatasına sebep olur.

40

Genel Programlama Hataları 2.18

== operatörünün = ile karıştırılması.

Bu karışıklığı önlemek için eşitlik operatörü “çift eşittir” olarak, atama operatörü ise “atama”

olarak okunmalıdır. İleride göreceğimiz üzere, bu iki operatörün karıştırılması kolaylıkla

giderilebilecek bir yazım hatası yanında oldukça önemli mantık hatalarında sebep olur.

Genel Programlama Hataları 2.19

if yapısındaki koşulu belirten parantezlerin sağına noktalı virgül ( ;) koymak

Şekil 2.13, kullanıcı tarafından girilen iki sayıyı karşılaştırmak için altı if yapısı

kullanmaktadır. Eğer herhangi bir if yapısı içindeki koşul gerçekleşirse, o if yapısıyla ilgili

printf fonksiyonu çalıştırılır. Program ve bu programın üç örneğe göre çıktıları aşağıdaki

şekilde gösterilmiştir.

1 /\* Şekil 2.13: prog02\_13.c

2 if yapılarını, karşılaştırma ve eşitlik

3 operatörlerini kullanmak \*/

4 #include <stdio.h>

5

6 int main( )

7 {

8 int sayi1, sayi2;

9

10 printf( "İki tamsayı girin\n" );

11 printf( "Bu iki sayının karşılaştırması yapılacaktır: " );

12 scanf( "%d%d", &sayi1, &sayi2 ); /\* iki sayının alınması \*/

13

14 if ( sayi1 == sayi2 )

15 printf( "%d eşittir %d\n", sayi1, sayi2 );

16

17 if ( sayi1 != sayi2 )

18 printf( " %d eşit değildir %d\n ", sayi1, sayi2 );

19

20 if ( sayi1 < sayi2 )

21 printf( "%d küçüktür %d\n", sayi1, sayi2 );

22

23 if ( sayi1 > sayi2 )

24 printf( "%d büyüktür %d\n", sayi1, sayi2 );

25

26 if ( sayi1 <= sayi2 )

27 printf( "%d küçüktür yada eşittir %d\n",

28 sayi1, sayi2 );

29

30 if ( sayi1 >= sayi2 )

31 printf( "%d büyüktür yada eşittir %d\n",

32 sayi1, sayi2 );

33

34 return 0; /\* program başarılı bir şekilde sona ermiştir \*/

41

35 }

İki tamsayı girin

Bu iki sayının karşılaştırması yapılacaktır: 3 7

3 eşit değildir 7

3 küçüktür 7

3 küçüktür yada eşittir 7

İki tamsayı girin

Bu iki sayının karşılaştırması yapılacaktır: 22 12

22 eşit değildir 12

22 büyüktür 12

22 büyüktür yada eşittir 12

İki tamsayı girin

Bu iki sayının karşılaştırması yapılacaktır: 7 7

7 eşittir 7

7 küçüktür yada eşittir 7

7 büyüktür yada eşittir 7

Şekil 2.13 Eşitlik operatörleri ve karşılaştırma operatörlerini kullanmak

Şekil 2.13’teki programın, iki tamsayıyı almak için scanf (12.satır) kullandığına dikkat ediniz.

Her dönüşüm belirteci, içinde değerin saklanacağı, ilgili bir argümana sahiptir. İlk %d, bir

değeri sayi1 değişkeni içinde tutulacak şekle dönüştürür ve ikinci %d, bir değeri sayi2 içinde

tutulacak bir şekle dönüştürür. Her if ifadesini içeriden başlatmak ve her if yapısının altına ve

üstüne bir satır boşluk yerleştirmek, programın okunurluğunu arttırır. Ayrıca, Şekil 2.13’teki

her if ifadesi içinde tek bir ifade yer aldığına da dikkat ediniz. 3.ünitede birden çok ifadeyi

içeren gövdeye sahip if ifadelerini nasıl yazacağımızı göreceğiz.

İyi Programlama Alıştırmaları 2.12

if yapısının gövdesi içindeki ifadeleri içeriden başlatmak.

İyi Programlama Alıştırmaları 2.13

Programlarda her kontrol yapısından önce ve sonra boş bir satır kullanmak programın

okunulabilirliğini arttırır.

İyi Programlama Alıştırmaları 2.14

Bir satırda birden fazla ifade bulunmamalıdır.

Genel Programlama Hataları 2.20

42

scanf ifadesi içindeki dönüşüm belirteçleri arasına (asla gerekmemesine rağmen) virgül

yerleştirmek.

Şekil 2.13’teki yorum satırı 3 satır sürmüştür. C programlarında, tab, yeni satır ve boşluklar

gibi boşluk karakterleri ihmal edilir. Bu sebepten, ifadeler ve yorumlar birden fazla satır

sürebilir.Ancak tanıtıcıları ayırmak doğru değildir.

İyi Programlama Alıştırmaları 2.15

Uzun bir ifade birden çok satır sürebilir. Eğer bir ifade birden fazla satır sürecekse,

ifadeyi mantıklı noktalardan ayırmak gerekir. (örneğin virgüllerle ayrılmış bir listede

virgülden sonra) Eğer bir ifade birden çok satır sürüyorsa, ifadenin sürdüğü tüm

satırlar içeriden başlatılmalıdır.

Şekil 2.14, bu ünitede gösterilen operatörlerin önceliklerini listelemektedir. Operatörlerin

önceliği yukarıdan aşağıya gidildikçe azalmaktadır. Eşit işaretininde bir operatör olduğuna

dikkat ediniz. Bütün bu operatörler, atama operatörü ( = ) hariç, soldan sağa doğru işlerler.

Atama operatörü ( = ), sağdan sola doğru işler.

İyi Programlama Alıştırmaları 2.16

Birden fazla operatör içeren deyimler yazdığınızda operatör önceliklerini gösteren

tabloya bakınız. Deyimin içindeki operatörlerin uygun biçimde kullanıldığından emin

olunuz. Eğer karmaşık bir deyim içinde hesaplama sırasından emin olamazsanız, sırayı

istediğiniz şekle getirmek için (matematikte olduğu gibi) parantezleri kullanın. C’in bazı

operatörlerinin (örneğin atama operatörü ( = ) gibi ) soldan sağa değil de, sağdan sola

doğru işlediğini gözden kaçırmayın.

Bu ünitedeki C programlarında kullandığımız bazı kelimeler ( int,return ve if ), bu dilin

anahtar kelimeleridir. C’deki tüm anahtar kelimeleri Şekil 2.15’te bulabilirsiniz. Derleyici için

bu kelimelerin özel anlamları vardır. Bu yüzden, programcı bu kelimeleri değişken isimleri

gibi tanıtıcılar biçiminde kullanmamaya dikkat etmelidir. Bu kitapta tüm anahtar kelimeleri

açıklayacağız.

Operatörler İşleyişleri

( ) soldan sağa

\* / % soldan sağa

+ - soldan sağa

< <= > >= soldan sağa

= = != soldan sağa

= sağdan sola

Şekil 2.14 Şu ana kadar anlatılan operatörlerin öncelikleri ve işleyişleri

Anahtar kelimeler

auto double int struct

break else long switch

case enum register typedef

char extern return union

const float short unsigned

continue for signed void

default goto sizeof volatile

43

do if static while

Şekil 2.15 C’nin anahtar kelimeleri

Bu ünitede, C programlama dilinin, verileri ekrana yazdırma, kullanıcıdan verileri alma,

işlemler yapma ve kararlar verme gibi önemli bir çok özelliğini tanıttık. 3.ünitede yapısal

programlamayı öğrendikçe bu tekniklerin üzerine devam edeceğiz. İçeriden başlatma

tekniklerini daha yakından tanıyacak ve ifadelerin hangi sırada çalıştırılacaklarına ( buna akış

kontrolü denir) nasıl karar vereceğimizi çalışacağız.

ÖZET

 Yorumlar /\* ile başlar ve \*/ ile biter. Programcılar yorumları, okunulabilirliği

arttırmak amacıyla ve programlarını daha açık bir hale getirmek için kullanırlar.

Yorumlar program çalışırken bilgisayara bir iş yaptırmazlar.

 #include <stdio.h> önişlemci komutu, derleyiciye standart giriş/çıkış öncü dosyasını

programa eklemesini söyler. Bu dosya, derleyicinin scanf ve printf gibi giriş ve çıkış

fonksiyonlarının doğru bir biçimde çağrıldıklarını onaylamasını sağlayan bilgiler içerir

 C programları biri main olan fonksiyonlar içerir. Her C programı main fonksiyonunu

çalıştırarak başlar.

 printf fonksiyonu tırnak içindeki dizeleri ve deyimlerin değerlerini yazdırmak için

kullanılabilir. Tamsayı değerlerini yazdırırken, printf fonksiyonunun ilk argümanı

(biçim kontrol dizesi) %d dönüşüm belirtecini ve yazdırılacak diğer karakterleri

içerir, ikinci argüman ise değeri yazdırılacak deyimdir. Eğer birden fazla tamsayı

yazdırılacaksa, biçim kontrol dizesi her tamsayı için bir %d içerir ve biçim kontrol

dizesini takip eden, virgüllerle ayrılmış argümanlar, değerleri yazdırılacak deyimleri

içerir.

 scanf fonksiyonu, kullanıcının klavyeden girdiği değerleri alır. Bu fonksiyonun ilk

argümanı, kullanıcı tarafından girilecek verinin tipinin ne olacağını bilgisayara

söyleyen biçim kontrol idzesidr. %d dönüşüm belirteci, verinin bir tamsayı olacağını

belirtir. Geriye kalan tüm argümanlar için biçim kontrol dizesi içinde ilgili bir

dönüşüm belirteci vardır. Bütün değişken isimlerinden önce adres operatörü adı

verilen & kullanılır. Adres operatörü, değişken ismi ile birleştiğinde, bilgisayara

verinin saklanacağı hafıza konumunu söyler. Sonra bilgisayar veriyi bu adreste saklar.

 C’de bütün değişkenler, programda kullanılmadan önce bildirilmelidirler.

 C’ de bir değişken ismi, geçerli herhangi bir tanıtıcı olabilir. Bir tanıtıcı harf, rakam ve

alt çizgi( \_ ) içerebilen karakter serileridir. Tanıtıcılar, rakamla başlayamaz. Tanıtıcılar

her uzunlukta olabilir ancak ANSI standardına göre yalnızca ilk 31 karakter önemlidir.

 C büyük/küçük harf duyarlıdır.

 Hesaplamaların büyük bir kısmı atama ifadeleriyle gerçekleştirilirler.

 Bilgisayarın hafızasında depolanan her değişkenin bir ismi, bir değeri ve tipi vardır.

 Yeni bir değer hafıza konumuna yerleştirildiğinde o konumda daha önceden bulunan

değerin üzerine yazılır. Daha önceki bu bilgi kaybolduğundan, hafıza konumuna bilgi

yazma süreci destructive read-in olarak adlandırılır.

 Hafızadan bir değer okuma işlemine nondestructive read-out denir.

 C’de aritmetik deyimler, matematikte kullanılan bazı özel yazımlardan farklı bir

biçimde yazılır.

 C, aritmetik deyimleri, operatör önceliği ve operatörlerin işleyişi gibi özel kurallarla

hesaplar.

44

 if ifadesi, programcının kesin bir koşul ile karşılaştığında karar vermesini sağlar. if

ifadesinin biçimi

if ( koşul )

ifade

şeklindedir. Eğer koşul doğru ise, if ifadesinin gövdesi içerisindeki ifade çalışır.

Eğer durum yanlış ise gövdedeki ifade atlanır.

 if ifadeleri içindeki koşullar, genellikle eşitlik operatörlerinin yada karşılaştırma

operatörlerinin kullanılmasıyla oluşurlar. Bu operatörleri kullanarak oluşturulan sonuç

her zaman “doğru” ya da “yanlış” biçiminde bir gözlemdir. Koşullar sıfır(yanlış) ya da

sıfır olmayan bir değer(doğru) üreten her hangi bir deyim olabilir.

ÇEVRİLEN TERİMLER

adress operator ...................................adres operatörü

ampersand...........................................ve karakteri ( & )

argument.............................................argüman / (Bilişim Sözlüğüne göre: bağımsız değişken )

arithmetic operators............................aritmetik operatörler

assignment operator ( = )....................atama operatörleri

assignment statement..........................atama ifadesi

associativity of operators....................operatörlerin işleyişi

asterisk................................................yıldız karakteri ( \* )

backslash ( \ ) escape sequence...........ters çizgi çıkış sırası ( Bilişim Sözlüğüne göre)

binary operators..................................ikili operatörler

body of a function...............................fonksiyon gövdesi

braces..................................................küme parantezleri { , }

case sensitive......................................harf duyarlı

character string....................................karakter dizesi

comment..............................................yorum

compile error.......................................derleme hatası

compile-time error...............................derleme zamanlı hata

condition..............................................koşul

control string........................................kontrol dizesi

conversion specifier.............................dönüşüm belirteci

desicion................................................karar

decision making...................................karar verme

declaration............................................bildirim

destructive read in..................................

equality operators..................................eşitlik operatörleri

escape character....................................çıkış karakteri

escape sequence....................................çıkış sırası

fatal error...............................................ölümcül hata

flow of control.......................................akış kontrolü

format control string..............................biçim kontrol dizesi

identifier................................................tanıtıcı (Bilişim Sözlüğüne göre)

if control structure.................................if kontrol yapısı

literal.....................................................hazır bilgi(Bilişim Sözlüğüne göre)

45

keywords...............................................anahtar kelimeler

modulus operator..................................mod operatörü ( % )

nested paranthesis..................................yuvalı parantezler

newline character...................................yeni satır karakteri

nondestructive read out..........................

nonfatal error.........................................ölümcül olmayan hata

precedence.............................................öncelik

prompt....................................................bilgi istemi/komut istemi(Bilişim Sözlüğüne göre)

relational operators.................................karşılaştırma operatörleri (ilişkisel operator de denir)

statement terminator...............................ifade sonlandırıcı

string.......................................................dize (Bilişim Sözlüğüne göre. String kavramı C’de

olukça önemlidir ve bu sebepten bu terimle çok sık karşılaşılır. Kitapta bazı yerlerde bu terim

aynen kullanılacaktır)

syntax error.............................................yazım hatası

whitespace characters.............................Boşluk karakterleri(Bilişim Sözlüğüne göre)

ÖZEL İSİMLER VE KISALTMALAR

int

stdio.h

GENEL PROGRAMLAMA HATALARI

2.1 Yorum satırının sonuna \*/ işaretini koymayı unutmak

2.2 Yorum satırına \*/ ile başlamak ve / veya yorum satırını /\* ile bitirmek.

2.3 printf yerine print yazmak hatadır.

2.4 Küçük harf kullanılması gereken bir yerde büyük harf kullanılması hatadır.Örneğin, main

yerine Main yazmak hatadır.

2.5 Değişken bildirimlerini çalıştırılabilir ifadelerden sonra yapmak hatadır.

2.6 Atama ifadelerinde hesaplama, = operatörünün sağ tarafında bulunmalıdır. Hesaplama

operatörün solunda yapılırsa yazım hatası ortaya çıkar.

2.7 printf ya da scanf içindeki biçim kontrol dizesinde tırnak karakterlerinden birini ya da

ikisini birden unutmak.

2.8 printf ya da scanf içindeki biçim kontrol dizesinde % dönüşüm belirleme karakterini

unutmak.

2.9 printf ya da scanf içindeki \n çıkış sırasını biçim kontrol dizesi dışına yerleştirmek.

2.10 Dönüşüm belirteçleri içeren bir printf ifadesi içine değerleri yazdırılacak ifadeleri dahil

etmeyi unutmak.

2.11 Bir ifade yazdırılacağında, printf biçim kontrol dizesi içinde bir dönüşüm belirteci

yazmamak.

2.12 Biçim kontrol dizesini yazdırılacak ifadelerinden ayırmak için kullanılması gereken

virgül ( , ) karakterini biçim kontrol dizesi içine yazmak.

2.13 scanf ifadesi içinde bir değişkenin başına & operatörü konması gerekirken bu karakteri

unutmak.

2.14 printf ifadesi içinde bir değişkenin başına & yazmak.

2.15 Bir sayıyı 0'a bölmek bilgisayar sistemlerinde tanımlı değildir ve genellikle ölümcül bir

hatadır. Ölümcül hatalar, programın çalışmasının aniden durmasına ve başarılı bir sonuç

46

vermemesine sebep olur. Ölümcül olmayan hatalar, programın yanlış sonuçlar vermesine

sebep olur.

2.16 = =, !=, >= ve <= operatörlerinin arasında boşluk kullanılması yazım hatasına sebep olur.

2.17 !=, >=, <= operatörlerinin ters çevrilerek =!, =>, =< şeklinde kullanılması dizim

hatasına sebep olur.

2.18 = = operatörünün = ile karıştırılması.

2.19 if yapısının koşulunu belirten parantezlerin sağına noktalı virgül ( ; ) koymak

2.20scanf ifadesi içindeki dönüşüm belirteçleri arasına (asla gerekmemesine rağmen)

virgül yerleştirmek.

İYİ PROGRAMLAMA ALIŞTIRMALARI

2.1 <stdio.h> öncü dosyasının eklenmesi tercihe bağlıdır fakat standart giriş /çıkış

fonksiyonlarının kullanıldığı programlara eklenmelidir. Bu sayede, derleyici, hataları

derleme anında bulabilecektir.Aksi takdirde, hatalar programın çalıştırıldığı anda ortaya

çıkar. Bu tür hataların düzeltilmesi oldukça güç olur.

2.2 Her fonksiyondan sonra fonksiyonu anlatan bir yorum satırı yazılmalıdır.

2.3 Yazdırma işlemi yapan bir fonksiyon tarafından yazdırılan son karakter ( \n ) olmalıdır.

Bu sayede, fonksiyonun ekran imlecini yeni satırın başlangıcına götürmesi sağlanır. Bu

tarz bize, yazılım geliştirme ortamlarında temel amaç olan yazılımın yeniden

kullanılabilirliğini artırma fırsatı verir.

2.4 Fonksiyon bloklarının içini yazarken, küme parantezlerinden içeride başlamak (3 boşluk

bırakarak) fonksiyonun gövdesini daha belirgin hale getirir.Bu sayede programımız daha

okunur hale gelecektir.

2.5 Kendinize göre bir girinti miktarı belirleyin ve gerekli tüm yerlerde bu girinti miktarını

kullanın. Girintiler yaratmak için tab tuşu kullanılabilir ancak kimi zaman tab

başlangıçları sorun yaratabilir.

2.6 Virgülden hemen sonra bir boşluk bırakmak programın okunurluğunu arttırır.

2.7 Anlamlı değişken isimleri kullanmak, programda daha az yorum satırı yazmamız

demektir.

2.8 Basit bir değişken olarak kullanılacak tanıtıcılar küçük harfle başlamalıdır. İleride büyük

harfle başlayacak ya da tüm harfleri büyük harf olan, özel öneme sahip değişkenlerden

bahsedeceğiz.

2.9 Bir çok kelimeden oluşan değişken isimleri programı daha okunabilir yapar.Ancak

kelimeleri birleşik yazmaktan kaçının.Bunun yerine kelimelerin arasında alt çizgi

kullanın.Eğer kelimeleri birleşik yazmak istiyorsanız, ikinci kelimeden sonrasını büyük

harfle başlatın. toplamkomisyon yerine toplam\_komisyon ya da toplamKomisyon

yazın.

2.10 Bildirimlerle, çalıştırılabilir ifadeler arsında boş bir satır bırakmak bildirimlerin sona

erdiğini vurgulamaya yarar.

2.11 Operatörün her iki tarafına da bir boşluk bırakılmalıdır. Bu sayede program daha

okunabilir olur.

2.12 if yapısının gövdesi içindeki ifadeleri içeriden başlatmak.

2.13 Programlarda her kontrol yapısından önce ve sonra boş bir satır kullanmak

programın okunulabilirliğini arttırır.

2.14 Bir satırda birden fazla ifade bulunmamalıdır.

2.15 Uzun bir ifade birden çok satır sürebilir.Eğer bir ifade birden fazla satır sürecekse,

ifadeyi mantıklı noktalardan ayırmak(örneğin virgüllerle ayrılmış bir listede virgülden

47

sonra) gerekir. Eğer bir ifade birden çok satır sürüyorsa, ifadenin sürdüğü tüm satırlar

içeriden başlatılmalıdır.

2.16 Birden fazla operatör içeren deyimler yazdığınızda operatör önceliklerini gösteren

tabloya bakınız.Deyimin içindeki operatörlerin uygun biçimde kullanıldığından emin

olunuz. Eğer karmaşık bir deyim içinde hesaplama sırasından emin olamazsanız,sırayı

istediğiniz şekle getirmek için (matematikte olduğu gibi) parantezleri kullanın.C’nin

bazı operatörlerinin (örneğin atama operatörü (=) gibi ) soldan sağa değil de,sağdan

sola doğru işlediğini gözden kaçırmayın.

TAŞINABİLİRLİK İPUÇLARI

2.1 Belirteçlerinizi 31 karakterden kısa tutmaya çalışın.Bu taşınırlığı artıracaktır.

CEVAPLI ALIŞTIRMALAR

2.1 Aşağıdaki boşlukları doldurunuz.

a) Bütün C programları \_\_\_\_\_\_\_ fonksiyonunun çağırılması ile başlar.

b) Bütün fonksiyonlar \_\_\_\_\_\_\_ ile başlar ve \_\_\_\_\_\_\_ ile biter

c) Bütün ifadeler \_\_\_\_\_\_\_ ile biter.

d) \_\_\_\_\_\_\_ standart kütüphane fonksiyonu bilgiyi ekrana yazdırır.

e) \n çıkış sırası \_\_\_\_\_\_\_ karakterini temsil eder ve bu karakter imlecin bir alt satırın

başına gitmesini sağlar.

f) \_\_\_\_\_\_\_ standart kütüphane fonksiyonu klavyeden veri girişini sağlar.

g) \_\_\_\_\_\_\_ dönüşüm belirteci, scanf ile kullanıldığında bir tamsayı girişi yapılması

gerektiğini, printf ile kullanıldığında ise ekrana bir tamsayı yazdırılacağını belirtir.

h) Hafızadaki bir bölgeye yeni bir değer yazıldığında bu bölgede eski değerin silinmesine

\_\_\_\_\_\_\_ read – in denir.

i) \_\_\_\_\_\_\_ ifadesi karar vermek için kullanılır.

2.2 Aşağıdakilerin hangilerinin doğru, hangilerinin yanlış olduğuna karar veriniz ve yanlış

olanların neden yanlış olduğunu açıklayınız.

a) printf fonksiyonu her çağrıldığında, yazdırmaya bir alt satırın başından başlar.

b) /\*ve \*/ arasına yazılan yorumlar ekrana yazdırılır.

c) \n çıkış sırası printf fonksiyonunda kullanıldığında, imlecin bir sonraki satırın

başlangıç pozisyonuna gelmesini sağlar.

d) Bütün değişkenler, kullanılmadan önce bildirilmelidirler.

e) Değişkenler bildirilirken tiplerinin belirlenmesi lazımdır.

f) C, sayi ve SaYi değişkenlerini eş kabul eder.

g) Değişken bildirimleri fonksiyonun içinde herhangi bir yerde yapılabilir.

h) printf’te kullanılan bütün argümanlardan önce & karakteri yazılmalıdır.

i) Mod operatörü (%) sadece tamsayılarla işlem gerçekleştirir.

j) \*, /, %, + ve – aritmetik operatörleri aynı öncelik sırasına sahiptirler.

k) Aşağıdaki değişken isimleri bütün ANSI C sistemlerinde eş olarak kabul edilirler.

bucoookuzunbirdegiskendir1234567

bucoookuzunbirdegiskendir1234568

48

l) Üç satırlık bir çıktı veren C programı mutlaka üç adet printf ifadesi içermelidir.

2.3 Aşağıdakileri gerçekleştiren C ifadelerini yazınız.

a) c, buDegisken, q76354 ve sayi değişkenlerini int tipinde olacak şekilde bildiriniz.

b) Kullanıcıya, bir sayı girmesini söyleyen bir mesaj içeren printf ifadesini yazın.

Mesajınızın iki nokta üst üste ( : ) karakteri ile bitirin.

c) Klavyeden bir sayı girişi yapın ve bunu int türündeki a değişkeninde saklayın.

d) Eğer sayi değişkeni 7 değilse “Bu sayı 7 değil” yazdırın.

e) “Bu, bir C programıdır.” mesajını ekranda tek satıra yazdırın.

f) “Bu, bir C programıdır.” mesajını ekranda iki satıra yazdırın. İlk satır, C ile bitsin.

g) “Bu, bir C programıdır.” mesajını ekranda her kelime ayrı bir satırda olacak şekilde

yazdırın.

h) “Bu, bir C programıdır.” mesajını ekranda her kelimenin arasında tab karakter

boşluğu olacak şekilde yazdırın.

2.4 Aşağıdakileri gerçekleştirmek için birer C ifadesi (yada yorumu) yazınız.

a) Programınızın, 3 tamsayının çarpımını bulacağını, programınız içerinde belirtin.

b) int tipinde x, y, z ve sonuc değişkenleri bildirin.

c) Kullanıcıya üç tamsayı girmesini söyleyen ifadeyi yazın.

d) Klavyeden 3 tamsayı girişi yapın ve bunları x, y ve z değişkenlerinde saklayın.

e) x, y ve z değişkenlerinin içerdiği üç tamsayının çarpımını bulun ve çarpımdan elde

edilen sonucu, sonuc değişkeni içinde saklayın.

f) sonuc değişkeninin içeriğini “Sonuç =” bilgisinden sonra gelecek şekilde ekrana

yazdırın.

2.5 Alıştırma 2.4’ te yazdığınız ifadelerden yararlanarak üç tamsayının çarpımını hesaplayan

bir program yazın.

2.6 Aşağıdaki ifadelerdeki hataları bulun ve düzeltin.

a) printf ( “ Sayı %d\n”, &sayi);

b) scanf( “%d%d”, &sayi1, sayi2);

c) if ( c < 7 );

printf ( “C 7’den küçüktür.\n” );

d) if ( c => 7 )

printf (“ C 7’den küçük ya da eşittir.\n “);

ÇÖZÜMLER

2.1 a) main. b) sol küme parantezi ({), sağ küme parantezi (}). c) noktalı virgül (;). d) printf.

e) yeni satır. f) scanf. g) %d h) destructive i) non-destructive j) if.

2.2

a) Yanlış. printf fonksiyonu, imleç en son hangi pozisyonda kaldıysa yazdırmaya oradan

devam eder.

49

b) Yanlış. Yorumlar programın çalışmasını kesinlikle etkilemezler. Programın

okunurluğunu artırmak için kullanılırlar.

c) Doğru

d) Doğru

e) Doğru

f) Yanlış. C, büyük ve küçük harfe duyarlıdır. Bu değişkenler farklı algılanır.

g) Yanlış. Değişkenlerin bildirimi, fonksiyonun sol küme parantezinden hemen sonra ({ )

ve diğer çalıştırılabilir ifadelerden önce yapılmalıdır.

h) Yanlış. printf fonksiyonun argümanlarından önce & karakteri kullanılmaz. Bu

karakter scanf argümanlarından önce kullanılır. Bu konu 6. ve 7. Ünitelerde

işlenecektir.

i) Doğru

j) Yanlış. \*, /, % operatörleri aynı öncelik seviyesine sahipken, + ve – operatörleri daha

düşük öncelik seviyesindedirler.

k) Yanlış. Bazı sistemlerde 31 karakterden daha uzun değişkenler bildirilebilir.

l) Yanlış. Tek bir printf ifadesi içerisinde \n kullanılarak birden fazla satırın yazılması

sağlanabilir.

2.3

a) int c, buDegisken, q76354, sayi;

b) printf ( “Bir sayı girin : ” );

c) scanf( “%d”, &a );

d) if ( sayi != 7)

printf ( “Bu sayı 7 değil.\n” );

e) printf ( “Bu, bir C programıdır.\n” );

f) printf ( “Bu, bir C\nprogramıdır.\n” );

g) printf ( “Bu,\nbir\nC\nprogramıdır.\n” );

h) printf ( “Bu,\tbir\tC\tprogramıdır.\n” );

2.4

a) /\* Üç tamsayının çarpımının hesaplanması \*/

b) int x, y, z, sonuc;

c) printf ( “Üç tamsayı girin : “ );

d) scanf( “%d%d%d”, &x, &y, &z);

e) sonuc = x \* y \* z;

f) printf ( “ Sonuç = %d\n“, sonuc);

2.5 Program aşağıdaki gibidir.

1 /\* Üç tamsayının çarpımının hesaplanması \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int x, y, z, sonuc;

8

9 printf ( “Üç tamsayı girin : “ );

10 scanf( “%d%d%d”, &x, &y, &z);

11 sonuc = x \* y \* z;

12 printf ( “ Sonuç = %d\n“, sonuc);

13

14 return 0;

50

15 }

2.6

a) Hata: &sayi. Düzeltme: & ‘nın silinmesi. İleride bu konu üzerinde durulacaktır.

b) Hata: sayi2, & karakteri kullanılmadan yazılmıştır. &sayi2 şeklinde kullanılmalıdır.

İleride bu konu üzerinde durulacaktır.

c) Hata: if ifadesinin sağ parantezinden sonra noktalı virgül kullanılması.

Düzeltme: Sözü geçen noktalı virgülün kaldırılması. Not: Bu hatanın sonucunda,

printf ifadesi, if ifadesinin içersindeki koşul sağlansada sağlanmasa da işlem görür.

Sağ parantezden sonraki noktalı virgül, boş bir ifadeymiş gibi algılanır.Boş ifadelerde

hiçbir işlem yapılmaz.

d) Hata: Karşılaştırma operatörü => , >= şeklinde kullanılmalıdır.

ALIŞTIRMALAR

2.7 Aşağıdaki ifadelerde hataları bulun ve düzeltin. (Not: Bir ifade birden fazla hata içeriyor

olabilir. )

a) scanf( “d”, sayi );

b) printf( “%d ve %d nin çarpımı %d dir.”\n, x, y );

c) ilkSayi + ikinciSayi = sayilarinToplami

d) if (sayi >= enBuyuk)

enBuyuk == sayi;

e) \* / Üç tamsayının en büyüğünü bulan program / \*

f) Scanf( “%d”, birTamsayi);

g) printf( “%d nin %d ye bölümünden kalan : \n”, x, y, x % y );

h) if ( x = y );

printf ( %d %d ye eşittir.\n”, x, y );

i) print( “Toplam %d dir.\n, ” x + y );

j) Printf( “Girdiğiniz Sayı : %d\n, &sayi );

2.8 Aşağıdaki boşlukları doldurun.

a) \_\_\_\_\_\_\_\_, program hakkında yorumların yazılması ve okunurluğu artırmak için

kullanılır.

b) Ekrana bilgi yazdırmak için kullanılan fonksiyon \_\_\_\_\_\_\_\_ dir.

c) Karar vermek için kullanılan C ifadesi, \_\_\_\_\_\_\_\_ dir.

d) Hesaplamalar normal olarak \_\_\_\_\_\_\_\_ ifadeleriyle yapılır.

e) \_\_\_\_\_\_\_\_ fonksiyonu klavyeden giriş yapmayı sağlar.

2.9 Aşağıdakiler için basit bir C ifadesi yazın.

a) “İki sayı girin” mesajını ekrana yazdırın.

b) b ve c değişkenlerinin çarpımını a değişkenine atayın.

c) Basit bir ücret hesaplaması yapan program yazın ( programınız, program hakkında

bilgi içeren bir metne yer versin)

d) Klavyeden üç tamsayı girişi yaptırın ve bu tamsayıları a, b, c değişkenlerine

yerleştirin.

51

2.10 Aşağıdakilerden hangilerinin doğru, hangilerinin yanlış olduğuna karar verin. Yanlış

olanlar için açıklama yapın.

a) C operatörleri, soldan sağa doğru işlem görür.

b) \_limit\_alti\_, m928134, t5, j7, onun\_satislari, onun\_toplam\_hesabi, a, b, c, z, z2

ifadelerinin her biri değişken olarak bildirilebilir.

c) printf ( “a = 5;” ); tipik bir atama ifadesi örneğidir.

d) Doğru yazılmış bir C aritmetik satırı, parantez içermiyorsa, soldan sağa işlem görür.

e) 3g, 87, 67h2, h22, 2h ifadelerinin her biri değişken olarak bildirilebilir.

2.11 Aşağıdaki boşlukları doldurun.

a) Hangi aritmetik operatörler çarpma ile aynı öncelik seviyesine sahiptir ? \_\_\_\_\_\_\_\_

b) Yuvalı parantezler içeren bir aritmetik işlemde hangi parantez çiftleri ilk önce

işlem görür ? \_\_\_\_\_\_\_\_

c) Programın çalışması sürecinde, farklı zamanlarda farklı değerler içeren hafıza

bölümüne \_\_\_\_\_\_\_\_ denir.

2.12 Aşağıdaki C ifadeleri çalıştığında eğer bir ekran çıktısı alınıyorsa bu nedir? Bir çıktı

alınmıyorsa “hiçbir şey” olarak cevaplandırın. x = 2 ve y = 3 olarak kabul edin.

a) printf ( “%d”, x );

b) printf ( “%d”, x + x );

c) printf ( “x=” );

d) printf ( “x=%d”, x );

e) printf ( “%d = %d”, x + y, y + x );

f) z = x + y;

g) scanf ( “%d%d”, &x, &y );

h) /\* printf ( “x + y = %d”, x + y ); \*/

i) printf ( “\n” );

2.13 Aşağıdaki C ifadelerden hangileri, eğer içeriyorsa, destructive read-in değişkenleri

içermektedir?

a) scanf ( “%d%d%d%d%d”, &b, &c, &d, &e, &f );

b) p = i + j + k + 7;

c) printf ( “Destructive read-in” );

d) printf ( “a = 5” );

2.14 Aşağıdakilerden hangisi ya da hangileri y = ax3

+ 7 eşitliğini doğru olarak ifade

etmektedir.

a) y = a \* x \* x \* x + 7;

b) y = a \* x \* x \* ( x + 7 );

c) y = ( a \* x ) \* x \* ( x + 7 );

d) y = ( a \* x ) \* x \* x + 7;

e) y = a \* ( x \* x \* x ) + 7;

f) y = a \* x \* ( x \* x + 7 );

52

2.15 Aşağıdaki C ifadelerin içerdiği operatörlerin öncelik sırasını belirtin ve her ifade için

x’in alacağı son değeri hesaplayın.

a) x = 7 + 3 \* 6 / 2 – 1;

b) x = 2 % 2 + 2 \* 2 – 2 / 2;

c) x = ( 3 \* 9 \* ( 3 + ( 9 \* 3 / ( 3 ) ) ) );

2.16 Kullanıcıdan iki tamsayı girmesini isteyen ve sayıları kullanıcıdan aldıktan sonra bu

sayıların toplamlarını, çarpımlarını, farklarını, bölümlerini, modlarını bulan bir program

yazınız.

2.17 1’ den 4’ e kadar olan tamsayıları ekrana tek satırda görülecek şekilde yazdıran bir

programı aşağıdaki metotları kullanarak yazın.

a) Bir printf ifadesi kullanarak ve hiç dönüşüm belirteci kullanmadan

b) Bir printf ifadesi ve dört dönüşüm belirteci kullanarak

c) Dört printf ifadesi kullanarak

2.18 Kullanıcıdan iki tamsayı girmesini isteyen, kullanıcıdan bu sayıları alan ve daha sonra

büyük olan sayı ile beraber “en büyüktür” ifadesini yazan bir program yazınız. Eğer sayılar

birbirine eşitse “Bu sayılar eşit” mesajını yazdırın. Bu ünitede görmüş olduğunuz tek seçimli

if ifadesini kullanın.

2.19 Klavyeden 3 farklı tamsayı girişi yaptıran, daha sonra bu sayıların toplamını,

ortalamasını, çarpımını, en küçüğünü ve en büyüğünü hesaplayan bir program yazın.

Programınız bu ünitede görmüş olduğunuz tek seçimli if ifadesini içersin. Ekran çıktısı

aşağıdaki şekilde olmalıdır:

Üç farklı tamsayı giriniz: 13 27 54

Toplam :54

Ortalama :18

Çarpım: 4914

En küçük: 13

En büyük: 27

2.20 Kullanıcıdan bir çemberin yarıçapını alan ve bu çemberin çapını, çevresini, alanını

hesaplayan bir program yazın. π için 3.14159 değerini kullanın. Bütün hesaplamalarınızı

kullandığınız printf ifadeleri içinde yaptırın. %f dönüşüm belirtecini kullanın. (Not : Bu

ünitede sadece tamsayı sabitleri ve değişkenlerinden bahsettik. 3. ünitede ondalıklı sayılardan

bahsedeceğiz. )

2.21 Ekrana aşağıdaki gibi bir dikdörtgen, elips, ok ve eşkenar dörtgen çizen bir program

yazın.

\*\*\*\*\*\*\*\*\* \*\*\* \* \*

\* \* \* \* \*\*\* \* \*

\* \* \* \* \*\*\*\*\* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \*

53

\* \* \* \* \* \* \*

\* \* \* \* \* \* \*

\*\*\*\*\*\*\*\*\* \*\*\* \* \*

2.22 Aşağıdaki kod ekrana ne yazdırır?

printf ( “\*\n\*\*\n\*\*\*\n\*\*\*\*\n\*\*\*\*\*\n”);

2.23 Kullanıcıdan beş tamsayı alan ve bu sayıların en büyüğünü ve en küçüğünü bulan ve

ekrana yazan bir program yazınız. Sadece bu ünitede gördüğünüz programlama tekniklerini

kullanın.

2.24 Kullanıcıdan bir tamsayı alan ve bu sayının tek bir tamsayı mı yoksa çift bir tamsayı

mı olduğunu hesaplayan ve sonucu ekrana yazan bir program yazınız. (İpucu: Mod

operatörünü kullanın. Çift sayılar ikinin tam katıdır. İkinin tam katı olan sayılar ikiye

bölündüklerinde 0 kalanını verirler.

2.25 Ad ve Soyadınızın baş harflerini kullanarak aşağıda ki gibi blok harfler oluşturun.

PPPPPPPP

P P

P P

P P

P P

JJ

J

J

J

JJJJJJJJJJJ

DDDDDDDDD

D D

D D

D D

DDDDD

2.26 İki tamsayı alan ve birinci tamsayının ikinci tamsayının tam katı olup olmadığını

hesaplayan ve sonucu ekranda gösteren programı yazınız. (İpucu: Mod operatörünü kullanın.)

2.27 Aşağıdaki deseni, sekiz printf ifadesiyle ekrana yazdıran bir program yazın. Daha

sonra aynı deseni, kullanabileceğiniz en az printf ifadesiyle yazdırın.

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

54

\* \* \* \* \* \* \* \*

2.28 Ölümcül programlama hataları ile ölümcül olmayan programlama hataları arasındaki

farkı açıklayınız. Neden ölümcül bir hata yapmak, ölümcül olmayan bir hata yapmaya göre

tercih edilir ?

2.29 Bu ünitede tamsayıları ve int tipini öğrendiniz. C, tabi ki büyük harfleri, küçük harfleri

ve çeşitli özel sembolleri de destekler. C’de küçük tamsayılar, değişik karakterleri

simgelerler. Bilgisayarın kullandığı bu karakter grubu ve bu karakterleri temsil eden

tamsayılara bilgisayarın karakter seti denir. Büyük A harfinin tamsayı eşitini aşağıdaki printf

ifadesiyle ekrana yazdırabilirsiniz.

printf( “%d”, ‘A’) ;

Bazı büyük harflerin, küçük harflerin, rakamların ve sembollerin tamsayı eşitlerini ekrana

yazdıran bir program yazınız. Örneğin : A B C a b c 0 1 2 $ \* + / ve boşluk karakterinin

tamsayı eşitlerini ekrana yazdırınız.

2.30 Beş basamaklı bir sayı girişi yapılan, bu sayıyı ayrı ayrı basamaklarına ayıran ve her

basamak arasına üç boşluk karakteri koyarak ekrana yazdıran bir program yazın. [İpucu : Tam

sayı bölme işlemlerini ve mod operatörünü beraber kullanın.] Örneğin, eğer kullanıcı 42139

girdiyse ekran çıktısı aşağıdaki gibi olmalıdır.

4 2 1 3 9

2.31 Sadece bu ünitede öğrendiğiniz programlama tekniklerini kullanarak 0’dan 10’a kadar

olan sayıların karelerini ve küplerini hesaplayıp, sonuçları ekrana aşağıda göründüğü biçimde

yazdıran bir program yazınız.(İpucu : sonuçları yazdırırken \t kullanın.)

sayı karesi küpü

0 0 0

1 1 1

2 4 8

3 9 27

4 16 64

5 25 125

6 36 216

7 49 343

8 64 512

9 81 729

10 100 1000

55

C'DE YAPISAL PROGRAM GELİŞTİRME

AMAÇLAR

 Temel problem çözme tekniklerini anlamak.

 Yukarıdan aşağıya adımsal iyileştirme süreciyle algoritmalar geliştirebilmek.

 if ve if/else yapılarını uygun yerde kullanarak işlemleri seçebilmek.

 while yapısını programlarda uygulayarak, ifadeleri tekrar tekrar çalıştırabilmek

 Sayıcı kontrollü döngü ve nöbetçi kontrollü döngüleri anlamak.

 Yapısal programlamayı anlamak

 Arttırma, azaltma ve atama işlemlerini kullanabilmek

BAŞLIKLAR

3.1 GİRİŞ

3.2 ALGORİTMALAR

3.3 SAHTE KODLAR

3.4 KONTROL YAPILARI

3.5 if SEÇİM YAPISI

3.6 if/else SEÇİM YAPISI

3.7 while DÖNGÜ YAPISI

3.8 ALGORİTMALARI UYGULAMAK : DURUM 1(SAYICI KONTROLLÜ DÖNGÜ)

3.9 YUKARIDAN AŞAĞI ADIMSAL KONTROL İLE ALGORİTMALARI

UYGULAMAK: DURUM 2 (NÖBETÇİ KONTROLLÜ DÖNGÜLER)

3.10YUKARIDAN AŞAĞI ADIMSAL KONTROL İLE ALGORİTMALARI

UYGULAMAK: DURUM 2 (YUVALI KONTROL YAPILARI)

3.11 ATAMA OPERATÖRLERİ

3.12 ARTTIRMA VE AZALTMA OPERATÖRLERİ

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Performans İpuçları\*

Taşınırlık İpuçları\*Yazılım Mühendisliği Gözlemleri\*Çözümlü Alıştırmalar\* Çözümler\*

Alıştırmalar

3.1 GİRİŞ

Bir problemi çözmeden önce, onu anlamak ve çözüm için en uygun çözüm yolunu planlamak

oldukça önemlidir. Önümüzdeki iki kısım boyunca, yapısal bilgisayar programları geliştirme

tekniklerini anlatacağız. Kısım 4.12’de, iki ünite boyunca anlattığımız tekniklerin özetini

bulacaksınız.

3.2 ALGORİTMALAR

Bir problemin çözümü, bir dizi işlemin belirli bir sırada çalıştırılmasını içerir. Bir problemin

çözülmesindeki yordam;

1- Uygulanacak işlemler ve

2- Bu işlemlerin hangi sırada uygulanacağı

56

algoritma olarak adlandırılır. Aşağıdaki örnek, işlemlerin uygulanma sıralarının doğru

belirlenmesinin önemli olduğunu gösterir.

Yataktan kalkıp, işe gidene kadar yapılacak işlemleri anlatan bir algoritmayı inceleyelim.

Yataktan kalk

Pijamalarını çıkar

Duş al

Giyin

Kahvaltı yap

İşe doğru yola çık

Bu algoritma, işe iyi hazırlanmış bir şekilde gitmeyi sağlatır. Şimdi de, aynı işlemlerin farklı

bir sırada uygulandığını düşünelim.

Yataktan kalk

Pijamalarını çıkart

Giyin

Duş al

Kahvaltı yap

İşe doğru yola koyul

Bu algoritma uygulanırsa işe ıslak elbiselerle gidilir. Bir programda, yapılacak işlemlerin

sırasını belirlemeye program kontrolü denir. Bu ünite ve 4. ünitede, C'nin program kontrol

yeteneklerinden bahsedeceğiz.

3.3 SAHTE KODLAR

Sahte kodlar, bir programcının algoritma yazmada kullandığı yapay ve mantıksal dildir.

Bu ünitede, yapısal C programlarına çevrilebilecek algoritmaları geliştirecek sahte kodları

yazmaya çalışacağız. Sahte kodlar, her gün konuştuğumuz dile oldukça yakındır.

Bu kodları bilgisayarda çalıştıramayız. Ancak bu kodlar programcıya, programını C gibi

herhangi bir programlama diliyle yazmadan önce, programı hakkında düşünme fırsatını verir.

Sahte kodların, yapısal C programları geliştirmede nasıl kullanılacağını gösteren bir çok örnek

vereceğiz.

Sahte kodlar, yalnızca karakterlerden oluştuğu için programcı sahte kodları bir editör program

sayesinde bilgisayara yazar. Bilgisayar, sahte kodlarla yazılmış programı ekranda gösterebilir

ya da yazıcıdan çıktı olarak verebilir. Dikkatlice hazırlanmış sahte kodlar, C programına

kolaylıkla çevrilebilir. Bunu yapmak için, çoğu durumda sahte kodlarla yazılmış ifadeleri

C’deki biçimleriyle değiştirmek yeterlidir.

Sahte kodlar sadece işlem ifadelerini içerir. İşlem ifadeleri, sahte kodlar C’ye çevrildiğinde,

C’de çalıştırılabilecek ifadelerdir. Bildirimler, çalıştırılabilir ifadeler değildir. Bunlar,

derleyiciye gönderilen mesajlardır. Örneğin;

int i;

bildirimi, derleyiciye i değişkeninin tipini bildirir ve derleyiciden, hafızada bu değişken için

yer ayırmasını ister. Ancak bu bildirim, program çalıştırıldığında giriş, çıkış ya da hesaplama

gibi bir işleme sebep olmaz. Çoğu programcı, sahte kodlar yazarken değişkenleri en başta

57

listelemeyi ve bu değişkenlerin amacını belirtmeyi uygun görür. Yeniden söylememiz

gerekirse, sahte kodlar bir program geliştirme aracıdır.

3.4 KONTROL YAPILARI

Genelde ifadeler, programa yazıldıkları sırada teker teker çalışırlar. Buna sıralı çalışma denir.

Ancak, bazı özel C komutları bu sırayı değiştirmemize yardımcı olur. Böylece, sıradaki ifade

yerine istenen ifadenin çalıştırılması sağlanabilir. Buna kontrolün transferi denir.

1960'larda yazılım geliştirme grupları, kontrol transferlerinin karışık bir biçimde kullanımının

büyük sorunlara yol açtığını fark ettiler. Bu zorluğun sebebi olarak, programcının goto

ifadesiyle program içerisinde oldukça geniş bir alanda hareket edebilmesi gösterildi. Yapısal

programlama, goto ifadelerinin elenmesiyle aynı anlama gelmeye başladı.

Bohm ve Jacobi'nin çalışmaları, programların goto ifadeleri kullanmadan yazılabileceğini

gösterdi. Programcılar, goto ifadesini daha az kullanmak zorunda kaldıkları bir döneme

girdiler. Yine de profesyonel programcıların, yapısal programlamayı ciddiye almaları

1970'lerde gerçekleşti. Bunu, yazılım gruplarının yapısal programlama tekniklerini kullanarak

geliştirme zamanını azaltmaları, sistemlerin zamanında dağıtımını sağlamaları gerçekleştirdi.

Başarının anahtarı, yapısal programlamayla daha açık programlar yazma ve ilk etapta

programlarda değişiklikler yapma ve hata ayıklamanın daha kolay gerçekleştirilmesiydi.

Bohm ve Jacobi'nin çalışması, programların yalnızca üç kontrol yapısıyla yazılabileceğini

göstermişti. Bunlar : sıra yapısı, seçme yapısı ve döngü yapısıydı. Dizi yapısı, C'yi oluşturur.

Aksi belirtilmedikçe bilgisayar, C ifadelerini yazıldıkları sırada çalıştırır. Şekil 3.1’deki akış

garfiği parçası, C'nin sıra yapısını gösterir.

toplam = toplam + not;

sayici = sayici +1;

Şekil 3.1 C'nin dizi yapısı akış grafiği.

Akış grafikleri, algoritmanın ya da algoritmanın bir kısmının grafik gösterimidir. Akış

grafikleri genelde dikdörtgen, çember, elmas gibi bazı özel kullanımlı sembollerden oluşur.

Bu semboller, akış çizgileri ile birbirlerine bağlanır.

Sahte kodlar gibi akış grafikleri de algoritmaları yazma ve geliştirmekte kullanılır. Ancak

sahte kodlar, programcılar tarafından daha yaygın olarak kullanılmaktadır. Akış grafikleri,

kontrol yapılarının nasıl çalıştığını gösterir. Biz de, kitabımızda, akış grafiklerini bu amaçla

kullandık. Şekil 3.1’de sıra yapısı için akış grafiği parçasını inceleyiniz. Dikdörtgen sembolü

notu toplama ekle

sayiciya 1 ekle

58

ya da işlem sembolü, giriş/çıkış ya da hesaplama gibi herhangi bir işlemi belirtmek için

kullanılır. Şekildeki akış çizgileri, işlemlerin sırasını belirtir: İlk önce not, toplam’a

eklenecek, sonra da sayiciya bir eklenecektir. C, sıra yapısıyla istediğimiz kadar işlem

yaptırmamıza izin verir. İleride göreceğimiz gibi, tek bir işlemi yerleştirdiğimiz yere sıralı bir

biçimde bir çok işlem yerleştirebiliriz.

Akış grafiğiyle tam bir algoritmayı göstermek istersek, akış grafiğinin başına, içinde

başlangıç yazan oval bir sembol yerleştiririz. Algoritmanın sonuna ise, içinde bitiş yazan

oval bir sembol yerleştiririz. Eğer Şekil 3.1’de olduğu gibi, akış grafiğinin sadece bir kısmını

göstermek istersek, bu kısmın başka kısımlara bağlanacağını belirten çember sembollerini

kullanırız.

Bir akış grafiğindeki en önemli işaret ise elmas sembolüdür. Bu işaret, aynı zamanda karar

işareti olarak da adlandırılır ve bir karar verileceğini gösterir. Bu sembolü ve kullanımlarını

gelecek kısımda tartışacağız.

C'nin üç çeşit seçim yapısı vardır. if seçim yapısı (Kısım 3.5), eğer koşul doğru ise işi yapar,

yanlışsa diğer satıra geçerek devam eder. if/else seçim yapısı(Kısım 3.6), eğer koşul doğru ise

bir işi, yanlış ise başka bir işi yapar. 4.ünitede anlatacağımız switch seçim yapısı, bir deyimin

değerine göre farklı işlemlerden birini yapar.

if yapısı, tekli seçim yapısı olarak bilinir çünkü tek bir işlemi seçer ya da ihmal eder. if /else

yapısı, çiftli seçim yapısı olarak bilinir çünkü iki farklı işlem arasından seçim yapar. switch

seçim yapısı, çoklu seçim yapısı olarak adlandırılır çünkü bir çok işlem arasından seçim yapar.

C'nin üç çeşit döngü yapısı vardır ; while (kısım 3.7), 4.ünitede anlatacağımız do-while ve

for döngü yapıları.

İşte tüm bunlar, C'yi oluşturur. C'de yalnızca yedi kontrol yapısı vardır. Sıra yapısı, üç adet

seçim ve üç adet döngü yapısıyla yazamayacağımız program yoktur. Her C programı, her

türden seçim yapısı ve/veya döngü yapısının belli bir sırada yerleştirilmesinden ibarettir. Şekil

3.1’deki sıra yapısında oluğu gibi her kontrol yapısının, biri kontrol yapısına girişte, diğeri

kontrol yapısından çıkışta olmak üzere iki çember sembolü vardır. Bu tek giriş/tek çıkışlı

kontrol yapıları, program oluşturmayı kolaylaştırır. Kontrol yapıları, birinin çıkış noktasıyla

diğerinin giriş noktasını bağlanarak birleştirilebilir. Bu, bir çocuğun blokları üst üste

yığmasına benzer. Bu sebepten adı "kontrol yapısı yığma"dır. İleride kontrol yapısı yuvalama

adı verdiğimiz bir bağlama biçimi daha öğreneceğiz. Böylece, bir C programını yedi kontrol

yapısı kullanarak ve bu kontrol yapılarını iki biçimde birleştirerek yazabileceğiz.

3.5 if SEÇİM YAPISI

Seçme yapısı, bir işin değişik yönlerinden birini seçmek için kullanılır. Bir sınavın geçme

notunun 60 olduğunu varsayalım.

Eğer (if) öğrencinin notu 60'dan büyükse ya da 60'a eşitse

“Geçtiniz” yazdır.

biçimindeki sahte kod, öğrencinin notunun 60'a eşit ya da 60’tan büyük olma koşulunun

doğru ya da yanlış olduğuna karar verir. Eğer koşul doğru ise “Geçtiniz” yazdırılır ve diğer

sahte koda geçilir. Eğer koşul yanlış ise yazdırma işlemi yaptırılmaz ve doğrudan bir sonraki

sahte koda geçilir ( Sahte kodun gerçek bir programlama dili olmadığını hatırlayınız ). Dikkat

59

edilirse, bu seçim yapısının ikinci satırı daha içeriden başlatılmıştır. Bu biçimde gösterim

tercihe bağlıdır. Ancak, yapısal programların yapısını vurgulaması açısından tavsiye edilir. Bu

biçimi, bu kitabın tümü boyunca uygulayacağız. C derleyicisi, boşluk karakterlerini, örneğin,

tab, yeni satır ve boşluk gibi karakterleri ihmal eder, önemsemez.

İyi Programlama Alıştırmaları 3.1

İçeriden başlayarak yazma gösteriminin uygulanması, programın okunulabilirliğini

artırır. İçeriden başlamak için üç boşluk bırakılmasını tavsiye ediyoruz.

Sahte kodu, C’de yazmak istersek

if ( not >= 60 )

printf ( "Geçtiniz\n" );

biçiminde yazmalıyız. Dikkat edilirse sahte kod, C koduna oldukça yakındır. Bu, sahte kodun

kullanışlı bir program geliştirme amacı olmasının sebeplerinden biridir.

İyi Programlama Alıştırmaları 3.2

Sahte kodlar genellikle program geliştirme sürecinde programı ortaya çıkartmak

amacıyla kullanılır. Daha sonra sahte kodlar, C koduna çevrilir.

Şekil 3.2’deki akış grafiği, tek seçimli if yapısını gösterir. Bu akış grafiği, akış grafiklerinin

en önemli elemanı olan elmas sembolünü içerir. Bu sembol aynı zamanda karar işareti olarak

da adlandırılır ve burada bir karar verileceğini gösterir. Karar işareti bir koşul içerir ve bu

koşul doğru ya da yanlış olabilir. Akış grafiğinde, elmas sembolünden çıkan iki çizgi görülür.

Bunlardan birincisi, koşul doğruysa izlenecek yolu, ikincisi ise koşul yanlışken izlenecek yolu

gösterir. 2. ünitede karar verme işlemlerinin, karşılaştırma operatörleri ya da eşitlik

operatörleriyle yapılabileceğini göstermiştik. Eğer deyim 0 değerini veriyorsa yanlış, eğer

0'dan farklı bir değer veriyorsa doğru olarak düşünülür.

Şekil3.2 Tek seçimli if yapısının akış grafiği

if yapısının da tek giriş ve tek çıkışa sahip olduğuna dikkat edin. İleride öğreneceğimiz gibi,

diğer kontrol yapılarının akış grafikleri de karar vermek için elmas sembollerini ve işlemlerin

yapılacağını gösteren dikdörtgen sembollerini gösterir. Bu, karar/işlem modeli olarak

adlandırdığımız ve anlatmaya çalıştığımız temel yapıdır.

“Geçtiniz” yazdır

doğru

yanlış

not>=6

0

60

Programcının yapması gereken, yedi kontrol yapısından uygun olanlarını seçmek ve

algoritmaya göre önce boş olarak elmas ve dikdörtgen sembollerini yerleştirmek sonra onları

uygun biçimde birbirine bağlamak, son olarak da işlem ve karar koşullarını sembollerini

işlemlerin içine yazmaktır. İleride karar verme ve işlem yazma çeşitlerini anlatacağız.

3.6 if/else SEÇİM YAPISI

if yapısı, koşul doğru ise belirlenen işi yapıyor, yanlış ise belirlenen işi atlıyordu. if/else

yapısı, programcıya koşul doğruysa belli işleri, yanlışsa belli başka işleri yaptırabilme fırsatı

verir. Örnek olarak, aşağıdaki sahte kodu inceleyelim.

Eğer ( if ) öğrencinin notu, 60 ya da daha büyükse

“Geçtiniz yazdır

aksi takdirde ( else )

“Kaldınız” yazdır

Bu kodla, notu 60 ya da 60' tan büyük olan öğrenciler için Geçtiniz mesajı, notu 60’ tan küçük

olan öğrenciler için ise Kaldınız mesajı yazdırılacaktır. Her iki koşulda da yazdırma

gerçekleştirildikten sonra sıradaki sahte kod işlenir. else yapısının gövdesinin de içeriden

başlatıldığına dikkat ediniz.

İyi Programlama Alıştırmaları 3.3

if/else yapısının gövdelerini içeriden başlatmak.

İyi programlama Alıştırmaları 3.4

Eğer birden fazla seviyede içeriden başlamanız gerekiyorsa, her seviye aynı miktarda

içeriden başlatılmalıdır.

Yukarıda yazdığımız sahte kodu, C'de yazmak istersek:

if ( not >= 60 )

printf ( "Geçtiniz\n" );

else

printf ( "Kaldınız\n" );

yazarız. Şekil 3.3’deki akış grafiğinde, if/else yapısının kullanımı gösterilmiştir. Bir kez daha,

bu akış grafiğinde yalnızca dikdörtgen ve elmas sembollerinin kullanıldığına dikkat ediniz.

Karar/işlem modelini vurgulamaya devam ediyoruz. Bir C programı oluşturmak için gerekli

olabilecek boş, çift seçimli yapılardan mümkün olabildiğince fazlasını düşünün. Programcının

görevinin, seçim yapılarını diğer kontrol yapılarıyla birleştirerek, algoritmanın gerektirdiği

biçimde dikdörtgen ve elmas sembollerinin içini işlem ve kararlardan uygun olanlarıyla

doldurmak olduğunu hatırlayın.

C, if/else yapısına oldukça benzeyen koşullu operatörü (?:) kullanmamıza imkan sağlar. Bu

operatör, C' in üçlü tek operatörüdür. Bu operatör, üç operand alır. İlk operand ve operatör,

koşullu deyimi oluşturur. İlk operand koşulun kendisidir. İkinci operand koşul doğru ise tüm

koşullu deyimin değeri, üçüncü operand koşul yanlış ise tüm koşullu deyimin değeridir.

Örneğin, aşağıdaki printf ifadesi

printf ( "%s\n", not >= 60 ? "Geçtiniz" : "Kaldınız" );

61

"not >= 60" koşulu doğru ise “Geçtiniz” bilgisini işleme koyan, yanlış ise “Kaldınız” bilgisini

işleme koyan bir koşullu deyim içermektedir. printf’deki biçim kontrol dizesi içinde yer alan

%s dönüşüm belirteci bir karakter dizesini yazdırmak için kullanılır. Böylelikle az önceki

printf ifadesi , if/else ifadesiyle aynı işlemi yapar.

Bir koşullu deyimdeki değerler, çalıştırılabilecek işlemler de olabilir. Örneğin, aşağıdaki

koşullu deyim ;

not >= 60 ? printf ( "Geçtiniz\n") : printf ( "Kaldınız\n" );

“Eğer not 60'tan büyükse ya da 60'a eşitse printf ( "Geçtiniz\n" ) ” , aksi takdirde “ printf

("Kaldınız\n" ) ” biçiminde okunur. Bu da, az önceki if/else yapısıyla karşılaştırılabilir.

İleride, if/else ifadelerinin kullanılamayacağı fakat koşullu operatörlerin kullanılabileceği bazı

durumlardan bahsedeceğiz.

Yuvalı if/else yapıları, if/else yapıları içerisine başka if/else yapıları yerleştirerek birden fazla

koşulu aynı anda test etmemizi sağlar. Örneğin, aşağıdaki sahte kod, öğrencinin notu 90'a eşit

ya da 90'dan büyükse A, 80' e eşit ya da 80'den büyükse B, 70' e eşit ya da 70'ten büyükse C,

60' a eşit ya da 60'tan büyükse D, diğer durumlarda F yazdıracak biçimde tasarlanmıştır.

Şekil 3.3 Çift seçimli if/else yapısının akış grafiği

Eğer(if) öğrencinin notu 90'a eşit ya da 90'dan büyükse

“A” yazdır

Aksi takdirde(else)

Eğer(if) öğrencinin notu 80'a eşit ya da 80'dan büyükse

“B” yazdır

Aksi takdirde(else)

Eğer(if) öğrencinin notu 70'a eşit ya da 70'dan büyükse

“C” yazdır

Aksi takdirde(else)

Eğer(if) öğrencinin notu 60'a eşit ya da 60'dan büyükse

“D” yazdır

Aksi takdirde(else)

“F” yazdır

not>=60

“kaldınız” yazdır “geçtiniz” yazdır

yanlış doğru

62

Bu sahte kod C'de aşağıdaki biçimde yazılabilir:

if ( not >= 90 )

printf("A\n");

else

if ( not >= 80 )

printf("B\n");

else

if (not >= 70)

printf ("C\n");

else

if ( not >= 60 )

printf ( "D\n" );

else

printf ("F\n" );

Eğer not, 90'a eşit ya da 90’dan büyükse ilk dört durumun tümü birden doğru olacak ancak

sadece ilk karşılaştırmadan sonraki printf çalıştırılacaktır. Bu printf uygulandıktan sonra,

if/else yapısının en dıştaki else kısmı atlanacaktır. Çoğu C programcısı, yukarıdaki if yapısını

aşağıdaki biçimde yazmayı tercih eder:

if ( not >= 90 )

printf("A\n");

else if ( not >= 80 )

printf("B\n");

else if (not >= 70)

printf ("C\n");

else if ( not >= 60 )

printf ( "D\n" );

else

printf ("F\n" );

C derleyicileri, boşlukları atladığı için yukarıdaki iki if yapısı da birbirine eşittir. İkinci olarak

yazdığımız if yapısı , kodun sürekli olarak sağa doğru ileriye gitmesini engellediği için daha

çok tercih edilir. İlk gösterdiğimiz şekilde içeri doğru yazma bir satırda çok az boşluk

kalmasına, bu sebepten de satırların ayrılmasına sebep olarak, programın okunulabilirliğini

azaltır.

if seçim yapısı, gövdesinde yalnızca tek bir ifade bekler. Bir if yapısının gövdesinde, birden

fazla ifade bulundurmak istiyorsak bu ifadeleri küme parantezinin içine almamız gerekir ({ }).

Küme parantezleri içinde yer alan ifadelere birleşik ifade denir.

Yazılım Mühendisliği Gözlemleri 3.1

Birleşik ifade, bir programda tek bir ifadenin yerleştirilebileceği her yere

yerleştirilebilir.

63

Aşağıdaki örneğimiz, if/else yapısının else kısmında birleşik ifade içermektedir

if ( not >= 60 )

printf ( "Geçtiniz\n" );

else {

printf (" Kaldınız\n" );

printf (" Bu dersi tekrar almalısınız\n" );

}

Bu durumda, not 60'tan küçük olduğunda program, else yapısının gövdesindeki iki printf

ifadesini de çalıştıracak ve ekrana

Kaldınız.

Bu dersi tekrar almalısınız

yazdıracaktır. else kısmındaki iki ifadenin de küme parantezleri içerisine yazıldığına dikkat

ediniz. Bu küme parantezleri önemlidir. Küme parantezleri olmadan

printf (" Bu dersi tekrar almalısınız\n ");

ifadesi, if / else yapısının else kısmının gövdesinin dışında kalacak ve notun 60'tan küçük ya

da büyük olmasına bakmaksızın mutlaka çalıştırılacaktı.

Genel Programlama Hataları 3.1

Bir birleşik ifadenin yazımında küme parantezlerini unutmak.

Yazım hataları derleyici tarafından yakalanır. Mantık hataları ise çalışma zamanında ortaya

çıkar. Ölümcül bir mantık hatası programın aniden sonlanmasına sebep olur. Ölümcül

olmayan mantıksal hatalar, programın çalışmaya devam etmesine izin verir. Ancak yanlış

sonuçlar üretir.

Genel Programlama Hataları 3.2

Tek seçimli if yapılarında koşuldan sonra noktalı virgül koymak bir mantık hatası, çift

seçimli if yapılarında ise, bir yazım hatasıdır.

İyi Programlama Alıştırmaları 3.5

Bazı programcılar, birleşik ifadeler yazacaklarında ifadeleri yazmadan önce küme

parantezlerini yazarlar. Daha sonrada, bu küme parantezlerinin arasına ifadeleri

yerleştirirler. Böylece küme parantezlerini unutma durumundan kurtulmuş olurlar.

Yazılım Mühendisliği Gözlemleri 3.2

Birleşik ifadelerin tek bir ifadenin yerleştirilebileceği her hangi bir yere yazılabilmesi

gibi , ifade yazmamak; yani boş bir ifade yazmakta mümkündür. Boş ifadeler, ifadenin

olması gereken yere yalnızca noktalı virgül konarak oluşturulur.

Bu kısımda birleşik bir ifadenin yazım biçimini tanıttık. Birleşik bir ifade, bildirimler de

içerebilir. ( main fonksiyonun gövdesinde yaptığımız gibi ) Eğer birleşik ifadede bildirim

yapılırsa buna blok denir. Bir blokta bildirimler, herhangi bir işlem ifadesinden önceye

yerleştirilmelidir. Blokların kullanımını 5.ünitede anlatacağız. 5. üniteye kadar, blokları

kullanmaktan kaçınınız. ( main fonksiyonunun gövdesi hariç )

64

3.7 while DÖNGÜSÜ

Bir döngü yapısı, programcıya bir koşul doğru olduğu sürece bir işlemi tekrarlatma imkanı

sağlar. Aşağıdaki sahte kod,

alışveriş listemde birden fazla malzeme bulunduğu sürece ( while )

bir sonraki malzemeyi al ve alışveriş listemden bu malzemeyi çıkart

alışveriş esnasındaki döngüyü tanımlamaktadır. “Alışveriş listemde birden fazla malzeme

bulunduğunda” koşulu, doğru ya da yanlış olabilir. Eğer doğru ise “ bir sonraki malzemeyi al

ve alışveriş listemden bu malzemeyi çıkart ” işlemi yapılacaktır. Bu işlem, koşul doğru olduğu

sürece tekrarlanır. while döngü yapısındaki ifadeler, while yapısının gövdesini oluşturur. while

yapısının gövdesi, tek bir ifadeden ya da birleşik ifadeden oluşabilir.

Herhangi bir anda, koşul yanlış hale gelebilir. (Alışveriş listesindeki en son malzeme satın

alındığında ve alışveriş listesinden çıkartıldığına ) Bu anda döngü sona erer ve döngü

yapısından sonraki ilk sahte kod çalıştırılır.

Genel Programlama Hataları 3.3

while koşulunu yanlış hale getirecek işlemi, while yapısının gövdesinde bulundurmamak.

Normal olarak bu döngü yapısı hiç bir zaman sonlanmaz. Bu hataya “sonsuz döngü” hatası

denir.

Genel Programlama Hataları 3.4

while kelimesini büyük harfle başlatmak ( C'nin harf duyarlı bir dil olduğunu hatırlayın ).

C'nin tüm anahtar kelimeleri yalnızca küçük harf içerebilir. Örneğin ; while, if ve else.

while yapısına örnek olarak, ikinin 1000'den büyük ilk üssünü bulacak biçimde tasarlanmış

program parçacığını inceleyelim. carpim adındaki tamsayı değişkenimizi ilk önce 2 sayısına

atayalım. Aşağıdaki while döngüsü sonlandığında, carpim değişkeni aradığımız cevabı

içermektedir.

carpim = 2;

while ( carpim <= 1000 )

carpim = 2 \* carpim;

Şekil 3.4’deki akış grafiği, while döngüsü içindeki kontrolün akışını göstermektedir. Bir kez

daha akış grafiğinin yalnızca dikdörtgen ve elmas sembolü içerdiğine dikkat edin. while

yapılarını temsil eden boş şekillerin diğer kontrol yapılarıyla birleştirilerek, algoritmanın akış

kontrolünü yapısal bir biçimde oluşturacağına dikkat edin. Bu boş dikdörtgen ve elmaslar

daha sonradan uygun işlem ve kararlarla doldurulur. Akış grafiği, döngüyü açık bir biçimde

göstermektedir. Dikdörtgenden çıkan akış çizgisi, bizi karar mekanizmamız yanlış olana

kadar karar verilen noktaya taşımakta, yanlış duruma gelindiğinde ise bir alt satıra

geçirmektedir. Bu noktada while yapısından çıkılmakta ve kontrol programdaki diğer ifadeye

geçmektedir.

65

Şekil 3.4 while döngü yapısının akış grafiği

while yapısına girildiğinde carpim'in değeri 2'dir. carpim değişkeni sürekli olarak 2 ile

çarpılır ve böylece 4, 8, 16, 32, 64, 128, 256, 512 ve 1024 değerlerini alır. carpim değişkeni

1024 olduğunda, while yapısındaki carpim <= 1000 koşulu yanlış hale gelecektir. Bu,

döngüyü sonlandırır. carpim değişkeninin son değeri 1024' tür. Program çalışmaya while

yapısından sonraki ifadeyle devam eder.

3.8 ALGORİTMALARI UYGULAMAK: DURUM1 (SAYICI

KONTROLLÜ DÖNGÜ)

Algoritmaların nasıl geliştirildiğini gösterebilmek için sınıf ortalaması bulma probleminin

değişik biçimlerini çözeceğiz. Aşağıdaki problemi inceleyiniz.

Bir sınıftaki 10 kişi bir sınava girmiştir. Notlar (0'dan 100'e kadar

tamsayılar) size verilmiş ve bu sınavın sonucunda sınıf ortalamasını

bulmanız istenmiştir.

Sınıf ortalaması, notların toplamının sınıftaki öğrenci sayısına bölünmesiyle bulunur. Bu

problemi bilgisayarda çözecek algoritma, bütün notların teker teker bilgisayara girilmesi,

ortalama hesabının yapılması ve sonucun ekrana yazdırılması biçimindedir. Sahte kodları

kullanalım ve yaptırılacak işleri listeleyip, onların hangi sırada kullanılacağını belirleyelim.

Notları girmek için sayıcı kontrollü döngüyü kullanacağız. Bu teknik sayici adındaki

bir değişken kullanarak, ifadelerin kaç kez tekrar edileceğini belirlememiz esasına dayanır. Bu

örnekte döngü, sayıcı 10'u geçtiğinde sonlanmaktadır. Bu kısımda sahte kodlardan oluşan

algoritmayı (Şekil 3.5) ve bu algoritmanın C ile yazılmış halini (Şekil 3.6) göstereceğiz.

Diğer kısımda, sahte kodlardan oluşan algoritmaların nasıl geliştirildiğini anlatacağız. Sayıcı

kontrollü döngüler, genellikle belirli döngüler olarak adlandırılırlar. Çünkü döngü

başlamadan önce döngünün kaçıncı tekrardan sonra sonlanacağını biliriz.

Algoritmada toplam ve sayici değişkenlerinden bahsedildiğine dikkat ediniz. toplam

değişkeni, bir dizi değerin toplamını depolamak için kullanılacaktır. sayici değişkeni ise

saymak amacıyla kullanılacaktır ( Burada kaç tane not girildiğini saymak için kullanacağız).

Toplamları depolamak için kullanılan değişkenlere, programda kullanılmadan önce 0

atanmalıdır. Aksi takdirde, bu değişkeni depoladığımız hafıza konumunda daha önceden

carpim<=1000 carpim = 2 \* carpim

doğru

yanlış

66

bulunan sayı da hesaplarımıza katılır. sayici değişkenine genellikle kullanımına göre 0 ya da

1 atanır ( Her ikisi içinde örnekler göstereceğiz ). İlk değeri verilmemiş değişkenler

genellikle, çöp (garbage) değer ( hafıza konumunda bu değişken için en son depolanmış

değer) içerirler.

Genel Programlama Hataları 3.5

Eğer sayıcı ya da toplam değişkenlerine ilk atamalar yapılmazsa program muhtemelen

yanlış çalışacaktır. Bu, mantıksal hatalara bir örnektir.

İyi Programlama Alıştırmaları 3.6

Sayıcı ve toplam değişkenlerine ilk değerler vermek.

toplamı 0'a ata

sayici 'yı 1'e ata

sayici 10' eşit ya da 10'dan küçükken (while)

Diğer notu gir.

Girilen notu, toplama ekle

sayici’ ya 1 ekle

Sınıf ortalamasını, toplamı 10'a bölerek bul

Sınıf ortalamasını yazdır.

Şekil 3.5 Sayıcı Kontrollü Döngülerle sınıf ortalaması problemini çözen algoritmanın sahte

kodlarla yazılmış biçimi

1 /\* Sekil. 3.6: fig03\_06.c

2 Sayıcı kontrollü döngü ile

3 sınıf ortalamasının bulunması \*/

4 #include <stdio.h>

5

6 int main( )

7 {

8 int sayici, not, toplam, ortalama;

9

10 /\* ilk değerlerin verilmesi \*/

11 toplam = 0;

12 sayici = 1;

13

14 /\* işlem kısmı \*/

15 while ( sayici <= 10 ) {

16 printf( "Notu girin: " );

17 scanf( "%d", &not );

18 toplam = toplam + not;

19 sayici = sayici + 1;

20 }

21

22 /\* bitiş kısmı \*/

23 ortalama = toplam / 10;

24 printf( "Sınıf ortalaması %d dir.\n", ortalama );

25

67

26 return 0; /\* programın başarılı bir şekilde bitti \*/

27 }

Notu girin: 98

Notu girin: 76

Notu girin: 71

Notu girin: 87

Notu girin: 83

Notu girin: 90

Notu girin: 57

Notu girin: 79

Notu girin: 82

Notu girin: 94

Sınıf ortalaması 81 dir.

Şekil 3.6 Sayıcı kontrollü döngü ile sınıf ortalaması problemini çözen C programı ve

programın örnek bir çıktısı.

Bu programda, sonuç olarak hesaplanan ortalamanın bir tamsayı olduğuna dikkat ediniz.

Aslında bu örnekteki notların toplamı 817 yapmaktadır ve 10'a bölündüğünde 81.17 sonucu

vermektedir. İleride bu tür sayılarla (ondalıklı sayılarla) ilgili konuları anlatacağız.

3.9 YUKARIDAN AŞAĞIYA,ADIMSAL İYİLEŞTİRMEYLE

ALGORİTMALARI UYGULAMAK:DURUM 2 (NÖBETÇİ

KONTROLLÜ DÖNGÜLER)

Ortalama problemimizi genelleştirelim.Aşağıdaki problemi inceleyiniz:

Program çalıştırıldığında, kaç kişinin ortalamasının hesaplanacağını önceden bilmeden,

sınıf ortalamasını bulacak bir program geliştirin.

İlk sınıf ortalaması örneğinde, notların sayısını (10) başlangıçta biliyorduk. Bu örneğimizde

ise kaç not girileceğini başlangıçta bilmiyoruz. Program, herhangi bir sayıda veriyi işlemek

zorundadır. Bu durumda, programımız notların girişinin sonlandığına nasıl karar verecektir?

Sınıf ortalamasını ne zaman hesaplayacağını ve yazdıracağını nasıl bilecektir?

Bu problemi çözmenin yolu, veri girişinin sonlandığını belirten bir özel değer, nöbetçi değer,

( sinyal değer ve ya işaretçi değer de denir) kullanmaktır. Kullanıcı, girmesi gereken tüm

verileri girdikten sonra son değeri girdiğini belirten bir nöbetçi değer girer. Nöbetçi kontrollü

döngüler, genelde belirsiz döngüler olarak adlandırılır çünkü döngü çalışmaya başlamadan

önce döngünün kaç kez tekrarlanacağı bilinmemektedir.

Nöbetçi değer, kabul edilebilir herhangi bir giriş değeriyle karıştırılmayacak biçimde

seçilmelidir. Not değerleri genellikle pozitif tamsayılar olduğundan, -1 bu örnek için uygun

bir nöbetçi değer olabilir. Böylece sınıf ortalama programı, 95,96,75,74,89 ve –1 gibi verileri

işleyecektir. Program, sınıf ortalamasını 95,96,75,74 ve 89 notları için hesaplayacak ve

ortalamayı yazdıracaktır.( -1 nöbetçi değerdir, bu sebepten ortalama hesabına katılmamalıdır.)

Genel Programlama Hataları 3.6

68

Veri olabilecek bir değeri nöbetçi değer seçmek.

Bu sınıf ortalama problemine, iyi yapısal programlar geliştirmek için ihtiyaç duyduğumuz ve

yukarıdan-aşağıya adımsal iyileştirme adını verdiğimiz bir teknikle yaklaşacağız. Bunun için

en yukarıya yapmak istediğimiz işin sahte kodunu yazalım.

Bu sınav için sınav ortalamasını belirle.

En yukarıya yazdığımız bu kod, tüm program için geçerli olacak ve tüm programın tanıtıcısı

konumunda bulunacaktır. Ancak çok az durumda, en yukarıya yazdığımız bu sahte kod bir C

programı yazmak için yeterli detaya sahiptir. Bu yüzden, süreci iyileştirmeye başlayacağız.

Bunun için, yukarıdaki kodu daha küçük görevlere bölüp, bu görevlerin hangi sırada

yapılacağını listeleyeceğiz. İlk iyileştirmemiz şu biçimde yapılabilir :

Değişkenleri belirle.

Notları gir,topla ve say.

Sınıf ortalamasını hesapla ve yazdır.

Burada yalnızca dizi yapısı kullanılmıştır ; adımlar çalıştırılacakları sırada birbiri ardına

sıralanmıştır.

Yazılım Mühendisliği Gözlemleri 3.3

Her iyileştirme algoritmanın bütünleştirilmesidir; yalnızca detay seviyeleri

değişmektedir.

Bir sonraki iyileştirme seviyesinde (ikinci iyileştirmemizde), bazı değişkenleri belirtmeliyiz.

Sayıların toplamına, kaç sayının işlendiğini bilmemize, her notun değerini girdi olarak alacak

bir değişkene ve hesaplanan ortalamayı tutacak bir değişkene ihtiyaç duyacağız.

Değişkenleri belirle

sahte kodunu

toplam değişkenini sıfır olarak belirle

sayıcı değişkenini sıfır olarak belirle biçiminde iyileştirebiliriz.

Yalnızca toplam ve sayıcıya ilk değer atanması gerektiğine dikkat ediniz ; ortalama ve not

değişkenleri için (sırasıyla hesaplanmış ortalama ve kullanıcı girişi için kullanılırlar) ilk değer

atanmasına gerek yoktur. Çünkü bu değerler, 2. ünitede anlattığımız destructive-read in

işlemi sayesinde, hafızada önceden bulunabilecek verileri siler.

Notları gir,topla ve say

sahte kodumuzun tüm notları alabilmesi için bir döngü yapısına ihtiyacı vardır. Kaç notun

girileceğini ve işleneceğini bilmediğimiz için nöbetçi kontrollü döngü kullanacağız. Kullanıcı

notları girdikten sonra nöbetçi değeri girecektir. Program her seferinde girilen sayının nöbetçi

değere eşit olup olmadığını kontrol etmeli, nöbetçi değer girildiği anda da döngüyü

sonlandırmalıdır. Öyleyse sahte kodumuzda yapacağımız iyileştirme

İlk notu gir

69

Kullanıcı nöbetçi değeri girmediği sürece(while)

Bu notu o andaki toplam değere ekle

Sayıcıyı bir arttır

Sıradaki notu al ( bu değer nöbetçi değer olabilir )

biçiminde yapılmalıdır. Sahte kodda, while yapısının gövdesindeki ifadeleri küme parantezleri

içine almadığımıza dikkat ediniz. Bunun yerine, while yapısı içindeki tüm ifadeleri içeriden

başlattık. Bir kez daha sahte kodların yalnızca bir program geliştirme aracı olduğuna dikkat

ediniz.

Ortalamayı hesapla ve yazdır sahte kodunu ise

Eğer ( if ) sayıcı sıfıra eşit değilse

Ortalamayı, notların toplamını sayıcıya bölerek hesapla

Ortalamayı yazdır

Aksi takdirde ( else )

“Not girilmemiştir” yazdır biçiminde düzeltebiliriz.

Burada, toplamın sıfıra bölünme ihtimalini ortadan kaldırdığımıza dikkat edin. Bir sayıyı

sıfıra bölmek engellenmezse ölümcül hata oluşturur.Şimdi de ikinci iyileştirmemizi topluca

şekil 3.7’de görelim.

Genel Programlama Hataları 3.7

Sıfıra bölmeye çalışmak ölümcül bir hata oluşturur.

İyi Programlama Alıştırmaları 3.7

Değeri sıfır olabilecek bir deyimi bölme işlemlerinde bölen olarak kullanacaksak bunu

programda ölümcül bir hata oluşturmayacak biçimde kullanmak gerekir.(Örneğin bir

hata mesajı yazdırarak)

Şekil 3.5 ve 3.7’de sahte kod içinde boş satırlar bırakarak okunurluğu arttırmaya

çalıştık.Aslında bu boş satırlar programları çeşitli kısımlara ayırmaktadır.

Toplam değişkenini sıfır olarak belirle

Sayıcı değişkenini sıfır olarak belirle

İlk notu gir

Kullanıcı nöbetçi değeri girmediği sürece (while)

bu notu o andaki değere ekle

Sayıcıyı bir arttır

Sıradaki notu al(bu değer nöbetçi değer olabilir)

Eğer ( if ) sayıcı sıfıra eşit değilse

Ortalamayı, notların toplamını sayıcıya bölerek hesapla

Ortalamayı yazdır

Aksi takdirde ( else )

“Not girilmemiştir” yazdır

Şekil 3.7 Sınıf ortalaması problemini nöbetçi kontrollü döngülerle çözen sahte kod

algoritması.

70

Yazılım Mühendisliği Gözlemleri 3.4

Programların çoğu mantıksal olarak 3 kısma bölünebilir:Program

değişkenlerinin bildirildiği ve değişkenlere ilk değer atandığı bildirim

safhası, girilen verilerin değerlerinin işlendiği ve program değişkenlerinin

ayarlandığı işleme safhası ve son olarak da sonuçların hesaplandığı ve

yazdırıldığı sonlandırma safhası.

Şekil 3.7’deki sahte kod algoritması, daha genel sınıf ortalaması problemlerini çözmektedir.

Algoritma iki iyileştirme seviyesinden sonra geliştirilmiştir. Çoğu zaman daha fazla seviyeye

ihtiyaç duyulur.

Yazılım Mühendisliği Gözlemleri 3.5

Programcı, yukarıdan-aşağıya adımsal iyileştirme sürecini, sahte kod algoritması programcı

tarafından C kodlarına çevrilebilecek kadar detaya sahip olduğunda sonlandırır. Daha sonra C

programının yazılması oldukça kolay olacaktır.

C programı ve örnek bir çıktısı Şekil 3.8’de gösterilmiştir. Notlar için yalnızca tamsayılar

girilmiş olsa da ortalama hesabı sonucunun, ondalıklı bir sayı olma ihtimali bulunmaktadır.

int tipi böyle bir sayıyı temsil edemez. Bu sebepten, programda yeni bir veri tipi olan float,

ondalıklı sayıları temsil edebilmek için kullanılmıştır. Ayrıca, ortalama hesabında iki tür

arasındaki dönüşümü sağlamak için dönüşüm operatörü (cast operatörü) kullanılmıştır.Bu

yeni özellikler programın yazılmasından sonra detaylı bir şekilde açıklanmıştır.

1 /\* Şekil 3.8: fig03\_08.c

2 Sayıcı kontrollü döngülerle

3 sınıf ortalaması bulan program \*/

4 #include <stdio.h>

5

6 int main( )

7 {

8 float ortalama ; /\* yeni veri tipi \*/

9 int sayici, not, toplam ;

10

11 /\* ilk değer atama \*/

12 toplam = 0;

13 sayici = 0;

14

15 /\* işlem \*/

16 printf( "Notu giriniz (Çıkış için –1) : " );

17 scanf( "%d", &not );

18

19 while ( not != -1 ) {

20 toplam = toplam + not;

21 sayici = sayici + 1;

22 printf( "Notu giriniz (Çıkış için –1) : " );

23 scanf("%d", &not);

71

24 }

25

26 /\* sonlandırma \*/

27 if ( sayici != 0 ) {

28 ortalama = ( float ) toplam / sayici;

29 printf ( "Sınıf ortalaması %.2f", ortalama);

30 }

31 else

32 printf ( "Hiç not girilmemiştir\n" );

33

34 return 0; /\* Program başarılı bir şekilde sonlanmıştır \*/

35 }

Notu giriniz (Çıkış için –1) : 75

Notu giriniz (Çıkış için –1) : 94

Notu giriniz (Çıkış için –1) : 97

Notu giriniz (Çıkış için –1) : 88

Notu giriniz (Çıkış için –1) : 70

Notu giriniz (Çıkış için –1) : 64

Notu giriniz (Çıkış için –1) : 83

Notu giriniz (Çıkış için –1) : 89

Notu giriniz (Çıkış için –1) : -1

Sınıf ortalaması 82.50

Şekil 3.8 Sınıf ortalaması problemini nöbetçi kontrollü döngülerle çözen C programı ve örnek

bir çıktısı.

Şekil 3.8‘de, while döngüsü (19.satır) içindeki birleşik ifadeye dikkat ediniz. Döngü ile dört

ifadenin de tekrarlanabilmesi için bu ifadelerin küme parantezi içine alınması gerektiğine

dikkat ediniz. Küme parantezleri olmadığında, son üç ifade döngü dışında kalır ve

bilgisayarın kodu aşağıdaki biçimde algılamasına sebep olur.

while( not != -1 )

toplam = toplam + not;

sayici = sayici + 1;

printf( “Notu giriniz (Çıkış için –1) :” );

scanf( “%d”, &not);

Bu, kullanıcı ilk not olarak –1 girmediğinde sonsuz döngü oluşmasına sebep olur.

İyi Programlama Alıştırmaları 3.8

Nöbetçi kontrollü döngülerde kullanıcıdan veri istenirken nöbetçi değerin her seferinde

hatırlatılması gerekir.

Ortalamaların hesabında her zaman tamsayı değerleri hesaplayamayız. Sıklıkla ortalama, 7.2

ya da –93.5 gibi ondalıklı bir kısım içeren bir değerdir. Bu değerler ondalıklı sayılar

(floating point numbers) ya da gerçek sayılar olarak adlandırılır ve float veri tipi ile temsil

edilirler. Hesaplamadaki ondalık kısmı tutabilmek için, ortalama değişkeni float tipinde

bildirilmiştir.Buna rağmen toplam/sayici işleminin sonucu bir tamsayıdır. Çünkü toplam ve

sayici değişkenleri tamsayı değişkenleridir. İki tamsayıyı bölmek bize ondalık kısmı

kaybolmuş bir tamsayı değeri verecektir. Hesaplama işlemi ilk önce yapıldığından, ondalık

72

kısım, sonuç ortalama değişkenine atanmadan önce kaybolur. Tamsayı değerleriyle ondalık

kısma sahip bir hesaplama yapabilmek için, işlemde kullanılacak değerleri geçici olarak

ondalıklı sayılara çevirmeliyiz. C, bu işlemi gerçekleştirmek için dönüşüm (cast) operatörünü

kullanır.Programın 28.satırındaki

ortalama = (float) toplam / sayici;

ifadesi bir dönüşüm operatörü , ( float ), içermektedir.Bu operatör sayesinde, bu operatörün

operandı olan toplam değişkeninin geçici olarak, ondalıklı sayı biçiminde bir kopyası

oluşturulur. toplam değişkeninde depolanan değer hala bir tamsayıdır. İşlem artık, ondalıklı

bir sayının (toplam değişkeninin geçici olarak float tipine çevrilmiş kopyası), sayici değişkeni

içinde tutulan tamsayı değerine bölünmesi haline gelmiştir. C derleyicisi, operandlarının tipi

aynı olan deyimleri hesaplayabilir. Operandların aynı tipte olmaları için, derleyici seçilen

operandlara terfi ( promotion ) adı verilen bir işlem uygular. Örneğin, int ve float veri tipini

içeren bir deyimde, ANSI standardı int operandlarının kopyalarının oluşturulmasını ve float

tipine terfi ettirilmesini söylemektedir. Örneğimizde, sayici değişkenin kopyası oluşturulup,

float tipine terfi edildikten sonra işlem yapılmakta ve ondalıklı biçimdeki sonuç ortalama

değişkenine atanmaktadır. ANSI standardı, değişik tipteki operandlar arasındaki terfi işlemleri

için bir takım kurallara sahiptir. 5.ünitede tüm standart veri tipleri ve terfi sıraları

anlatılacaktır.

Dönüşüm operatörleri, her veri tipi için geçerlidir. Dönüşüm operatörleri, veri tipi isminin

parantez içine alınmasıyla oluşturulur. Dönüşüm operatörü tekli bir operatördür. Yani tek bir

operand kullanılır. İkinci ünitede, ikili aritmetik operatörleri çalışmıştık. C, ayrıca artı ( + ) ve

eksi ( - ) operatörlerinin tekli biçimlerini de içermektedir. Böylece programcı –7 ya da +5 gibi

deyimler yazabilmektedir. Dönüşüm operatörleri, sağdan sola doğru işler ve diğer tekli

operatörlerle, örneğin, tekli artı ( + ) ve tekli eksi ( - ) operatörleriyle aynı seviyede önceliğe

sahiptir. Bu öncelik, \*, / ve % operatörlerinden bir seviye üstte ve parantez operatöründen bir

seviye alttadır.

Şekil 3.8’de, 29.satırdaki printf ifadesindeki dönüşüm belirteci ortalama değişkeninin

değerini yazdırmak için %.2f biçiminde kullanılmıştır. f, ondalıklı bir değerin yazdırılacağını

belirtmektedir. .2 ise, değerin hangi duyarlık ile gösterileceğini belirtir ve gösterilecek

değerin, noktadan sonra iki basamak içerebileceği anlamına gelir. Eğer %f dönüşüm belirteci

tek başına kullanılırsa, değerleri 6 duyarlığında yazdırır. Yani noktadan sonra 6 basamak

yazdırır. Bu, %.6f yazmak ile aynıdır. Ondalıklı sayılar duyarlık ile yazdırıldıklarında,

yazdırılan değerin belirtilen sayıda ondalıklı kısım içerebilmesi için değer yuvarlanır.

Hafızadaki değer değiştirilmez. Aşağıdaki ifadeler çalıştırıldığında, 3.45 ve 3.4 değerleri

yazdırılır.

printf(“%.2f\n”,3.446); /\*3.45 yazdırılır\*/

printf(“%.1f\n”,3.446); /\*3.4 yazdırılır\*/

Genel Programlama Hataları 3.8

scanf ifadesi içindeki biçim kontrol dizesi içinde dönüşüm belirtecini, duyarlık ile birlikte

kullanmak hatadır. Duyarlık yalnızca printf dönüşüm belirteçleriyle kullanılır.

Genel Programlama Hataları 3.9

Ondalıklı sayıların mükemmel bir biçimde gösterilebileceklerini düşünerek bu sayıları

kullanmak hatalı sonuçlar üretilmesine sebep olur. Ondalıklı sayılar çoğu bilgisayarda

yaklaşık olarak temsil edilirler.

73

İyi Programlama Alıştırmaları 3.9

Eşitlik söz konusu olduğunda ondalıklı sayıları karşılaştırmayınız.

Ondalıklı sayılar, her zaman %100 kesin olmasalar da bir çok uygulamada kullanılırlar.

Örneğin, 37.6 sıcaklığının normal vücut sıcaklığı olduğunu söylediğimizde çok fazla ondalıklı

basamak belirtmemize gerek yoktur. Sıcaklığı termometreden 37.6 olarak okuduğumuzda,

vücut sıcaklığının gerçek değeri 37.5999473210643 olabilir. Burada anlatılan, bu değer yerine

37.6 kullanılmasının çoğu uygulamada yeterli olacağıdır. Bu konu hakkında daha fazla bilgiyi

ileride yeniden vereceğiz. Ondalıklı sayıların oluşmasındaki bir diğer sebep de bölme

işlemidir. Onu üçe böldüğümüzde sonuç 3.3333333... dır ve üçlerin dizisi sonsuza kadar

devam etmektedir. Bilgisayar, böyle bir değeri tutmak için yalnızca belli sayıda boşluk

ayıracağından ondalıklı sayıların yalnızca bir tahmin olduğu açıktır.

3.10 YUKARIDAN AŞAĞIYA ADIMSAL İYİLEŞTİRMEYLE

ALGORİTMALAR YAZMAK: DURUM3 (YUVALI KONTROL YAPILARI)

Şimdi başka bir problem üzerinde çalışalım. Algoritmamızı yine sahte kod ve yukarıdan

aşağıya adımsal iyileştirmeyle oluşturacağız ve bu algoritmanın C kodunu yazacağız. Daha

önceden kontrol yapılarının birbirleri üzerine (bir dizide) eklenebildiğini görmüştük. Şimdi ise

C’de kontrol yapılarını, yapısal bir biçimde birleştirebilecek diğer yolu çalışacağız. Bu yola,

bir kontrol yapısını diğeri içine yuvalamak denir.Aşağıdaki problemi inceleyiniz.

Bir kurs öğrencilerini bir lisans sınavına hazırlamaktadır. Geçen sene, bu kursu tamamlayan

öğrencilerden bir kısmı lisans sınavına girmiştir. Kurs yöneticileri, öğrencilerin sınavdaki

başarılarını öğrenmek istemektedir ve size sonuçları özetleyen bir program yazmanızı

söylemişlerdir. Bu sınava giren 10 öğrencinin isimlerinin yer aldığı bir liste size verilmiştir.

Bu listede eğer öğrenci sınavı geçmişse isminin yanında 1, eğer sınavdan kalmışsa isminin

yanında 2 yazmaktadır.

Programınızın sınav sonuçlarını aşağıdaki şekilde analiz etmesi gerekmektedir.

Sizden;

1.Her sınav sonucunu girmenizi ve program başka bir sınav sonucunu alacağında ekrana

“Sonucu girin” mesajını yazdırmanızı

2.Her tipte sınav sonucunun sayısını bulmanızı

3.Kaç öğrencinin sınavı geçtiğini ve kaçının kaldığını özetleyen bir gösterge hazırlamanızı

4.Eğer 8’den fazla öğrenci sınavı geçtiyse “yüksek başarı” mesajını yazdırmanızı

istemektedirler.

Problemi dikkatlice okuduktan sonra aşağıdaki gözlemleri yaparız:

1.Program 10 test sonucunu işleyecektir. Sayıcı kontrollü döngü kullanılacaktır.

2.Her test sonucu 1 ya da 2 gibi bir sayıdır. Program yeni bir sonuç okuduğunda bu sonucun 1

mi yoksa 2 mi olduğuna karar vermelidir. Algoritmamızda 1 olması durumunu inceleyeceğiz.

Eğer sayı 1 değilse 2 olduğunu düşüneceğiz. ( Bu ünitenin sonunda bu kabullenmeye benzer

bir dizi alıştırma bulacaksınız )

3.İki sayıcı kullanılacaktır. Bunlardan birincisi sınavı geçen öğrenci sayısını, ikincisi ise

sınavdan kalan öğrenci sayısını saymak için kullanılacaktır.

4.Program tüm sonuçları işledikten sonra, sınavı geçen öğrenci sayısının 8’den fazla olup

olmadığına karar vermelidir.

74

Şimdi yukarıdan aşağıya adımsal iyileştirmeyi kullanalım. En başa aşağıdaki sahte kodu

yazalım:

Sınav sonuçlarını incele ve yüksek başarı durumunun gerçekleşip gerçekleşmediğini belirle.

Bu kodun tüm programın bir özeti olduğunu tekrar vurgulamak istiyoruz. Ancak bu kodu C

programına çevrilebilecek kadar detaylandırabilmek için birkaç iyileştirme yapmalıyız. İlk

iyileştirmemiz şu şekildedir:

Değişkenleri tanımla, 10 notu gir ve kalanlarla geçenleri say,

Sınav sonuçlarının özetini yazdır ve yüksek başarı sağlandı mı karar ver.

Bu iyileştirme sonucunda da tüm programın bütünü anlatılmış olsa bile hala iyileştirmeler

yapmalıyız. Geçen ve kalanları kaydetmek için sayıcılara, döngü sürecini kontrol etmek için

bir başka sayıcıya ve kullanıcının gireceği veriyi tutacağımız bir değişkene ihtiyacımız

vardır.Bu sebepten,

Değişkenleri tanımla

sahte kodunu

Geçenler değişkenini sıfıra ata.

Kalanlar değişkenini sıfıra ata

Öğrenci değişkenini bire ata

biçiminde iyileştirebiliriz.

On notu gir ve geçenlerle kalanları say

sahte kodu, her sınavın sonucunu başarılı bir şekilde girebileceğimiz bir döngüye ihtiyaç

duymaktadır. Burada, 10 adet sınav sonucu olduğunu kesin olarak bildiğimizden sayıcı

kontrollü döngü kullanabiliriz. Döngünün içinde (döngünün içine yuvalanmış), bir çiftli

seçim yapısı kullanarak, sınav sonucunun geçer bir not ya da kalır bir not olduğuna karar

verip uygun sayıcıları arttırırız. Sahte kodun iyileştirilmiş hali

Öğrenci sayıcısı 10’a eşit ya da 10’dan küçükken yeni sınav sonucunu al

Eğer ( if ) öğrenci geçmişse

Geçenlere bir ekle

Aksi takdirde ( else )

Kalanlara bir ekle

Öğrenci sayıcısına bir ekle

Boş satırların if/else kontrol yapısını açığa çıkartmak için konulduğuna dikkat ediniz.

Böylelikle programın okunurluğu arttırılmıştır.

Sınav sonuçlarının özetini yazdır ve yüksek başarı sağlandı mı karar ver

Kodunu aşağıdaki biçimde iyileştirebiliriz:

75

Geçenlerin sayısını yazdır

Kalanların sayısını yazdır

Eğer(if) 8‘den fazla öğrenci geçmişse

“Yüksek başarı” yazdır.

İkinci iyileştirmenin tamamı, Şekil 3.9’da gösterilmiştir. Boş satırların, while yapısını

vurgulamak için bırakıldığına ve programın okunurluğunun arttırıldığına dikkat ediniz.

Bu sahte kod C’ye dönüştürülebilecek kadar iyileştirilmiştir. C programı ve bu programın iki

örnek çıktısı Şekil 3.10’da gösterilmiştir. Programda, değişkenler bildirilirken aynı anda

değişkenlere ilk değer atandığına dikkat ediniz. Bu, C’nin önemli özelliklerinden biridir. Bu

şekildeki atamalar derleme zamanında gerçekleşir.

Performans İpuçları3.1

Değişkenler bildirilirken, değişkenlere ilk değer atamak programın çalışma

zamanını kısaltır.

Yazılım Mühendisliği Gözlemleri 3.6

Bilgisayarda bir problemi çözmedeki en zor kısmın çözüm algoritması geliştirmek olduğu tecrübeyle

sabittir. Doğru algoritma geliştirildikten sonra çalışan bir C programı geliştirme süreci oldukça

kolaydır.

Yazılım Mühendisliği Gözlemleri 3.7

Çoğu programcı, programlarını sahte kod gibi program geliştirme araçları

kullanmadan yazarlar ve esas hedeflerinin problemi bilgisayarda çözmek

olduğunu düşünürler. Bu sebepten de sahte kod yazmanın sonuçları üretme

zamanını geciktirdiğini düşünürler.

Gecenler değişkenini sıfıra ata.

Kalanlar değişkenini sıfıra ata

Öğrenci değişkenini bire ata

Öğrenci sayıcısı 10’a eşit ya da 10’dan küçükken yeni sınav sonucunu al

Eğer öğrenci geçmişse

Geçenlere bir ekle

Aksi takdirde

Kalanlara bir ekle

Öğrenci sayicisına bir ekle

Gecenlerin sayısını yazdır

Kalanların sayısını yazdır

Eğer 8‘den fazla öğrenci geçmişse

“Yüksek başarı” yazdır

Şekil 3.9 Sınav sonuçları programı için sahte kod.

1 /\* Şekil 3.10: fig03\_10.c

2 Sınav sonuçlarının analizi \*/

76

3 #include <stdio.h>

4

5 int main( )

6 {

7 /\* Bildirimde değişkenlere ilk değer vermek \*/

8 int gecenler = 0, kalanlar = 0, ogrenci = 1, sonuc;

9

10 /\* sayıcı kontrollü döngüyle 10 sonucun işlenmesi\*/

11 while ( ogrenci <= 10 ) {

12 printf ( "Sonucu girin (1 = geçti, 2=kaldı ): " );

13 scanf( "%d", &sonuc );

14

15 if ( sonuc == 1 ) /\* if/else, while içinde yuvalanmıştır. \*/

16 gecenler = gecenler + 1;

17 else

18 kalanlar = kalanlar + 1;

19

20 ogrenci = ogrenci + 1;

21 }

22

23 printf ( "Geçenler %d\n", gecenler);

24 printf( "Kalanlar %d\n", kalanlar);

25

26 if (gecenler > 8 )

27 printf( " Yüksek başarı\n" );

28

29 return 0; /\* Program başarılı bir şekilde sonlanmıştır. \*/

30 }

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 2

Sonucu girin (1 = geçti, 2=kaldı ): 2

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 2

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 2

Geçenler 6

Kalanlar 4

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 2

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

77

Sonucu girin (1 = geçti, 2=kaldı ): 1

Sonucu girin (1 = geçti, 2=kaldı ): 1

Geçenler 9

Kalanlar 1

Yüksek başarı

Şekil 3.10 Sınav sonuçları problemi için C programı ve bu programın iki örnek çıktısı.

3.11 ATAMA OPERATÖRLERİ

C, atama ifadelerini kısaltmak için bir çok atama operatörüne sahiptir.Örneğin,

c = c+3;

ifadesi toplama atama operatörüyle,+=, aşağıdaki biçimde kısaltılabilir.

c += 3;

+= operatörü, operatörün sağındaki deyimin değerini operatörün solundaki değişkenin

değerine ekler ve sonucu operatörün solundaki değişkene kaydeder.

değişken = değişken operatör deyim;

şeklindeki her ifade

değişken operatör =deyim;

şeklinde yazılabilir. Burada operatör, +,-,\*,/ ya da % gibi tekli operatörler ya da 10.Ünitede

anlatacaklarımızdan birisi olabilir.

Böylece, c += 3; ifadesinin, c’ye 3 eklediğini anlamış olduk. Şekil 3.11, aritmetik atama

operatörlerini, bu operatörleri kullanan örnek ifadeleri ve açıklamaları göstermektedir.

Atama Operatörü Örnek Deyim Açıklama Atar

int c = 3, d = 5, e = 4, f = 6, g = 12; olduğunu kabul ediniz.

+= c += 7 c = c + 7 c’ ye 10’ u

-= d -= 4 d = d – 4 d’ ye 1’i

\*= e \*= 5 e = e \* 5 e’ ye 20’ yi

/= f /= 3 f = f / 3 f’ e 2’ yi

%= g %= 9 g = g % 9 g’ ye 3’ü

Şekil 3.11 Aritmetik atama operatörleri

Performans İpuçları 3.2

Şu ana kadar bahsettiğimiz performans ipuçlarından çoğu küçük geliştirmeler

yapmaktadır, bu sebepten okuyucu bunları önemsememeyi düşünebilir. Buradaki

önemli nokta performans ipuçlarının bütününün birlikte yapacağı etkidir. Bu etki,

programlarınızı önemli bir biçimde hızlandırabilir. Ayrıca küçük bir iyileştirme çok

fazla sayıda tekrar edilen bir döngü içine yerleştirilirse önemli bir geliştirme sağlanır.

3.12 ARTIRMA VE AZALTMA OPERATÖRLERİ

C, tekli artırma operatörü ( ++ ) ve tekli azaltma operatörünü ( -- ) kullanmamıza izin verir.Bu

operatörlerin özetini Şekil 3.12’de bulabilirsiniz.Eğer c değişkeni 1 arttırılacaksa, c = c +1 ya

78

da c += 1 yerine artırma operatörü de kullanılabilir. Eğer artırma ya da azaltma operatörleri

değişkenden önce yerleştirilirse, sırasıyla ön arttırma ( preincrement ) ya da ön azaltma

(predecrement) olarak adlandırılır. Eğer artırma ya da azaltma operatörleri değişkenden sonra

yerleştirilirse, sırasıyla son artırma ( postincrement ) ya da son azaltma( postdecrement )

olarak adlandırılırlar. Ön artırma (ön azaltma) ile öncelikle değişkenin değeri bir arttırılır

(azaltılır ) ve değişkenin yeni değeri, değişkenin içinde bulunduğu deyimde kullanılır. Son

artırma (son azaltma) ile değişkenin o andaki değeri deyimde kullanıldıktan sonra değişkenin

değeri bir arttırılır. ( azaltılır )

Şekil 3.13’te, ++ operatörü için ön artırma ve son artırma biçimleri incelenmiş ve aralarındaki

fark gösterilmiştir. c değişkenine son artırma uygulanması, c değişkeninin değerinin printf

ifadesi içinde kullanıldıktan sonra arttırılmasına sebep olmuştur. c değişkenine ön artırma

uygulanması, c değişkeninin değerinin printf ifadesinden önce arttırılmasına ve yeni değerin

printf ifadesi içinde kullanılmasına sebep olmuştur.

Operatör Örnek

deyim

Açıklama

++ ++a a’yı bir arttır ve a’nın yeni değerini a’nın içinde bulunduğu

deyimde kullan

++ a++ a’nın değerini a’nın içinde bulunduğu deyimde kullan ve daha sonra

a’yı bir arttır.

-- --b b’yi bir azalt ve b’nin yeni değerini b’nin içinde bulunduğu deyimde

kullan

-- b-- b’nin değerini b’nin içinde bulunduğu deyimde kullan ve daha sonra

b’yi bir azalt

Şekil 3.12 Arttırma ve azaltma operatörleri

1 /\* Şekil 3.13: fig03\_13.c

2 Ön arttırma ve son artırma \*/

3 #include <stdio.h>

4

5 int main ( )

6 {

7 int c = 5;

8

9 printf( "%d\n", c );

10 printf( "%d\n", c++ ); /\* Ön arttırma \*/

11 printf( "%d\n\n", c );

12

13 c = 5;

14 printf( "%d\n", c );

15 printf( "%d\n", ++c ); /\* Son artırma \*/

16 printf( "%d\n", c );

17

18 return 0; /\* Program başarılı bir şekilde sonlanmıştır \*/

19 }

5

79

5

6

5

6

6

Şekil 3.13 Ön arttırma ve son azaltma arasındaki fark

Program c’nin değerini ++ operatörü kullanılmadan önce ve sonra göstermektedir. Azaltma

operatörü(--) benzer biçimde çalışmaktadır.

İyi programlama Alıştırmaları 3.10

Tekli operatörlerle, operandları arasında boşluk bırakılmamalıdır.

Şekil 3.10’daki 3 atama ifadesi

gecenler =gecenler+1;

kalanlar =kalanlar+1;

ogrenci =ogrenci+1;

Atama operatörleriyle

gecenler += 1;

kalanlar += 1;

ogrenci += 1;

Ön arttırma operatörleriyle

++gecenler;

++kalanlar ;

++ogrenci ;

Son arttırma operatörleriyle

gecenler++;

kalanlar++;

ogrenci++;

biçiminde yazılabilirdi. Eğer bir ifadede değişkenin kendisi artırılıyor ya da azaltılıyorsa ön

artırma ya da son arttırmanın aynı etkiyi yaratacağını bilmek önemlidir. Ön arttırma ve son

arttırma (benzer olarak ön azaltma ve son azaltma), yalnızca değişken daha geniş bir deyimin

içinde yer alıyorsa farklı etkilere sahiptir.

Arttırma ve azaltma operatörlerinin operandları olarak yalnızca değişken isimleri

kullanılabilir.

Genel Programlama Hataları 3.10

Artırma ve azaltma operatörlerini değişken isimleri yerine bir deyimle birlikte kullanmaya

çalışmak.Örneğin, ++(x+1) yazmak bir yazım hatasıdır.

80

İyi Programlama Alıştırmaları 3.11

ANSI standardı, genellikle operatörün operandının hesaplama sırasını

belirlememiştir.(Bu konudaki istisnaları 4.ünitede göreceğiz) Bu sebepten, programcı

artırma ve azaltma operatörlerinin kullanıldığı ifadelerde belli bir değişkenin birden

fazla kez arttırılması ya da azaltılmasından kaçınmalıdır.

Şekil 3.14’te, şu ana kadar gösterdiğimiz operatörlerin öncelik sıralarını ve işleme biçimlerini

bulacaksınız. Operatörlerin önceliği yukarıdan aşağıya gidildikçe azalmaktadır. İkinci sütunda

operatörlerin işleme biçimlerini bulacaksınız. Koşullu operatör ( ?: ), tekli arttırma ( ++ ) tekli

azaltma ( -- ), artı ( + ), eksi ( - ) ve dönüşüm operatörleriyle =, +=, -=, \*=, /=, %= atama

operatörlerinin sağdan sola doğru işlediğine dikkat ediniz. Üçüncü sütun çeşitli operatör

gruplarının isimlerini belirtmektedir.Şekil 3.14’teki diğer tüm operatörler soldan sağa doğru

işlerler.

Operatörler İşleyiş Tip

( ) soldan sağa parantez

++ -- + - (tip) sağdan sola tekli

\* / % soldan sağa multiplicative

+ - soldan sağa additive

< <= > >= soldan sağa karşılaştırma

= = != soldan sağa eşitlik

?: sağdan sola koşullu

= += -= \*= /= sağdan sola atama

Şekil 3.14 Şu ana kadar anlatılan operatörlerin öncelikleri

ÖZET

 Bir problemin çözümü, belli sırada bir dizi işlemin yapılmasını içerir. Bir problemin

çalıştırılacak işlemler ve bu işlemlerin çalıştırılma sıraları bakımından çözümüne

algoritma denir.

 Bir bilgisayar programında, ifadelerin çalıştırılma sıralarını belirlemeye program

kontrolü denir.

 Sahte kod, programcıların algoritma geliştirmelerine yardımcı olan yapay bir dildir.

Sahte kodlar, günlük konuşma diline oldukça yakındır. Sahte kodlar bilgisayarda

çalıştırılamazlar. Sahte kodlar, programcının programını herhangi bir programlama

dili kullanarak (örneğin C) yazmasından önce program hakkında düşünmesine

yardımcı olur.

 Sahte kodlar, yalnızca karakterlerden oluştuğundan sahte kodlardan oluşan programlar

bilgisayarda yazılabilir, üzerinde düzeltmeler yapılabilir ve saklanabilir.

 Sahte kodlar yalnızca çalıştırılabilir ifadelerin yazımında kullanılır. Bildirimler,

derleyiciye değişkenlerin özelliklerinin belirtildiği ve derleyicinin değişkenlere

hafızada yer ayırmasını belirten mesajlardır.

 Seçim yapıları bir çok işlem arasından birini seçmek için kullanılırlar.

 if seçim yapısı, belirlenmiş bir işi yalnızca koşul doğru ise çalıştırır.

 if/else seçim yapısı, koşul doğru ve yanlışken ayrı işlemleri çalıştırır.

 Yuvalı bir if/else yapısı, birden fazla koşulu test edebilir. Eğer birden fazla koşul

doğru ise yalnızca ilk koşuldan sonraki ifadeler çalıştırılır.

81

 Tek bir ifadenin kullanılması beklenen bir yerde, birden fazla ifade kullanılacaksa

ifadeler küme parantezleri içine alınmalıdır. Birleşik bir ifade, tekli ifadelerin

yerleştirilebildiği her yere yerleştirilebilir.

 Hiçbir işlem yapılmayacağını belirten boş ifade, normalde bir ifadenin bulunacağı yere

noktalı virgül ( ; ) yazarak belirtilir.

 Döngü yapısı, bir işlemin bazı koşullar doğru olarak kaldığı sürece tekrarlanmasını

sağlatır.

 while döngü yapısının biçimi aşağıda gösterilmiştir.

while ( koşul )

ifade ;

 while döngü yapısı içindeki ifade ( birleşik ifade ya da blok ), döngünün gövdesini

oluşturur.

 while gövdesi içinde belirtilen ifadelerden biri koşulun yanlış hale gelmesini

sağlatmalıdır. Aksi takdirde döngü hiçbir zaman sonlanmaz. Sonsuz döngü adı verilen

bir hata oluşur.

 Sayıcı kontrollü döngüler, döngünün ne zaman sonlanacağını belirlemek için bir sayıcı

kullanırlar.

 Toplam değişkeni, bir dizi sayının toplandığı değişkendir.Toplam değişkenleri genelde

program çalıştırılmadan sıfıra atanmalıdır.

 Akış grafiği, algoritmanın grafik ile gösterimidir. Akış grafikleri dikdörtgen, çember

gibi özel sembollerle bu sembolleri birbirine bağlayan akış çizgilerinden oluşur.

Semboller bir işlem yapılacağını, akış çizgileri işlemlerin hangi sırada yapılacağını

belirtir.

 Oval biçimindeki sembol, sonlandırma sembolü olarak da bilinir ve her algoritmanın

başlangıç ve bitişini belirtir.

 Dikdörtgen sembolü, işlem sembolü olarak da bilinir ve herhangi bir tipte hesaplama

ya da giriş çıkış işlemi belirtir. Dikdörtgen sembolleri genellikle atama ifadeleri ve

printf ve scanf gibi standart kütüphane fonksiyonları tarafından gerçekleştirilen

giriş/çıkış işlemlerini belirtir.

 Elmas sembolü aynı zamanda karar sembolü olarak da adlandırılır ve bir karar

verileceğini belirtir. Elmas sembolü doğru ya da yanlış olabilecek bir deyim içerir. Bu

sembolden iki akış çizgisi çıkar. Bunlardan birincisi koşul doğru ise izlenecek yolu,

ikincisi koşul yanlış ise izlenecek yolu belirtir.

 Ondalıklı bir kısım içeren değerler, ondalıklı sayılar olarak adlandırılır ve float veri

tipi ile temsil edilir.

 İki tamsayının bölümü, eğer varsa ondalıklı kısmı kaybolmuş bir tamsayı sonucu verir.

 C, tekli dönüşüm operatörü (float) ile operandının ondalıklı biçimde bir kopyasının

oluşturulmasına imkan sağlar. Her tipte veri için dönüşüm operatörleri bulunmaktadır.

 C derleyicisi, operandlarının tipi aynı olan deyimleri hesaplayabilir. Operandların aynı

tipte olmaları için derleyici, seçilen operandlara terfi ( promotion ) adı verilen bir

işlem uygular. Örneğin, int ve float veri tipini içeren bir ifadede ANSI standardı int

operandlarının kopyalarının oluşturulmasını ve float tipine terfi ettirilmesini

söylemektedir. ANSI standardı değişik tipteki operandlar arasındaki terfi işlemleri için

bir takım kurallara sahiptir.

 Ondalıklı sayılar, printf ifadesi içinde %f dönüşüm belirteciyle noktadan sonra kaç

basamak yazdırılacağını belirten bir duyarlık ile yazdırılırlar. %.2f dönüşüm

belirteciyle 3.456 değeri 3.46 olarak yazdırılır. Eğer duyarlık belirtmeden %f

kullanılırsa, noktadan sonra 6 basamak yazdırılır.

82

 C, aritmetik atama ifadelerini kısaltmak için bir çok atama operatörüne sahiptir.Bu

operatörler +=, -=, \*=, /= ve %= operatörleridir.

değişken = değişken operatör deyim; biçimindeki atamalar

değişken operatör = deyim; biçiminde yazılabilir.

Burada operatör +, -, \*, / ya da % operatörlerinden biri olabilir.

 C arttırma, ++, ve azaltma, -- , operatörleriyle bir değişkeni bir artırabilir ya da bir

azaltabilir. Bu operatörler bir değişkenin önüne ya da sonuna eklenebilir. Eğer

operatör değişkenin önüne eklenmişse, değişken önce bir arttırılır ya da bir azaltılır,

daha sonrada deyimde kullanılır. Eğer operatör değişkenin sonuna eklenmişse,

değişken önce deyimde kullanılır daha sonra ise bir artırılır ya da azaltılır.

ÇEVRİLEN TERİMLER

action.............................................................................. işlem

action symbol................................................................... işlem sembolü

algorithm........................................................................... algoritma

arithmetic assignment operators....................................... aritmetik atama operatörleri

body of a loop................................................................... döngü gövdesi

cast operator..................................................................... dönüşüm operatörü

compound statement......................................................... birleşik ifade

conditional operator......................................................... koşullu operator

control structure............................................................... kontrol yapısı

counter............................................................................. sayıcı / sayaç

counter-controlled repetition........................................... sayıcı kontrollü döngü

decision........................................................................... karar

decision symbol.............................................................. karar sembolü

decrement operator......................................................... azaltma operatörü

default precision............................................................. varsayılan duyarlık

definite repetition........................................................... belirli döngü

diamond symbol............................................................. elmas sembolü

double-selection structure.............................................. çiftli seçim yapısı

empty statement............................................................. boş ifade

end symbol..................................................................... bitiş sembolü

first refinement............................................................... ilk iyileştirme

floating-point number.................................................... ondalıklı/gerçek sayılar

flowchart........................................................................ akış grafiği

increment operator......................................................... artırma operatorü

indefinite repetition........................................................ belirsiz döngü

infinite loop.................................................................... sonsuz döngü

initialization................................................................... ilk değer vermek/atamak

nested control structures................................................ yuvalı/iç içe geçmiş kontrol

yapıları

oval symbol.................................................................... oval sembolü

postdecrement operator.................................................. son azaltma

postincrement operator.................................................... son artırma

precision.......................................................................... duyarlık

predecrement operator..................................................... ön azaltma

preincrement operator...................................................... ön artırma

promotion......................................................................... terfi

pseudocode....................................................................... sahte kodlar

83

rectangle symbol............................................................... dikdörtgen sembolü

rounding........................................................................... yuvarlama

sentinel value................................................................... nöbetçi değer

sequence structure........................................................... sıra yapısı

stepwise refinement......................................................... adımsal iyileştirme

ternary operator............................................................... üçlü operatör

truncation......................................................................... ondalık kısmı kaybolmuş

GENEL PROGRAMLAMA HATALARI

3.1 Bir birleşik ifadenin yazımında küme parantezlerini unutmak.

3.2 Tek seçimli if yapılarında koşuldan sonra noktalı virgül koymak bir mantık

hatası, çift seçimli if yapılarında ise bir yazım hatasıdır.

3.3 while koşulunu yanlış hale getirecek işlemi, while yapısının gövdesinde bulundurmamak.

Normal olarak bu döngü yapısı hiç bir zaman sonlanmaz. Bu hataya sonsuz döngü hatası

denir.

3.4 while kelimesini büyük harfle başlatmak (C'nin harf duyarlı bir dil olduğunu hatırlayın).

C'nin tüm anahtar kelimeleri yalnızca küçük harf içerebilir. Örneğin while, if ve else.

3.5 Eğer sayıcı ya da toplam değişkenlerine ilk atamalar yapılmazsa program muhtemelen

yanlış çalışacaktır. Bu, mantıksal hatalara bir örnektir.

3.6 Veri olabilecek bir değeri, nöbetçi değer seçmek.

3.7 Sıfıra bölmeye çalışmak ölümcül bir hata oluşturur.

3.8 scanf ifadesi içindeki biçim kontrol dizesi içinde dönüşüm belirtecini duyarlık ile birlikte

kullanmak hatadır. Duyarlık yalnızca printf dönüşüm belirteçleriyle kullanılır.

3.9 Ondalıklı sayıların, mükemmel bir biçimde gösterilebileceklerini düşünerek bu sayıları

kullanmak hatalı sonuçlar üretilmesine sebep olur.Ondalıklı sayılar çoğu bilgisayarda

yaklaşık olarak temsil edilirler.

3.10Artırma ve azaltma operatörlerini değişken isimleri yerine bir deyimle birlikte

kullanmaya çalışmak. Örneğin, ++(x+1) yazmak bir yazım hatasıdır.

İYİ PROGRAMLAMA ALIŞTIRMALARI

3.1 İçeriden başlayarak yazma gösteriminin uygulanması, program okunulabilirliğini

artırır.

İçeriden başlamak için üç boşluk bırakılmasını tavsiye ediyoruz.

3.2 Sahte kodlar, genellikle program geliştirme sürecinde programı ortaya çıkartmak

amacıyla

kullanılır. Daha sonra sahte kodlar, C koduna çevrilir.

3.3 if/else yapısının gövdelerini içeriden başlatmak.

3.4 Eğer birden fazla seviyede içeriden başlamanız gerekiyorsa, her seviye aynı miktarda

içeriden başlatılmalıdır.

3.5 Bazı programcılar, birleşik ifadeler yazacaklarında ifadeleri yazmadan önce küme

parantezlerini yazarlar. Daha sonrada bu küme parantezlerinin arasına ifadeleri

yerleştirirler. Böylece küme parantezlerini unutma durumundan kurtulmuş olurlar.

3.6 Sayıcı ve toplam değişkenlerine ilk değerler vermek.

3.7 Değeri sıfır olabilecek bir deyimi bölme işlemlerinde bölen olarak kullanacaksak bunu

programda ölümcül bir hata oluşturmayacak biçimde kullanmak gerekir.(Örneğin bir

hata

mesajı yazdırarak)

84

3.8 Nöbetçi kontrollü döngülerde, kullanıcıdan veri istenirken nöbetçi değerin her

seferinde

hatırlatıması gerekir.

3.9 Eşitlik söz konusu olduğunda ondalıklı sayıları karşılaştırmayınız.

3.10 Tekli operatörlerle, operandları arasında boşluk bırakılmamalıdır.

3.11 ANSI standardı genellikle operatörün operandının hesaplama sırasını

belirlememiştir.(Bu konudaki istisnaları 4.ünitede göreceğiz)Bu sebepten, programcı

artırma ve azaltma operatörlerinin kullanıldığı ifadelerde belli bir değişkenin birden

fazla arttırılması ya da azaltılmasından kaçınmalıdır.

PERFORMANS İPUÇLARI

3.1 Değişkenler bildirilirken değişkenlere ilk değer atamak programın çalışma zamanını

kısaltır.

3.2 Şu ana kadar bahsettiğimiz performans ipuçlarından çoğu küçük geliştirmeler

yapmaktadır, bu sebepten okuyucu bunları önemsememeyi düşünebilir.Buradaki

önemli

nokta performans ipuçlarının bütününün birlikte yapacağı etkidir. Bu etki

programlarınızı

önemli bir biçimde hızlandırabilir. Ayrıca küçük bir iyileştirme çok fazla sayıda tekrar

edilen bir döngü içine yerleştirilirse önemli bir geliştirme sağlanır.

YAZILIM MÜHENDİSLİĞİ GÖZLEMLERİ

3.1 Birleşik ifade bir programda, tek bir ifadenin yerleştirilebileceği her yere

yerleştirilebilir..

3.2 Birleşik ifadelerin tek bir ifadenin yerleştirilebileceği her hangi bir yere yazılabilmesi

gibi,ifade yazmamak yani boş bir ifade yazmakta mümkündür. Boş ifadeler, ifadenin

olması gereken yere yalnızca noktalı virgül konarak oluşturulur.

3.3 Her iyileştirme algoritmanın bütünleştirilmesidir; yalnızca detay seviyeleri

değişmektedir.

3.4 Programların çoğu mantıksal olarak 3 kısma bölünebilir:Program değişkenlerinin

bildirildiği ve değişkenlere ilk değer atandığı bildirim safhası, girilen verilerin

değerlerinin işlendiği ve program değişkenlerinin ayarlandığı işleme safhası ve son

olarak

da sonuçların hesaplandığı ve yazdırıldığı sonlandırma safhası.

3.5 Programcı yukarıdan aşağıya adımsal iyileştirme sürecini sahte kod algoritması programcı

tarafından C kodlarına çevrilebilecek kadar detaya sahip olduğunda sonlandırır.Daha

sonra C programının yazılması oldukça kolay olacaktır.

3.6 Bilgisayarda bir problemi çözmedeki en zor kısmın çözüm algoritması geliştirmek olduğu

tecrübeyle sabittir. Doğru algoritma geliştirildikten sonra çalışan bir C programı geliştirme

süreci oldukça kolaydır.

3.7 Çoğu programcı programlarını sahte kod gibi program geliştirme araçları kullanmadan

yazarlar ve esas hedeflerinin problemi bilgisayarda çözmek olduğunu düşünürler.Bu

sebepten de sahte kod yazmanın sonuçları üretme zamanını geciktirdiğini düşünürler.

ÇÖZÜMLÜ ALIŞTIRMALAR

85

3.1 Aşağıdaki boşlukları doldurunuz.

a) Bir problemi çözmede, problemi çalıştırılacak olay cinsinden ifade etmek ve bu

olayları sıraya koyma işlemine \_\_\_\_\_\_\_ denir

b) Bilgisayarın, çalıştırılacak ifadelerin sırasını belirlemesine \_\_\_\_\_\_\_ denir.

c) Bütün programlar üç kontrol yapısı içererek yazılabilir: \_\_\_\_\_\_\_, \_\_\_\_\_\_\_ ve

\_\_\_\_\_\_\_.

d) \_\_\_\_\_\_\_ seçim yapısı, bir koşul doğru olduğunda bazı ifadeleri, yanlış olduğunda ise

başka ifadeleri çalıştırır.

e) Küme parantezleri ({ ve }) arasına yazılarak gruplandırılan ifadelere \_\_\_\_\_\_\_ denir.

f) \_\_\_\_\_\_\_ döngü yapısı, bir grup ifadenin, belli bir koşul sağlandığı sürece çalışmasını

sağlar.

g) Bir grup komutun, belli bir sayıda tekrar tekrar çalıştırılmasına \_\_\_\_\_\_\_ denir.

h) Bir grup ifadenin bir döngüde kaç kez çalıştırılacağı bilinmiyorsa, \_\_\_\_\_\_\_

kullanılarak döngü durdurulabilir.

3.2 x tamsayı değişkenine 1 ekleyen dört farklı ifade yazınız.

3.3 Aşağıdaki ifadeleri gerçekleştirecek C ifadelerini yazınız.

a) x ve y’nin toplamını z’ye atayın ve bu işlemin yapılmasından sonra x’i 1 artırın.

b) carpim değişkenini \*= operatörü kullanarak 2 ile çarpın.

c) = ve \* operatörlerini kullanarak, carpim değişkenini 2 ile çarpın.

d) sayici değişkeninin değerinin 10’dan büyük olup olmadığını kontrol edin. Eğer

büyükse, “sayici 10’dan büyüktür yazdırın.”

e) x değişkeninin değerini 1 azaltın ve toplam değişkeninden çıkartın.

f) x değişkenini toplam değişkenine ekleyin ve x ’in değerini 1 azaltın.

g) q , bolen’e bölündükten sonra kalanı hesaplayın ve sonucu q’ya yazdırın. Bunu iki

farklı yoldan yapın.

h) 123.4567 değerini 2 basamak duyarlıkta yazdırın. Sonuç ne olur?

i) 3.14159 ondalıklı sayısını 3 basamak duyarlıkta yazdırın. Sonuç ne olur?

3.4 Aşağıdaki ifadeleri gerçekleştirecek birer C ifadesi yazınız.

a) toplam ve x değişkenlerini int türünde bildirin.

b) x değişkeninin ilk değerini 1 olarak atayın.

c) toplam değişkeninin ilk değerini 0 olarak atayın.

d) x değişkenini toplam ile toplayın ve sonucu toplam’a atayın.

e) “Toplam :” mesajını toplam değişkeni takip edecek şekilde ekrana yazdırınız.

3.5 Alıştırma 3.4 de yazdığınız ifadeleri bir araya getirerek 1’ den 10’ a kadar olan

tamsayıların toplamını bulan bir C programı yazınız. Hesaplamalarda while döngü

yapısını kullanın. x, 11 olduğunda döngüden çıkılmasını sağlayınız.

3.6 Aşağıdaki hesaplamalar çalıştırıldığında her değişkenin alacağı son değeri hesaplayınız.

Her ifade çalışmaya başladığında değişkenlerin değerlerinin 5 olduğunu kabul edin.

a) carpim \*= x++;

b) sonuc = ++x + x;

86

3.7 Aşağıdakileri gerçekleştirecek birer C ifadesi yazınız.

a) scanf kullanarak bir x tamsayı değişkeni alın.

b) scanf kullanarak bir y tamsayı değişkeni alın.

c) i değişkeninin ilk değerini 1 yapın.

d) kuvvet tamsayı değişkeninin ilk değerini 1 olarak atayın.

e) kuvvet değişkenini x ile çarparak sonucu tekrar kuvvet’e atayın.

f) y değişkenin 1 artırın.

g) y‘nin x’e eşit ya da x’den küçük olmasını test edin.

h) kuvvet tamsayı değişkenini printf kullanarak ekrana yazdırın.

3.8 Alıştırma 3.7’de yazdığınız ifadeleri kullanarak x’in y. kuvvetini bulan bir program

yazınız. Programınız kontrol yapısı olarak while döngüsü içermeli.

3.9 Aşağıdaki hataları bulun ve düzeltin.

a) while (c <= 5) {

carpim \*= c;

++c;

b) scanf(“%.4f”, &deger);

c) if (cinsiyet == 1)

printf (“Kadın\n”);

else;

printf (“Erkek\n”);

3.10 Aşağıdaki while döngü yapısındaki yanlışlık nedir?

while (z >= 0)

sum += z;

Çözümler

3.1 a) algoritma b) Program kontrolü c) sıra, seçim, döngü d) if/else e) bileşik ifadeler f)

while g) Sayıcı kontrollü h) nöbetçi

3.2 x = x + 1;

x += 1;

++x;

x++;

3.3

a) z = x++ +y;

b) carpim \*= 2;

c) carpim = carpim \* 2;

d) if (sayici > 10)

printf (“Sayici 10’dan büyüktür.\n”);

e) toplam -= --x;

f) toplam += x--;

g) q %= bolen;

q = q % bolen;

h) printf (“%.2f”, 123.4567);

87

123.46 ekrana yazdırılır.

i) printf (“%.3f\n”, 3.14159);

3.142 ekrana yazdırılır.

3.4

a) int toplam, x;

b) x = 1;

c) toplam = 0;

d) toplam += x; ya da toplam = toplam + x;

e) printf (“Toplam : %d\n”, toplam);

3.5

1 /\* 1’ den 10’a kadar olan tamsayıların toplamı \*/

2 #include <stdio.h>

3

4 int main( )

5 {

6 int toplam, x;

7

8 x = 1;

9 toplam = 0;

10 while (x <= 10) {

11 toplam += x;

12 ++x;

13 }

14

15 printf (“Toplam :%d\n”, toplam);

16 return 0;

17 }

3.6

a) carpim = 25, x = 6;

b) sonuc = 12, x = 6;

3.7

a) scanf(“%d”, &x);

b) scanf(“%d”, &y);

c) i = 1

d) kuvvet = 1;

e) kuvvet \*= x;

f) y++;

g) if(y <= x)

h) printf (“%d”, kuvvet);

3.8

1 /\* x’in y. kuvveti \*/

2 #include <stdio.h>

3

4 int main( )

5 {

88

6 int x, y, i, kuvvet;

7

8 i = 1;

9 kuvvet = 1;

10 scanf(“%d”, &x);

11 scanf(“%d”, &y);

12

13 while (i <= y) {

14 kuvvet \*= x;

15 ++i;

16 }

17

18 printf(“%d”, kuvvet);

19 return 0;

20 }

3.9

a) Hata: while yapısında sağ küme parantezinin unutulması

Düzeltme: ++c; ifadesinden sonra sağ küme parantezinin eklenmesi

b) Hata: scanf ifadesinde duyarlık kullanılması

Düzeltme: scanf’ten .4 karakterlerinin silinmesi

c) Hata: if/else yapısının else ifadesinden sonra noktalı virgül kullanılması bir mantık

hatasına yol açar. İkinci printf ifadesi her koşulda çalışır.

Düzeltme: else’den sonraki noktalı virgülün kaldırılması.

3.10 while yapısı içerisinde z’nin değeri hiçbir zaman değişmemektedir. Sonuç olarak sonsuz

bir döngü meydana gelmiştir. Döngünün sonsuz kez dönmesini engellemek için z’nin 1’er

1’er azaltılması gerekir.

ALIŞTIRMALAR

Aşağıdakilerde hataları bulun ve düzeltin (Not: Her kod parçasında birden fazla hata olabilir.)

3.11

a) if (yas >= 65);

printf (“ Yaş 65’ten büyük ya da 65’e eşittir\n”);

else

printf (“Yaş 65’ten küçüktür.\n”);

b) int x = 1, toplam;

while (x <= 10) {

toplam += x;

++x;

}

c) While (x <= 100)

toplam += x;

++x;

89

d) while

printf (“%d\n”, y);

++y;

}

3.12 Aşağıdaki boşlukları doldurunuz.

a) Herhangi bir problemin çözümü, biri dizi ifadenin belli bir \_\_\_\_\_\_\_ içinde

uygulanmasıdır.

b) İşlemin eş anlamlısı \_\_\_\_\_\_\_ dir.

c) Birkaç sayının toplamını tutan değişkene \_\_\_\_\_\_\_ denir.

d) Belli değişkenlere programın başında belli değerlerin verilmesine \_\_\_\_\_\_\_ denir.

e) “Veri girişi sonunu” ifade eden özel değere \_\_\_\_\_\_\_ denir. Bu değer \_\_\_\_\_\_\_,

\_\_\_\_\_\_\_, \_\_\_\_\_\_\_ ya da \_\_\_\_\_\_\_.

f) \_\_\_\_\_\_\_, bir algoritmanın grafiksel gösterimidir.

g) Bir akış grafiğinde, adımların hangi sırayla gerçekleştirileceği \_\_\_\_\_\_\_

sembolleriyle ifade edilir.

h) Bitiş sembolü, her algoritmada \_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ belirtir.

i) Dikdörtgen semboller, normalde \_\_\_\_\_\_\_ ifadeleri tarafından hesaplamalara ve

\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ standart kütüphane fonksiyonlarının çağrılmasıyla

gerçekleştirilen giriş/çıkış operasyonlarına karşılık gelir.

j) Bir karar sembolü içine yazılan nesneye \_\_\_\_\_\_\_ denir.

3.13 Aşağıdaki programın çıktısı nasıl olur?

1 #include <stdio.h>

2

3 int main( )

4 {

5 int x = 1, toplam = 0, y;

6

7 while (x <= 10) {

8 y = x \* x;

9 printf(“%d\n”, y);

10 toplam += y;

11 ++x;

12 }

13

14 printf (“Toplam : %d\n”, toplam);

15 return 0;

16 }

3.14 Aşağıdakileri açıklayan birer sahte kod yazınız.

a) “İki Sayı giriniz” mesajını ekrana yazdırın.

b) x, y ve z değişkenlerinin toplamını p değişkenine atayınız.

90

c) Aşağıdaki koşul, bir if/else seçim yapısı içerisinde test edilmektedir: m’nin değeri

v’nin iki katından büyüktür.

d) s, r ve t değişkenleri için klavyeden değerler alın.

3.15 Aşağıdakiler için birer sahte kod algoritması yazınız.

a) Klavyeden iki sayı alınız, toplamlarını hesaplayıp sonucu ekrana yazdırınız.

b) Klavyeden iki sayı alınız, eğer varsa iki sayıdan büyük olanını ekrana yazdırınız.

c) Klavyeden birkaç pozitif sayı alınız,ve sayıların toplamını hesaplayıp ekrana

yazdırınız. “Veri girişi sonunu” belirtmek için –1 nöbetçi değerini kullanıldığını

kabul ediniz.

3.16 Aşağıdakilerin hangilerinin doğru yada yanlış olduğuna karar veriniz. Yanlış olanların

neden yanlış olduklarını açıklayınız.

a) Deneyimler gösteriyor ki, bilgisayarda bir problemi çözmenin en zor yolu, çalışan

bir C programı üretmektir.

b) Nöbetçi değer, makul veri değerleriyle karışabilecek bir değer olmamalı.

c) Akış çizgileri, yapılacak işlemleri belirtir.

d) Karar sembolleri içerisine yazılan koşullar her zaman aritmetik operatörler içerir

(Örneğin +, -, \*, / ve %)

e) Yukarıdan aşağı adımsal iyileştirme işleminde her iyileştirme algoritmanın bütün

gösterimidir.

3.17’den 3.21‘e kadar olan alıştırmalar için aşağıdaki adımların her birini

gerçekleştirin.

1. Problemdeki ifadeyi okuyun.

2. Algoritmayı, sahte kod ve yukarıdan aşağı adımsal iyileştirme kullanarak

oluşturunuz.

3. Bir C programı yazınız.

4. C programını test edin, hata ayıklayın ve çalıştırın.

3.17 Benzinin yüksek fiyatından dolayı, sürücüler arabalarıyla kaç kilometre yol yaptıklarına

dikkat etmeye başladılar. Bir sürücü, kaç kilometre yol yaptığı ile, kaç galon benzin

harcadığını hesaplamaya başladı. Kaç kilometre yol alındığını ve kaç galon benzin

harcandığını kullanıcıdan alan bir C programı yazınız. Programınız, her galon için kaç

kilometre yol alındığını hesaplasın. Bütün giriş bilgileri alındıktan sonra programınız,

gidilen toplam kilometre için harcanan benzin miktarını bulmalı .

Kaç galon benzin harcandı (çıkış için –1) : 12.8

Kaç kilometre yol alındı: 287

Kilometre / galon : 22.42875

Kaç galon benzin harcandı (çıkış için –1) : 10.3

Kaç kilometre yol alındı: 200

Kilometre / galon : 19.417475

Kaç galon benzin harcandı (çıkış için –1) : 5

91

Kaç kilometre yol alındı: 120

Kilometre / galon : 24.000000

Kaç galon benzin harcandı (çıkış için –1) : -1

Toplam ortalama kilometre/galon : 21.601423

3.18 Bir mağaza müşterisinin kredi limitini aşıp aşmadığını hesaplayan bir C programı

geliştiriniz. Her müşteri için aşağıdaki bilgiler bilinmektedir.

1. Hesap Numarası

2. Ayın başlangıcındaki bakiyesi

3. Bu aydaki müşterinin toplam harcaması

4. Bu ay, bu müşterinin hesabına aktarılan kredi

5. İzin verilen kredi limiti

Programa yukarıdaki bilgilerin her biri girilmelidir. Yeni bakiye ( = başlangıç bakiyesi +

harcamalar – krediler) hesaplanarak, müşterinin kredi limitini aşıp aşmadığına karar

verilmelidir. Kredi limiti aşılan müşterilerin ise hesap numarası, kredi limiti ve yeni bakiyesi

ile beraber “Kredi limiti aşıldı.” mesajı ekrana yazdırılmalıdır.

Hesap numarasını girin: (Çıkış için –1): 100

İlk bakiyeyi girin: 5394.78

Toplam harcamaları girin: 1000.00

Toplam kredileri girin: 500.00

Kredi limitini girin: 5500.00

Hesap Numarası: 100

Kredi limiti: 5500.00

Bakiye: 5894.78

Kredi limiti aşıldı.

Hesap numarasını girin: (Çıkış için –1): 200

İlk bakiyeyi girin: 1000.00

Toplam harcamaları girin: 123.45

Toplam kredileri girin: 321.00

Kredi limitini girin: 1500.00

Hesap numarasını girin: (Çıkış için –1): 300

İlk bakiyeyi girin: 500.00

Toplam harcamaları girin: 274.73

Toplam kredileri girin: 100.00

Kredi limitini girin: 800.00

Hesap numarasını girin: (Çıkış için –1): -1

3.19 Bir ilaç şirketi, satış elemanlarına ücretlerini komisyon şeklinde ödemektedir. Bir satış

elemanı haftalık 200$ ve haftalık brüt satışından %9 almaktadır. Örneğin, 5000$ tutarında bir

haftalık satış yapan satış elemanı 200$ ve 5000$ ‘in %9 ‘unu kazanmaktadır, yani 650$. Son

92

haftadaki satış elemanlarının satışlarını kullanıcıya girdiren ve bu satış elemanlarının ne kadar

kazandıklarını hesaplayıp ekrana yazdıran bir program yazınız. Her seferinde bir satış

elemanın işlemlerini yapınız.

Dolar cinsinden satış tutarını giriniz (Çıkış için –1): 5000.00

Maaş: 650.00$

Dolar cinsinden satış tutarını giriniz (Çıkış için –1): 1234.56

Maaş: 311.11$

Dolar cinsinden satış tutarını giriniz (Çıkış için –1): 1088.89

Maaş: 298.00$

Dolar cinsinden satış tutarını giriniz (Çıkış için –1): -1

3.20 Bir borcunun faizi, basit olarak aşağıdaki formülle hesaplanabilir:

faiz = anapara \* oran \* gunler / 365

Bu formülde oran, yıllık faiz yüzdesi olarak kabul edildiğinden 365’e ( günler ) bölümü

içermektedir. Kullanıcıdan birkaç kez anapara, oran ve gunler‘i alarak her borçlanma için

faiz’i yukarıdaki formülü kullanarak hesaplayan bir program yazınız.

Anaparayı girin (çıkış için –1): 1000.00

Faiz oranını girin: .1

Kaç günlük faiz: 365

Faiz ücreti 100$ dır.

Anaparayı girin (çıkış için –1): 1000.00

Faiz oranını girin: .08375

Kaç günlük faiz: 224

Faiz ücreti 51.40$ dır.

Anaparayı girin (çıkış için –1): 10000.00

Faiz oranını girin: .09

Kaç günlük faiz: 1460

Faiz ücreti 3600$ dır.

Anaparayı girin (çıkış için –1): -1

3.21 Birkaç çalışanın brüt maaşlarını hesaplayan bir C programı yazınız. Şirket çalışanlarına

ilk 40 saatlik çalışmaları için “sabit saat” ücret ve 40 saatlik çalışma sonrası içinde “saat

ve yarısı” şeklinde fazla mesai ücreti vermektedir. Size geçtiğimiz haftada her çalışanın

kaç saat çalıştığını ve her işçinin saat başı ücretini gösteren bir liste verilmiştir.

Programınız kullanıcıdan bu bilgileri alarak her çalışanın brüt maaşını hesaplamalı ve

ekrana yazdırmalıdır.

93

Çalışma saatini girin: (Çıkış için –1): 39

Çalışanın saatlik ücretini girin: ($00.00): 10.00

Çalışanın maaşı 390.00$ dır.

Çalışma saatini girin: (Çıkış için –1): 40

Çalışanın saatlik ücretini girin: ($00.00): 10.00

Çalışanın maaşı 400.00$ dır.

Çalışma saatini girin: (Çıkış için –1): 41

Çalışanın saatlik ücretini girin: ($00.00): 10.00

Çalışanın maaşı 415.00$ dır.

Çalışma saatini girin: (Çıkış için –1): -1

3.22 Ön azaltma ve son azaltma arasındaki farkı -- çıkarma operatörü kullanarak gösteren

bir program yazınız.

3.23 1’ den 10’a kadar sayıları aynı satıra aralarında 3 boşluk olacak şekilde, bir döngü

kullanarak yazdıran bir program yazınız.

3.24 En büyük sayıyı (bir grup sayının en büyüğü) bulma işleme bilgisayar uygulamalarında

sıklıkla kullanılır. Örneğin, bir satış müsabakasında en fazla satış yapan satıcının

bulunması. En fazla ürün satan satıcı müsabakayı kazanmaktadır. Programcıya 10 sayı

girdiren ve en büyüğünü bulup ekrana yazdıran bir sahte kod ve C programı yazınız.

İpucu: Programınız aşağıdaki üç değişkeni kullanmalı

sayici : 10’a kadar sayacak bir sayaç ( kaç sayının girildiğini ve 10 sayınında

girildiğini anlamada kullanılmalı.

sayi : Programa girilecek sayı

enBuyuk : Bulunan en büyük sayı

3.25 Aşağıdaki değerler tablosunu döngü kullanarak ekrana yazdırınız.

N 10\*N 100\*N 1000\*N

1 10 100 1000

2 20 200 2000

3 30 300 3000

4 40 400 4000

5 50 500 5000

6 60 600 6000

7 70 700 7000

8 80 800 8000

9 90 900 9000

10 100 1000 10000

94

\t tab karakterini programınızda kullanabilirsiniz.

3.26 Aşağıdaki değerler tablosunu döngü kullanarak ekrana yazdırınız.

A A+2 A+4 A+6

3 5 7 9

6 8 10 12

9 11 13 15

12 14 16 18

15 17 19 21

3.27 Alıştırma 3.24’teki yaklaşıma benzer bir yaklaşımla 10 sayının içerisindeki en büyük iki

sayıyı bulun. Not: Her sayıyı sadece bir kez girin.

3.28 Şekil 3.10’daki programı girdilerini onaylayacak şekilde değiştirin. Herhangi bir girişte

eğer girilen sayı 1 ya da 2’den farklı ise, kullanıcı doğru bir sayı girene kadar program

döngü içinde kalsın.

3.29 Aşağıdaki program ekrana ne yazdırır?

#include <stdio.h>

main( )

{

int sayac = 1;

while (sayac <= 10) {

printf (“%s\n”, sayac % 2 ? “\*\*\*\*” : “++++++++”);

++sayac;

}

return 0;

}

3.30 Aşağıdaki program ekrana ne yazdırır?

1 #include <stdio.h>

2

3 int main( )

4 {

5 int satir = 10, sutun;

6

7 while (satir >= 1) {

8 sutun = 1;

9

10 while ( sutun <= 10) {

95

11 printf (“%s”, satir % 2 ? “<” : “>”);

12 ++sutun;

13 }

14

15 --satir;

16 printf (“\n”);

17 }

18

19 return 0;

20 }

3.31 (zorlayıcı problem) x = 9 , y = 11 olduğunda ve x = 11, y = 9 olduğunda aşağıdaki

ifadelerin çıktısının ne olacağına karar veriniz. Derleyicinin bir C programındaki satır

başlarındaki girintileri ihmal ettiğini unutmayın. Aksi küme parantezleri kullanılarak

belirtilmediği takdirde, C bir else ifadesini her zaman bir önceki if ifadesi ile eşleştirir.

İlk şıkta, programcı else ifadesinin hangi if ifadesine ait olduğunu anlamayabilir. Bu tarz

problemlere karışık else problemleri denir. Aşağıdaki programlardan satır girintilerini

anlaşılmaları daha zor olmaları için kaldırdık. (İpucu:Öğrendiğiniz satır girintilerini

uygulayınız.)

a) if (x < 10)

if (y > 10)

printf(“\*\*\*\*\*\n”);

else

printf(“#####\n”);

printf(“$$$$$\n”);

b) if (x < 10) {

if (y > 10)

printf(“\*\*\*\*\*\n”);

}

else {

printf(“#####\n”);

printf(“$$$$$\n”);

}

3.32 (Başka bir zorlayıcı problemi) Aşağıdaki programı, gösterilen çıktıları verecek şekilde

değiştirin. Parantez eklemekten başka değişiklikler yapmayın.

if ( y == 8)

if (x == 5)

printf (“@@@@@\n”);

else

printf(“#####\n”);

printf(“$$$$$\n”);

printf(“&&&&&\n”);

a) x = 5 ve y = 8 olarak kabul edildiğinde aşağıdaki çıktı aşağıdaki gibi olmuştur.

@@@@@

96

$$$$$

&&&&&

b) x = 5 ve y = 8 olarak kabul edildiğinde aşağıdaki çıktı aşağıdaki gibi olmuştur.

@@@@@

c) x = 5 ve y = 8 olarak kabul edildiğinde aşağıdaki çıktı aşağıdaki gibi olmuştur.

@@@@@

&&&&&

d) x = 5 ve y = 7 olarak kabul edildiğinde aşağıdaki çıktı aşağıdaki gibi olmuştur. Not :

Son üç printf ifadesi bir bileşik ifadenin parçalarıdır.

#####

+++++

&&&&&

3.33 Bir karenin kenarını kullanıcıdan alan ve o kareyi yıldız karakterlerinden oluşacak

şekilde çizen bir program yazınız.

\* \* \* \*

\* \* \* \*

\* \* \* \*

\* \* \* \*

3.34 Alıştırma 3.33 de yazdığınız programı içi boş kare çizecek şekilde değiştiriniz. Örneğin,

kenarı 5 olarak verilen kare aşağıdaki gibi çizilmelidir.

\* \* \* \* \*

\* \*

\* \*

\* \*

\* \* \* \* \*

3.35 Tersten ve düzden okunduğunda aynı okunan kelimelere palindrome denir. Örneğin,

12321, 55555, 45554, 11611 beş basamaklı tam sayıları birer palindrome’dur.

Kullanıcının girdiği beş basamaklı bir sayının palindrome olup olmadığına karar verip

ekrana yazdıran bir program yazdınız.

3.36 Sadece 0 ve 1’lerden oluşan bir tamsayı (ikilik sistem) girişi yaptırın ve bu sayıyı

10’luk sistemde yazdırın.(İpucu: mod ve bölme operatörlerini kullanarak sayının

basamaklarını teker teker sağdan sola doğru alabilirsiniz. 10’luk sistemde en sağdaki

sayının pozisyon değeri 1 ve sonrakilerin 10, 100, 1000 olacak şekilde 10’un

kuvvetlerinde arttığı gibi, ikilik sistemde de 1 ile başlayıp 2 nin kuvvetleri şeklinde,2, 4,

8 gibi artmaktadır. Örneğin 10’luk sistemdeki 234 sayısı 4 \* 1 + 3 \* 10 + 2 \* 100

97

şeklinde gösterilir ve 1101 ikilik sistem sayısının 10’luk sistemdeki karşılığı 1 \*1 + 0 \*

2 + 1 \* 4 + 1 \* 8 ya da 1 + 0 + 4 + 8 yada 13’ tür.)

3.37 Çok hızlı bilgisayarların nasıl olduğunu duymaya devam ediyoruz. Kendi

bilgisayarınızın ne kadar hızlı çalıştığını nasıl anlayabilirsiniz? 1’ den 3,000,000 e kadar

sayan bir while döngüsü içeren bir program yazınız. Sayacınız 1,000,000 her katında

ekrana bu sayıyı yazdırsın. Saatinizi kullanarak her bir milyonluk döngünün ne kadar

zamanda çalıştığını ölçün.

3.38 Bir seferde ekrana 100 yıldız karakteri yazdıran bir program yazınız. Her onuncu yıldız

karakterinden sonra programınız yeni satır karakterini yazdırsın.(İpucu: 1’ den 100’ e

kadar döngü kurun. Mod operatörünü kullanarak sayacınızın 10’un katı olduğunu

anlayın.)

3.39 Bir tam sayı alan ve bu tam sayının basamaklarının kaç tanesinin 7 olduğunu bulan bir

program yazınız.

3.40 Aşağıdaki deseni ekrana yazdıran bir program yazdırınız.

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

Sadece üç printf ifadesi kullanabilirsiniz. Biri :

printf(“\* “);

şeklinde, diğeri :

printf(“ “);

şeklinde, diğer ise :

printf(“\n“);

şeklinde olmalıdır.

3.41 2’nin katlarını, 2, 4, 8, 16, 32, 64, vs. ekrana yazdıran bir program yazdırınız.

Döngüden çıkmayınız (yani sonsuz bir döngü oluşturunuz.)

3.42 Bir çemberin yarı çapını (float türünde) okuyan ve çapını, çevresini ve alanını

hesaplayan bir program yazınız. π için 3.14159 değerini kullanınız.

3.43 Aşağıdaki ifadede yanlışlık nedir? İfadeyi, programcının yapmak istediği olayı

gerçekleştirebileceği şekilde tekrar yazınız.

printf (”%d”, ++(x + y));

3.44 float türünde üç sayı alan ve bu sayıların bir üçgenin üç kenarı olup olamayacağına

karar veren bir program yazınız.

98

3.45 Üç tamsayı alan ve bu sayıların bir dik üçgenin üç kenarı olup olamayacağına karar

veren bir program yazınız.

3.46 Bir şirket bazı verilerini telefon aracılığıyla başka bir yere iletmek istemektedir ama

telefonlarının dinlendiğinden kuşkulanmaktadırlar. Bütün veriler dört basamaklı

tamsayılar şeklinde iletilecektir. Size verilen görev ise bu verileri daha güvenli bir

şekilde iletilmeleri için şifrelemenizdir. Programınız şifrelemeyi şu şekilde yapmalıdır:

Her basamağı 7 ile toplamının 10’luk sistemdeki eşitiyle ve birinci basamağı üçüncü

basamak, ikinci basamağı da dördüncü basamak ile yer değiştirmelisiniz. Bu

şifrelenmiş sayıları alan ve eski haline getiren başka bir programda yazınız.

3.47 Negatif olmayan bir tamsayının faktöriyeli n! Şeklinde yazılır (“n faktöriyel” okunur)

ve aşağıdaki gibi tanımlanır.

n! = n \* (n – 1) \* (n – 2) \* ..... \*1 (n’ in 1 den büyük ya da 1’ eşit değerleri için)

ve

n! = 1 (n = 0 için)

Örneğin 5! = 5.4.3.2.1, yani 120 dir.

a) Negatif olmayan bir tamsayı alan ve bu sayının faktöriyelini hesaplayıp ekrana

yazdıran bir program yazınız.

b) Bir matematik sabiti olan e ‘nin değerini aşağıdaki formülle hesaplayan bir

program yazınız.

e = 1 + 1/1! + ½! + 1/3! + ....

c) Aşağıdaki formülü kullanarak ex i hesaplayan bir program yazınız.

e

x = 1 + x/1! + x2

/2! + x3

/3!

99

C’DE PROGRAM KONTROLÜ

AMAÇLAR

 for ve do/while döngü yapılarını kullanabilmek.

 switch seçim yapısını kullanarak çoklu seçimler yapabilmek

 Program kontrolünde break ve continue kullanabilmek

 Mantık operatörlerini kullanabilmek

BAŞLIKLAR

4.1 GİRİŞ

4.2 DÖNGÜLERİN TEMELLERİ

4.3 SAYICI KONTROLLÜ DÖNGÜLER

4.4 for DÖNGÜ YAPISI

4.5 for DÖNGÜ YAPISIYLA İLGİLİ NOTLER VE GÖZLEMLER

4.6 for YAPISIYLA İLGİLİ ÖRNEKLER

4.7 switch ÇOKLU SEÇİM YAPISI

4.8 do/while DÖNGÜ YAPISI

4.9 break ve continue İFADELERİ

4.10 MANTIK OPERATÖRLERİ

4.11 EŞİTLİK VE ATAMA OPERATÖRLERİ

4.12 YAPISAL PROGRAMLAMA ÖZETİ

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Performans İpuçları\*

Taşınırlık İpuçları\*Yazılım Mühendisliği Gözlemleri\*Çözümlü Alıştırmalar\* Çözümler\*

Alıştırmalar

4.1 GİRİŞ

Bu noktada okuyucu, basit ancak tamamlanmış C programları yazma sürecinde kendini rahat

hissediyor olmalıdır. Bu ünitede, döngü yapıları daha detaylı anlatılacak ve for yapısı ile do/

while yapısı gibi yeni döngü kontrol yapılarından bahsedilecektir. switch çoklu seçim yapısı

tanıtılacaktır. break ifadesiyle, belli kontrol yapılarından istendiği anda nasıl çıkış

yapılacağını ve continue ifadesiyle bir döngünün gövdesinin geri kalan kısmını atlayarak,

döngünün diğer kısımlarını çalıştırmayı tartışacağız. Koşulları birleştirmekte kullanılan

mantık operatörlerini açıklayacağız. Bu üniteyi, 3. ve 4. ünitelerde anlattığımız yapısal

programlamanın temel kurallarını özetleyerek sonlandıracağız.

4.2 DÖNGÜLERİN TEMELLERİ

100

Çoğu program, tekrar ya da döngüler içerir. Döngü, döngü-devam koşulları doğru olarak

kaldığı sürece bilgisayarın çalıştırdığı bir grup emirdir. İki tür döngüden bahsettik :

1. Sayıcı kontrollü döngüler

2. Nöbetçi kontrollü döngüler

Sayıcı kontrollü döngüler, belirli döngüler olarak adlandırılır çünkü döngünün kaç kez

tekrarlanacağı önceden bilinmektedir. Nöbetçi kontrollü döngüler, belirsiz döngüler olarak

adlandırılır çünkü döngünün kaç kez tekrarlanacağı daha önceden bilinmemektedir.

Sayıcı kontrollü döngülerde, bir kontrol değişkeni tekrarların sayısını sayar. Kontrol

değişkeni, emir grupları çalıştırıldıktan sonra arttırılır ( genellikle 1 arttırılır ).Kontrol

değişkeni, doğru sayıda tekrarın yapıldığını gösterdiği anda döngü sona erer ve bilgisayar

döngüden sonraki ilk ifadeyi çalıştırarak programa devam eder.

Nöbetçi değerler, döngüyü aşağıdaki durumlarda kontrol eder.:

1. Döngünün kaç kez tekrarlanacağı bilinmediğinde ve

2. Döngünün içinde döngünün her tekrarında veri alacak ifadeler bulunduğunda

Nöbetçi değer, veri girişinin sonlandığını belirtir. Nöbetçi değer, uygun bütün veri değerleri

girildikten sonra girilir. Nöbetçi değerler, uygun veri değerlerinden farklı olmak zorundadır.

4.3 SAYICI KONTROLLÜ DÖNGÜLER

Sayıcı kontrollü döngüler aşağıdakilere ihtiyaç duyar:

1. Kontrol değişkeninin ( ya da döngü sayıcısının ) ismine.

2. Kontrol değişkeninin ilk değerine.

3. Kontrol değişkeninin döngü içinde arttırılarak ya da azaltılarak değiştirilmesine .

4. Kontrol değişkeninin son değerini kontrol edecek bir koşula. ( döngünün devam edip

etmeyeceğini belirlemek için )

Şekil 4.1’deki, birden ona kadar sayıları sırayla yazdıran programı inceleyiniz.

int sayici = 1;

bildirimi değişkenin ismini verir (sayici), değişkenin tamsayı türünde olduğunu belirtir,

değişkene hafızada yer ayırır ve ayrılan bu yere 1 değerini yazar. Bildirim, çalıştırılabilir bir

ifade değildir.

sayici değişkenini bildirme ve değişkene ilk değer verme işlemi aşağıdaki ifadelerle de

yapılabilirdi:

int sayici;

sayici = 1;

Bildirim çalıştırılamaz ancak atama çalıştırılabilir. Değişkenlere ilk değer verirken iki

yöntemi de kullanacağız.

101

++sayici;

ifadesi, döngü değişkenini döngünün her tekrarından sonra bir arttırır. while yapısı içindeki

döngü-devam koşulu kontrol değişkeninin 10’a eşit ya da 10’dan küçük olmasını kontrol eder.

10 değeri koşulun doğru olarak gerçeklendiği son değerdir. while yapısının gövdesinin,

kontrol değişkeni 10 olduğunda da tekrarlandığına dikkat ediniz. Döngü, kontrol değişkeni 10

değerini geçtiğinde sonlanır. ( örneğin, sayici değişkeni 11 olduğunda)

1 /\* Şekil. 4.1: fig04\_01.c

2 Sayıcı kontrollü döngü \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int sayici = 1; /\* ilk değerin atanması\*/

8

9 while ( sayici <= 10 ) { /\* döngü koşulu \*/

10 printf ( "%d\n", sayici );

11 ++sayici; /\* artırma \*/

12 }

13

14 return 0;

15 }

1

2

3

4

5

6

7

8

9

10

Şekil 4.1 Sayıcı kontrollü döngü

C programcıları, Şekil 4.1’deki programı sayici değişkenine ilk değer olarak 0 verip, while

yapısını

while ( ++sayici<=10 )

printf ( “%d\n”,sayici );

yazarak kısaltırlar. Bu kod ile program bir ifade kısalmaktadır çünkü arttırma işlemi koşul

kontrol edilmeden önce while içinde yapılmaktadır. Ayrıca, bu kod while yapısının

gövdesindeki küme parantezlerinden de kurtulmayı sağlar, çünkü while yapısının gövdesinde

yalnızca bir ifade kalmaktadır. Bu biçimde kod yazabilmek alıştırma yapmayı gerektirir.

Genel Programlama Hataları 4.1

102

Ondalıklı sayılar yalnızca gerçeğe yakın birer tahmin olduğundan, döngülerde kontrol

değişkeni olarak kullanılması kesin olmayan sayıcı değerleri elde edilmesine ve

sonlandırma için yanlış değerlere sahip olunmasına sebep olur.

İyi Programlama Alıştırmaları 4.1

Döngüleri tamsayı değerleriyle kontrol etmek

İyi Programlama Alıştırmaları 4.2

Kontrol yapılarının gövdelerini içeriden başlatmak

İyi Programlama Alıştırmaları 4.3

Kontrol yapısından önce ve sonra boşluk bırakarak kontrol yapılarını programda

belirgin hale getirmek

İyi Programlama Alıştırmaları 4.4

Çok fazla yuvalama kullanmak programın anlaşılabilirliğini zorlaştırır. Genel bir kural

olarak üç seviyeden fazla yuvalama kullanılmamalıdır.

İyi Programlama Alıştırmaları 4.5

Kontrol yapısından önce ve sonra boşluk bırakarak kontrol yapılarını programda

belirgin hale getirmek ve kontrol yapılarının gövdelerini, kontrol yapılarının

başlıklarından daha içeriden başlatmak programın iki boyutlu hale gelmesini

sağlayarak okunurluğu geliştirir.

4.4 for DÖNGÜ YAPISI

for döngü yapısı, sayıcı kontrollü döngülerin bütün detaylarını otomatik olarak kolaylıkla

uygular. for yapısının gücünü anlatabilmek için, Şekil 4.1’deki programı for kullanarak tekrar

yazalım. Program Şekil 4.2’de gösterilmiştir.

Program şu şekilde çalışmaktadır : for yapısı çalıştırıldığında kontrol değişkeni olan sayici, 1

değerine atanır. Daha sonra, döngü-devam koşulu olan sayici <=10 kontrol edilir. sayici

değişkeninin ilk değeri 1 olduğundan, koşul sağlanır ve printf ifadesi ( 12.satır ) sayici

değişkeninin değerini ( 1 ) yazdırır. Kontrol değişkeni olan sayici‘nin değeri sayici++ deyimi

ile arttırılır ve döngü yeniden döngü devam kontrolünü yapar. Kontrol değişkeni olan sayici,

artık 2’ye eşit olduğundan son değer aşılmamıştır. printf ifadesi yeniden çalıştırılır. Bu süreç

kontrol değişkeni olan sayici , 11 olana kadar devam eder. sayici değişkeninin değeri 11

olduğunda, döngü devam şartı yanlış hale gelir ve döngü sona erer. Program for döngüsünden

sonraki ilk ifadeyi çalıştırarak devam eder. ( Bu programda return ifadesini çalıştırır )

1 /\* Şekil 4.2: fig04\_02.c

2 for yapısı ile sayıcı kontrollü döngü\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int sayici;

8

103

9 /\* ilk değer ataması, döngü koşulu, ve artırmanın

10 hepsi birden for yapısının başlığı içindedir \*/

11 for ( sayici = 1; sayici <= 10; sayici++ )

12 printf( "%d\n", sayici );

13

14 return 0;

15 }

Şekil 4.2 Sayıcı kontrollü döngülerin for yapısıyla uygulanması

Şekil 4.3‘de, Şekil 4.2‘de kullanılan for yapısına daha yakından bakılmıştır. for yapısının,

sayıcı kontrollü döngülerde ihtiyaç duyulan her şeyi tek başına belirlediğine dikkat ediniz.

for yapısının gövdesinde birden fazla ifade bulunacaksa bu ifadeler küme parantezleri içine

alınmalıdır.

Şekil 4.2’de, döngü devam şartının sayici <= 10 olduğuna dikkat ediniz.Eğer programcı

yanlışlıkla sayici < 10 yazsaydı, döngü 9 kez tekrarlanıp sona erecekti. Bu, genellikle

karşılaşılan bir mantık hatasıdır.

Genel Programlama Hataları 4.2

while ya da for döngüsü içinde yanlış karşılaştırma operatörü kullanmak ya da döngü

sayıcısı için yanlış son değerler vermek mantık hatası oluşturur.

İyi Programlama Alıştırmaları 4.6

while ve for yapısı içinde döngünün son değerini <= karşılaştırma operatörüyle birlikte

kullanmak mantık hatalarını engellemeye yardımcı olur. Örneğin, 1’den 10’a kadar değerleri

yazdıracak bir döngünün döngü devam koşulu sayici < 11 ya da sayici < 10 yerine

sayici<=10 olmalıdır.

for yapısının genel biçimi

for ( deyim1; deyim2; deyim3)

ifade

olarak gösterilebilir. Burada, deyim1 döngü kontrol değişkenine ilk değer vermekte, deyim2

döngü devam koşulunu belirlemekte ve deyim3 kontrol değişkenini arttırmaktadır. Çoğu

durumda for yapısı aşağıda gösterildiği gibi, while yapısı biçimine çevrilebilir:

deyim1;

while(deyim2){

ifade

deyim3;

}

Bu kural için bir istisnayı Kısım 4.9’da göstereceğiz.

Sıklıkla, deyim1 ve deyim3 deyimlerin virgüllerle ayrılmış listeleridir. Burada virgüllerin

kullanılması, virgül operatörünün ( , ) deyimlerin soldan sağa doğru hesaplanmasını garanti

altına almasıdır. Virgülle ayrılmış bir listenin değeri ve tipi listede en sağda bulunan deyimin

değeri ve tipidir. Virgül operatörü, for yapıları içinde oldukça sık kullanılır. Bu kullanımın

esas faydası, programcının birden çok deyime ilk değer atama ve birden çok deyimin değerini

104

arttırmasıdır. Örneğin, tek bir for yapısı içinde ilk değer atanması ve değerlerini arttırması

gereken iki kontrol değişkeni bulunabilir.

for ( sayici = 1; sayici <= 7; ++sayici )

Şekil 4.3 Tipik bir for yapısı başlığının bileşenleri

İyi Programlama Alıştırmaları 4.7

for yapısı içine yalnızca kontrol değişkenlerine ilk değer atama kısımlarını ve for yapısında

arttırma yapılan kısımları yerleştirmek. Diğer değişkenlerle ilgili işlemler eğer yalnızca bir

kez yapılacaklarsa döngüden önce, eğer birden fazla tekrarlanacaklarsa döngünün içine

yerleştirilmelidir.

for yapısının içindeki 3 deyimin de kullanımı tercihe bağlıdır. Eğer deyim2 çıkartılırsa C,

değerin doğru olduğunu kabul eder ve bu da sonsuz döngü oluşmasına sebep olur. deyim1,

eğer değişkene ilk değer verme işlemi programda başka bir yerde yapılmışsa çıkartılabilir.

Eğer arttırma işlemi for yapısının gövdesinde tanımlanmışsa ya da arttırmaya ihtiyaç

duyulmuyorsa deyim3 çıkartılabilir. for içindeki arttırma deyimi, for yapısının sonunda tek

başına duran bir ifade gibi kullanıldığından

sayici = sayici + 1

sayici += 1

++sayici

sayici++

deyimlerinin hepsi for içinde aynı biçimde çalıştırılır. Çoğu C programcısı, arttırma döngü

sonunda yapılacağından sayici++ biçimini tercih eder. Aslında, burada yapılacak ön arttırma

ya da son arttırma hiçbir deyim içinde yer almadığından aralarında bir fark yoktur. for yapısı

içinde mutlaka iki adet noktalı virgül bulunmalıdır.

Genel Programlama Hataları 4.3

for anahtar kelimesi kontrol değişkeninin ismi

koşulun doğru olabileceği

kontrol değişkeninin son

değeri

kontrol değişkeninin ilk değeri

döngü devam koşulu

kontrol değişkeninin artırılması

105

for yapısının başlığı içinde noktalı virgül yerine virgül kullanmak.

Genel Programlama Hataları 4.4

for yapısının başlığının dışına noktalı virgül koymak o for yapısının gövdesini boş bir ifade

haline getirir.Bu, bir mantık hatasıdır.

4.5 for DÖNGÜ YAPISIYLA İLGİLİ NOTLAR VE GÖZLEMLER

1. İlk değer verme, döngü devam koşulu ve artırma deyimleri aritmetik operatörler

içerebilir. Örneğin, x = 2 ve y = 10 olsun,

for ( j = x ; j <= 4 \* x \* y; j += y / x )

ifadesi

for ( j = 2; j <= 80; j += 5 )

ifadesi ile eşdeğerdir.

2. Arttırma negatif olabilir. ( Bu durumda döngü değişkeni azaltılır ve aşağıya doğru

saydırılır )

3. Eğer döngü devam koşulu en baştan yanlışsa, for yapısının gövdesi tümden atlanır ve

for yapısından sonraki ilk satır çalıştırılır.

4. Kontrol değişkeni, döngü gövdesinde sıklıkla yazdırılır ya da işlemlere sokulur. Ancak

genelde bu yapılmamalıdır. En uygun olan, kontrol değişkenini döngüyü kontrol

etmek için kullanmak ve döngünün gövdesi içinde bir daha kullanmamaktır.

5. for yapısı, while yapısının akış grafiğine benzer bir biçimde şekillendirilir. Örneğin,

for ( sayici=1 ; sayici<=10 ; sayici++);

printf ( “%d”, sayici );

gibi bir for yapısının akış grafiği şekil 4.4’te gösterilmiştir. Bu akış grafiği, ilk değer

verme işleminin yalnızca bir kez uygulandığını ve arttırma işleminin gövde içindeki

işlemlerden sonra yapıldığını açıkça göstermektedir. Akış grafiğinin (çemberler ve akış

çizgileri dışında) yalnızca dikdörtgen ve elmas sembollerini içerdiğine dikkat ediniz. Bir

kez daha, programcının algoritma oluşturmak için, istediği kadar for yapısını diğer

kontrol yapılarının üzerine dizebileceğini ya da içlerine yuvalayabileceğini hatırlayınız.

Programcı algoritmasını tamamlayabilmek için dikdörtgen ve elmas sembollerinin

içlerini tamamlayacaktır.

İyi Programlama Alıştırmaları 4.8

Kontrol değişkenini for döngüsünün gövdesi içinde değiştirmek mümkündür. Ancak bu,

hatalara yol açabilir. En iyisi kontrol değişkenini değiştirmemektir.

4.6 for YAPISIYLA İLGİLİ ÖRNEKLER

106

Aşağıdaki örnekler for yapısında kontrol değişkeninin nasıl değiştirilebileceğini

göstermektedir.

1. Kontrol değişkenini 1’den 100’e kadar birer birer arttır.

for ( i = 1; i <= 100; i++)

2. Kontrol değişkenini 100’den 1’e kadar birer birer azalt.

for ( i = 100; i <= 1; i--)

3. Kontrol değişkenini 7’den 77’ye kadar yedişer yedişer arttır.

for ( i = 7; i <= 77 ; i += 7)

---

Şekil 4.4 Tipik bir for döngü yapısının akış grafiği

4. Kontrol değişkenini 20’den 2’ye kadar ikişer ikişer azalt.

for ( i = 20; i <= 2 ;i -= 2)

5. Kontrol değişkenini 2, 5, 8, 11, 14, 17, 20 değerlerini alacak biçimde değiştir.

for ( j = 2; j <=20; j += 3)

sayici = 1

sayici<=10 printf ( “%d”, sayici ); sayici++

doğru

yanlış

kontrol

değişkeninin

ulaştığı son

değere karar

verilmesi

döngünün gövdesi

(birkaç ifadeden

oluşabilir).

kontrol

değişkeninin

artırılması

107

6. Kontrol değişkenini 99, 88, 77, 66, 55, 44, 33, 22, 11, 0 değerlerini alacak biçimde

değiştir.

for ( j = 99; j <= 0 ;j -= 11)

Şimdiki 2 örneğimiz, for yapısının basit uygulamalarını göstermektedir. Şekil 4.5, for

yapısını 2’den 100’e kadar tüm çift sayıları toplamak için kullanmaktadır.

Şekil 4.5’te, for yapısının gövdesinin, aşağıdaki biçimde virgül kullanarak for içine

alınabileceğine dikkat ediniz.

for (sayi=2 ; sayi<=100 ; toplam+=sayi , sayi+=2)

; /\*boş gövde\*/

İyi Programlama Alıştırmaları 4.9

for yapısı gövdesindeki ifadeler, for yapısı içine alınabilse de bunu yapmaktan kaçınmalıyız.

Çünkü programın okunurluğu zorlaşır.

İyi Programlama Alıştırmaları 4.10

Kontrol yapısını eğer mümkünse tek bir satıra sığdırmak.

1 /\* Şekil 4.5: fig04\_05.c

2 for ile toplama \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int toplam = 0, sayi;

8

9 for ( sayi = 2; sayi <= 100; sayi += 2 )

10 toplam += sayi;

11

12 printf( "Toplam %d\n", toplam );

13

14 return 0;

15 }

Toplam 2550

Şekil 4.5 for ile toplama

Diğer örneğimiz ise for yapısı kullanarak birleşik faiz hesaplamaktadır.Aşağıdaki problemi

inceleyiniz:

Bir kişi $1000’ını %5 faizle bankaya yatırmıştır. Bütün faizin hesaptaki paraya

eklendiğini düşünerek 10 yıl boyunca, her yıl sonunda hesapta birikecek parayı

hesaplayıp yazdırınız. Aşağıdaki formülü kullanınız:

108

a = p (1 + r)n

a: n yıl sonra hesapta birikecek miktar

p: ilk yatırılan miktar

n: yıl sayısı

r: faiz oranı

Bu problem, 10 yıl boyunca her yıl sonunda hesapta biriken parayı bulmamız için yukarıdaki

formülü kullanmamızı istiyor. Öyleyse bir döngü kullanmalıyız. Çözüm, Şekil 4.6’da

gösterilmiştir. [ not : Çoğu UNIX C derleyicilerinde Şekil 4.6’daki programı derleyebilmek

için –lm yazmanız gerebilir. ( Örneğin : cc -lm fig04\_06.c ) ]

1 /\* Şekil 4.6: fig04\_06.c

2 Birleşik faizin hesaplanması \*/

3 #include <stdio.h>

4 #include <math.h>

5

6 int main( )

7 {

8 int yil;

9 double miktar, anapara = 1000.0, oran = .05;

10

11 printf( "%4s%21s\n", "Yıl", "Depozito Miktarı " );

12

13 for ( yil = 1; yil <= 10; yil++ ) {

14 miktar = anapara \* pow( 1.0 + oran, yil );

15 printf( "%3d%21.2f\n", yil, miktar );

16 }

17

18 return 0;

19 }

Yıl Depozito Miktarı

1 1050.00

2 1102.50

3 1157.62

4 1215.51

5 1276.28

6 1340.10

7 1407.10

8 1477.46

9 1551.33

10 1628.89

Şekil 4.6 for ile birleşik faiz hesabı

109

for yapısı ( 13.satır ) kontrol değişkenini 1’den 10’a kadar birer birer arttırarak, gövdesini 10

kez çalıştırmaktadır. C, üs almak için özel bir operatöre sahip olmasa da bunun yerine standart

kütüphane fonksiyonlarından pow fonksiyonunu bu amaçla kullanabilir. pow ( x, y )

fonksiyonu x’in y’ninci kuvvetini hesaplar. Bu fonksiyon, double tipte iki eleman alır ve

double tipinde bir sonuç döndürür. double tipi, float tipine oldukça benzer ancak float tipi

kullanılarak saklanabilecek ondalıklı sayıları daha büyük bir duyarlıkta saklayabilir ve double

tipindeki bir değer, float tipiyle temsil edilebilecek bir değerden daha büyük değerler temsil

edebilir. pow gibi bir matematik fonksiyonu programda kullanıldığında, math.h ( satır 4 )

öncü dosyasının programa eklenmesi gerektiğine dikkat ediniz. Bu program, math.h

programa eklenmeseydi hatalı çalışırdı. pow fonksiyonu double tipte elemanlar kullanır. sene

değişkenin tamsayı olduğuna dikkat ediniz. math.h dosyası, derleyiciye fonksiyonu

çağırmadan önce, yil değişkeninin değerini geçici olarak double tipte temsil etmesi

gerektiğini söyleyen bilgiler içerir. Bu bilgi, pow fonksiyonunun prototipinde yer almaktadır.

Fonksiyon prototipleri, ANSI/ISO C’nin önemli özelliklerindendir ve 5.Ünitede detaylı bir

biçimde açıklanmıştır. pow fonksiyonu ve diğer matematik kütüphane fonksiyonlarını da

5.ünitede daha detaylı olarak bulabilirsiniz.

Bu örneğimiz için, anapara, oran ve miktar değişkenlerimizi double tipte tanıttığımıza

dikkat ediniz. Bunu, programda basitlik sağlaması için yaptık çünkü burada dolarların

ondalıklı kısımlarıyla ilgileniyoruz.

İyi Programlama Alıştırmaları 4.11

float ve double tipte değişkenleri parayla ilgili hesaplamalarda kullanmayınız. Ondalıklı

sayıların kesin olarak gösterilememesinden dolayı hatalar oluşabilir. Örneklerde,

tamsayıların para ile ilgili hesaplarda kullanımını araştıracağız.

Şimdi, float ve double tiplerinin dolar miktarlarını göstermek için kullanıldıklarında nelerde

sorun yaşayabileceğimize bakalım.

float tipiyle temsil edilen iki dolar miktarının hafızada, 14.234 ( %.2f ile 14.23 olarak

yazdırılacaktır ) ve 18.763 ( %.2f ile 18.67 olarak yazdırılacaktır ) saklandıklarını düşünelim.

Bu miktarlar toplandıklarında 32.907 sonucunu verir ve %.2f ile bu sonuç yazdırıldığında

32.91 sonucunu verir.Aşağıda, yapılan işlemin çıktısı gösterilmiştir:

14.23

+ 18.67

----------

32.91

Ancak toplamınızın sonucu 32.90 olmalıydı! Daha önceden sizi uyarmıştık.

Programımızda miktar değişkenin değerini yazdırmak için %21.2f dönüşüm belirtecini

kullandık. Dönüşüm belirtecindeki 21 , değerin yazdırılacağı alan genişliğini belirtir. 21 alan

genişliği, yazdırılacak değerin 21 pozisyon olarak gözükeceği anlamına gelir. 2 ise duyarlığı

(noktadan sonra kaç basamak yazdırılacağını) gösterir. Eğer yazdırılacak değer alan

genişliğinden daha az yer kaplıyorsa, değer otomatik olarak sağa dayalı olarak yazdırılır. Bu,

aynı duyarlığa sahip ondalıklı sayılar için kullanışlı bir yöntemdir. Bir değeri alan içinde sola

yaslamak için % işaretiyle alan genişliği arasına eksi ( - ) işareti konmalıdır. Eksi işareti

tamsayıları ve karakter stringlerini sola yaslamakta da kullanılabilir. (Örneğin, %-6d ve %-

8s ) printf ve scanf fonksiyonlarının biçimlendirme yeteneklerini 9.ünitede anlatacağız.

110

4.7 switch ÇOKLU SEÇİM YAPISI

3.Ünitede if tekli-seçim ve if/else çiftli-seçim yapılarını anlatmıştık. Gerçekte bir algoritmada,

bir değişken ya da ifadenin ayrı ayrı sabitlerle karşılaştırılması ve buna bağlı olarak farklı

işlemlerin yapılması gibi bir dizi seçim yer alabilir. C, bu tarz seçim yapılarını switch çoklu

seçim yapısıyla yapabilir.

switch yapısı, case ve tercihe bağlı olarak default kısımlarından oluşur. Şekil 4.7’de, switch

yapısı kullanılarak bir sınavda öğrencilerin aldıkları değişik harf notlarının sayısı

bulunmaktadır.

1 /\* Şekil 4.7: fig04\_07.c

2 Harf notlarının sayılması \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int not;

8 int aSay = 0, bSay = 0, cSay = 0,

9 dSay = 0, fSay = 0;

10

11 printf( "Harf notlarını girin.\n" );

12 printf( "Çıkış için EOF karakteri girin.\n" );

13

14 while ( ( not = getchar( ) ) != EOF ) {

15

16 switch ( not ) { /\* while içine yuvalanmış switch \*/

17

18 case 'A': case 'a': /\* not büyük A \*/

19 ++aSay; /\* ya da küçük a iken\*/

20 break;

21

22 case 'B': case 'b': /\* not büyük B \*/

23 ++bSay; /\* ya da küçük b iken\*/

24 break;

25

26 case 'C': case 'c': /\* not büyük C \*/

27 ++cSay; /\* ya da küçük c iken \*/

28 break;

29

30 case 'D': case 'd': /\* not büyük D \*/

31 ++dSay; /\* ya da küçük d iken \*/

32 break;

33

34 case 'F': case 'f': /\* not büyük F \*/

35 ++fSay; /\* ya da küçük f iken \*/

111

36 break;

37

38 case '\n': case ‘ ': /\* bunları veri olarak kabul etme \*/

39 break;

40

41 default: /\* diğer tüm karakterleri yakala\*/

42 printf( "Yanlış harf notu girildi." );

43 printf( " Yeni bir not girin.\n" );

44 break;

45 }

46 }

47

48 printf( "\n Her harf notu için toplam:\n" );

49 printf( "A: %d\n", aSay );

50 printf( "B: %d\n", bSay );

51 printf( "C: %d\n", cSay );

52 printf( "D: %d\n", dSay );

53 printf( "F: %d\n", fSay );

54

55 return 0;

56 }

Harf notlarını girin.

Çıkış için EOF karakteri girin.

A

B

C

C

A

D

F

C

E

Yanlış harf notu girildi. Yeni bir not girin.

D

A

B

Her harf notu için toplam:

A: 3

B: 2

D: 3

D: 2

F: 1

Şekil 4.7 switch örneği

Kullanıcı programda, sınıf için harf notlarını girmektedir.while yapısının başlığında (14.Satır)

112

while ( ( not =getchar ( ) ) != EOF )

ilk önce parantez içindeki atama ( not = getchar ( ) ) çalıştırılır. getchar fonksiyonu (standart

giriş/çıkış kütüphanesi içindedir), klavyeden bir karakter okur ve bu karakteri not tamsayı

değişkeni içinde depolar. Karakterler genellikle char tipiyle saklanırlar. C’nin önemli

özelliklerinden biri de karakterlerin, tamsayı veri tipi ile de saklanabilmelidir. Çünkü

karakterler genellikle bilgisayarlarda bir byte uzunluğunda tamsayılardır. Bu sebepten,

karakterleri, karakter ya da sayı olarak kullanabilmek mümkündür. Örneğin,

printf ( “ ( %c ) karakteri %d değerine sahiptir. \n” , ‘a’, ‘a’ ) ;

ifadesi, %c ve %d dönüşüm belirteçlerini kullanır ve sırasıyla a karakterini ve bu karakterin

tamsayı değerini yazdırır. Yukarıdaki printf ifadesinin sonucu aşağıda gösterilmiştir:

a karakteri 97 değerine sahiptir.

97 tamsayısı, karakterin bilgisayardaki sayısal gösterimidir. Bugünkü çoğu bilgisayar, ASCII

(American Standart Code for Information Interchange) karakter kümesini kullanır. Bu

kümede 97, küçük a harfini ( ‘a’ ) temsil etmektedir.ASCII karakterlerinin listesi ve tamsayı

karşılıkları Ekler D’de gösterilmiştir. Karakterler, scanf ve %c dönüşüm belirteci kullanılarak

okunabilir.

Atama ifadelerinin bütününün bir değeri vardır. Bu değer genellikle = operatörünün solundaki

değişkene atanan değerdir. not = getchar( ) atamasının değeri getchar fonksiyonunun

döndürdüğü karakterdir ve bu değer not değişkenine atanmıştır.

Atama ifadelerinin değerleri olduğu gerçeği, farklı değişkenlere aynı ilk değeri verirken

oldukça kullanışlıdır. Örneğin,

a = b = c = 0 ;

ilk önce c = 0 atamasının değerini hesaplar (çünkü = operatörü sağdan sola doğru çalışır).

Daha sonra b değişkeni, c = 0 atamasının değerine ( bu değer sıfırdır ) atanır. En son olarak da

a değişkeni b = ( c = 0 ) atamasının değerine ( bu değer sıfırdır ) atanır. Programda not =

getchar ( ) atamasının değeri, EOF ile karşılaştırılmıştır. EOF, İngilizce “end of file”

deyiminin kısaltmasıdır ve dosya sonuna gelindiğini belirten bir sayıdır. EOF genellikle –1

değerine sahiptir. Programımızda, EOF değerini nöbetçi değeri olarak kullandık. Kullanıcı

kullandığı sisteme bağlı olarak belli tuşlara bastığında EOF “end of file” değerini girer ve

artık daha fazla veri girmeyeceğini belirtir. EOF, <stdio.h> içinde tanımlanmış sembolik bir

sabittir. (6.Ünitede sembolik sabitler tanımlamayı öğreneceğiz) Eğer not değişkenine atanan

değer EOF’ ye eşit olursa, program sonlanır. Bu programda karakterleri int tipinde kullandık

çünkü EOF bir tamsayı değerine sahiptir ( genellikle –1 )

Taşınırlık İpuçları 4.1

EOF değerini girmek için gerekli olan tuş kombinasyonu, sistemlerde farklılık gösterebilir.

Taşınırlık İpuçları 4.2

-1 değeri yerine EOF ile test etmek daha taşınırlığı arttırır. ANSI standardına göre EOF

negatif bir değerdir.Bu sebepten EOF farklı sistemlerde farklı değerlere sahip olabilir.

113

UNIX sistemleri ve birçok diğer sistemde EOF değeri

<return> <ctrl-d>

kombinasyonuyla girilir. Bu gösterim return ( giriş ) tuşuna basıldıktan sonra ctrl tuşuyla d

tuşunun aynı anda basılacağı anlamına gelir. Bazı diğer sistemlerde, örneğin Digital

Equipment Corporation’ın VAX VMS ve Microsoft Corporation’ın MS-DOS sistemlerinde

EOF, <ctrl-z> ile girilir.

Kullanıcı notları klavyeden girmektedir. Kullanıcı return tuşuna bastığında, getchar

fonksiyonu her seferinde bir karakter okur. Eğer girilen karakter EOF değerine eşit değilse,

switch yapısı (satır 16) çalışır. switch anahtar kelimesinden sonra, not değişkeni parantez

içinde yazılır. Bu, kontrol deyimi olarak adlandırılır. Bu deyimin değeri, case kısımlarının her

biriyle karşılaştırılır. Kullanıcının not değeri olarak C harfini girdiğini düşünelim.C otomatik

olarak switch içindeki her case kısmıyla karşılaştırılır. Eğer eşleme gerçekleşirse (case ‘C’ : ),

o case kısmındaki ifadeler çalıştırılır. C harfi girilmesi durumunda cSay değişkeni bir arttırılır

ve break ifadesi sayesinde switch yapısından çıkılır.

break ifadesi, programın switch yapısından sonraki ilk ifadeyle devam etmesini

sağlatır. Eğer break yapısı kullanılmazsa, switch yapısı içindeki tüm case kısımları birlikte

çalışır. Eğer break yapısı switch yapısı içinde hiçbir yerde kullanılmamışsa, yapıdaki

herhangi bir eşlemede geride kalan tüm case kısımları birlikte çalışır ( Bu bazı durumlarda

kullanışlı olabilir ). Eğer herhangi bir eşleme olmazsa, default kısmı çalıştırılır ve bir hata

mesajı yazdırılır.

Her case kısmı bir ya da daha fazla işlem içerebilir. switch yapısı diğer kontrol

yapılarından farklıdır; switch içindeki case kısımlarına yazılacak işlemleri küme parantezine

almaya gerek yoktur. switch çoklu seçim yapısının ( her case kısmında bir break yer aldığı

düşünülmüştür ) akış grafiği, Şekil 4.8’de gösterilmiştir.

.

.

.

case a

case b

case z

case a’ nın olayları

case b’ nin olayları

case b’ nin olayları

doğru

doğru

doğru

yanlış

yanlış

yanlış

break

break

break

114

Şekil 4.8 switch çoklu seçim yapısı

Akış grafiğinde görüldüğü üzere, her case kısmının sonunda yer alan break ifadesi switch

yapısından çıkılmasını sağlamaktadır. Bu akış grafiğinde de (çember ve akış çizgileri dışında)

yalnızca dikdörtgen ve elmas sembollerinin bulunduğuna dikkat ediniz. Bir kez daha

programcının, algoritma oluşturmak için, istediği kadar switch yapısını diğer kontrol

yapılarının üzerine dizebileceğini ya da içlerine yuvalayabileceğini hatırlayınız. Programcı

algoritmasını tamamlayabilmek için dikdörtgen ve elmas sembollerinin içlerini

tamamlayacaktır.

Genel Programlama Hataları 4.5

switch yapısında gerekli yerlerde break kullanmamak

İyi Programlama Alıştırmaları 4.12

switch yapılarının default kısmını içermesini sağlayın.switch yapısında test edilmeyen case

yapıları ihmal edilir.default kullanmak programcının istisnai durumları işleyebilmesine

yardımcı olur.Bazı durumlarda default kısmını kullanmaya gerek olmayabilir.

İyi Programlama Alıştırmaları 4.13

case ve default kısımları switch yapısında istenen sırada yer alabilir ancak default kısmını

yapının sonunda kullanmak iyi bir alışkanlıktır.

İyi Programlama Alıştırmaları 4.14

switch yapılarında default kısmı yapının en sonuna yerleştirildiğinde break ifadesine gerek

kalmaz. Ancak bazı programcılar case kısımlarıyla uyumun bozulmaması için default

kısmında da break kullanırlar.

Şekil 4.7’deki switch yapısında

case ‘\n’: case ‘ ’:

break;

satırları, programın boşluk karakterini ve yeni satır karakterlerini atlamasını sağlatır.

Karakterleri teker teker okumak bazı problemler yaratabilir. Programın karakterleri

okuyabilmesi için, karakter yazıldıktan sonra return(giriş) tuşuna basılmalıdır. Bu, yeni satır

karakterinin işlemek istediğimiz karakterin arkasına yerleştirilmesine sebep olur. Sıklıkla, bu

yeni satır karakteri programın doğru olarak çalışabilmesi için özel olarak işlenmelidir.

Programımızda yukarıdaki kodları switch yapısında kullanarak, yeni satır ya da boşluk

karakterlerinin girilmesi durumunda default kısmındaki hata mesajının yazdırılmasını

engellemiş olduk.

Genel Programlama Hataları 4.6

Karakter okurken yeni satır karakterlerini özel olarak işlememek mantık hatalarına

sebep olabilir.

default olaylar

115

İyi Programlama Alıştırmaları 4.15

Karakter okurken yeni satır karakterlerini işleyecek kodlar yazmayı unutmayın

Şekil 4.7’de, bazı case kısımlarının birlikte kullanıldıklarına dikkat ediniz. ( case:’D’ : case

‘d’:) Bu, birlikte listelenen case kısımları için aynı işlemlerin yapılacağını göstermektedir.

switch yapısını kullanırken, bu yapının karakter ya da tamsayı sabitleri gibi yalnızca sabit

deyimlerin test edilmesinde kullanılabileceğini hatırlayınız. Karakter sabitleri, karakterin tek

tırnak içine yazılmasıyla (örneğin, ‘A’) oluşturulurlar. Tamsayı sabitleri, tamsayı değerlerinin

kendisidir. Örneğimizde karakter sabitlerinin kullandık. Karakterlerin gerçekte küçük tamsayı

değerleri olduğunu hatırlayınız.

C gibi taşınılabilir diller, esnek veri tipi çeşitlerine sahip olmalıdır. Farklı uygulamalar,

farklı büyüklüklerde tamsayılara ihtiyaç duyulabilir. C, tamsayıları temsil edebilmek için bir

çok veri tipine sahiptir. Her tip için tamsayıların kullanılabileceği aralık kullanılan

bilgisayarın yapısına bağlıdır. int ve char tipinin yanında C, short (short int için kullanılan

kısaltmadır) ve long (long int için kullanılan kısaltmadır) tiplerine de sahiptir. ANSI

standardı, short tamsayılar için en küçük aralığı 32767 olarak belirlemiştir. Fakat bazı

uygulamalar için daha büyük tamsayılara ihtiyaç duyulur. Bu tür uygulamalar için long tipi

genelde uygun olur. Standarda göre long tipindeki tamsayılar için en küçük aralık

2147483647 olarak belirlenmiştir. Çoğu bilgisayarda int tipi short ya da long ile eşittir.

Standarda göre int için sayıların alabileceği değer aralığı en az short için belirlenen değer

aralığı kadar olmalı ve long tipi için belirlenen değer aralığını geçmemelidir. char veri tipi

127 aralığındaki tamsayıları ya da bilgisayarın karakter setinde bulunan karakterleri temsil

etmek için kullanılabilir.

Taşınırlık İpuçları 4.3

int farklı sistemlerde farklılık gösterebildiğinden, eğer kullanacağınız değerlerin 32767 aralığı

dışında olmasını bekliyorsanız ve programınızı farklı sistemlerde kullanabilmeyi istiyorsanız long

tamsayıları kullanın.

Performans İpuçları 4.1

Hafızanın sınırlı ya da hızın gerekli olduğu performansa yönelik durumlarda küçük tamsayı

boyutları kullanmak gerekebilir.

4.8 do/while DÖNGÜ YAPISI

do/while döngü yapısı, while yapısına oldukça benzer. while yapısında döngü devam koşulu,

döngünün gövdesinden önce test ediliyordu. do/while yapısında ise döngü devam koşulu,

döngünün gövdesi çalıştırıldıktan sonra kontrol edilir. Bu sebepten, döngünün gövdesi en az

bir kez çalıştırılır. do/while yapısı sonladığında program, while cümlesinden sonraki ifadeden

devam eder. Eğer do/while yapısının gövdesi tek bir ifadeden oluşuyorsa, küme parantezlerini

kullanmaya gerek yoktur. Buna rağmen, while yapısıyla do/while yapısının karışmasını

önlemek için küme parantezleri kullanılır. Örneğin,

while( koşul )

normal olarak bir while yapısının başlığı olarak algılanır. Küme parantezleri kullanılmadan

yazılacak bir do/while yapısı ise aşağıdaki biçimde görünür:

116

do

ifade

while( koşul ) ;

bu da bazen yanıltıcı olabilir. Son satır, okuyucu tarafından boş bir ifade içeren while yapısı

gibi anlaşılabilir. Bu sebepten, tek ifadeye sahip do/while yapıları bu karışıklığı önlemek için

do{

ifade

}while(koşul);

biçiminde yazılır.

İyi Programlama Alıştırmaları 4.17

Bazı programcılar küme parantezine ihtiyaç duyulmasa bile do/while yapısında küme

parantezlerini kullanırlar.Bu, tek ifade içeren do/while yapıları ile while yapılarının

karıştırılmasını engeller.

Genel Programlama Hataları 4.7

while, for ya da do/while yapılarında döngü devam şartı asla yanlış hale gelmiyorsa, sonsuz

döngüler oluşur. Bunu önlemek için, while ya da for yapılarının başlatıldığı kısmın sonuna

noktalı virgül koymadığınıza emin olun. Sayıcı kontrollü döngülerde, kontrol değişkeninin

döngü gövdesinde arttırılmasına ( ya da azaltılmasına ) dikkat edin. Nöbetçi kontrollü

döngülerde nöbetçi değerin girildiğine emin olun

Şekil 4.9, do/while yapısıyla 1’den 10’a kadar olan sayıları yazdırmaktadır. Kontrol değişkeni

olan sayici değişkeninin, döngü devam koşulu içinde ön arttırma ile arttırıldığına dikkat edin.

Ayrıca do/while yapısının tek ifadeden oluşan gövdesinin, küme parantezleri içine alındığına

da dikkat edin.

do/while yapısının akış grafiği Şekil 4.10’da gösterilmiştir. Bu akış grafiği, döngü

devam koşulunun işlemler en az bir kez yapılmadan çalıştırılmadığını açıkça göstermektedir.

Bu akış grafiğinde de (çember ve akış çizgileri dışında) yalnızca dikdörtgen ve elmas

sembollerinin bulunduğuna dikkat ediniz. Bir kez daha, programcının algoritma oluşturmak

için istediği kadar do/while yapısını diğer kontrol yapılarının üzerine dizebileceğini ya da

içlerine yuvalayabileceğini hatırlayınız. Programcı, algoritmasını tamamlayabilmek için

dikdörtgen ve elmas sembollerinin içlerini tamamlayacaktır.

1 /\* Şekil 4.9: fig04\_09.c

2 do/while döngü yapısını kullanma \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int sayici = 1;

8

9 do {

10 printf( "%d ", sayici);

11 } while ( ++sayici <= 10 );

12

13 return 0;

117

14 }

1 2 3 4 5 6 7 8 9 10

Şekil 4.9 do/while yapısını kullanmak

Şekil 4.10 do/while döngü yapısının akış grafiği

4.9 break ve continue İFADELERİ

break ve continue ifadeleri, kontrol akışını değiştirmek için kullanılır. break komutu while,

for, do/while ve switch ile kullanıldığında o yapıdan çıkışı sağlar ve program yapıdan sonraki

ilk ifadeyi çalıştırarak devam eder. break ifadesinin en genel kullanımı, bir döngüden istenen

anda çıkmak ve switch yapısında olduğu gibi ( Şekil 4.7’ye bakınız ) döngünün belli bir

kısmından kurtulmaktır. Şekil 4.11’de, break ifadesinin for yapısıyla kullanımı gösterilmiştir.

if yapısı x’in 5 olduğunu tespit ettiğinde break ifadesi çalıştırılır. Bu, for döngüsünü

sonlandırır ve program for yapısından sonraki printf ifadesiyle devam eder. Döngü yalnızca

dört kez çalışır.

continue ifadesi while, for ve do/while yapıları içinde çalıştığında döngü gövdesinin kalan

kısmını atlar ve döngünün diğer tekrara geçmesini sağlar. while ve do/while yapılarında

döngü devam koşulu, continue ifadesi çalıştırıldıktan hemen sonra kontrol edilir. for

yapısında arttırma deyimi çalıştırılır daha sonra ise döngü devam koşulu kontrol edilir. Daha

önce while yapılarının for yapılarını temsil edebildiğini söylemiştik. Burada bir istisnayla

karşılaşıyoruz. Eğer while yapısı içinde arttırma deyimi continue ifadesinden sonra

kullanılmışsa, while yapısı for yapısı yerine kullanılamaz. Bu durumda, arttırma döngü

devam koşulundan önce çalıştırılmayacaktır ve while yapısı for yapısı gibi çalışmayacaktır.

Şekil 4.12, continue ifadesi for yapısı içinde printf ifadesini atlamak ve döngünün başına

dönmek için kullanılmıştır.

işlem

koşul

doğru

yanlış

118

İyi Programlama Alıştırmaları 4.17

Bazı programcılar break ve continue ifadelerinin yapısal programlama modeline

uymadığını düşünürler. Bu programcılar, break ve continue yerine ileride öğreneceğimiz

bazı yapısal programlama teknikleriyle aynı etkiyi yaratırlar.

1 /\* Şekil 4.11: fig04\_11.c

2 for yapısı içerisinde break ifadesinin kullanılması \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int x;

8

9 for ( x = 1; x <= 10; x++ ) {

10

11 if ( x == 5 )

12 break; /\* döngüden sadece x == 5 olduğunda çık\*/

13

14 printf( "%d ", x );

15 }

16

17 printf( "\n Döngüden x == %d olduğunda çıkıldı.\n", x );

18 return 0;

19 }

1 2 3 4

Döngüden x==5 olduğunda çıkıldı.

Şekil 4.11 break ifadesini for yapısı içinde kullanmak

1 /\* Şekil 4.12: fig04\_12.c

2 for yapısı içerisinde continue ifadesinin kullanılması \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int x;

8

9 for ( x = 1; x <= 10; x++ ) {

10

11 if ( x == 5 )

12 continue; /\* sadece x == 5 olduğunda döngüyü

13 atla \*/

14

15 printf( "%d ", x );

16 }

17

18 printf( "\ncontinue, 5 değerinin atlanması için kullanıldı\n" );

119

19 return 0;

20 }

1 2 3 4 6 7 8 9 10

continue 5 değerinin atlanması için kullanıldı

Şekil 4.12 continue ifadesini for yapısı içinde kullanmak

Performans İpuçları 4.2

break ve continue eğer uygun bir biçimde kullanılırsa, aynı etkiyi yaratacak yapısal

programlama tekniklerinden daha hızlı çalışırlar.

Yazılım Mühendisliği Gözlemleri 4.1

Yazılım mühendisliğindeki kalite ile en iyi çalışan yazılımı yazmak arasında bir denge vardır.

Bu hedeflerden birine ulaşmak için genelde diğerinden vazgeçilmesi gerekir.

4.10 MANTIK OPERATÖRLERİ

Şimdiye kadar, sayici<=10, toplam>1000 ve sayi != nobetci\_deger gibi basit durumları

çalıştık. Bu koşulları > , < , >= , <= gibi karşılaştırma operatörleri ve == , != gibi eşitlik

operatörleri sayesinde ifade ettik. Her karar yalnızca bir koşulu test ediyordu. Eğer karar

verme sürecinde birden fazla koşulu test etmek istersek, bu testleri yuvalı if ya da if/else

yapıları sayesinde gerçekleştiriyorduk.

C, basit koşulları birleştirerek daha karmaşık koşullar yaratmamamıza imkan veren mantık

operatörlerine sahiptir. Mantık operatörleri, && (mantıksal ve), || (mantıksal veya) ve !

(mantıksal değil) olarak belirlenmiştir. Her biriyle ilgili örnekler vereceğiz.

Bir işlemin çalışmasını yönlendirmek için, iki koşulun aynı anda doğru olması gerektiğini

düşünelim. Bu durumda && operatörü aşağıdaki biçimde kullanılır:

if ( cinsiyet ==1 && yas >= 65)

++kadin\_emekli;

Bu if ifadesi, iki basit koşul içermektedir.cinsiyet == 1 koşulu, bir kişinin kadın olup

olmadığını belirlemek için hesaplanabilir. yas >= 65 koşulu ise, bu kişinin emekli olup

olmadığını belirlemek için kullanılabilir. İki basit koşul önce hesaplanır çünkü >= ve ==

operatörleri && operatörüne göre önceliklidir. if ifadesi aşağıdaki birleştirilmiş koşulu

dikkate almaktadır:

cinsiyet ==1 && yas >= 65

Bu koşul, yalnızca iki basit koşul da doğru ise doğrudur. Son olarak, eğer bu koşul doğru ise

kadin\_emekli değişkeni 1 arttırılacaktır. Eğer bu iki basit koşuldan herhangi biri yanlış ise

program if yapısını atlayacak ve çalışmaya if yapısından sonraki ifadeyle devam edecektir.

Şekil 4.13, && operatörünü özetlemektedir. Tablo, deyim1 ve deyim2 için doğru ve yanlış

olabilme ihtimallerinin tümünü göstermektedir. Bu tür tablolara doğruluk tabloları denir. C,

karşılaştırma operatörleri, eşitlik operatörleri ve/veya mantık operatörlerini içeren tüm

120

deyimleri 0 ya da 1 olarak hesaplar. Ayrıca C, doğru bir değeri 1 olarak kullanmasına rağmen,

0 haricindeki tüm değerleri de doğru olarak kabul eder.

deyim1 deyim2 deyim1&&deyim2

0 0 0

0 sıfırdan farlı değer 0

sıfırdan farlı değer 0 0

sıfırdan farlı değer sıfırdan farlı değer 1

Şekil 4.13 && (mantıksal ve) operatörü için doğruluk tablosu

Şimdi de || (mantıksal veya) operatörünü inceleyelim. Programımızda bir işlemin çalışmasını

yönlendirmek için iki koşulun aynı anda ya da iki koşuldan birinin doğru olması gerektiğini

düşünelim. Bu durumda, || operatörü aşağıdaki program parçacığında olduğu gibi kullanılır:

if ( donem\_ortalamasi >= 90 || final\_sinavi >= 90)

printf ( “Öğrenci ortalaması A’ dır\n” );

Bu ifade de iki basit koşul içermektedir. donem\_ortalamasi>= 90 koşulu, öğrencinin dönem

boyunca gösterdiği performanstan dolayı dersten A notu alması gerektiği koşulunu

hesaplamaktadır. final\_sinavi >=90 koşulu, öğrencinin final sınavından 90 ve üstü bir not

almasından dolayı dersten A notu alması gerektiği koşulunu hesaplamaktadır. if yapısı

aşağıdaki birleştirilmiş koşulu dikkate almaktadır.

donem\_ortalamasi >=90 || final\_sinavi >=90

ve eğer iki koşuldan biri ya da ikisi de birden doğru ise öğrenciyi A notuyla

ödüllendirmektedir. “Öğrenci ortalaması A’dır” mesajının yalnızca iki koşulda yanlışken

yazdırılmadığına dikkat ediniz. Şekil 4.14, mantıksal veya ( || ) operatörünün doğruluk

tablosunu göstermektedir.

deyim1 deyim2 deyim1||deyim2

0 0 0

0 sıfırdan farlı değer 1

sıfırdan farlı değer 0 1

sıfırdan farklı değer sıfırdan farlı değer 1

Şekil 4.14 Mantıksal veya operatörü (||) için doğruluk tablosu

&& operatörü, || operatörüne göre daha yüksek önceliğe sahiptir. İki operatörde soldan sağa

doğru çalışmaktadır. && ya da || operatörlerini içeren bir deyim yalnızca doğruluk ya da

yanlışlık durumu bilinene kadar hesaplanır. Bu sebepten,

cinsiyet = =1 && yas >= 65

121

koşulunun hesaplanması eğer cinsiyet == 1 deyimi yanlışsa ( yani tüm deyim yanlışsa )

duracaktır ve cinsiyet 1’e eşitse ( tüm deyim, eğer yas>=65 ise doğru olabilir ) devam

edecektir.

Performans İpuçları 4.3

&& operatörünü kullanan deyimlerde en sola yanlış olma ihtimali daha fazla olan deyimi

yerleştirin. || operatörünü kullanan deyimlerde doğru olma ihtimali daha fazla olan deyimi en

sola yerleştirin. Bu, programın çalışma zamanını kısaltacaktır.

C, programcının bir koşulun anlamını tersine çevirmesine imkan sağlayan !(mantıksal değil)

operatörüne sahiptir. && ve || operatörleri iki koşulu birleştirirken (bu sebepten ikili

operatörlerdir), mantıksal değil operatörü yalnızca tek bir koşulu operand olarak kullanır. (bu

sebepten tekli bir operatördür)

Mantıksal değil operatörü, koşul yanlış olduğunda izlenecek yolu belirlemek amacıyla

koşulun başına yerleştirilir. Aşağıdaki program parçacığını inceleyiniz:

if ( ! ( not == nobetci\_deger ) )

printf ( “Bir sonraki not %f’dir\n”,not );

not == nobetci\_deger koşulu, mantıksal değil operatörü eşitlik operatöründen daha yüksek

önceliğe sahip olduğundan parantez içine alınmalıdır. Şekil 4.15’te, mantıksal değil

operatörünün doğruluk tablosunu görebilirsiniz.

deyim !deyim

0 1

sıfırdan farklı değer 0

Şekil 4.15 mantıksal değil (!) operatörünün doğruluk tablosu.

Çoğu durumda programcılar, mantıksal değil operatörü yerine deyimi başka bir şekilde,

karşılaştırma operatörleri kullanarak, yazıp aynı etkiyi yaratırlar. Örneğin, az önceki ifadeyi

aşağıdaki biçimde yazmak mümkündür:

if ( not ! = nobetci\_deger )

printf ( “Bir sonraki not %f ’dir\n”,not);

Şekil 4.16, şu ana kadar gösterilen operatörlerin öncelik sıralarını ve çalışma biçimlerini

göstermektedir. Operatörlerin öncelikleri yukarıdan aşağıya doğru gidildikçe azalmaktadır.

4.11 EŞİTLİK VE ATAMA OPERATÖRLERİ

Genelde ne kadar tecrübeli olursa olsun, C programcılarının yaptıkları bir hata vardır.Bu

sebepten bu hataya ayrı bir kısım ayırmayı uygun gördük. Bu hata, == ( eşitlik ) ve = ( atama )

operatörlerinin karıştırılmasından kaynaklanır. Bu hata, bir yazım hatasından daha ciddidir

çünkü bu hataların yer aldığı ifadeler doğru bir biçimde derlenirler ancak çalışma zamanında

programın hatalı sonuçlar üretmesine sebep olurlar.

Operatörler İşeyiş Biçimleri Tipleri

( ) soldan sağa parantez

++ -- + - ! (tip) sağdan sola tekli

\* / % soldan sağa multiplicative

122

+ - soldan sağa additive

< >= > >= soldan sağa karşılaştırma

= = != soldan sağa eşitlik

&& soldan sağa mantıksal ve

|| soldan sağa mantıksal veya

?: sağdan sola koşullu

= += -= \*= /= %= sağdan sola atama

, soldan sağa virgül

Şekil 4.16 Operatörlerin öncelikleri ve işleyiş biçimleri

C’nin bu tür hataları üretmesinin iki sebebi vardır. Bunlardan birincisi, C’de değer üretebilen

herhangi bir deyimin, kontrol yapılarının karar kısımlarında kullanılabilmesidir. Eğer değer 0

ise yanlış olarak değerlendirilir. Eğer değer sıfırdan farklı ise doğru olarak kabul edilir. İkinci

sebep ise C’de atamaların bir değer üretmesidir. Bu değer, atama operatörünün solundaki

değerdir. Örneğin,

if ( puan == 4)

printf ( “Bonus kazandınız\n” );

yazmak isterken yanlışlıkla

if ( puan = 4)

printf ( “Bonus kazandınız\n” );

yazdığımızı düşünelim.

İlk if ifadesi, eğer kişinin puanı 4 ise o kişiye bonus vermektedir. İkinci if ifadesi ise yanlış

bir biçimde yazıldığından, öncelikle atama operatörünün değerini hesaplar. Bu değer sıfırdan

farklı bir değer olduğundan, doğru olarak kabul edilir. Bu da, if yapısının sürekli doğru olarak

çalışması ve kişinin puanı kaç olursa olsun kişiye bonus verilmesi hatasına sebep olur.

Genel Programlama Hataları 4.8

== operatörünü atama ya da = operatörünü eşitlik için kullanmak.

Programcılar x ==7 gibi koşulları sabit değer sağda, değişken ismi solda olacak şekilde, 7 ==

x biçiminde yazarlar. Böylece, == operatörünü yanlışlıkla = operatörüyle karıştırmaktan

kurtulurlar. Derleyiciler, bu yazım hatasını tespit edebilirler çünkü bir atama işleminde

değişken ismi yalnızca değişkenin solunda bulunabilir. En azından böylelikle, çalışma

zamanlı bir hata yapma ihtimali ortadan kalkmış olur.

Değişken isimleri, sol taraf değeri olarak adlandırılırlar çünkü değişken isimleri atama

işlemlerinde operatörün solunda bulunurlar. Sabit değerler ise sağ taraf değeri olarak

adlandırılırlar çünkü yalnızca atama operatörünün sağında bulunabilirler. Sol taraf değerleri,

sağ taraf değeri olabilir ancak bunun tersi geçerli değildir.

İyi Programlama Alıştırmaları 4.18

Eşitlik deyimi, x == 1 gibi bir değişken ismi ve bir sabit içerdiğinde,bazı programcılar

deyimi sabit değer solda, değişken ismi sağda bulunacak şekilde yazarak, yanlışlıkla ==

operatörüyle = operatörünü karıştırmaktan kaynaklanabilecek mantık hatalarını engellerler.

123

Madalyonun öteki yüzü de aynı biçimde can sıkıcıdır. Programcının bir değişkene sabit bir

değer atamak istediğini düşünelim:

x = 1;

yazacağına yanlışlıkla

x ==1;

yazması bir yazım hatası oluşturmaz. Derleyici, karşılaştırma deyimini hesaplar. Eğer x, 1’e

eşitse koşul doğrudur ve deyim, 1 değerini döndürecektir. Eğer x, 1’e eşit değilse koşul yanlış

olacaktır ve deyim, 0 değerini döndürecektir. Hangi değer döndürülürse döndürülsün, atama

operatörü bulunmadığından değer kaybolacak ve x değişmeden kalacaktır. Bu da muhtemelen

çalışma zamanlı bir mantık hatası üretecektir. Maalesef , bu problemi ortadan kaldıracak bir

yola sahip değiliz.

4.12 YAPISAL PROGRAMLAMA ÖZETİ

Programcılar programlarını, mimarların binalar tasarlarken bütün bilgilerini tecrübeleriyle

birleştirdikleri gibi yazmalıdırlar. Bizim alanımız daha oldukça yenidir ve tecrübelerimiz de

oldukça azdır. Ama son 50 yıl içinde çok fazla şey öğrendik ve belki de en önemlisi yapısal

olmayan programlara göre test etmesi, hata ayıklaması, değiştirilmesi ve anlaşılması daha

kolay olan yapısal programlamayı öğrendik.

3 ve 4.Ünitelerde C’nin kontrol yapılarını öğrendik. Her yapı önce tanıtıldı, akış grafikleri

gösterildi ve çeşitli örneklerle incelendi. Şimdi, 3 ve 4.üniteden çıkarttığımız sonuçları

özetleyerek yapısal programlamanın özelliklerini ve bir takım kurallarını tanıtacağız.

Şekil 4.17, 3 ve 4. ünitede tartışılan kontrol yapılarını özetlemektedir. Şekilde kullanılan

çemberler, her yapının tekli giriş ya da tekli çıkış yapabildiği noktaları belirtmektedir. Akış

grafiklerini birbirine rasgele bağlamak, yapısal olmayan programlar yazılmasına sebep

olabilir. Bu sebepten, programlamada akış grafikleri yalnızca sınırlı sayıda kontrol yapısı

oluşturacak biçimde ve kontrol yapıları yalnızca iki basit şekilde birleştirilmiştir. Kolaylığı

sağlamak için tekli giriş ve tekli çıkışa sahip kontrol yapıları kullanılmıştır. Her kontrol

yapısına yalnızca tek bir noktadan girilebilir ve kontrol yapılarından çıkmak için tek bir nokta

kullanılabilir. Kontrol yapılarını bir dizi biçiminde bağlamak oldukça basittir. Bir kontrol

yapısının çıkış noktası diğer kontrol yapısının giriş noktasına bağlanır. Kontrol yapıları

programda birbiri ardına sıralanır, buna kontrol yapısı yığma adı verilir. Yapısal programlama

kuralları ayrıca kontrol yapılarının yuvalı biçimde yerleştirilmesine de imkan tanır.

Şekil 4.18, uygun biçimde yapısal programlama yapmak için uyulacak kuralları gösterir.

Kurallar, dikdörtgen işaretinin giriş/çıkış işlemleri de dahil olmak üzere işlemleri gösterdiğini

kabul eder.

Şekil 4.18’deki kuralları uygulamak, her zaman düzgün ve bloklar biçiminde akış grafikleri

elde etmemizi sağlar. Örneğin, kural 2‘yi en basit akış grafiğine sürekli uygulamak bir dizi

dikdörtgenin ard arda sıralandığı bir akış grafiği elde etmemizi sağlar. (Şekil 4.20) Kural 2,

kontrol yapılarından bir yığın oluşturmaktadır, bu sebepten bu kural yığma kuralı olarak ta

adlandırılır.

124

Kural 3, yuvalama kuralı olarak bilinir. Kural 3’ü en basit akış grafiğine ard arda uygulamak

kontrol yapılarının yuvalı bir biçimde yerleştirildiği bir akış grafiği elde etmemizi sağlar.

Örneğin, Şekil 4.21’de en basit akış grafiğindeki dikdörtgen işareti ilk önce çiftli seçim

yapısıyla (if/else) değiştirilmiştir. Ardından kural 3, çiftli seçim yapısının içindeki

dikdörtgenlere yeniden uygulanmıştır ve dikdörtgenlerden her biri çiftli seçim yapılarıyla

değiştirilmiştir. Her çiftli seçim yapısının etrafındaki noktalı çizgiler, ilk akış grafiğinde

değiştirilen dikdörtgeni temsil etmektedir.

Yapısal Program Oluşturma Kuralları

Yapısal Program oluşturma kuralları :

1) En basit akış grafiğiyle başlanır ( Şekil 4.19 )

2) Her dikdörtgen ( işlem) dizi halinde iki dikdörtgen ( işlem ) ile değiştirilebilir.

3) Her dikdörtgen herhangi bir kontrol yapısıyla ( sıra kontrol yapısı, if, if/else, switch,

while, do/while veya for ) ile değiştirilebilir.

4) Kural 2 ve kural 3 istenen sıklıkta ve istenen sırada uygulanabilir.

Şekil 4.18 Yapısal Program Oluşturma Kuralları

Şekil 4.19 En basit akış grafiği

125

Dddd

break

break

break

.

.

.

.

.

.

Seri

if yapısı

(tek seçimli)

t

f

if/else yapısı

(çift seçimli)

f t

switch yapısı

(çoklu seçim)

t

f

t

t

f

while yapısı

t

f

do/while yapısı

for yapısı

t

f

t

f

Döngü

Şekil 4.17 C ’in tekli giriş/tekli çıkış dizi, seçim,

döngü yapıları

126

Şekil 4.20 Kural 2’yi en basit akış grafiğine tekrar tekrar uygulamak.

Kural 4, daha geniş, daha içerikli ve daha çok yuvalanmış yapılar üretir. Şekil 4.18’deki

kuralları uygulayarak oluşturulan akış grafikleri, muhtemel bütün yapısal akış grafiklerini

oluşturur. Dolayısıyla da, muhtemel bütün yapısal programları oluşturur.

goto ifadesi elendiği için bloklar birbirleri içine geçemezler. Yapısal programlamanın güzel

tarafı, yalnızca az sayıda tekli giriş/tekli çıkış noktası kullanmamız ve bunları yalnızca iki

basit yolla birleştirmemizdir. Şekil 4.22, kural 2’nin uygulanmasıyla elde edilen blok

yığınlarını ve kural3’ün uygulamasından elde edilen yuvalı blokları göstermektedir. Şekil,

ayrıca blokların birbirleri içine geçemeyeceğini de göstermektedir. (goto ifadesi elendiği için)

Şekil 4.8’deki kurallara uyulursa, Şekil 4.23’teki gibi yapısal olmayan bir akış grafiği

oluşturulamaz. Eğer herhangi bir anda akış grafiğinizin yapısal olup olmadığından emin

olamazsanız, Şekil 4.18’deki kuralları tersine doğru uygulayarak en basit akış grafiğine

ulaşmayı deneyin. Eğer akış grafiği en basit akış grafiğine indirgenebiliyorsa akış grafiğiniz

yapısaldır, aksi takdirde de yapısal değildir.

Yapısal programlama basitlik sağlar. Bohm ve Jacoponi’nin çalışmaları 3 çeşit kontrole

ihtiyaç duyulduğunu göstermiştir:

 Sıra

 Seçim

 Döngü

Sıra yapısı tüm programlarda geçerlidir. Seçim yapısı 3 yöntemle sağlanır:

 if yapısı(tekli seçim)

 if/else yapısı(çiftli seçim)

 switch yapısı(çoklu seçim)

.

.

127

Şekil 4.21 Kural3’ü en basit akış grafiğine uygulamak

kural3

kural3

kural3

128

Şekil 4.22 Yığın halinde , yuvalı ve iç içe geçmiş bloklar

Şekil 4.23Yapısal olmayan akış grafiği.

Aslında, basit if yapısıyla her türlü seçim gerçekleştirilebilir. if/else ya da switch yapısıyla

yapılabilecek her şey if yapısıyla gerçekleştirilebilir.

Döngü 3 yöntemden biriyle yapılır:

 while yapısı

 do/while yapısı

 for yapısı

while yapısının, herhangi bir formda döngü oluşturmak için yeterli olabileceği ispatlanabilir.

do/while ya da for yapılarıyla yapılabilecek her şey while yapısıyla gerçekleştirilebilir.

kontrol yapısı yığma blokları yuvalama

129

Bu sonuçlar birleştirildiğinde, bir C programında ihtiyaç duyulan herhangi bir kontrol

yapısının aşağıdaki 3 kontrol yapısıyla sağlanılabileceği görülür:

\* sıra

\* if yapısı(seçim)

\* while yapısı(döngü)

Bu kontrol yapıları yalnızca iki biçimde birleştirilebilir : yığma ve yuvalama.

3 ve 4. ünitelerde, işlem ve kararlar içeren kontrol yapılarıyla nasıl program yazacağımızı

tartıştık. 5. Ünitede diğer bir yapısal programlama biriminden ( fonksiyon ) bahsedeceğiz.

Kontrol yapılarıyla oluşturulmuş fonksiyonları birleştirerek nasıl büyük programlar

oluşturduğumuzu öğreneceğiz. Ayrıca fonksiyonların kullanımının yazılımın yeniden

kullanılabilirliğini sağlamasını tartışacağız.

ÖZET

 Döngü, bilgisayarın belli sonlandırma koşulları sağlanana kadar tekrar ettiği bir grup

emirdir. Döngünün iki çeşidi vardır. Bunlar sayıcı kontrollü döngü ve nöbetçi

kontrollü döngüdür.

 Döngü sayıcısı, döngünün kaç kez tekrarlanacağını saymak için kullanılır. Döngünün

her tekrarında sayıcı arttırılır ( genellikle 1 artırılır)

 Nöbetçi değerler, döngünün kaç kez tekrarlanacağı bilinmediğinde ve döngü her

tekrarında yeni bir veri alıyorsa kullanılır.

 Nöbetçi değer, programın işleyeceği tüm geçerli veriler girildikten sonra girilir.

Nöbetçi değerler, verilerle karışmayacak bir biçimde dikkatlice seçilmelidir.

 for döngü yapısı, sayıcı kontrollü döngülerin tüm detaylarını otomatik olarak

yapabilir. for yapısının genel biçimi aşağıdaki gibidir:

for(deyim1;deyim2;deyim3)

ifade

 do/while döngü yapısı, while döngü yapısına benzerdir. Ancak do/while döngü

yapısında, döngü devam koşulu döngünün sonunda kontrol edildiğinden döngü en az

bir kere çalışır. do/while yapısının genel biçimi aşağıdaki gibidir:

do

ifade

while( koşul );

 break ifadesi, döngü yapıları içinde(for,while ve do/while) çalıştırıldığında yapıdan

anında çıkışı sağlar. Programın çalışması döngüden sonraki ilk ifadeyle devam eder.

 continue ifadesi, döngü yapıları içinde(for,while ve do/while) çalıştırıldığında

yapının gövdesindeki diğer ifadeleri atlar ve döngünün bir sonraki tekrarını başlatır.

 switch ifadesi, bir değişkenin ya da ifadenin alabileceği değerlere göre farklı işlemler

yapma kararını halleder. switch ifadesi içindeki her case kısmı birçok ifadenin

çalıştırılmasını sağlatabilir. Çoğu programlarda her case kısmının break ifadesi

içermesi gerekebilir. Aksi takdirde program break ifadesiyle karşılaşıncaya ya da

130

switch yapısının sonuna gelinceye dek her case kısmı içindeki ifadeleri gerçekleştirir.

Birden fazla case kısmı aynı ifadeleri çalıştırmaları için ifadelerden önce yan yana

sıralanabilir. switch yapısı yalnızca sabit değerleri test edebilir.

 getchar fonksiyonu, klavyeden (standart giriş) alınan bir karakteri tamsayı değeri

olarak döndürür.

 UNIX sistemlerinde ve diğer çoğunda EOF karakteri aşağıdaki tuş kombinasyonuyla

girilir;

<return><ctrl-d>

 VMS ve DOS sistemlerinde EOF karakteri aşağıdaki tuş kombinasyonuyla girilir;

<crtl-z>

 Mantık operatörleri koşulları birleştirerek karmaşık koşullar yaratmakta kullanılabilir.

Mantık operatörleri şunlardır: &&(mantıksal ve), ||(mantıksal veya) ve !(mantıksal

değil)

 Sıfırdan farklı bir değer, doğru olarak kabul edilir.

 Sıfır değeri, yanlış olarak kabul edilir.

ÇEVRİLEN TERİMLER

ASCII character set.................................................... ASCII karakter kümesi

conrol variable........................................................... kontrol değişkeni

end of file.................................................................. dosya sonu belirteci

field width................................................................. alan genişliği

left justify................................................................. sola hizalamak/yaslamak

logical AND ( && )................................................. mantıksal VE

logical OR ( | | )....................................................... mantıksal VEYA

logical negation ( ! )................................................ mantıksal DEĞİL

loop-continuation condition ................................ döngü devam koşulu

loop-control variable........................................... döngü konrol değişkeni

loop counter........................................................ döngü sayıcısı

nesting rule.............................................................. yavalama kuralı

off-by-one error....................................................... bir eksik hatası

stacking rule............................................................. yığma kuralı

truth table................................................................. doğruluk tablosu

unary operator........................................................... tekli operatör

GENEL PROGRAMLAMA HATALARI

4.1 Ondalıklı sayılar yalnızca gerçeğe yakın birer tahmin olduğundan, döngülerde

kontrol değişkeni olarak kullanılması kesin olmayan sayıcı değerleri elde edilmesine

ve sonlandırma için yanlış değerlere sahip olunmasına sebep olur.

4.2 while ya da for döngüsü içinde yanlış karşılaştırma operatörü kullanmak ya da döngü

sayıcısı için yanlış son değerler vermek mantık hatası oluşturur.

4.3 for içinde noktalı virgül yerine virgül kullanmak

4.4 for yapısının başlığının dışına noktalı virgül koymak o for yapısının gövdesini boş bir

ifade haline getirir.Bu, bir mantık hatasıdır.

131

4.5 switch yapısında gerekli yerlerde break kullanmamak

4.6 Karakter okurken yeni satır karakterlerini özel olarak işlememek mantık hatalarına

sebep olabilir.

4.7 while, for ya da do/while yapılarında döngü devam şartı asla yanlış hale gelmiyorsa

sonsuz döngüler oluşur. Bunu önlemek için while ya da for yapılarının başlatıldığı

kısmın sonuna noktalı virgül koymadığınıza emin olun. Sayıcı kontrollü döngülerde

kontrol değişkeninin döngü gövdesinde arttırılmasına(ya da azaltılmasına) dikkat edin.

Nöbetçi kontrollü döngülerde nöbetçi değerin girildiğine emin olun

4.8 == operatörünü atama ya da = operatörünü eşitlik için kullanmak.

İYİ PROGRAMLAMA ALIŞTIRMALARI

4.1 Döngüleri tamsayı değerleriyle kontrol etmek

4.2 Kontrol yapılarının gövdelerini içeriden başlatmak

4.3 Kontrol yapısından önce ve sonra boşluk bırakarak kontrol yapılarını programda

belirgin hale getirmek

4.4 Çok fazla yuvalama kullanmak programın anlaşılabilirliğini zorlaştırır. Genel bir

kural

olarak üç seviyeden fazla yuvalama kullanılmamalıdır.

4.5 Kontrol yapısından önce ve sonra boşluk bırakarak kontrol yapılarının

gövdelerini,

kontrol yapısının başlığından daha içeriden başlatarak programda belirgin hale

getirmek ve kontrol yapılarının gövdelerini içeriden başlatmak programın iki

boyutlu

hale gelmesini sağlayarak okunurluğu artırır.

4.6 while ve for yapısı içinde döngünün son değerini <= karşılaştırma operatörüyle

birlikte kullanmak mantık hatalarını engellemeye yardımcı olur. Örneğin 1’den 10’a

kadar değerleri yazdıracak bir döngünün döngü devam koşulu sayici<11 ya da sayici

<10 yerine sayici<=10 olmalıdır.

4.7 for yapısı içine yalnızca kontrol değişkenlerine ilk değer atama kısımlarını ve for

yapısında arttırma yapılan kısımları yerleştirmek. Diğer değişkenlerle ilgili işlemler

eğer yalnızca bir kez yapılacaklarsa döngüden önce, eğer birden fazla

tekrarlanacaklarsa döngünün içine yerleştirilmelidir.

4.8 Kontrol değişkenini for döngüsünün gövdesi içinde değiştirmek

mümkündür.Ancak

bu, hatalara yol açabilir.En iyisi kontrol değişkenini değiştirmemektir.

4.9 for yapısı gövdesindeki ifadeler for yapısı içine alınabilse de bunu yapmaktan

kaçınmalıyız çünkü programın okunurluğu zorlaşır.

4.10 Kontrol yapısını eğer mümkünse tek bir satıra sığdırmak.

4.11 float ve double tipte değişkenleri parayla ilgili hesaplamalarda kullanmayınız.

Ondalıklı sayıların kesin olarak gösterilememesinden dolayı hatalar

oluşabilir.

4.12 switch yapılarının default kısmını içermesini sağlayın.switch yapısında test

edilmeyen case yapıları ihmal edilir. default kullanmak programcının istisnai

durumları işleyebilmesine yardımcı olur.Bazı durumlarda default kısmını

kullanmaya gerek olmayabilir.

4.13 case ve default kısımları switch yapısında istenen sırada yer alabilir ancak default

kısmını yapının sonunda kullanmak iyi bir alışkanlıktır.

132

4.14 switch yapılarında default kısmı yapının en sonuna yerleştirildiğinde break

ifadesine gerek kalmaz. Ancak bazı programcılar case kısımlarıyla uyumun

bozulmaması için default kısmında da break kullanırlar.

4.15 Karakter okurken yeni satır karakterlerini işleyecek kodlar yazmayı unutmayın

4.16 Bazı programcılar, küme parantezine ihtiyaç duyulmasa bile do/while yapısında

küme parantezlerini kullanırlar. Bu, tek ifade içeren do/while yapıları ile while

yapılarının karıştırılmasını engeller.

4.17 Bazı programcılar, break ve continue ifadelerinin yapısal programlama modeline

uymadığını düşünürler. Bu programcılar, break ve continue yerine ileride

öğreneceğimiz bazı yapısal programlama teknikleriyle aynı etkiyi yaratırlar.

4.18 Eşitlik deyimi, x == 1 gibi bir değişken ismi ve bir sabit içerdiğinde, bazı

programcılar deyimi sabit değer solda, değişken ismi sağda bulunacak şekilde

yazarak, yanlışlıkla == operatörüyle = operatörünü karıştırmaktan

kaynaklanabilecek mantık hatalarını engellerler.

PERFORMANS İPUÇLARI

4.1 Hafızanın sınırlı ya da hızın gerekli olduğu performansa yönelik durumlarda küçük

tamsayı boyutları kullanmak gerekebilir

4.2 break ve continue eğer uygun bir biçimde kullanılırsa, aynı etkiyi yaratacak yapısal

programlama tekniklerinden daha hızlı çalışırlar.

4.3 && operatörünü kullanan ifadelerde en sola yanlış olma ihtimali daha fazla olan deyimi

yerleştirin. || operatörünü kullanan ifadelerde doğru olma ihtimali daha fazla olan deyimi

en sola yerleştirin. Bu, programın çalışma zamanını kısaltacaktır.

TAŞINIRLIK İPUÇLARI

4.1 EOF değerini girmek için gerekli olan tuş kombinasyonu sistemlerde farklılık

gösterebilir.

4.2 -1 değeri yerine EOF ile test etmek daha taşınırlığı arttırır. ANSI standardına göre

EOF

negatif bir değerdir. Bu sebepten, EOF farklı sistemlerde farklı değerler sahip olabilir.

4.3 int farklı sistemlerde farklılık gösterebildiğinden eğer kullanacağınız değerlerin 32767

aralığı dışında olmasını bekliyorsanız ve programınızı farklı sistemlerde kullanabilmeyi

istiyorsanız long tamsayıları kullanın.

YAZILIM MÜHENDİSLİĞİ GÖZLEMLERİ

4.1 Yazılım mühendisliğindeki kalite ile en iyi çalışan yazılımı yazmak arasında bir denge

vardır. Bu hedeflerden birine ulaşmak için genelde diğerinden vazgeçilmesi gerekir.

ÇÖZÜMLÜ ALIŞTIRMALAR

a) Sayıcı kontrollü döngüler aynı zamanda \_\_\_\_\_\_\_ döngüler olarak da bilinir, çünkü

döngünün kaç kez tekrarlanacağı bellidir.

b) Nöbetçi kontrollü döngüler aynı zamanda \_\_\_\_\_\_\_ döngüler olarak da bilinir. Çünkü

döngünün kaç kez tekrarlanacağı belli değildir.

133

c) Sayıcı kontrollü bir döngüde \_\_\_\_\_\_\_ bir grup komutun kaç kez tekrar edilerek

çalıştırılacağını sayar.

d) \_\_\_\_\_\_\_ ifadesi, döngü içinde çalıştırıldığında döngü bir sonraki turuna başlar.

e) \_\_\_\_\_\_\_ ifadesi bir döngü içinde ya da switch yapısında kullanıldığında o yapıdan

çıkış sağlanır.

f) \_\_\_\_\_\_\_ bir değişkeni ya da ifadeyi, alabileceği her tam sayı değeri için test eder.

4.2 Aşağıdaki ifadelerin doğru ya da yanlış olduklarına karar veriniz. Yanlış olanların neden

yanlış olduğunu açıklayınız.

a) default kısmına switch yapısı içerisinde yer verilmesi zorunludur.

b) break ifadesine, switch yapısının default kısmında yer verilmesi zorunludur.

c) (x > y) && (a < b) deyimi x > y olduğunda ya da a < b olduğunda doğrudur.

d) || operatörü kullanılan bir ifade, operandlarının her ikisi ya da yalnız biri doğru

olduğunda doğru olur.

4.3 Aşağıdaki istenenleri gerçekleştirecek C ifadelerini yazınız.

a) for yapısı kullanarak 1'den 99'a kadar olan tek tamsayıları toplatın. toplam ve sayac

isimli tamsayı değişkenlerinin daha önceden bilidirlmiş olduğunu kabul edin.

b) 333.46372 sayısını 15 genişliğinde bir alana 1, 2, 3, 4, 5 duyarlığında sırasıyla

yazdırın. Çıktı sola dayalı olsun. Bu beş çıktı nasıl olur?

c) 2.5'in üçüncü kuvvetini pow fonksiyonunu kullanarak bulun ve 10 genişliğinde bir

alana 2 kesinliğinde yazın. Çıktı nasıl olur?

d) Bir while döngüsü ve sayac değişkeni olarak verilen x'i kullanarak 1'den 20'ye kadar

olan tamsayıları ekrana yazdırın. x değişkeninin daha önce ilk değerinin atanmadan

bildirilmiş olduğunu kabul edin. Her satıra sadece 5 sayı yazdırın. İpucu: x % 5

işlemini kullanın. Bu işlemin sonucu 0 ise yeni bir satır karakteri, değilse /t karakteri

basılsın.

e) Alıştırma 4.3 d) deki ifadeyi for döngüsü ile gerçekleştirin.

4.4 Aşağıdaki kod parçalarındaki hataları bulun ve düzeltin.

a) x = 1;

while (x <= 10);

x++;

}

134

b) for (y = .1; y != 1.0; y += .1)

printf ("%f\n", y);

c) switch (n) {

case 1:

printf ("1. sayı\n");

case 2:

printf ("2. sayı\n");

break;

default:

printf ("sayı 1 veya 2 değil\n");

break;

}

d) Aşağıdaki kod 1'den 10'a kadar sayıları yazmalı

n = 1;

while (n < 10)

printf ("%d ", n++);

Çözümler

4.1 a) belirli b) belirsiz c) kontrol değişkeni ya da sayaç d) continue e) break f) switch seçim

yapısı

4.2

a) Yanlış. default kısmı tercihe bağlıdır. Gerek yoksa kullanılmaz.

b) Yanlış. break ifadesi switch yapısından çıkmak için kullanılır. default kısmı en son

kısım olduğu için break ifadesine gerek duymaz.

c) Yanlış. && ifadesinin doğru olması için operatörün sağ ve solundaki her iki ifadede

doğru olmalıdır.

d) doğru

4.3

a) toplam = 0;

for (sayac = 1; sayac <= 99; sayac += 2)

toplam += sayac;

b) printf("%-15.1f\n", 333.546372); /\* 333.5 yazdırır \*/

printf("%-15.2f\n", 333.546372); /\* 333.55 yazdırır \*/

printf("%-15.3f\n", 333.546372); /\* 333.546 yazdırır \*/

135

printf("%-15.4f\n", 333.546372); /\* 333.5464 yazdırır \*/

printf("%-15.5f\n", 333.546372); /\* 333.54637 yazdırır \*/

c) printf ("%10.2f\n", pow(2.5, 3)); /\* 15.63 yazdırır \*/

d) x = 1;

while (x <= 20) {

printf ("%d", x);

if (x % 5 == 0)

printf ("\n");

else

printf ("\t");

x++;

}

ya da

x = 1;

while (x <= 20)

if (x % 5 == 0)

printf ("%d\n", x++);

else

printf ("%d\t", x++);

ya da

x = 0;

while (++x <= 20)

if (x % 5 == 0)

printf ("%d\n", x);

else

printf ("%d\t", x);

e) for (x = 1; x <= 20; x++) {

printf ("%d", x);

if (x % 5 == 0)

printf ("\n");

else

printf ("\t");

}

ya da

for (x = 1; x <= 20; x++)

if (x % 5 == 0)

printf ("%d\n", x);

else

printf("%d\t", x);

136

4.4

a) Hata: while ’dan sonraki noktakı virgül ( ; ) sonsuz döngüye yol açar.

Düzeltme: ; yerine { konulması ya da ; ve } karakterlerinin kaldırılması.

b) Hata: for yapısında ondalıklı sayıların kullanılması

Düzeltme: Tamsayılar kullanılarak istenilen işlemlerin yapılması.

c) Hata: İlk case içerisinde break ifadesinin bulunmaması

Düzeltme: İlk case içersindeki ifadelerin sonuna break ifadesinin konulması. Not:

Eğer programcı case 1’in her çalışmasından sonra case 2'nin de çalışmasını istiyorsa

bu bir hata değildir.

d) Hata: while yapısı içersinde yanlış karşılaştırma operatörünün kullanılması.

Düzeltme: < yerine <= kullanılması

ALIŞTIRMALAR

4.5 Aşağıdaki ifadelerdeki hataları bulun, bir ifadede birden fazla hata olabilir.

a) For (x = 100, x >=1, x++)

printf ("%d\n", x);

b) Aşağıdaki kod bir tamsayının tek mi yoksa çift mi olduğunu ekrana yazdırır.

switch( deger % 2) {

case 0:

printf ("Çift tam sayı\n");

case 1:

printf ("Tek tam sayı\n");

}

c) Aşağıdaki kod kullanıcıya bir tamsayı ve bir karakter girişi yaptırır ve girilenleri

ekrana yazdırır. Kullanıcının 100 ve A girdiğini kabul ediniz.

scanf("%d", &tamsayi);

karakter=getchar( );

printf ("Tamsayı: %d\n karakter: %c\n", tamsayi, karakter);

d) for (x = .000001; x <= .0001; x += .000001)

printf ("%7f\n", x);

e) Aşağıdaki kod 999'dan 1'e kadar olan tek tamsayıları ekrana yazdırır.

for (x = 999; x >=1; x+=2)

printf ("%d\n", x);

f) Aşağıdaki kod 2'den 100'e kadar olan çift tamsayıları ekrana yazdırır.

sayac = 2;

Do {

if (sayac %2 == 0)

printf ("%d\n", sayac);

sayac += 2;

} While (sayac < 100);

g) Aşağıdaki kod 100'den 150'ye kadar olan tamsayıları toplar.(toplam değişkeninin ilk

değerinin 0 olarak daha önceden atanmış olduğunu kabul edin)

for (x = 100; x <= 150; x++)

137

toplam += x;

4.6 Aşağıdaki for ifadelerinde x 'in hangi değerleri ekrana yazdırılır.

a) for (x = 2; x <= 13; x+=2)

printf ("%d\n", x);

b) for (x = 5; x <= 22; x+=7)

printf ("%d\n", x);

c) for (x = 3; x <= 15; x+=3)

printf ("%d\n", x);

d) for (x = 1; x <= 5; x+=7)

printf ("%d\n", x);

e) for (x = 12; x >= 2; x-=3)

printf ("%d\n", x);

4.7 Aşağıdaki dizileri ekrana yazdıran for ifadelerini yazınız.

a) 1, 2, 3, 4, 5, 6, 7

b) 3, 8, 13, 18, 23

c) 20, 14, 8, 2, -4, -10

d) 19, 27, 35, 43, 51

4.8 Aşağıdaki program ne yapar?

#include <stdio.h>

main( )

{

int i, , j, x, y;

printf ("1-20 arasında iki tamsayı girin.: ");

scanf ("%d%d", &x, &y);

for ( i = 1; i <= y; i++) {

for (j = 1; j <= x; j++)

printf ("@");

printf ("\n");

}

return 0;

}

4.9 Bir dizi tamsayıyı toplayan bir program yazınız. scanf tarafından alınan ilk tamsayının

toplanacak kaç sayı olacağını gösterdiğini kabul edin. Her scanf ifadesiyle sadece bir tamsayı

girişi yaptırın. Örnek bir girdi dizisi:

5 100 200 300 400 500

olabilir. 5, 5 tamsayının toplanacağını gösterir.

138

4.10 Birkaç sayının ortalamasını bulan ve ekrana yazdıran bir program yazınız. scanf ile

alınacak en son sayı 9999 olmalıdır.Örnek bir dizi şu şekildedir :

10 8 11 7 9 9999

9999’dan önceki sayıların ortalaması alınmalıdır.

4.11 Kullanıcı tarafından girilen birkaç tamsayının en küçüğünü bulan bir program yazınız.

Girilen ilk tamsayı daha sonra kaç sayı girileceğini belirtsin.

4.12 2'den 30'a kadar olan çift tamsayıların toplamını hesaplayan ve ekrana yazdıran bir

program yazınız.

4.13 1'den 15'e kadar olan tek tamsayıların çarpımını hesaplayan ve ekrana yazdıran bir

program yazınız.

4.14 faktöriyel fonksiyonu olasılık problemlerinde sıklıkla kullanılır. n pozitif tamsayısının

faktöriyeli (n! olarak yazılır ve n faktöriyel olarak okunur.) 1 'den n’ e kadar olan tamsayıların

çarpımına eşittir.1' den 5'e kadar olan sayıların faktöriyelini hesaplayan ve çizelge şeklinde

ekrana yazdıran bir program yazınız. 20 faktöriyeli hesaplamada karşınıza çıkacak sorun

nedir?

4.15 Kısım 4.6’daki programı %5, %6, %7, %8, %9 ve %10 faiz oranları için işlemleri tekrar

yapacak şekilde değiştirin.Faiz oranını değiştirmek için for döngüsü kullanın.

4.16 Aşağıdaki desenleri ekrana alt alta yazdıran bir program yazın. Programınızda for döngü

yapısını kullanın. Bütün yıdız karakterleri(\*) tek bir printf ifadesiyle yazdırılabilir. printf

("\*"); gibi. ( Bu, asterikslerin yan yana yazdırılmasını sağlar.) İpucu: Son iki desene her

satırda belli bir sayıda boşluk bırakılması ile başlanmalıdır.

(A) (B) (C) (D)

\* \*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\* \*

\*\* \*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\* \*\*

\*\*\* \*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\* \*\*\*

\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*

\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\*\*

\*\*\*\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*\*\*\*

\*\*\*\*\*\*\*\* \*\*\* \*\*\* \*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\* \*\* \*\* \*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\* \* \* \*\*\*\*\*\*\*\*\*\*

4.17 Şirketlerin kriz sırasında para toplaması oldukça zordur. Bu yüzden bu şirketler kredi

limitlerini düşürerek hesapların çok büyük değerlerde olmasını engellerler. Bu durumda bir

şirket, müşterilerinin kredi limitlerini yarıya indirmiştir. Eğer bir müşterinin 2000$ lık kredi

limiti varsa artık 1000$ olmuştur ya da 5000$lık kredi limiti varsa artık 2500$ olmuştur.

Bu şirketin üç müşterisinin kredi durumunu inceleyen bir program yazınız. her müşteriyle

ilgili olarak aşağıdaki bilgiler verilmiştir.

139

1-Müşterinin hesap numarası

2-Müşterinin önceki kredi limiti

3-Müşterinin şu andaki bakiyesi

Programınız her müşterinin yeni kredi limiti hesaplayıp ekrana yazdırmalı ve hangi

müşterilerin yeni kredi limitlerini aştıklarına karar verip ekrana yazdırmalı.

4.18 İlginç bir bilgisayar uygulaması da grafik ve çubuk çizelgeleridir. Kullanıcıdan 5 sayı

alan ( 1 - 30 arasında) ve her satıra o sayı kadar yıldız karakteri(\*) yazan bir program yazınız.

Örneğin, eğer 7 sayısı girilirse ekrana \*\*\*\*\*\*\* yazdırılmalıdır.

4.19 Bir posta havalesi firması 5 çeşit ürün satmaktadır. Bu ürünlerin perakende fiyatları

aşağıdadır.

Ürün No Perakende fiyatı

1 2.98$

2 4.50

3 9.98

4 4.49

5 6.87

1.Ürün numarası

2.Bir günde satış miktarı

özelliklerini birkaç kez kullanıcıdan alan ve switch yapısını kullanarak her ürünün toplam

perakende fiyatını ve satışı yapılan bütün ürünlerin toplam perakende fiyatını hesaplayan ve

ekrana yazdıran bir program yazınız.

4.20 Aşağıdaki boşlukları 0 veya 1 ile doldurunuz.

Koşul1 Koşul2 Koşul1 && Koşul2

0 0 0

0 0 olmayan 0

0 olmayan 0 \_\_\_\_\_\_\_\_\_

0 olmayan 0 olmayan \_\_\_\_\_\_\_\_\_

Koşul1 Koşul2 Koşul1 || Koşul2

0 0 0

0 0 olmayan 1

0 olmayan 0 \_\_\_\_\_\_\_\_\_

0 olmayan 0 olmayan \_\_\_\_\_\_\_\_\_

140

Koşul1 !Koşul1

0 1

0 olmayan \_\_\_\_\_\_\_

4.21 Şekil 4.2’deki programı, sayac değişkeninin ilk değeri for yapısında değil de, değişkenin

bildirilmesi sırasında verileceği şekilde tekrar yazınız.

4.22 Şekil 4.7’deki programı sınıfın ortalama notunu hesaplayacak şekilde değiştiriniz.

4.23 Şekil 4.6’daki programı birleşik faizi hesaplamak için yalnızca tamsayıları kullanacak

biçimde değiştiriniz. (İpucu:Tüm para değerlerini doların yüzde biri olan sent ile ifade

ediniz.Sonucun dolar ve sent kısmını bölme ve mod alma ile ayırınız.)

4.24 i = 1, j = 2, k = 3 ve m = 2 olduğuna göre, aşağıdaki ifadeler ekrana ne yazdırır.?

a) printf ("%d", i == 1);

b) printf ("%d", j == 3);

c) printf ("%d", i >= 1 && j < 4);

d) printf ("%d", m <= 99 && k < m);

e) printf ("%d", j >= i || k == m);

f) printf ("%d", k + m < j || 3 - j >= k);

g) printf ("%d", !m);

h) printf ("%d", !(j - m));

i) printf ("%d", !(k > m));

j) printf ("%d", !(j > k));

4.25 Onluk, ikilik, sekizlik ve onaltılık sayıların bir tablosunu ekrana yazdıran bir program

yazınız. Bu sayı sistemleri hakkında bilginiz yoksa bu programı yazmandan önce Ek-E 'ye

bakınız.

4.26

π = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 +....

sonsuz serisinden faydalanarak π’yi hesaplayın. her terimden sonra π’ye ne kadar

yaklaşıldığını bir tabloda gösterin. 3.14, 3.141, 3.1415, 3.14159 bulabilmeniz için serinin kaç

terimini kullanmanız gerekir.

4.27 (Pisagor teoremi) Bir dik üçgen, tamamı tamsayılar olan üç kenara sahip olabilir. Bu üç

kenar, iki kenarın kareleri toplamı hipotenüsün karesine eşittir,bağıntısını sağlamalıdır.

Kenar1, Kenar2 ya da hipotenüsü 500’den büyük olmayan tüm dik üçgenleri bulunuz.Üç

yuvalı bir for döngüsünü tüm olasılıkları bulmak için kullanınız.

4.28 Bir şirket, müdürlerine(sabit haftalık ücret alırlar), vardiyalı işçilerine(40 saate kadar

sabit saatlik ücret ve sonrası için saatlik ücretin 1.5 katı alırlar), komisyon işçilerine (sabit

250$ ve haftalık brüt satışların %5.7 sini alırlar) yada parça işçilerine(ürettiği her malzeme

başına sabit ücret alırlar -- her parça işçisi tek bir malzeme üzerinde çalışır--) maaş

vermektedir. Her çalışan tipinin haftalık maaşını hesaplayan bir program yazınız. Kaç çalışan

olduğunu bilmiyorsunuz. Her tipte çalışanın kendi maaş kodu vardır. müdürler 1, vardiyalı

işçiler 2, komisyon işçileri 3 ve parça işçileri 4. Bu maaş kodlarına göre çalışanların maaşları

141

hesabı için, switch yapısı kullanın. switch içerisinde kullanıcının gerekli değerleri girmesini

sağlayın.

4.29 (De Morgan kanunu) Bu ünitede && || ve ! mantık operatörlerini inceledik. De Morgan

kanunu ile bazı mantık operatörlerini daha uygun şekilde ifade edebiliriz. Bu kanuna göre

!(kosul1 && kosul2), (!kosul1 || !kosul2) ye eşittir. Tabi ki !(kosul1 || kosul2) de (!kosul1

&& !kosul2)'ye eşittir. Bu kanunu kullanarak aşağıdaki mantıksal deyimleri yazınız ve bu

eşitlikleri ispatlayan bir program yazınız.

a) !(x < 5) && !(y <= 7)

b) !(a == b) || !(g != 5)

c) !((x <= 8) && (y > 4))

d) !((i > 4) || (j <=6))

4.30 Şekil 4.7’deki programı switch yapısı yerine yuvalı if/else yapısı kullanarak tekrar

yazınız. default durumuna dikkat ediniz. Daha sonra aynı programı yuvalı if/else yapısı

yerine birden fazla if ifadesi kullanarak yazınız. Burada da default durumuna dikkat ediniz

(buradaki default durumu, yuvalı if/else yapısındakinden daha zor olacaktır.) Bu örnek

switch’in uygunluğunu ve herhangi bir switch ifadesinin sadece tek seçimli ifadelerle

yazılabileceğini gösterir.

4.31 Aşağıdaki eşkenar dörtgeni ekrana yazdıran bir program yazınız. printf ifadelerinizle

sadece bir yıldız karakteri ya da sadece bir boşluk yazdırabilirsiniz. Maksimum döngü(yuvalı

for döngüsü) ve minimum printf ifadesi kullanın.

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

4.32 4.31’deki programı kullanıcıya 1 ile 19 arasında tek bir tamsayı girdirecek ve bu tamsayı

kadar satıra eşkenar dörtgeni çizecek şekilde değiştirin.

4.33 Romen rakamları hakkında bilginiz varsa 1-100 arasındaki romen rakamlarının onluk

sistemdeki eşitlerini ekrana yazdıran bir program yazınız.

4.34 1'den 256'ya kadar olan onluk sistemdeki sayıların ikilik, sekizlik ve onaltılık

sistemlerdeki karşılıklarını yazınız. Eğer bu sayı sistemleri hakkında yeterli bilginiz yoksa

Ekler E 'ye bakınız.

4.35 do/while döngüsünü, while döngüsüne dönüştürebilmek için gereken işlemleri anlatınız.

Bir while döngüsünü, do/while döngüsü ile değiştirmeye çalıştığımızda ne gibi bir problemle

142

karşılaşırız. Eğer size bir while döngüsünü, do/while ile değiştirmeniz söylenirse hangi

kontrol yapısını eklemeye ihtiyaç duyarsınız ve bu test yapısını programın sonucunun

değişmediğini anlayacak şekilde nasıl kullanırsınız?

4.36 Kullanıcıdan 1994 ile 1999 arasında bir yıl girdisi yaptıran ve for döngüleri kullanarak

kısa ve düzenli bir takvimi ekrana yazdıran bir program yazınız. Artık yıllara dikkat edin.

4.37 break ve continue ifadeleri yapısızdırlar. break ve continue ifadeleri yapısal ifadelerle

değiştirilebilir. Tabi ki bu pek düzenli olmaz. Bir programdaki bir döngü içersindeki break

ifadesini yapısal bir eşitiyle nasıl değiştirebileceğinizi açıklayınız (İpucu: break, döngünün

bir kısmından çıkmaya yarar. Diğer bir yol ise döngünün devamını bütünüyle durdurarak

çıkmaktır. Döngü devamlılığı testinde ikinci bir test kullanın: break ifadesi için erken çıkış)

Bu tekniği 4.11’de kullanarak bütün break ifadelerini kaldırın.

4.38 Aşağıdaki program parçacığı ne yapar?

for (i = 1; i <= 5; i++) {

for (j = 1; j <= 2; j++) {

for (k = 1; k <= 4; k++)

printf ("\*");

printf ("\n");

}

printf ("\n");

}

4.39 Genel olarak bir döngüdeki continue ifadesini nasıl yapısal bir ifadeyle değiştiririz?

Bulduğunuz tekniği 4.12’de kullanarak continue ifadelerini kaldırın.

4.40 Bir switch yapısındaki break ifadelerini nasıl bir yapısal eşitiyle değiştirirsiniz? Pek

düzenli olmasa da bu tekniği şekil 4.7’de kullanın ve break ifadelerini kaldırın.

143

FONKSİYONLAR

AMAÇLAR

 Fonksiyon adı verilen küçük parçalarla modüler programlar oluşturabilmeyi anlamak.

 C standart kütüphanesinde içinde yer alan genel matematik fonksiyonlarını tanıtmak

 Yeni fonksiyonlar oluşturabilmek

 Fonksiyonlar arasında bilgi aktarımını sağlayan yöntemleri anlamak

 Rasgele sayılar üreterek benzetim ( simulation ) tekniklerini tanıtmak

 Kendi kendini çağırabilen fonksiyonları yazabilmek ve nasıl kullanılacaklarını

anlamak.

BAŞLIKLAR

5.1 GİRİŞ

5.2 C’DE PROGRAM MODÜLLERİ

5.3 MATEMATİK KÜTÜPHANESİNDEKİ FONKSİYONLAR

5.4 FONKSİYONLAR

5.5 FONKSİYON TANIMLARI

5.6 FONKSİYONLARIN İLKHALLERİ ( PROTOTİPLERİ )

5.7 ÖNCÜ ( HEADER ) DOSYALAR

5.8 FONKSİYONLARI ÇAĞIRMAK: DEĞERE GÖRE ÇAĞIRMAK ve

REFERANSA GÖRE ÇAĞIRMAK

5.9 RASTGELE SAYI ÜRETMEK

5.10 ÖRNEK:ŞANS OYUNU

5.11 DEPOLAMA SINIFLARI

5.12 FAALİYET ALANI KURALLARI

5.13 YİNELEME

5.14 YİNELEMELERİ KULLANAN ÖRNEK : FIBONACCI SERİLERİ

YİNELEME ve TEKRAR

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Performans İpuçları\*

Taşınırlık İpuçları\*Yazılım Mühendisliği Gözlemleri\*Çözümlü Alıştırmalar\* Çözümler\*

Alıştırmalar

5.1 GİRİŞ

Gerçek problemleri çözen çoğu bilgisayar programları, ilk ünitelerde yazdıklarımızdan çok

daha geniştir. Tecrübeler bu tür geniş programları yazmanın en iyi yolunun, küçük parçaları

ya da her biri orijinal programdan daha kolay kullanılabilecek modülleri (daha önceden

hazırlanmış program parçacıkları) birleştirmek olduğunu göstermiştir. Bu tekniğe, böl ve

zaptet (divide&conquer) denir. Bu ünite C dilinin , geniş programların tasarım, uygulama,

işlem ve kontrol aşamalarını kolaylaştıran özelliklerini açıklamaktadır.

144

5.2 C’DE PROGRAM MODÜLLERİ

C’de modüllere fonksiyon denir. C programları, genellikle programcının yazacağı yeni

fonksiyonlarla, daha önceden C standart kütüphanesi içinde tanımlanmış fonksiyonların

birleştirilmesiyle yazılır. Bu ünitede, her iki türde de fonksiyonları açıklayacağız. C standart

kütüphaneleri genel matematik işlemleri, karakter işlemleri, giriş/çıkış işlemleri ve diğer

birçok önemli işlemi yerine getiren birçok fonksiyonu içermektedir. Bu, programcının işini

kolaylaştıracaktır çünkü bu fonksiyonlar programcının ihtiyaç duyacağı bir çok yeteneği

sağlamaktadır.

İyi Programlama Alıştırmaları 5.1

ANSI C standart kütüphanesi içindeki fonksiyonları dikkatlice inceleyin.

Yazılım mühendisliği Gözlemleri 5.1

Tekerleği yeniden icat etmekten kaçının. Mümkün olduğunda yeni fonksiyonlar yazmak yerine ANSI C

standart kütüphanesi içindeki fonksiyonları kullanın. Bu, program geliştirme zamanını azaltacaktır.

Taşınırlık İpuçları 5.1

ANSI C standart kütüphanesi içindeki fonksiyonları kullanmak daha taşınır programlar yazmamıza

yardımcı olur.

Aslında teknik olarak, standart kütüphane fonksiyonları C dilinin bir kısmı olmasa da ANSI C

sistemlerinde değişmez olarak yer alırlar. printf ,scanf ,pow fonksiyonları gibi daha önceki

ünitelerde kullandığımız fonksiyonlar standart kütüphane fonksiyonlarıdır.

Bir programcı, bir programın çoğu kısmında kullanılabilecek belirli işlemleri

tanımlamak için fonksiyonlar yazabilir. Bunlara, programcı tarafından tanımlanmış

fonksiyonlar denir. Fonksiyonu tanımlayan asıl ifadeler bir kez yazılır ve bu ifadeler diğer

fonksiyonlardan gizlenirler.

Fonksiyonlar, fonksiyon çağrıları sayesinde çağrılırlar. Fonksiyon çağırmak demek

fonksiyonu kullanmamız gerektiği anda onu programa dahil etmek demektir. Fonksiyon

çağrıları, fonksiyonun ismini ve fonksiyonun görevini yerine getirebilmesi için gerekli olan

bilgileri (argümanlar olarak) içerir. Anlattıklarımıza uygun bir benzerlik, yönetimin hiyerarşi

düzenidir. Patron ( çağırıcı fonksiyon ya da çağırıcı ), işçiye ( çağrılan fonksiyon ) bir görev

verir ve işin sonunda da kendisine rapor vermesini söyler. Örneğin, ekrana bir bilgiyi

yazdırmak isteyen fonksiyon, printf işçi fonksiyonunu çağırır ve bu görevi yerine getirmesini

söyler. printf fonksiyonu, bilgiyi ekrana yazdırdıktan kendini çağıran fonksiyona geri döner

ve rapor verir. Patron fonksiyon, işçi fonksiyonun görevi nasıl yerine getirdiğini bilmez. İşçi,

başka işçi fonksiyonlar çağırabilir ve patronun bundan haberi olmaz. İleride uygulama

detaylarının gizlenmesinin yazılım mühendisliğini nasıl iyileştirdiğini göreceğiz. Şekil 5.1,

main fonksiyonunun çeşitli işçi fonksiyonlarla hiyerarşik bir düzende nasıl haberleştiğini

göstermektedir. işçi1’in işçi4 ve işçi5’e patron fonksiyon olarak davrandığına dikkat ediniz.

Fonksiyonlar arasındaki ilişki, burada gösterilen hiyerarşik düzenden farklı olabilir.

main

İşçi 1 İşçi 2 İşçi 3

İşçi 4 İşçi 5

145

Şekil 5.1 Patron fonksiyon/işçi fonksiyon yapısının hiyerarşik düzeni.

5.3 MATEMATİK KÜTÜPHANESİNDEKİ FONKSİYONLAR

Matematik kütüphane fonksiyonları, programcının bazı genel matematik işlemlerini

yapmasını sağlatır. Burada, fonksiyon kavramını tanıtmak için çeşitli matematik

fonksiyonlarını kullanacağız. İleride, C standart kütüphanesindeki diğer fonksiyonlardan da

bahsedeceğiz.

Fonksiyonlar bir programda, fonksiyonun ismi ve ismin sağında parantez içinde argüman

(bağımsız değişken ) ya da virgülle ayrılmış argüman listesi yazarak kullanılır. Örneğin,

programcı 900.0 sayısının karekökünü bulmak ve yazdırmak istiyorsa şu kodu yazabilir:

printf ( “ % . 2f ”, sqrt ( 900.0 ) );

Bu ifade çalıştırıldığında, sqrt matematik kütüphanesi fonksiyonu parantezin içindeki 900.0

sayısının karekökünü bulmak için çağrılmıştır. Burada, 900.0 sayısı sqrt fonksiyonunun

argümanıdır. Az önceki ifade, 30.00 sayısını yazdıracaktır. sqrt fonksiyonu, double tipte

argüman kullanır ve double tipinde sonuçlar döndürür. Matematik kütüphane

fonksiyonlarının hepsi double veri tipinde veri döndürürler.

İyi Programlama Alıştırmaları 5.2

Bir programda matematik kütüphanesi fonksiyonları kullanıyorsak, programımızın başına #include

<math.h> önişlemci komutunu yazarak matematik öncü ( header ) dosyasını programımıza katmalıyız.

Genel Programlama Hataları 5.1

Matematik kütüphane fonksiyonlarını kullanırken matematik öncü dosyasını eklemeyi unutmak garip

sonuçlara yol açabilir.

Fonksiyon argümanları sabit sayılar, değişkenler ya da deyimler olabilir. Eğer, c1=13.0

d=3.0 ve f=4.0 olsaydı,

printf ( “%.2f”,sqrt ( c1+d\*f ) );

ifadesi, 13.0 + 3.0 \* 4.0 = 25.0 sayısının karekökünü hesaplayıp yazdıracaktı. Hesaplama

sonucu 5.00 olacaktır.

Şekil 5.2’de, matematik kütüphane fonksiyonları özetlenmiştir.x ve y, double veri tipindedir.

FONKSİYON TANIM ÖRNEK

sqrt(x) x’in karekökü sqrt(900.0) 30.0 değerini verir

146

sqrt(9.0) 3.0 değerini verir

exp(x) e

x

üssel fonksiyon exp(1.0) 2.718252 değerini verir

exp(2.0) 7.389056 değerini verir

log(x) x’in e tabanına göre

logaritması

log (2.718252) 1.0 değerini verir

log (7.389056) 2.0 değerini verir

log10(x) x’in 10 tababına göre

logaritması

log10(1.0) 0.0 değerini verir

log10(10.0) 1.0 değerini verir

log10(100.0) 2.0 değerini verir

fabs(x) x’in mutlak değeri x>0 ise x değerini verir

x=0 ise 0.0 değerini verir

x<0 ise –x değerini verir

ceil(x) x’i kendinden büyük ilk

tamsayıya yuvarlar

ceil(9.2) 10.0 değerini verir

ceil(-9.8) –9.0 değerini verir

floor(x) x’i kendinden küçük ilk

tamsayıya yuvarlar

floor(9.2) 9.0 değerini verir

floor(-9.8) –10.0 değerini verir

pow(x) x

y

x üzeri y pow(2,7) 128.0 değerini verir

pow(9,.5) 3 değerini verir

fmod(x,y) x/y işleminin kalanını

bulur

fmod(13.657,2.333) 1.992 değerini verir

sin(x) x’in sinüsünü hesaplar(x

radyan)

sin(0.0) 0.0 değerini verir

cos(x) x’in cosinüsünü hesaplar

(x radyan)

cos(0.0) 1.0 değerini verir

tan(x) x’in tanjantını hesaplar (x

radyan)

tan(0.0) 0.0 değerini verir

Şekil 5.2 Sıklıkla kullanılan matematik kütüphane fonksiyonları.

5.4 FONKSİYONLAR

Fonksiyonlar, programcının programını modüler hale getirmesini sağlar. Bir fonksiyon içinde

bildirilmiş tüm değişkenler, yerel değişkenlerdir ve sadece bildirildikleri fonksiyon içinde

bilinirler. Çoğu fonksiyon, parametre listelerine sahiptir. Bu parametreler, fonksiyonların

birbirleri arasındaki haberleşmelerini sağlar. Bir fonksiyonun parametreleri de yerel

değişkenlerdir.

Yazılım Mühendisliği Gözlemleri 5.2

Birden fazla fonksiyon kullanılan programlarda, main fonksiyonu programın esas görevini yerine

getiren fonksiyonların çağırıcısı olarak kullanılmalıdır.

Bir programı fonksiyonlar ile yazmanın bir çok nedeni vardır. Böl ve zaptet ( divide and

conquer) yaklaşımı, program geliştirmeyi kolaylaştırır. Başka bir sebep ise yazılımın yeniden

kullanılmasıdır. Bu sayede, önceden yazılmış fonksiyonlar bloklar halinde birbirleri ardına

yerleştirilerek yeni programlar yazılabilir. Yazılımın yeniden kullanılabilmesi, nesneye

yönelik programlamanın temel etmenlerindendir. Fonksiyona iyi isim verilir ve fonksiyon iyi

tanımlanırsa, programlar belirli işleri yapan standartlaştırılmış fonksiyonları kullanarak

yazılabilir. Bu teknik, özetleme ( abstraction ) olarak bilinir. Özetleme tekniğini, printf, scanf

ve pow gibi standart kütüphane fonksiyonlarını kullanarak yazdığımız programlarda

uygulamıştık. Üçüncü sebep ise programda kodları tekrar etmekten kurtulmaktır. Kodları

147

fonksiyon haline getirerek paketlemek, kodların program içinde birçok noktada yalnızca

fonksiyonu çağırarak kullanılmasını sağlatır.

Yazılım Mühendisliği Gözlemleri 5.3

Her fonksiyon, iyi olarak tanımlanmış tek bir işi yapacak şekilde sınırlandırılmalıdır ve fonksiyon ismi

fonksiyonun görevini etkili bir biçimde açıklamalıdır. Bu, özetlemeyi ve yazılımın yeniden

kullanılabilirliğini sağlatır.

Yazılım Mühendisliği Gözlemleri 5.4

Eğer fonksiyonun görevini açıklayacak etkili bir isim bulamıyorsanız, muhtemelen yazdığınız

fonksiyon birden fazla görevi yerine getirmeye çalışmaktadır. Bu tarzda fonksiyonları daha küçük

fonksiyonlara bölmek en iyi yoldur.

5.5 FONKSİYON TANIMLARI

Şu ana kadar yazdığımız tüm programlar, standart kütüphane fonksiyonlarını işlerini

yaptırmak için çağıran main adında bir fonksiyon içeriyordu. Şimdi ise programcıların kendi

fonksiyonlarını nasıl yazdıklarını inceleyeceğiz.

Birden ona kadar olan tam sayıların karesini alan kare fonksiyonunu kullanan bir program

inceleyelim. ( Şekil 5.3 )

İyi Programlama Alıştırmaları 5.3

Programın okunurluğunu arttırmak ve fonksiyonları ayırmak için fonksiyon tanımlarından önce bir

satır boşluk bırakmak.

1 /\* Şekil 5.3:fig05\_03.c

2 Programcı tarafından tanımlanmış kare fonksiyonu \*/

3 #include <stdio.h>

4

5 int kare( int ); /\* fonksiyonun ilk hali(prototipi) \*/

6

7 int main( )

8 {

9 int x;

10

11 for ( x = 1; x <= 10; x++ )

12 printf( "%d ", kare( x ) );

13

14 printf( "\n" );

15

16 return 0;

17 }

18

19 /\*Fonksiyon tanımı\*/

20 int kare( int y )

21 {

22 return y \* y;

23 }

148

1 4 9 16 25 36 49 64 81 100

Şekil 5.3 Programcı tarafından tanımlanmış fonksiyon kullanmak.

kare fonksiyonu, main altındaki printf ifadesi ( 12.satır ) içinden çağrılmıştır.

printf ( “ %d ” , kare ( x ) );

kare fonksiyonu, x değerinin kopyasını y parametresi sayesinde alır. Daha sonra y \* y

hesabını yapar. Sonuç, main içindeki printf fonksiyonuna döndürülür ve printf sonucu

yazdırır. Bu süreç, for döngü yapısı sayesinde 10 kez tekrarlanır.

kare fonksiyonunun tanımı, kare ’in bir y tamsayı parametresi beklediğini gösterir.

Fonksiyon isminden önceki int anahtar kelimesi, bize kare’ in sonucunun yine bir tamsayı

olarak döndürüleceğini gösterir. kare içindeki return ifadesi, hesaplamanın sonucunu çağırıcı

fonksiyona döndürür.

5. satırdaki

int kare ( int );

fonksiyon prototipidir. Parantezin içindeki int , derleyiciye kare fonksiyonunun çağırıcı

fonksiyondan bir tamsayı almayı beklediğini bildirir. Fonksiyon isminin solundaki int ise

derleyiciye, kare fonksiyonunun kendini çağıran fonksiyona bir tamsayı sonucu

döndüreceğini bildirir. Derleyici, kare fonksiyonu çağrıldığında fonksiyonun ilk hali

(prototipi) ile karşılaştırma yaparak, çağrının doğru tipte geri dönüş değerine sahip oluşunu,

doğru sayıda argüman ve doğru argüman tipleri kullanmasını ve argümanların doğru sırada

oluşlarını kontrol eder. Fonksiyonların ilk halleri(prototipleri), kısım 5.6’da detaylı olarak

anlatılacaktır.

Bir fonksiyon tanımının biçimi şu şekildedir:

geri dönüş tipi fonksiyonun ismi ( parametre listesi )

{

bildirimler

ifadeler

}

Fonksiyon ismi, geçerli herhangi bir tanıtıcı olabilir. Geri dönüş tipi, çağırıcı fonksiyona

döndürülen sonucun veri tipini gösterir. void tipinde geri dönüş değeri, fonksiyonun herhangi

bir değer geri döndürmeyeceğini gösterir. Belirlenmemiş geri dönüş tipi, derleyici tarafından

her zaman int tipinde algılanır.

Genel Programlama Hataları 5.2

Fonksiyon tanımlamalarında geri dönüş değerini unutmak eğer fonksiyonun ilk hali(prototipi) int

tipinden başka bir geri dönüş tipi ile belirtilmişse yazım hatası oluşturur.

Genel Programlama Hataları 5.3

Bir değer ile dönmesi beklenen bir fonksiyonun, geri dönüş değerinin belirtilmemesi beklenmeyen

hatalara yol açabilir. ANSI standardı, bu ihmalin sonuçlarını belirlememiştir.

149

Genel Programlama Hataları 5.4

Geri dönüş tipi void olarak bildirilmiş bir fonksiyonun, bir değer geri döndürmesi bir yazım hatasıdır.

İyi Programlama Alıştırmaları 5.4

Geri dönüş tipi ihmal edildiğinde, derleyici geri dönüş tipini int olarak belirlese de her zaman geri

dönüş tipini belirleyiniz. Ancak , main fonksiyonunun geri dönüş tipi normal olarak ihmal edilebilir.

Parametre listesi, fonksiyon çağrıldığında fonksiyonun alacağı parametrelerin bildirimlerini

içeren, virgüllerle ayrılmış bir listedir. Eğer bir fonksiyon herhangi bir değer almıyorsa

parametre listesi void olur. Her parametre için ( eğer tipi int değilse) parametre tipi ayrı ayrı

belirtilmelidir. Eğer tip listelenmezse int olarak algılanacaktır.

Genel programlama hataları 5.5

Aynı tipte fonksiyon parametrelerini, double x, double y yerine double x , y olarak bildirmek.

double x, y biçiminde parametre bildirmek y parametresinin tipinin int olmasına sebep olur. Çünkü

belirtilmeyen parametre tipi otomatik olarak int tipinde varsayılır.

Genel programlama hataları 5.6

Parametre listesini yazdığımız parantezlerin dışına noktalı virgül koymak

yazım hatasıdır.

Genel programlama hataları 5.7

Bir fonksiyon parametresini, daha sonradan fonksiyon içinde yerel bir değişken olarak kullanmak bir

yazım hatasıdır.

İyi Programlama Alıştırmaları 5.5

Parametre listesindeki tüm parametrelerin tipini , belirtilmeyenler otomatik olarak int tipinde

kullanılacak olsa da mutlaka belirtiniz.

İyi Programlama Alıştırmaları 5.6

Yanlış olmasa da fonksiyona aktarılan argümanlarla bu argümanların yerine kullanılacak

parametrelerin aynı isimde olmamasına özen gösteriniz. Bu, belirsizlikten kurtulmamızı sağlar.

Parantezlerin içinde yer alan bildirimler ve ifadeler, fonksiyonun gövdesini oluşturur.

Fonksiyon gövdesi aynı zamanda blok olarak da adlandırılır. Bir blok en basit anlamda,

bildirimler içerebilen birleşik bir ifadedir. Değişkenler herhangi bir blok içinde bildirilebilir

ve bloklar yuvalanabilirler. Bir fonksiyonun tanımı hiçbir koşul altında başka bir fonksiyonun

içinde yapılamaz.

Genel Programlama Hataları 5.8

Bir fonksiyon içinde başka bir fonksiyon tanımlamak yazım hatasıdır.

İyi Programlama Alıştırmaları 5.7

Anlamlı fonksiyon isimleri ve anlamlı parametre isimleri kullanmak programları daha okunur yapar

ve yorumların çok fazla kullanılmasını engeller.

150

Yazılım Mühendisliği Gözlemleri 5.5

Bir fonksiyon genellikle bir sayfadan daha uzun olmamalıdır. Hatta en iyisi yarım sayfadan uzun

olmamalıdır. Küçük fonksiyonlar, yazılımın yeniden kullanılabilmesini sağlatır.

Yazılım Mühendisliği Gözlemleri 5.6

Programlar, küçük fonksiyonların bir araya getirilmesiyle yazılmalıdır.Bu, programların daha kolay

yazılması,değiştirilmesi ve hatalarının giderilmesini sağlar.

Yazılım Mühendisliği Gözlemleri 5.7

Çok fazla sayıda parametreye ihtiyaç duyan fonksiyonlar, birden fazla görevi yerine getiriyor olabilir.

Böyle fonksiyonları, ayrı görevleri gerçekleştiren daha küçük fonksiyonlara bölmek gerekir.

Fonksiyonun başlığı mümkünse bir satıra sığmalıdır.

Yazılım Mühendisliği Gözlemleri 5.8

Fonksiyonun ilk hali(prototipi) , fonksiyonun başlığı ve fonksiyon çağrısı , argüman ve parametre

sayısı, tipi ve sırasıyla, geri dönüş değerinin tipi bakımından uyumlu olmalıdır.

Bir fonksiyonun çağrıldığı yere geri dönmesini kontrol etmek için 3 yol vardır :

1-) Eğer bir fonksiyon bir sonuç ile geri dönmeyecekse, kontrol fonksiyonun en son

parantezine ulaşıldığında ya da

2-) return;

ifadesinin çalıştırılmasıyla döndürülür.

3-) Eğer fonksiyon bir sonuç ile geri dönecekse

return deyim;

ifadesi, deyimin değerini çağırıcıya döndürür.

İkinci örneğimiz, programcı tarafından tanımlanmış maksimum adlı fonksiyonu kullanarak 3

tamsayının en büyüğünü bulmakta ve geri döndürmektedir. Üç tamsayıda scanf ile alınsın.

Daha sonra tamsayılar, maksimum adını verdiğimiz fonksiyona geçirilsin. En büyük sayı,

maksimum fonksiyonun return ifadesi ile main fonksiyonuna geri döndürülsün. Geri

döndürülen değer printf ile yazdırılsın.

1 /\*Şekil 5.4:fig05\_04.c

2 Üç tamsayının en büyüğünü bulmak\*/

3 #include <stdio.h>

4

5 int maksimum( int, int, int ); /\*fonksiyon prototipi\*/

6

7 int main( )

8 {

9 int a, b, c;

10

11 printf( "3 tamsayı giriniz: " );

12 scanf( "%d%d%d", &a, &b, &c );

13 printf( "Maksimum : %d dir.\n", maksimum( a, b, c ) );

14

15 return 0;

16 }

151

17

18 /\*maksimum fonksiyonunun tanımı\*/

19 int maksimum( int x, int y, int z )

20 {

21 int maks = x;

22

23 if ( y > maks )

24 maks = y;

25

26 if ( z > maks )

27 maks = z;

28

29 return maks;

30 }

3 tamsayı giriniz: 22 85 17

Maksimum : 85 dir.

3 tamsayı giriniz: 85 22 17

Maksimum : 85 dir.

3 tamsayı giriniz: 22 17 85

Maksimum : 85 dir.

Şekil 5.4 Programcı tarafından tanımlanmış maksimum fonksiyonu.

5.6 FONKSİYON PROTOTİPLERİ

ANSI C’nin en önemli özelliklerinden biriside fonksiyonların ilk halleridir.(prototipleridir)

Bu özellik, ANSI C komitesi tarafından C++ geliştiricilerinden alınmıştır. Bir fonksiyon

prototipi, derleyiciye fonksiyon tarafından döndürülen verinin tipini, fonksiyonun almayı

beklediği parametre sayısını, parametrelerin tiplerini ve parametrelerin sırasını bildirir.

Derleyici, fonksiyonların ilk hallerini ( prototiplerini ) fonksiyon çağrılarını onaylamakta

kullanır. C’nin daha önceki sürümleri bu tarzda bir kontrol yapmazdı. Bu sebepten, derleyici

hataları belirleyemeden fonksiyonlar çağrılabiliyordu. Bu tür fonksiyon çağrıları, çalışma

zamanlı ölümcül hatalara ya da güç fark edilen mantık hatalarına sebep oluyordu. ANSI C ,bu

eksikliği gidermiştir.

152

İyi Programlama Alıştırmaları 5.8

C’nin kontrol yeteneklerinden faydalanabilmek için tüm fonksiyonların ilk hallerini (prototiplerini)

programa dahil etmeliyiz. Uygun kütüphanelerdeki öncü dosyalardan, standart kütüphane

fonksiyonlarının ilk hallerini ( prototiplerini ) elde etmek için #include önişlemci komutlarını kullanın.

Ayrıca siz ve/ve ya arkadaşlarınızın kullandığı fonksiyon prototiplerini içeren öncü dosyaları elde

etmek için de #include kullanın.

Şekil 5.4 ‘deki maksimum fonksiyonunun prototipi

int maksimum ( int, int, int );

şeklindedir.

Bu fonksiyon prototipi, maksimum fonksiyonunun int tipinde 3 argüman alacağını ve sonuç

olarak int tipinde bir sonuç döndüreceğini belirtir. Dikkat edilirse, fonksiyon prototipiyle

maksimum fonksiyonunun tanımının ilk satırı, parametrelerin isimlerinin (x, y ve z)

bulunmayışı haricinde aynıdır.

İyi Programlama Alıştırmaları 5.9

Parametre isimleri, belgeleme amaçlı olarak fonksiyon prototipleri içinde yazılabilir. Derleyici, bu

isimleri ihmal eder.

Genel Programlama Hataları 5.9

Fonksiyon prototipinin sonuna noktalı virgül koymamak bir yazım hatasıdır.

Fonksiyon prototipiyle eşleşmeyen bir fonksiyon çağrısı, yazım hatası oluşmasına sebep olur.

Ayrıca eğer fonksiyon tanımıyla fonksiyon prototipi birbirine uymazsa, başka bir hata daha

oluşturulur. Örneğin, Şekil 5.4’ deki örneğimizde fonksiyon prototipini

void maksimum (int,int,int) ;

şeklinde yazmış olsaydık, derleyici fonksiyon başlığındaki geri dönüş tipi olan int ile

fonksiyon prototipindeki geri dönüş değeri void birbirinden farklı olduğundan hata üretecekti.

Fonksiyon prototiplerinin önemli bir özelliği de argümanların zorlanmasıdır yani

argümanların uygun tipte olmasının sağlanmasıdır. Örneğin, sqrt matematik fonksiyonunun

math.h dosyasındaki prototipi double argüman ile belirlenmiş olsa da tamsayı argümanıyla

da çağrılabilir ve fonksiyon hala doğru olarak çalışır. Örneğin,

printf ( “%.3f\n ”, sqrt ( 4 ) );

ifadesi, sqrt ( 4 ) değerini doğru olarak hesaplayacak ve 2.000 değerini yazdıracaktır.

Fonksiyon prototipi, derleyicinin tamsayı olan 4 değerini sqrt fonksiyonuna geçirmesinden

önce bu değeri double değer olan 4.0’a çevirmesini sağlar. Genel olarak, fonksiyon prototipi

içindeki parametre listesiyle uyuşmayan argüman değerleri, fonksiyon çağrılmadan uygun

tipe dönüştürülür.

Bu dönüşümler eğer C’nin dönüştürme kurallarına uyulmazsa, yanlış sonuçlara sebep

olabilir. Bu kurallar, bir tipten öteki tipe herhangi bir hata olmadan çevrimlerin nasıl

yapılacağını belirtir. Mesela, yukarıdaki sqrt örneğimizde int, değerini kaybetmeden

otomatik olarak double tipe dönüştürülmüştür. Ancak, double tipte bir veri int tipine

153

çevrilseydi, o zaman double tipindeki verinin ondalıklı kısmı kaybolacaktı. Büyük tamsayı

tipleri küçük tamsayı tiplerine dönüştürülürken (örneğin, long short’a çevrilirse) yine

kayıplar olur.

Dönüştürme kuralları, otomatik olarak 2 ya da daha fazla veri tipi içeren ifadelere uygulanır.

Bu tarzda ifadelerde her değerin tipi, otomatik olarak ifadedeki en yüksek veri tipine

dönüştürülür. (aslında her değerin geçici bir kopyası oluşturulur ve bu kopya ifadede

kullanılır-orijinal değerler değişmeden kalır) Şekil 5.5‘de, veri tiplerini en yüksek tipten en

düşük tipe doğru sıraladık ve her tipin printf ve scanf dönüşüm belirteçleriyle kullanımlarını

gösterdik .

VERİ TİPLERİ printf ile kullanımları scanf ile kullanımları

long double %Lf %Lf

double %f %lf

float %f %f

unsigned long int %lu %lu

long int %ld %ld

unsigned int %u %u

int %d %d

short %hd %hd

char %c %c

Şekil 5.5 Veri tipleri için dönüşüm hiyerarşisi

Değerleri düşük tiplere çevirmek genelde yanlış değerler hesaplanmasına sebep olur. Bu

sebepten, bir değeri daha düşük bir tipe dönüştürmek için değer özel olarak düşük tipte bir

değişkene atanır ya da dönüşüm operatörleri kullanılır. Fonksiyon argüman değerleri,

fonksiyon prototiplerindeki parametre tiplerine, o tipte değişkenlere atama yapılıyormuş gibi

dönüştürülür. Eğer tamsayı parametresi kullanan kare fonksiyonu ( Şekil 5.3 ), ondalıklı tipte

( float ) argümanlarla çağrılsaydı, argüman int tipine (daha düşük bir tipe) dönüştürülecekti

ve square fonksiyonu yanlış bir değer döndürecekti. Mesela square ( 4.5 ), 20.25 yerine 16

değerini verecekti.

Genel Programlama Hataları 5.10

Dönüşüm hiyerarşisinde, yüksek tipte bir veri tipi daha düşük bir veri tipine dönüştürülürse verinin

değeri değişebilir.

Eğer bir fonksiyon prototipi programın içerisinde yer almazsa, derleyici fonksiyonla ilk kez

karşılaştığında o fonksiyonun için kendisine göre bir prototip oluşturur. Fonksiyonun ilk

göründüğü an, fonksiyon tanımı ya da fonksiyon çağrısı olabilir. Derleyici, otomatik olarak

fonksiyonun geri dönüş değeri tipini int olarak kabul eder ve argümanlar için herhangi bir

işlem yapmaz. Bu sebepten, fonksiyona geçirilen argümanlar hatalıysa derleyici bunları tespit

edemez..

154

Genel Programlama Hataları 5.11

Fonksiyon prototipinin unutulması eğer fonksiyonun geri dönüş tipi int değilse ve fonksiyon tanımı

fonksiyon çağrısından daha sonra bulunmuyorsa, yazım hatalarına sebep olur. Aksi takdirde,

fonksiyon prototipini unutmak çalışma zamanlı ya da beklenmeyen hatalara yol açabilir.

Yazılım Mühendisliği Gözlemleri 5.9

Herhangi bir fonksiyon tanımı dışına yerleştirilmiş fonksiyon prototipi, dosyada fonksiyon prototipinin

yazıldığı yerden itibaren fonksiyonun tüm çağrılarında geçerli olur. Eğer fonksiyon prototipi, bir

fonksiyon tanımının içinde yer alırsa sadece o fonksiyon içinden yapılan çağrılara uygulanır.

5.7 ÖNCÜ ( HEADER ) DOSYALAR

Her standart kütüphane, o kütüphanedeki her fonksiyonun prototiplerinin yer aldığı ve bu

fonksiyonlar tarafından kullanılabilecek çeşitli veri tipleriyle, bazı sabitlerin bulunduğu bir

öncü dosyaya sahiptir. Şekil 5.6’da, programlara dahil edilebilecek standart kütüphane öncü

dosyaları alfabetik bir sırada listelenmiştir. Şekil 5.6’da, açıklamalarda sıklıkla kullanılan

makro terimi, 13. ünitede daha ayrıntılı bir biçimde tartışılmıştır.

Standart kütüphane Öncü Dosyası AÇIKLAMA

<assert.h> Programın hatalarının ayıklanmasında yardımcı olması için eklenen teşhislerin

makroları ve bilgilerini içerir.

<ctype.h> karakterleri belli özelliklere göre test eden fonksiyonların ve küçük harfi büyük

harfe çeviren fonksiyonların (terside geçerlidir) prototiplerini tutar

<errno.h> hata mesajlarını iletmek için gereken makroları tanımlar

<float.h> sistemin ondalıklı sayılar için limitlerini tutar

<limits.h> sistemin integral limitlerini tutar

<locale.h> Programın çalıştırıldığı yerdeki yerel bilgilere göre değiştirilebilmesini

sağlayan fonksiyon prototiplerini ve bunların ihtiyaç duyabileceği bilgileri

içerir. Yerel gösterimler bilgisayar sisteminin, tarih, zaman, para birimleri ve

dünya üzerindeki büyük sayılar için değişik gösterim biçimlerini doğru bir

şekilde kullanmasını sağlatır.

<math.h> matematik kütüphane fonksiyonlarının prototiplerini tutar.

<setjmp.h> fonksiyon çağrıları ve geri dönüşleri arasındaki geçişlere izin veren

fonksiyonların prototiplerini tutar

<signal.h> programın çalıştırılması esnasında oluşabilecek çeşitli durumları gerçekleştiren

makroları ve fonksiyon prototiplerini tutar

<stdarg.h> sayısı ve tipleri belli olmayan argüman listesine sahip fonksiyonların

çalışmasını sağlayan makroları tutar

<stddef.h> C’de belli hesaplamaları yaptığımız tiplerinin genel tanımlarını tutar

<stdio.h> standart giriş/çıkış kütüphane fonksiyonlarının prototiplerini ve bunlar

tarafından kullanılan bilgileri tutar

<stdlib.h> sayıları yazılara, yazıları sayılara çeviren, rastgele sayılar üreten, hafıza

ayrılmasını sağlayan fonksiyonların prototiplerini tutar

<string.h> string işlemlerini yapan fonksiyonların prototiplerini tutar

<time.h> zamanla ve tarihle ilgili işlemler yapan fonksiyonların prototiplerini tutar.

155

Şekil 5.6 Bazı standart kütüphane öncü dosyaları

Programcının kendiside öncü dosyalar oluşturabilir. Programcı tarafından tanımlanan öncü

dosyalar da .h uzantısıyla bitmelidir. Programcı tarafından oluşturulan öncü dosyalar,

programlara #include önişlemci komutuyla dahil edilir. Örneğin, square.h gibi bir öncü

dosyayı programa

#include “square.h”

şeklinde dahil ederdik ve bu satırı programımızın en başına yazardık. Kısım 13.2, öncü

dosyalar eklemekle ilgili daha detaylı bilgi içermektedir.

5.8 FONKSİYONLARI ÇAĞIRMAK:DEĞERE GÖRE ve

REFERANSA GÖRE ÇAĞIRMAK

Fonksiyonları çağırmak için çoğu programlama dilinde iki yöntem kullanılır: değere göre

çağırma ve referansa göre çağırma. Argümanlar değere göre çağırma ile geçirilirse,

argümanın değerinin bir kopyası oluşturulur ve çağırılan fonksiyona geçirilir. Oluşturulan

kopyadaki değişiklikler, çağırıcıdaki orijinal değişkenin değerini etkilemez. Bir argüman

referansa göre çağrıldığında ise çağırıcı, çağrılan fonksiyonun değişkenin orijinal değerini

ayarlamasına izin verir.

Değere göre çağırma, çağrılan fonksiyonun çağırıcının orijinal değerini değiştirmeyeceği

durumlarda kullanılmalıdır. Bu, yanlışlıkla kaynaklanabilecek ve doğru ve güvenilir yazılım

sistemleri geliştirilmesini önemli oranda etkileyecek sorunları engeller. Referansa göre

çağırma, orijinal değeri değiştirmesi gereken güvenilir fonksiyonlarla birlikte kullanılmalıdır.

C’de tüm çağırmalar değere göre yapılır. 7. ünitede göreceğimiz gibi, adres operatörlerini

kullanarak referansla çağırmada yapılabilir. 6.ünitede, dizilerin referansa göre çağırma ile

otomatik olarak geçirilebileceklerini göreceğiz. 7. üniteye kadar beklediğinizde bunların ne

anlama geldiğini daha iyi anlayacaksınız. Şimdilik sadece, değere göre çağırma üzerine

yoğunlaşacağız.

5.9 RASTGELE SAYILAR ÜRETME

Şimdide, kısaca popüler programlama uygulamalarından olan eğlenceli bir yöne bakalım;

oyunlar ve simülasyonlar yazacağız. Bu kısımda ve gelecek kısımda yapısal programlama ile

birden çok fonksiyonu içeren bir oyun programı geliştireceğiz. Bu program, şu ana kadar

kullandığımız kontrol yapılarının çoğunu kullanmaktadır.

Oyun oynanan mekanlarda, en zor oyunlardan en kolay oyunlara kadar oyuncuları

heyecanlandıran bir faktör vardır. Bu, şans faktörüdür. Oyuncuların ceplerindeki bir miktar

parayı bir servete dönüştürme ihtimalleri bulunmaktadır. Şans faktörü, bilgisayar

uygulamalarına C standart kütüphanesinde bulunan rand fonksiyonu sayesinde uygulanabilir.

Şimdi

i = rand( );

ifadesine bakalım. rand fonksiyonu 0 ile RAND\_MAX (<stdlib.h> öncü dosyasında tanımlı

bir sembolik sabit) değeri arasında bir tamsayı yaratır. ANSI standartlarına göre

RAND\_MAX, iki byte (16 bit) tamsayıların alabileceği en büyük değer olan 32767’den

156

küçük olamaz. Bu kısımdaki programlar, RAND\_MAX değeri en çok 32767 olan bir sistem

için yazılmıştır.

Eğer rand düzgün olarak çalışarak tamsayılar oluşturursa, 0 ile RAND\_MAX arasındaki tüm

sayıların üretilme şansı (ihtimali) rand her çağrıldığında aynı olacaktır.

rand tarafından üretilen sayıların dizisi, genellikle bir uygulamada ihtiyaç duyulanlardan

farklıdır. Örneğin, yazı tura oyununu gerçekleştiren bir uygulamada yazıyı belirtmek için sıfır,

turayı belirtmek için bir kullanmak yeterli olur. Zar atma programı, 6 yüzlü bir zar için birden

altıya kadar sayılara ihtiyaç duyacaktır.

rand fonksiyonunu daha iyi anlamak için, 6 yüzlü bir zarın 20 kez atılışını gerçekleyen bir

program geliştirelim ve her atışta gelen sayıyı yazdıralım. rand fonksiyonunun prototipi

<stdlib.h> içinde bulunabilir. Mod operatörünü (%) rand ile aşağıdaki biçimde

kullandığımızda

rand( )%6

0 ile 5 dizisindeki (0,1,2,3,4,5) tamsayılar üretilecektir.Buna derecelendirme (scaling) denir. 6

sayısı, derecelendirme faktörü olarak adlandırılır.Daha sonra, önceki sonucumuza 1 ekleyerek

sayıların dizisini kaydırırız. Şekil 5.7 , sonuçların 1-6 dizisi içinde olduğunu onaylamaktadır.

1 /\* Şekil 5.7:fig05\_07.c

2 1+rand( )%6 ile üretilmiş kaydırılmış ve derecelendirilmiş tamsayılar \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 int main( )

7 {

8 int i;

9

10 for ( i = 1; i <= 20; i++ ) {

11 printf( "%10d", 1 + ( rand( ) % 6 ) );

12

13 if ( i % 5 == 0 )

14 printf( "\n" );

15 }

16

17 return 0;

18 }

5 5 3 5 5

2 4 2 5 5

5 3 2 2 1

5 1 4 6 4

Şekil 5.7 1+rand( )%6 ile üretilmiş tamsayılar

157

Bu sayıların yaklaşık olarak eşit ihtimalle ortaya çıktıklarını göstermek için, Şekil 5.8’de zarın

6000 kez atılmasını gerçekleyelim.1’den 6’ya kadar tüm tamsayılar yaklaşık olarak 1000 kez

zarın üstüne gelmelidir.

1 /\* Şekil 5.8:fig05\_08.c

2 6 yüzlü bir zarı 6000 kez atmak \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 int main( )

7 {

8 int yuz, atis, frekans1 = 0, frekans2 = 0,

9 frekans3 = 0, frekans4= 0,

10 frekans5= 0, frekans6= 0;

11

12 for ( atis = 1; atis <= 6000; atis++ ) {

13 yuz = 1 + rand() % 6;

14

15 switch ( yuz ) {

16 case 1:

17 ++frekans1;

18 break;

19 case 2:

20 ++frekans2;

21 break;

22 case 3:

23 ++frekans3;

24 break;

25 case 4:

26 ++frekans4;

27 break;

28 case 5:

29 ++frekans5;

30 break;

31 case 6:

32 ++frekans6;

33 break;

34 }

35 }

36

37 printf( "%s%13s\n", "Yüz", "Frekans" );

38 printf( "1%15d\n", frekans1 );

39 printf( "2%15d\n", frekans2 );

40 printf( "3%15d\n", frekans3 );

41 printf( "4%15d\n", frekans4 );

42 printf( "5%15d\n", frekans5 );

43 printf( "6%15d\n", frekans6 );

44 return 0;

45 }

158

Yüz Frekans

1 987

2 984

3 1029

4 974

5 1004

6 1022

Şekil 5.8 6 yüzlü bir zarı 6000 kaz atmak

Program çıktılarından da görüldüğü gibi, derecelendirme ve kaydırma ile rand fonksiyonunun

6 yüzlü bir zarın atışını gerçekçi bir biçimde gerçeklemesini sağlattık. switch yapısında

default kısmının kullanılmadığına dikkat ediniz. Ayrıca, sütunların başlıkları olarak

yazdırılan “Yüz”ve “Frekans” karakter stringleri için %s dönüşüm belirteci kullandığımıza

dikkat ediniz. 6.Ünitede dizileri çalıştıktan sonra, tüm switch yapısını tek satırda nasıl

yazabileceğimizi göstereceğiz.

Şekil 5.7’deki programı yeniden çalıştırdığımızda

5 5 3 5 5

2 4 2 5 5

5 3 2 2 1

5 1 4 6 4

sonucunu elde ederiz.

Bu sonucun, daha önce yazdırılan sonuç ile aynı olduğuna dikkat ediniz. Öyleyse bu sayılar

nasıl rasgele sayılar olabilir? Bu tekrar, rand fonksiyonunun önemli bir özelliğidir. Bir

programın hatası ayıklanırken, bu tekrar programda yapılan düzeltmelerin doğru bir biçimde

çalıştığını kanıtlamak için önemlidir.

rand fonksiyonu gerçekte, sahte rasgele sayılar üretir. rand fonksiyonunu tekrar tekrar

çağırmak, rasgele gibi görünen bir dizi sayı oluşmasına sebep olur. Ancak bu dizi, program

her çalıştırıldığında kendini tekrar etmektedir. Programın hataları tamamen ayıklandığında,

her çalıştırılmada rasgele sayıların farklı bir dizisinin üretilmesi sağlatılabilir. Buna,

rassallaştırma denir ve standart kütüphane fonksiyonu olan srand sayesinde yapılır. srand

fonksiyonu, unsigned tipte bir tamsayıyı argüman olarak kullanır ve rand fonksiyonunu

besleyerek, programın her çalıştırılışında farklı bir dizide rasgele sayılar oluşturulmasını

sağlar.

srand fonksiyonunun kullanımı, Şekil 5.9’da gösterilmiştir. Programda, unsigned int için

kısaltma olarak unsigned veri tipini kullandığımıza dikkat ediniz. int , hafızada en az iki byte

içinde saklanır ve negatif ya da pozitif değerler alabilir. unsigned tipinde bir değişken de en

az iki byte içinde saklanır. İki byte bir unsigned int yalnızca, 0’dan 65535’e kadar olan

pozitif değerleri alabilir. Dört byte bir unsigned int yalnızca, 0’dan 4294967295’e kadar olan

pozitif tamsayıları alabilir. srand fonksiyonu, unsigned bir değeri argüman olarak alır. %u

159

dönüşüm belirteci, scanf ile unsigned bir değeri okumakta kullanılır. srand fonksiyonunun

prototipi <stdlib.h> içinde bulunur.

1 /\* Şekil 5.9: fig05\_09.c

2 Rasgele zar atma programı \*/

3 #include <stdlib.h>

4 #include <stdio.h>

5

6 int main( )

7 {

8 int i;

9 unsigned besleme;

10

11 printf( “Beslemeyi girin: " );

12 scanf( "%u", &besleme );

13 srand( besleme );

14

15 for ( i = 1; i <= 10; i++ ) {

16 printf( "%10d", 1 + ( rand() % 6 ) );

17

18 if ( i % 5 == 0 )

19 printf( "\n" );

20 }

21

22 return 0;

23 }

Beslemeyi girin: 67

1 6 5 1 4

5 6 3 1 2

Beslemeyi girin: 432

4 2 5 4 3

2 5 1 4 4

Beslemeyi girin: 67

1 6 5 1 4

5 6 3 1 2

Şekil 5.9 Zar atma programını rastgele hale getirme.

Programı bir çok kez çalıştırıp, sonuçları inceleyelim. Program her çalıştırıldığında ve farklı

bir besleme sağlandığında, farklı bir dizide rastgele sayılar oluşturulduğuna dikkat ediniz.

160

Eğer her seferinde besleme girmeden rasgele hale getirmek istiyorsak, aşağıdaki gibi bir ifade

kullanabiliriz:

srand ( time ( NULL ) );

Bu, bilgisayarın besleme değeri olarak otomatik bir biçimde kendi saatini okumasına sebep

olur. time fonksiyonu, o andaki saati saniye biçiminde oluşturur. Bu değer, unsigned bir

tamsayıya dönüştürülür ve rasgele sayı üretiminde besleme olarak kullanılır. time fonksiyonu

argüman olarak NULL kullanır. (time programcının o günkü zamanı temsil eden bir dizeyi

(string) elde etmesini sağlar ; NULL bu özelliği, time fonksiyonuna yapılan belirli bir

çağrıda ortadan kaldırır.) time fonksiyonunun prototipi <time.h> içindedir.

rand ile elde edilen değerler her zaman 0  rand( )  RAND\_MAX aralığındadır.

Daha önce, 6 yüzlü bir zarın atılışını gerçeklemek için nasıl bir ifade yazılacağını

göstermiştik;

yuz =1 + rand ( ) % 6;

Bu ifade her zaman, yuz değişkenine 1  yuz  6 aralığından rasgele olarak bir tamsayı atar.

Bu aralığın genişliğinin (aralıkta ard arda gelen tamsayı sayısı) 6 olduğuna ve aralığın

başlangıcının 1 olduğuna dikkat ediniz. Az önceki ifadeye dönersek, aralığın genişliğinin

rand’ı derecelendirmek için mod operatörüyle birlikte kullanılan sayı olduğunu görüyoruz.

(bu örnekte 6) Aralığın başlangıç sayısı ise rand % 6’ya eklenen sayıdır.(bu örnekte 1).Bu

sonucu aşağıdaki biçimde genelleştirebiliriz:

n = a + rand ( ) % b;

Burada a, kaydırma değeri (istenen aralığın başlangıç değeridir) b ise derecelendirme

faktörüdür ( istenen arlığın genişliğidir). Alıştırmalarda ard arda gelen sayılar yerine bir

kümedeki sayılar içinden rasgele sayı seçmenin de mümkün olduğunu göreceğiz.

Genel Programlama Hataları 5.12

Rasgele sayılar üretirken rand yerine srand kullanmak.

5.10 ÖRNEK: ŞANS OYUNU

Şans oyunlarından en popüler olanı, gazinolarda ve tüm dünyada sokaklarda oynanan barbut

adı verilen bir zar oyunudur. Oyunun kuralları açıktır:

Oyuncu iki zarı aynı anda atar. İki zarında altı yüzü vardır. Bu

yüzlerde 1,2,3,4,5 ve 6 adet nokta bulunur. Zarlar durduktan

sonra her iki zarında üste gelen yüzleri toplanır. Eğer toplam ilk

atışta 7 ya da 11 ise oyuncu kazanır. Eğer toplam ilk atışta 2,3

ya da 12 gelirse (buna barbut denir) oyuncu kaybeder. Eğer ilk

atışta toplam 4,5,6,8,9,10 ise bu toplam oyuncunun sayısı haline

gelir.Kazanmak için oyuncu sayısını bulana kadar zarları atmaya

devam eder.Zarları atmaya devam ederken kendi sayısı yerine 7

atarsa kaybeder.

161

Şekil 5.10 barbut oyununun gerçeklenmesini ve Şekil 5.11 programın çalıştırılmasından

ortaya çıkan sonuçları göstermektedir.

1 /\* Şekil. 5.10: fig05\_10.c

2 Barbut \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5 #include <time.h>

6

7 int zarAtma( void );

8

9 int main( )

10 {

11 int oyunDurumu, toplam, oyuncuPuani;

12

13 srand( time( NULL ) );

14 toplam = zarAtma( ); /\* zarın ilk atılışı \*/

15

16 switch( toplam ) {

17 case 7: case 11: /\* ilk atışta kazanma \*/

18 oyunDurumu = 1;

19 break;

20 case 2: case 3: case 12: /\* ilk atışta kaybetme \*/

21 oyunDurumu = 2;

22 break;

23 default: /\* hatırlatma noktası \*/

24 oyunDurumu = 0;

25 oyuncuPuani= toplam;

26 printf( "Oyuncunun kazanacağı zar: %d\n", oyuncuPuani );

27 break;

28 }

29

30 while (oyunDurumu == 0 ) { /\* zar atmaya devam et \*/

31 toplam= zarAtma( );

32

33 if ( toplam == oyuncuPuani) /\* kazanılacak zarı atma \*/

34 oyunDurumu = 1;

35 else

36 if ( toplam == 7 ) /\* 7 atma ile kaybetme \*/

37 oyunDurumu = 2;

38 }

39

40 if (oyunDurumu == 1 )

41 printf( "Oyuncu Kazanır \n" );

42 else

43 printf( "Oyuncu kaybeder\n" );

44

45 return 0;

46 }

47

162

48 int zarAtma( void )

49 {

50 int zar1, zar2, toplamZar;

51

52 zar1 = 1 + ( rand( ) % 6 );

53 zar2 = 1 + ( rand( ) % 6 );

54 toplamZar = zar1 + zar2;

55 printf( "Oyuncu %d + %d = %d attı \n", zar1, zar2, toplamZar );

56 return toplamZar;

57 }

Şekil 5.10 Barbut oyununun gerçeklendiği program

Oyuncu 6 + 5 = 11 attı

Oyuncu Kazanır

Oyuncu 6 + 6 = 12 attı

Oyuncu Kaybeder

Oyuncu 4 + 6 = 10 attı

Oyuncunun kazanacağı zar: 10

Oyuncu 2 + 4 = 6 attı

Oyuncu 6 + 5 = 11 attı

Oyuncu 3 + 3 = 6 attı

Oyuncu 6 + 4 = 10 attı

Oyuncu Kazanır

Oyuncu 1 + 3 = 4 attı

Oyuncunun kazanacağı zar: 4

Oyuncu 1 + 4 = 5 attı

Oyuncu 5 + 4 = 9 attı

Oyuncu 4 + 6 = 10 attı

Oyuncu 6 + 3 = 9 attı

Oyuncu 1 + 2 = 3 attı

Oyuncu 5 + 2 = 7 attı

Oyuncu Kaybeder

Şekil 5.11 Barbut oyunu programının çalıştırılmasına örnekler.

163

Oyuncunun ilk atışta ve daha sonra gelen atışlarda iki zarı birden atması gerektiğine dikkat

ediniz. zarAtma adında bir fonksiyon, zarların atışından gelen toplamı hesaplayıp, yazdırmak

için tanımlanmıştır. zarAtma fonksiyonu bir kez tanımlanmış ancak programda iki yerde

çağrılmıştır. İlginç olan zarAtma fonksiyonunun argüman almamasıdır. Bu sebepten,

fonksiyonun parametre listesinde void kullanılmıştır. zarAtma fonksiyonu iki zarın üstüne

gelen sayıların toplamını döndürmektedir ve bu sebepten, fonksiyon başlığında geri dönüş tipi

int olarak bildirilmiştir.

Program oldukça kapsamlıdır. Oyuncu ilk atışında kazanabilir ya da kaybedebilir ya da daha

sonraki atışlarında kazanabilir ya da kaybedebilir. oyunDurumu değişkeni, bütün bunların

kaydını tutmak için kullanılmıştır

Oyun ilk ya da daha sonraki atışlarda kazanıldığında, oyunDurumu değişkeni 1 yapılmıştır.

Oyun ilk ya da daha sonraki atışlarda kaybedildiğinde, oyunDurumu değişkeni 2 yapılmıştır.

Aksi takdirde, oyunDurumu değişkeni 0 olarak gösterilmiştir ve program devam edecektir.

İlk atıştan sonra eğer oyun biterse, oyundurumu 0’a eşit olmadığı için while yapısı atlanır.

Program if/else yapısını çalıştırır ve oyundurumu 1 ise “Oyuncu kazanır” , oyunDurumu 2

ise “Oyuncu kaybeder “ yazdırır.

İlk atıştan sonra oyun bitmezse toplam , oyuncuPuani içinde saklanır. oyunDurumu 0

olduğu için program while yapısıyla devam eder. while yapısı her çalıştığında zarAtma

fonksiyonu çağrılarak yeni bir toplam oluşturulur. Eğer toplam, oyuncuPuani ile eşleşirse

oyunDurumu oyuncunun kazandığını belirtmek için 1 olur, while testi yanlış olacağından

while yapısı atlanır, if/else yapısı “Oyuncu kazanır” yazdırır ve programın çalışması sona

erer. Eğer toplam 7’ye eşitse oyunDurumu oyuncunun kaybettiğini belirtmek için 2 olur,

while testi yanlış olacağından while yapısı atlanır, if/else yapısı “Oyuncu kaybetder”

yazdırır ve programın çalışması sona erer.

Programın ilginç kontrol yapısına dikkat ediniz. main ve zarAtma adında iki fonksiyon,

switch, while, if/else ve yuvalı if yapılarını kullandık. Alıştırmalarda barbut oyununun ilginç

özelliklerini inceleyeceğiz.

5.11 DEPOLAMA SINIFLARI

2. üniteden 4. üniteye kadar değişken isimleri için tanıtıcılar kullandık. Değişkenlerin

özellikleri isim, tip ve değer içermektedir. Bu ünitede, tanıtıcıları ayrıca programcı tarafından

tanımlanan fonksiyonların isimleri olarak da kullandık. Aslında bir programdaki her tanıtıcı

depolama sınıfı,depolama süreci,faaliyet alanı ve bağlama özelliklerine de sahiptir.

C auto,register,extern ve static depolama sınıfı belirteçleriyle belirlenebilen dört depolama

sınıfına sahiptir. Bir tanıtıcının depolama sınıfı, depolama süreci, faaliyet alanı ve bağlama

özelliklerinin belirlenmesine yardımcı olur. Bir tanıtıcının depolama süreci o tanıtıcının

hafızada tutulduğu zaman aralığıdır. Bazı tanıtıcılar oldukça kısa, bazıları yeniden yaratılıp

yok edilerek, bazıları ise programın çalıştığı tüm süre boyunca hafızada tutulur. Bir tanıtıcının

faaliyet alanı tanıtıcının program içinde kullanılabileceği yerdir. Bazı tanıtıcılar, tüm program

boyunca, diğerleri ise programın bazı kısımlarında kullanılabilirler. Bir tanıtıcının bağlaması,

çok kaynaklı bir program ( Bu konuyu 14. ünitede inceleyeceğiz) içinde tanıtıcının yalnızca o

andaki kaynak dosyada mı yoksa uygun bildirimlerle herhangi bir kaynak dosyada mı geçerli

164

olacağını belirler. Bu kısım, depolama sınıflarını ve depolama süreçlerini açıklamaktadır.

Kısım 5.12, tanıtıcıların faaliyet alanlarını açıklamaktadır. 14. ünite, tanıtıcı bağlama ve

birden çok kaynak dosyası ile programlamayı açıklamaktadır.

4 depolama sınıfı belirteci, 2 depolama sürecine ayrılabilir: Otomatik depolama süreci ve

statik depolama süreci. auto ve register anahtar kelimeleri, otomatik depolama süreçli

değişkenler bildirmek için kullanılırlar. Otomatik depolama süreçli değişkenler, bildirildikleri

blok içine girildiğinde yaratılır, blok aktif iken varolur ve bloktan çıkıldığında yok edilirler.

Yalnızca değişkenler otomatik depolama sürecine sahip olabilirler. Bir fonksiyonun yerel

değişkenleri ( parametre listesi yada fonksiyon gövdesi içinde bildirilenler) otomatik

depolama sürecine sahiptirler. auto anahtar kelimesi, otomatik depolama süreçli değişkenler

bildirmek için kullanılır. Örneğin, aşağıdaki bildirim double tipteki x ve y değişkenlerinin

otomatik yerel değişkenler olduğunu ve yalnızca bildirimin yer aldığı fonksiyon gövdesi

içinde varolduğunu belirtir:

auto double x, y;

Yerel değişkenler aksi belirtilmedikçe otomatik depolama sürecine sahip olduklarından, auto

anahtar kelimesi nadiren kullanılır. Bu noktadan itibaren otomatik depolama süreçli

değişkenlere kısaca otomatik değişkenler diyeceğiz.

Performans İpuçları 5.1

Otomatik depolama hafızayı korumak için kullanılır.Çünkü otomatik değişkenler yalnızca ihtiyaç

duyulduklarında varolurlar.Bunlar, bildirildikleri fonksiyon çalıştırıldığında yaratılır, fonksiyonun

çalıştırılması sona erdiğinde yok edilirler.

Yazılım Mühendisliği Gözlemleri 5.10

Otomatik depolama, en az yetki prensibinin bir başka örneğidir. Değişkenler neden hafızada

depolansın ve neden gerçekte ihtiyaç duyulmamalarına rağmen erişilebilsin? Bir programın makine

dili versiyonunda veri, hesaplamalar ve diğer işlemler için genellikle yazmaçlara ( register ) yüklenir.

Performans İpuçları 5.2

register depolama sınıfı belirteci, otomatik değişken bildiriminden önceye yerleştirilerek derleyiciye,

değişkeni, bilgisayarın yüksek hızlı donanım yazmaçlarından birine yerleştirmesi önerilebilir.Eğer

sayıcı ya da toplam gibi sıklıkla kullanılan değişkenler donanım yazmaçları içine yerleştirilebilirse,

değişkenleri hafızadan yazmaçlara sık sık yüklemek ve sonuçları hafızaya yazmak yükü ortadan

kaldırılabilir.

Derleyici register bildirimlerini ihmal edebilir. Örneğin derleyicinin kullanımına uygun

yeterli sayıda yazmaç olmayabilir. Aşağıdaki bildirim, sayici değişkeninin tamsayı olduğunu

ve bilgisayarın yazmaçlarından birine yerleştirileceğini ve ayrıca ilk değer olarak 1 değerine

atanacağını belirtmektedir:

register int sayici = 1;

register anahtar kelimesi yalnızca otomatik depolama süreçli değişkenlerle kullanılabilir.

165

Performans İpuçları 5.3

Genellikle register bildirimleri gereksizdir. Bugünkü derleyicilerin bir çoğu sıklıkla kullanılan

değişkenleri tanıyıp, programcının register bildirimi yapmasına gerek kalmadan değişkeni

yazmaçlardan birinin içine koymaya karar verir.

extern ve static anahtar kelimeleri, statik depolama süreçli değişken ve fonksiyon tanıtıcıları

bildirmek için kullanılırlar. Statik depolama süreçli tanıtıcılar, program çalışmaya başladığı

andan itibaren var olurlar. Değişkenler için program çalışmaya başladığında bir kereliğine

depolama için yer ayrılır ve ilk değerler verilir. Fonksiyonlar için fonksiyon ismi program

çalışmaya başladığında varolur ancak değişkenler ve fonksiyon isimleri program çalışmaya

başladığı andan itibaren varolsalar da bu, tanıtıcıların program boyunca kullanılacakları

anlamına gelmez. Depolama süreci ve faaliyet alanı kısım 5.12’de göreceğimiz gibi ayrı

konulardır.

Statik depolama zamanlı iki tip tanıtıcı vardır: dış tanıtıcılar (global değişkenler ve fonksiyon

isimleri gibi) ve static depolama zamanı belirteciyle bildirilmiş yerel değişkenler. Global

değişkenler ve fonksiyon isimleri, aksi belirtilmedikçe extern depolama sınıfındadırlar.

Global değişkenler, değişken bildirimleri herhangi bir fonksiyon tanımının dışında yapılarak

oluşturulurlar ve değerlerini programın çalışma zamanı boyunca korurlar. Global değişkenler

ve fonksiyonlar, bildirimlerini ve tanımlanmalarını izleyen fonksiyonlar tarafından ya da

dosya tanımlamalarında kullanılabilirler. Bu, fonksiyon prototiplerinin kullanılmasının bir

sebebidir. printf çağıran bir programa stdio.h eklediğimizde, fonksiyon prototipi programın

en başında yer alır ve böylece printf dosyanın geri kalanında bilinir.

Yazılım Mühendisliği Gözlemleri 5.11

Bir değişkeni yerel değil de global olarak bildirmek, bir değişkene erişmemesi gereken bir

fonksiyonun, değişkeni yanlışlıkla değiştirmesi gibi istenmeyen yan etkilere sebep olabilir.Genelde

bazı belirli ve çok özel durumlar hariç (14.Ünitede anlatıldığı gibi) global değişkenlerin

kullanımından kaçınılmalıdır.

İyi Programlama Alıştırmaları 5.10

Yalnızca belli bir fonksiyonda kullanılan değişkenler o fonksiyon içinde yerel olarak bildirilmelidir.

static anahtar kelimesiyle bildirilen yerel değişkenler yalnızca bildirildikleri fonksiyon içinde

bilinmektedirler ancak otomatik değişkenlerden farklı olarak, static yerel değişkenler

fonksiyondan çıkıldıktan sonrada değerlerini korurlar. Fonksiyonun bir sonraki çağrısında,

static yerel değişken, fonksiyondan en son çıkıldığındaki değeri tutmaktadır. Şimdiki

ifademiz say yerel değişkenini static olarak bildirmekte ve değişkene ilk değer olarak 1

vermektedir.

static int say = 1;

Statik depolama zamanlı tüm nümerik değişkenler, programcı tarafından özel olarak bir başka

değere atanmadıkça ilk değer olarak 0’a atanırlar. (Gösterici değişkenleri 7.Ünitede anlatıldığı

gibi ilk değer olarak NULL’a atanırlar)

Genel Programlama Hataları 5.13

Bir tanıtıcı için birden çok depolama sınıfı belirteci kullanmak.Bir değişkene yalnızca bir depolama

sınıfı belirteci uygulanabilir.

166

extern ve static anahtar kelimeleri , dış tanıtıcılara uygulandıklarında özel anlamlara gelirler.

14.Ünitede, extern ve static anahtar kelimelerinin dış tanıtıcılarla ve çok kaynak dosyalı

programlarla özel olarak nasıl kullanılacağını anlatacağız.

5.12 FAALİYET ALANI KURALLARI

Bir tanıtıcının faaliyet alanı, tanıtıcının kod içinde kullanılabileceği program kısmıdır.

Örneğin, bir blok içinde yerel değişken bildirirsek, bu değişken yalnızca o blok içinde ya da o

bloğun içine yuvalanmış bloklarda kullanılabilir. Bir tanıtıcının 4 faaliyet alanı şu şekilde

adlandırılır: fonksiyon faaliyet alanı ,dosya faaliyet alanı, blok faaliyet alanı ve fonksiyon

prototipi faaliyet alanı.

Etiketler (bir tanıtıcının sonuna iki nokta üst üste konurak oluşturulur, örneğin basla: )

fonksiyon faaliyet alanına sahip tek tanıtıcıdır. Etiketler, bulundukları fonksiyon içinde her

yerde kullanılabilirler ancak fonksiyon gövdesi dışında kullanılamazlar. Etiketler, switch

yapılarında (case etiketleri gibi) ve goto ifadelerinde (14.Üniteye bakınız) kullanılırlar.

Etiketler, fonksiyonların diğerlerinden gizledikleri uygulama detaylarıdır. Bu saklama ( daha

teknik olarak bilgi saklama ), iyi yazılım mühendisliğinin en temel prensiplerinden biridir.

Herhangi bir fonksiyonun dışında bildirilmiş tanıtıcılar, dosya faaliyet alanına sahiptir. Bu

tarzda bir tanıtıcı, bildirildiği yerden dosyanın sonuna kadar tüm fonksiyonlar tarafından

bilinir. Global değişkenler, fonksiyon tanımları ve fonksiyon dışına yerleştirilmiş fonksiyon

prototipleri dosya faaliyet alanına sahiptir.

Bir blok içinde bildirilmiş tanıtıcılar, blok faaliyet alanına sahiptir. Blok faaliyet alanı, bloğu

sonlandıran küme parantezine( } ) ulaşıldığında sona erer. Fonksiyonun başında bildirilen

yerel değişkenler, fonksiyon parametreleri gibi blok faaliyet alanına sahiptir ve fonksiyon için

yerel değişkenler olarak bilinirler. Herhangi bir blok, değişken bildirimleri içerebilir. Bloklar

yuvalandığında ve dıştaki bloktaki tanıtıcı içteki bloktaki tanıtıcıyla aynı isme sahip

olduğunda, dış bloktaki tanıtıcı iç blok sona erene kadar saklanır. Bu, içteki blok

çalıştırılırken içteki bloğun yalnızca kendi yerel tanıtıcısının değerini göreceği ve dışındaki

blokta yer alan ve aynı isme sahip olan tanıtıcının değerini kullanmayacağı anlamına gelir.

static olarak bildirilmiş yerel değişkenler, program çalışmaya başladığı andan itibaren var

olsalar da hala blok faaliyet alanına sahiptirler. Bu sebepten, depolama süreci bir tanıtıcının

faaliyet alanını etkilemez.

Fonksiyon prototipi faaliyet alanına sahip tek tanıtıcı, fonksiyon prototipinin parametre

listesinde kullanılan tanıtıcılardır. Daha önceden bahsedildiği gibi, fonksiyon prototiplerinin

parametre listelerinde isim olması gerekmez, derleyici bu isimleri ihmal eder. Fonksiyon

prototipinde kullanılan tanıtıcılar, programın herhangi bir yerinde karışıklık olmadan yeniden

kullanılabilir.

Genel Programlama Hataları 5.14

Programcı dış bloktaki tanıtıcının iç blok çalışırken aktif olmasını isterken, yanlışlıkla iç blokta

kullandığı tanıtıcı ismiyle dış blokta kullandığı tanıtıcı isminin aynı olması.

İyi Programlama Alıştırmaları 5.11

Dış faaliyet alanlarında, isimleri gizleyen değişken isimlerinden kaçının. Bu, bir programda aynı

tanıtıcı ismini bir kez daha kullanmayarak sağlanabilir.

167

Şekil 5.12 global değişkenler, otomatik yerel değişkenler ve static yerel değişkenler için

faaliyet alanı konularını göstermektedir. Global olarak bir x değişkeni bildirilmiş ve bu

değişkene ilk değer olarak 1 atanmıştır. Bu global değişken, x adında bir değişkenin

bildirildiği herhangi bir bloktan (ya da fonksiyon) gizlenmiştir. main içinde x yerel değişkeni

bildirilmiş ve bu değişkene ilk değer olarak 5 atanmıştır. Daha sonra bu değişken yazdırılarak

main içinde global değişken olan x’in saklandığı gösterilmiştir. main içinde yeni bir blok

tanımlanmış ve bu blok içinde başka bir yerel değişken olan x bildirilmiş ve bu değişkene ilk

değer olarak 7 atanmıştır. Bu değer yazdırılarak x’in dış blok olan main’den saklandığı

gösterilmiştir. 7 değerine sahip olan x değişkeni, bloktan çıkıldığında otomatik olarak yok

edilmiştir ve main dış bloğu içindeki yerel değişken olan x yeniden yazdırılarak artık bu

değişkenin daha fazla saklanmadığı gösterilmiştir. Program, argüman almayan ve geriye

değer döndürmeyen üç fonksiyon tanımlamıştır. a fonksiyonu, otomatik bir x değişkeni

bildirmiş ve bu değişkene 25 ilk değerini atamıştır. a çağrıldığında değer yazdırılmış,

arttırılmış ve fonksiyondan çıkılmadan bir kez daha yazdırılmıştır. Fonksiyon her

çağrıldığında x otomatik değişkeni yeniden 25 değerine atanmıştır. b fonksiyonu, static bir x

değişkenini bildirmekte ve bu değişkene ilk değer olarak 50 atamaktadır. static olarak

bildirilen yerel değişkenler, faaliyet alanı dışında olsalar bile değerlerini korurlar. b

çağrıldığında x yazdırılmış , arttırılmış ve fonksiyondan çıkılmadan önce bir kez daha

yazdırılmıştır. Bu fonksiyonun bir sonraki çağrısında, static yerel değişken x, 51 değerini

içerecektir. c fonksiyonu, herhangi bir değişken bildirmemektedir. Bu sebepten, x

değişkeninden bahsettiğinde, global x değişkeni kullanılmaktadır. c çağrıldığında global

değişken yazdırılmakta, 10 ile çarpılmakta ve fonksiyondan çıkılmadan önce bir kez daha

yazdırılmaktadır. Sonuç olarak program, main içindeki yerel değişken x’i, fonksiyon

çağrılarının hiçbirinin x’ in değerini değiştirmediğini çünkü tüm fonksiyonların başka faaliyet

alanlarındaki değişkenlerden söz ettiğini göstermek için yeniden yazdırılmıştır.

1 /\* Fig. 5.12: fig05\_12.c

2 Bir faaliyet alanı örneği \*/

3 #include <stdio.h>

4

5 void a( void ); /\* fonksiyon prototipi\*/

6 void b( void ); /\* fonksiyon prototipi \*/

7 void c( void ); /\* fonksiyon prototipi \*/

8

9 int x = 1; /\* global değişken \*/

10

11 int main( )

12 {

13 int x = 5; /\* main’e yerel değişken \*/

14

15 printf ("main’in faaliyet alanı dışındaki yerel x değişkeni %d olur.\n", x );

16

17 { /\* yeni faaliyet alanına başla \*/

18 int x = 7;

19

20 printf( "main’in faaliyet alanı içindeki yerel x değişkeni %d olur.\n", x );

21 } /\* yeni faaliyet alanını bitir\*/

22

23 printf( "main’in faaliyet alanı dışındaki yerel x değişkeni %d olur.\n", x );

24

168

25 a( ); /\* a automatic yerel x’ e sahiptir.\*/

26 b( ); /\* b static yerel x’ e sahiptir. \*/

27 c( ); /\* c global x kullanır. \*/

28 a( ); /\* a automatic yerel x’ e tekrar ilk değer atar.\*/

29 b( ); /\* static yerel x önceki değerini korur \*/

30 c( ); /\* global x ‘ de değerinin korur. \*/

31

32 printf( "main içindeki yerel x %d olur.\n", x );

33 return 0;

34 }

35

36 void a( void )

37 {

38 int x = 25; /\* a her çağrıldığında ilk değeri atanır \*/

39

40 printf( "\na’ya girildikten sonra yerel x %d olur.\n", x );

41 ++x;

42 printf( "a’dan çıkmadan önce yerel x %d olur.\n", x );

43 }

44

45 void b( void )

46 {

47 static int x = 50; /\* sadece static ilk değer atama \*/

48 /\* b’nin ilk çağırılması \*/

49 printf( "\nb’ye girerken static x %d olur.\n", x );

50 ++x;

51 printf( "b’den çıkarken önce static x %d olur.\n", x );

52 }

53

54 void c( void )

55 {

56 printf( "\nc’ ye girerken global x %d olur.\n", x );

57 x \*= 10;

58 printf( "c’ den çıkarken global x %d olur.\n", x );

59 }

main’in faaliyet alanı dışında yerel x değişkeni 5 olur.

main’in faaliyet alanı içindeki yerel x değişkeni 7 olur.

main’in faaliyet alanı dışında yerel x değişkeni 5 olur.

a’ya girildikten sonra yerel x 25 olur.

a’dan çıkmadan önce yerel x 26 olur.

b’ye girerken static x 50 olur.

b’den çıkarken static x 51 olur.

c’ ye girerken global x 1 olur.

c’ den çıkarken global x 10 olur.

a’ya girildikten sonra yerel x 25 olur.

169

a’dan çıkmadan önce yerel x 26 olur.

b’ye girerken static x 51 olur.

b’den çıkarken static x 52 olur.

c’ ye girerken global x 10 olur.

c’ den çıkarken global x 100 olur.

main içindeki yerel x 5 olur.

Şekil 5.12 Faaliyet Alanı Örneği

5.13 YİNELEME

Şimdiye kadar tartıştığımız programlar, fonksiyonların birbirlerini hiyerarşik bir düzende

çağırdıkları yapısal programlardı. Bazı problem tipleri için fonksiyonların kendi kendilerini

çağırması kullanışlı olabilir. Bir yineleme fonksiyonu ( recursive function ), kendi kendini

doğrudan ya da bir başka fonksiyon içinden çağıran fonksiyondur. Yineleme, yüksek seviyeli

bilgisayar derslerinde uzunca tartışılan karışık bir konudur. Bu ve sonraki kısımda

yinelemenin basit örnekleri gösterilecektir. Bu kitap yineleme konusuna, 5. Üniteden 12.

üniteye kadar özel bir önem göstermiştir. Kısım 5.15’te bu kitap boyunca yinelemeyle ilgili

verilmiş 31 örnek ve alıştırmanın bir özetini bulacaksınız.

Öncelikle yineleme kavramı üstünde duracak daha sonra da yineleme fonksiyonları içeren

örnekler inceleyeceğiz. Yinelemeli problem çözme yaklaşımlarının genelde, birden çok

elemanı vardır. Yineleme fonksiyonu, bir problemi çözmek için çağrılır. Bu fonksiyon,

yalnızca en basit durumu ya da temel durum olarak adlandırılan durumu nasıl çözeceğini

bilmektedir. Eğer fonksiyon temel bir durumla çağrılırsa, fonksiyon bir sonuç geri döndürür.

Eğer fonksiyon daha karmaşık bir problemle çağrılırsa, fonksiyon problemi iki kavramsal

parçaya ayırır : Fonksiyonun nasıl yapacağını bildiği parça ve fonksiyonun nasıl yapacağını

bilmediği parça. Yinelemeyi mümkün kılmak için sonraki parça orijinal probleme

benzemelidir, fakat orijinal problemin daha basit ya da daha küçük bir versiyonu olmalıdır.

Bu yeni problem orijinal probleme benzediğinden, fonksiyon bu küçük problem üzerinde

çalışmak için yeni bir kopyasını çağırır. Buna yineleme çağrısı ve de yineleme adımı denir.

Yineleme adımı, return anahtar kelimesini içerir çünkü bu adımın sonucu, fonksiyonun

problemin nasıl çözeceğini bildiği kısmıyla birleştirilerek, orijinal çağırıcıya döndürülecek

sonucu oluşturur.(orijinal çağırırcı muhtemelen main’dir)

Yineleme adımı fonksiyona yapılan çağrı açıkken, yani henüz çalışması sonlanmadan çalışır.

Yineleme adımı daha fazla yineleme çağrısına sebep olabilir. Fonksiyon her problemi

bölmeye devam ederken iki kavramsal kısım ile çağrılır. Yinelemeden çıkılabilmesi için , her

seferinde fonksiyon kendini problemin biraz daha basit versiyonuyla çağırır. Bu basit ve daha

basit problemlerin dizisi en sonunda temel duruma ulaşmalıdır. Bu noktada fonksiyon temel

durumu tanır, fonksiyonun bir önceki kopyasına bir sonuç aktarır ve sonuçların döndürüldüğü

bir dizi, fonksiyonun orijinal çağrısının en son sonucu main’e döndürmesine kadar yukarıya

doğru devam eder. Bütün bunlar, şu ana kadar kullanmaya alıştığımız problem çözme

170

teknikleriyle karşılaştırıldığında oldukça ilginç gelebilir. Aslında, bu sürecin doğal gelmesi

için bir çok yineleme programı yazma alıştırması yapmak gerekmektedir. Bu kavramların bir

örneği olarak, popüler bir matematik hesaplamasını gerçekleştiren bir yineleme programı

yazalım.

Negatif olmayan bir n tamsayısının faktoriyeli, n! biçiminde yazılır (“ n faktoriyel” şeklinde

okunur) ve

n \* (n-1) \* (n-2) \*......\* 1

çarpımı olarak tanımlanır. 1! ve 0! faktoriyel 1 olarak tanımlanmıştır. Örneğin , 5! 120’ye

eşit olan 5 \* 4 \* 3 \* 2 \* 1 çarpımıdır.

Sıfırdan büyük yada sıfıra eşit bir tamsayının (sayi) faktoriyeli, yineleme olmadan for

kullanarak aşağıdaki biçimde hesaplanabilir:

faktoriyel = 1;

for ( sayici = sayi; sayici >= 1; sayici-- )

faktoriyel \*= sayici;

Faktoriyel fonksiyonu için bir yineleme tanımına

n! = n \* ( n –1 )!

bağıntısı gözlemlenerek ulaşılabilir.

Örneğin, 5! aşağıda açıkça gösterildiği gibi 5 \* 4! ’e eşittir:

5!=5\*4\*3\*2\*1

5!=5\*(4\*3\*2\*1)

5!=5\*(4!)

5!’in hesaplanışı, şekil 5.13’te gösterildiği biçimde ilerler. Şekil 5.13-a , yineleme çağrılarının

başarısının, yinelemeyi sonlandıran 1!‘in bir olarak hesaplanmasını sağlayana kadar nasıl

sürdüğünü göstermektedir. Şekil 5.13-b , en son sonuca ulaşılıp bu sonuç döndürülene kadar

her yineleme çağrısından döndürülen sonucu göstermektedir.

5!

5 \* 4!

4 \* 3!

3 \* 2!

2 \* 1!

5!

5 \* 4!

4 \* 3!

Son değer = 120

5! = 5 \* 24 = 120 geri gönderilmiştir

4! = 4 \* 6 = 24 geri gönderilmiştir

3! = 3 \* 2 = 6 geri gönderilmiştir

171

---

Şekil 5.13 5!’in yineleme ile hesaplanması

Şekil 5.14, 0’dan 10’a kadar olan tamsayıların faktoriyellerini hesaplamak ve yazdırmakta

yinelemeyi kullanmaktadır. (long veri tipinin kullanılışı az sonra açıklanacaktır) faktoriyel

yineleme fonksiyonu, ilk önce bir sonlandırma koşulunun ( sayi 1’den küçük ya da 1’e eşitse)

doğru olup olmadığını kontrol etmektedir. sayi 1’den küçük ya da 1’e eşitse, faktoriyel

fonksiyonu 1 sonucunu döndürmektedir ve daha fazla yinelemeye gerek kalmadığından

program sonlanmaktadır. Eğer sayi 1’den büyükse

return sayi \* faktoriyel ( sayi - 1);

ifadesi problemi, sayi ile sayi-1’in faktoriyelini hesaplayan faktoriyel fonksiyonuna bir

yineleme çağrısının çarpımı biçiminde ifade etmektedir. faktoriyel (sayi-1)’in orijinal

hesaplama olan faktoriyel (sayi)’dan biraz daha basit bir problem olduğuna dikkat ediniz.

1 /\* Fig. 5.14: fig05\_14.c

2 Yineleme faktoriyel fonksiyonu \*/

3 #include <stdio.h>

4

5 long faktoriyel ( long );

6

7 int main( )

8 {

9 int i;

10

11 for ( i = 1; i <= 10; i++ )

12 printf( "%2d! = %ld\n", i, faktoriyel(i ) );

13 return 0;

14 }

15

16 long faktoriyel( long sayi)

17 {

18 if ( sayi <= 1 )

19 return 1;

20 else

21 return ( sayi \* faktoriyel(sayi - 1 ) );

1

3 \* 2!

2 \* 1!

2! = 2 \* 1 = 2 geri gönderilmiştir

1 geri gönderilmiştir.

a) Yineleme çağrılarının işleyişi b) Her bir yineleme çağrısından geri gönderilen değerler

172

22 }

1! = 1

2! = 2

3! = 6

4! = 24

5! = 120

6! = 720

7! = 5040

8! = 40320

9! = 362880

10! = 3628800

Şekil 5.14 Faktoriyel hesaplamalarını yineleme fonksiyonuyla yapmak

faktoriyel fonksiyonu, long tipte bir parametre alacak ve long tipte bir sonuç üretecek

biçimde bildirilmiştir. Bu, long int gösteriminin kısaltmasıdır. ANSI standardı long int

tipinde bir değişkenin en az 4 byte içinde depolanacağını ve bu sebepten +2147483647 kadar

büyük bir değer tutabileceğini belirlemiştir. Şekil 5.14’te görülebileceği gibi faktoriyel

değerleri hızlı bir biçimde büyümektedir. long veri tipini, küçük tamsayılar (2 byte gibi)

kullanan bilgisayarlarda 7!’den daha büyük faktoriyel değerlerini hesaplayabilmek için seçtik.

long değerleri yazdırmak için %ld belirteci kullanılır. Maalesef, faktoriyel fonksiyonu büyük

değerleri o kadar hızlı üretir ki, long int bir değişkenin büyüklüğüne ulaşılana kadar, bir çok

faktoriyel değerini long int kullansak bile yazdırmak mümkün olmaz.

Alıştırmalarda araştıracağımız gibi, daha büyük sayıların faktoriyellerini hesaplamak için

kullanıcı tarafından double kullanmaya ihtiyaç duyulabilir. Bu, C’nin (ve diğer bir çok

programlama dilinin) bir zayıflığını gösterir. Dil, çeşitli uygulamaların ihtiyaçlarını

karşılamak için kolaylıkla genişletilememektedir. İleride göreceğimiz gibi, C++ istediğimizde

büyük sayılar elde edebilmemiz için kolaylıkla genişletilebilir.

Genel Programlama Hataları 5.15

İhtiyaç duyulmasına rağmen, bir yinelemeli fonksiyondan değer geri döndürmeyi unutmak.

Genel Programlama Hataları 5.16

Temel durumu dahil etmemek ya da yineleme adımını temel duruma ulaşmayacak yanlış bir biçimde

yazmak, neticede hafızayı yoran sonsuz yineleme yaratır. Bu, yinelemeli olmayan bir çözümde sonsuz

döngü problemiyle eşdeğerdir. Sonsuz yineleme beklenmeyen bir giriş yapıldığında da oluşabilir.

5.14 YİNELEMELERİ KULLANAN ÖRNEK: FIBONACCI SERİLERİ

Fibonacci serileri

0,1,1,2,3,5,8,13,21,......

0 ve 1 ile başlar ve sonradan gelen her Fibonacci sayısının, kendinden önceki iki Fibonacci

sayısının toplanması özelliğine sahiptir.

173

Seriler doğada bulunmaktadır ve özel olarak spiral biçimini tanımlar. Fibonacci sayılarının

oranı, sabit bir değer olan 1.618....’e yaklaşır. Bu sayı da doğada sık sık tekrarlanır ve altın

oran ya da altın orta olarak adlandırılır. İnsanlar altın ortayı estetik olarak hoş bulma

eğilimindedir. Mimarlar genellikle pencereleri, odaları ve binaları, uzunluk ve genişlikleri

altın orta oranında olacak biçimde tasarlarlar. Kartpostallar genellikle altın orta oranına sahip

bir biçimde yapılırlar.

Fibonacci serileri yinelemeli olarak aşağıdaki biçimde tanımlanabilir:

fibonacci(0)=0

fibonacci(1)=1

fibonacci(n)=fibonacci(n-1)+fibonacci(n-2)

Şekil 5.15, fibonacci fonksiyonunu kullanarak i. Fibonacci sayısını yinelemeli olarak

hesaplamaktadır. Fibonacci sayılarının hızlı bir biçimde büyüme eğiliminde olduklarına

dikkat ediniz. Bu sebepten, fibonacci fonksiyonunda geri dönüş tipi ve parametre tipi için

long tipini seçtiğimize dikkat ediniz. Şekil 5.15’te çıktıların her iki satırı programın ayrı bir

çalışmasından elde edilen sonucu göstermektedir.

1 /\* Fig. 5.15: fig05\_15.c

2 Yinelemeli fibonacci fonksiyonu \*/

3 #include <stdio.h>

4

5 long fibonacci( long );

6

7 int main( )

8 {

9 long sonuc, sayi;

10

11 printf( "Bir tamsayı giriniz: " );

12 scanf( "%ld", & sayi);

13 sonuc = fibonacci(sayi);

14 printf( "Fibonacci( %ld ) = %ld\n", sayi, sonuc);

15 return 0;

16 }

17

18 /\* Yinelemeli tanımlanmış fibonacci fonksiyonu \*/

19 long fibonacci( long n )

20 {

21 if ( n == 0 || n == 1 )

22 return n;

23 else

24 return fibonacci( n - 1 ) + fibonacci( n - 2 );

25 }

Bir tamsayı giriniz: 0

Fibonacci(0) = 0

174

Bir tamsayı giriniz: 1

Fibonacci(1) = 1

Bir tamsayı giriniz: 2

Fibonacci(2) = 1

Bir tamsayı giriniz: 3

Fibonacci(3) = 2

Bir tamsayı giriniz: 4

Fibonacci(4) = 3

Bir tamsayı giriniz: 5

Fibonacci(5) = 5

Bir tamsayı giriniz: 6

Fibonacci(6) = 8

Bir tamsayı giriniz: 10

Fibonacci(10) = 55

Bir tamsayı giriniz: 20

Fibonacci(20) = 6765

Bir tamsayı giriniz: 30

Fibonacci(30) = 832040

Bir tamsayı giriniz: 35

Fibonacci(35) = 9227465

Şekil 5.15 Yinelemeli olarak Fibonacci sayıları oluşturmak.

main içinden fibonacci fonksiyonuna yapılan çağrı yinelemeli çağrı değildir, ancak bundan

sonra fibonacci fonksiyonuna yapılan çağrıların hepsi yinelemelidir. fibonacci her

çağrıldığında temel durumu, yani n’in 0 ya da 1’e eşit oluşunu kontrol eder. Eğer bu

doğruysa, n döndürülür. İlginç bir şekilde eğer n 1’den büyükse, yineleme her biri

fibonacci’ye yapılan orijinal çağrıdan daha basit bir problem olan iki yinelemeli çağrı yapar.

Şekil 5.16, fibonacci fonksiyonunun fibonacci ( 3 ) değerini nasıl hesapladığını gösterir.

Şekil 5.16’da fibonacci yerine f yazarak şeklin daha kolay anlaşılmasını sağladık.

Bu şekil (Şekil 5.16), C derleyicilerinin, operatörlerin operandlarını hangi sırada

değerlendirecekleriyle ilgili ilginç konuları ortaya çıkartmaktadır. Bu, operatörlerin

operandlara hangi sırada uygulandıklarından, yani operatör önceliklerinden daha farklıdır.

Şekil 5.16, f ( 3 ) hesaplanırken iki yinelemeli çağrının ; f ( 2 ) ve f ( 1 )’in yapıldığını

göstermektedir. Fakat bu çağrılar hangi sırada yapılacaktır? Çoğu programcı operandların

soldan sağa doğru hesaplanacağını düşünecektir. Ancak ANSI standardı, çoğu operatörün

(+ operatörü de dahil olmak üzere) operandlarının değerlendirilme sıralarını belirtmemiştir.

Bu sebepten, programcı bu çağrıların çalıştırılma sıraları hakkında yorum yapamaz. Bu

çağrılar önce f ( 2 ) daha sonra f ( 1 ) çalıştırılarak ya da tam ters bir biçimde önce f ( 1 ) sonra

175

f ( 2 ) çalıştırılarak yapılabilir. Bu programda ve diğer bir çok programda en son sonuç aynı

olacaktır. Ancak bazı programlarda, bir operandın değerlendirilmesi ifadenin en son değerini

etkileyebilecek yan etkilere sebep olabilir. C’nin bir çok operatörü arasından, ANSI standardı

yalnızca 4 operatörün operandlarını değerlendirme sıralarını belirlemiştir. Bunlar &&, || ,

virgül operatörü ( , ) ve ?: operatörleridir. Bunlardan ilk üçü, ikili operatörlerdir ve

operandlarının soldan sağa doğru değerlendirilmesi garanti altına alınmıştır. Son operatör ise

C’nin üçlü tek operatörüdür. Bu operatörün en soldaki operandı her zaman ilk olarak ele

alınır; eğer en soldaki operand sıfır harici bir değer olarak hesaplanırsa, ortadaki operand

hesaplanır ve en son operand ihmal edilir , eğer en soldaki ifade 0 olarak hesaplanırsa en

sağdaki operand ele alınır ve ortadaki operand ihmal edilir.

Genel Programlama Hataları 5.17

&& , || , ?: ve virgül ( , ) operatörleri dışındaki operatörlerin operandlarını değerlendirme sıralarına

bağımlı olarak yazılan programlar hatalara sebep olabilir.Çünkü derleyiciler operandları

programcının beklediği gibi ele almayabilir.

Taşınırlık İpuçları 5.2

&& ,|| ,?: ve virgül(,) operatörleri dışındaki operatörlerin operandlarını değerlendirme sıralarına

bağımlı olarak yazılan programlar farklı sistemlerde ve farklı derleyicilerle farklı bir şekilde

çalışabilirler.

Burada yineleme programlarıyla ilgili dikkat edilecek bir noktadan bahsetmek istiyoruz.

fibonacci fonksiyonundaki her yineleme, çağrı sayılarını iki katına çıkartmaktadır yani n.

fibonacci sayısını bulmak için çalıştırılacak yineleme çağrısı sayısı 2n

dir. Bu çok hızlı bir

şekilde kontrolden çıkabilir. 20.fibonacci sayısını hesaplamak için 220 ya da milyonlarca çağrı

yapmak, 30.fibonacci sayısını hesaplamak için 230 çağrı ya da milyarlarca çağrı yapmak

gerekir. Bilgisayarla uğraşan bilim adamları bu üssel karışıklığa dikkati çekerler. Bu tarzdaki

problemler dünyanın en güçlü bilgisayarlarını bile zorlar! Genel karmaşa konuları ve üssel

karmaşa konuları genellikle yüksek seviyeli bilgisayar bilimlerinde “Algoritmalar” adlı

kurslarda tartışılır.

Performans İpuçları 5.4

Çağrıların üssel bir biçimde arttığı Fibonnacci tarzında yinelemeli programlardan kaçınınız.

f(3)

return f(2) + f(1)

return f(1) + f(0)

return 1

176

Şekil 5.16 fibonacci fonksiyonu için yineleme çağrıları

5.15 YİNELEME ve TEKRAR

Önceki kısımlarda kolaylıkla yinelemeli ya da tekrarlı bir biçimde gösterilebilecek iki

fonksiyon inceledik. Bu kısımda iki yaklaşımı karşılaştırarak, programcının özel durumlarda

hangisini diğerine tercih edebileceğini tartışacağız.

Yineleme ve tekrar bir kontrol yapısına dayanır : tekrar bir döngü yapısı, yineleme bir seçim

yapısı kullanır. Tekrar ve yinelemenin ikisi de döngü içerir. Tekrar özellikle döngü yapısını

kullanırken, yineleme döngüyü fonksiyon çağrılarının tekrarında kullanır. Tekrar ve

yinelemenin ikisi de bir sonlandırma testi içerirler. Yineleme temel bir durumla

karşılaşıldığında, tekrar ise döngü devam koşulu yanlış hale geldiğinde sona erer. Sayıcı

kontrollü döngü içeren tekrar ve yineleme yavaş yavaş sonlanmaya yaklaşır : tekrarlama

sayıcıyı, sayıcı döngü devam şartını yanlış hale getirecek bir değer alana kadar değiştirmeye

devam eder; yineleme ise temel duruma ulaşılıncaya kadar orijinal problemin daha basit

versiyonlarını yaratmaya devam eder. Tekrar ve yinelemenin ikisi de sonsuz olabilir: Sonsuz

bir döngü eğer döngü devam şartı asla yanlış hale gelmiyorsa ; sonsuz yineleme, yineleme

adımı problemi temel duruma yaklaştıracak biçimde indirgemiyorsa oluşur.

Yineleme bir çok negatif özelliğe sahiptir.Yineleme, mekanizmayı sürekli çağırarak

fonksiyon çağrılarının artmasına sebep olur. Bu, işlemci zamanı ve hafızada fazladan yük

demektir. Her yineleme çağrısı ,fonksiyonun başka bir kopyasının oluşmasına(aslında

yalnızca fonksiyonun değişkenlerinin) sebep olur, bu da hafızayı fazladan işgal etmek

demektir. Tekrar, genellikle bir fonksiyon içinde yer aldığından, fonksiyonların sürekli olarak

çağrılması ve fazladan hafıza kullanılması engellenir. O halde neden yineleme seçilsin?

Yazılım Mühendisliği Gözlemleri 5.12

Yinelemeli olarak çözülen her problem tekrarlı bir biçimde çözülebilir. Yineleme yaklaşımı genelde

problemi daha iyi yansıttığı ve daha kolay anlaşılan ve hataları kolay ayıklanan programlar

yazılmasını sağlattığı için, tekrar yaklaşımına göre tercih edilebilir. Yinelemeli çözümleri seçmenin

başka bir sebebi de tekrarlı çözümün kolaylıkla bulunamayışıdır.

Performans İpuçları 5.5

Performansın önemli olduğu durumlarda yinelemeden kaçının.Yineleme çağrıları fazladan vakit ve

hafıza gerektirir.

Genel Programlama Hataları 5.18

Yanlışlıkla, kendi kendini doğrudan ya da başka bir fonksiyon içinden çağıran, yinelemeli olmayan bir

fonksiyona sahip olmak.

Çoğu programlama kitabı, yinelemeyi bizim burada yaptığımızdan daha geç tanıtır.

Yinelemenin oldukça zengin ve karmaşık bir konu olduğunu düşündüğümüzden, bu konuyu

şimdi anlatmayı ve örnekleri kitabın geri kalanına yaymayı uygun gördük. Şekil 5.17, kitap

boyunca verilen 31 yineleme örneğinin hangi ünitelerde bulunduğunu özetlemektedir.

return 1 return 0

177

Bu üniteyi, kitap boyunca tekrar tekrar yaptığımız birkaç gözlemle kapatalım. İyi yazılım

mühendisliği önemlidir. Yüksek performans önemlidir. Ancak maalesef bu hedefler genelde

birbirlerini engellerler. İyi yazılım mühendisliği, ihtiyaç duyduğumuz daha büyük ve

karmaşık yazılım sistemlerinin geliştirilmesinde temel noktadır. Yüksek performans ise

gelecekteki donanım üzerinde daha fazla işlem gerektiren sistemlerin

gerçekleştirilebilmesindeki temel noktadır. Öyleyse fonksiyonlar nereye uymaktadır?

Yazılım Mühendisliği Gözlemleri 5.13

Programları düzgün,hiyerarşik bir düzende fonksiyonlardan oluşturmak iyi yazılım mühendisliğini

destekler. Ancak bunun da bir bedeli vardır.

Performans İpuçları 5.6

Fonksiyonların yoğun bir biçimde kullanıldığı programlar, fonksiyonların yer almadığı tek parça

programlarla karşılaştırıldığında, çok fazla sayıda fonksiyon çağrısı yapacaktır ve bu da bilgisayarın

işlemcisinin zamanını çok fazla alır. Ancak tek parça programları yazmak, test etmek, hatalarını

ayıklamak ve geliştirmek oldukça zordur.

Bu yüzden programlarınızı performans ve iyi yazılım mühendisliği arasındaki dengeyi göz

önünde tutarak fonksiyonelleştirin.

Ünite Yineleme Örnekleri ve Alıştırmaları

Ünite 5 Faktoriyel fonksiyonu

Fibonacci fonksiyonları

En büyük ortak bölen

İki tamsayının toplanması

İki sayının çarpılması

Bir tamsayının tamsayı kuvvetini bulmak

Hanoi’nin kuleleri

Yinelemeli main

Klavye girdilerini tersten yazdırmak

Yinelemeyi görselleştirmek

Ünite 6 Bir dizinin elemanlarını toplamak

Bir diziyi yazdırmak

Bir diziyi tersten yazdırmak

Bir stringin palindrom olup olmadığını kontrol etmek

Bir dizideki en küçük eleman

Seçimli sıralama

Hızlı sıralama

Lineer arama

İkili arama

Ünite 7 Sekiz vezir

Labirent yolculuğu

Ünite 8 Klavyeden girilen stringleri tersten yazdırmak

Ünite 12 Bağlı liste ekleme

Bağlı liste silme

Bağlı listede arama yapma

178

Bağlı bir listeyi tersten yazdırmak

İkili ağaç ekleme

İkili ağaçta öncesol ilerleme

İkili ağaçta önceçocuk ilerleme

İkili ağaçta öncedüğüm ilerleme

Şekil 5.17 Kitaptaki yineleme örnekleri ve alıştırmaları

ÖZET

 Büyük programlar geliştirmenin en iyi yolu, onu orijinal programdan daha kolay idare

edilebilir küçük program modüllerine bölmektir. Modüller C’de fonksiyon olarak

yazılırlar.

 Bir fonksiyon, fonksiyon çağrısıyla çağrılır. Fonksiyon çağrısı, fonksiyondan ismiyle

bahseder ve çağrılan fonksiyonun görevini yapabilmesi için ihtiyaç duyabileceği

bilgileri ( argüman ) sağlar.

 Fonksiyonlar için bilgi saklamanın amacı yalnızca görevlerini yerine getirmelerine

yarayacak bilgiler ulaşmalarıdır. Bu, iyi yazılım mühendisliğinin temel

prensiplerinden biri olan en az yetki prensibinin uygulanmasıdır.

 Fonksiyonlar bir programda fonksiyonun ismi ve ismin yanında, parantez içinde

argümanı (ya da virgüllerle ayrılmış argüman listesi) yazılarak çağrılır.

 double veri tipi, float veri tipi gibi ondalıklı bir veri tipidir. double tipte bir değişken

float ile tutulabilecek değerlerden daha büyük ve daha duyarlı değerler tutabilir.

 Bir fonksiyonun her argümanı bir sabit , deyim ya da değişken olabilir.

 Yerel bir değişken yalnızca fonksiyon tanımı içinde bilinir. Diğer fonksiyonların, bir

fonksiyonun yerel değişkenlerinin isimlerini bilmeye ve bir fonksiyonun, diğer bir

fonksiyonun uygulama detaylarını bilmeye hakkı yoktur.

Fonksiyon tanımının genel biçimi

geri\_dönüş\_değeri fonksiyon\_ismi (parametre\_listesi)

{

bildirimler

ifadeler

}

biçimindedir. Geri dönüş tipi, çağırıcı fonksiyona döndürülen sonucun veri tipini

gösterir. Eğer fonksiyon bir değer geri döndürmüyorsa, geri dönüş değeri void olarak

bildirilir. Fonksiyon ismi geçerli herhangi bir tanıtıcı olabilir. Parametre listesi,

fonksiyon çağrıldığında fonksiyonun alacağı parametrelerin bildirimlerini içeren

virgüllerle ayrılmış bir listedir. Eğer bir fonksiyon herhangi bir değer almıyorsa,

179

parametre listesi void olarak bildirilir. Fonksiyon gövdesi, fonksiyonu oluşturan

ifade ve bildirimlerin bir kümesidir.

 Fonksiyona geçirilen argümanlar, fonksiyon tanımındaki parametrelerle sayı,tip, ve

sıra bakımından uyuşmalıdır.

 Program fonksiyonla karşılaştığında, kontrol çağırma anından itibaren çağrılan

fonksiyona aktarılır, çağrılan fonksiyonun ifadeleri çalıştırılır ve kontrol çağırıcıya

döner.

 Bir fonksiyonun çağrıldığı yere geri dönmesini kontrol etmek için 3 yol vardır. Eğer

bir fonksiyon bir sonuç ile geri dönmeyecekse, kontrol, fonksiyonun en son

parantezine ulaşıldığında ya da

return;

ifadesinin çalıştırılmasıyla döndürülür.

Eğer fonksiyon bir sonuç ile geri dönecekse

return deyim;

ifadesi deyimin değerini çağırıcıya döndürür.

 Bir fonksiyon prototipi, fonksiyon tarafından döndürülen verinin tipini, fonksiyonun

almayı beklediği parametre sayısını, parametrelerin tiplerini ve parametrelerin sırasını

bildirir. Fonksiyonların ilk halleri (prototipleri), derleyicinin fonksiyon çağrılarının

doğru yapıldığını onaylamasına imkan sağlar.

 Derleyici, fonksiyon prototipi içindeki değişken isimlerini ihmal eder.

 Her standart kütüphane, o kütüphanedeki her fonksiyonun prototiplerinin yer aldığı ve

bu fonksiyonlar tarafından kullanılabilecek çeşitli veri tipleriyle bazı sabitlerin

bulunduğu bir öncü dosyaya sahiptir.

 Programcılar kendi öncü dosyalarını yaratabilir ve programlara dahil edebilirler.

 Argümanlar değere göre çağırma ile geçirilirse, değişkenin değerinin bir kopyası

oluşturulur ve çağırılan fonksiyona geçirilir. Oluşturulan kopyadaki değişiklikler

çağırıcıdaki orijinal değişkenin değerini etkilemez.

 C’deki tüm çağrılar değere göre çağırma ile yapılır.

 rand fonksiyonu 0 ile RAND\_MAX arsında bir tamsayı üretir ve RAND\_MAX

ANSI standardına göre en az 32767 olabilir.

 rand ve srand fonksiyonlarının prototipleri <stdlib.h> içindedir.

 Bir programı rassallaştırmak için C standart kütüphane fonksiyonu olan srand

fonksiyonunu kullanın.

 srand ifadeleri, programın hataları tamamıyle ayıklandıktan sonra programa her

zamanki gibi yerleştirilir. Hata ayıklama esnasında srand fonksiyonunu çıkartın. Bu,

tekrarlamayı garanti altına alır. Bu, rasgele sayı üretme programında yapılan

değişikliklerin doğru bir biçimde çalıştığını ispatlamak için oldukça önemlidir.

 Eğer her seferinde besleme girmeden rassallaştırma yapmak istiyorsak aşağıdaki gibi

bir ifade kullanabiliriz:

srand ( time ( NULL ) );

time fonksiyonu o andaki saati saniye biçiminde oluşturur. time fonksiyonunun

prototipi <time.h> içindedir.

Rasgele bir sayıyı kaydırmak ve derecelendirmek için genel denklem

180

n = a + rand( ) % b;

biçimindedir. Burada a kaydırma değeridir.(ard arda gelen tamsayılardan oluşan

istenen aralığın başlangıç değeridir) b ise derecelendirme faktörüdür. (ard arda gelen

tamsayılardan oluşan istenen aralığın genişliğidir)

 Bir programdaki her tanıtıcı depolama sınıfı,depolama süreci,faaliyet alanı ve bağlama

özelliklerine sahiptir.

 C, auto, register,extern ve static depolama sınıfı belirteçleriyle belirlenebilen dört

depolama sınıfına sahiptir.

 Bir tanıtıcının depolama süreci, o tanıtıcının hafızada tutulduğu zaman aralığıdır.

 Bir tanıtıcının faaliyet alanı, tanıtıcının program içinde kullanılabileceği yerlerdir

 Bir tanıtıcının bağlaması, çok kaynaklı bir program içinde tanıtıcının yalnızca o andaki

kaynak dosyada mı yoksa uygun bildirimlerle herhangi bir kaynak dosyada mı geçerli

olacağını belirler.

 Otomatik depolama süreçli değişkenler, bildirildikleri blok içine girildiğinde yaratılır,

blok aktif iken varolur ve bloktan çıkıldığında yok edilirler. Bir fonksiyonun yerel

değişkenleri ,otomatik depolama sürecine sahiptirler.

 register depolama sınıfı belirteci, otomatik değişken bildirimden önceye

yerleştirilerek derleyiciye değişkeni bilgisayarın yüksek hızlı donanım yazmaçlarından

birine yerleştirmesi önerilebilir. Derleyici, register bildirimlerini ihmal edebilir.

register anahtar kelimesi yalnızca otomatik depolama süreçli değişkenlerle

kullanılabilir.

 extern ve static anahtar kelimeleri, statik depolama süreçli değişken ve fonksiyon

tanıtıcıları bildirmek için kullanılırlar.

 Statik depolama süreçli tanıtıcılar, program çalışmaya başladığı andan itibaren var

olurlar.

 Statik depolama zamanlı iki tip tanıtıcı vardır: dış tanıtıcılar (global değişkenler ve

fonksiyon isimleri gibi) ve static depolama zamanı belirteciyle bildirilmiş yerel

değişkenler.

 Global değişkenler, değişken bildirimleri herhangi bir fonksiyon tanımının dışında

yapılarak oluşturulurlar ve değerlerini programın çalışma zamanı boyunca korurlar.

 static olarak bildirilmiş yerel değişkenler, fonksiyondan çıkıldıktan sonrada

değerlerini korurlar.

 Statik depolama zamanlı tüm nümerik değişkenler programcı tarafından özel olarak

bir başka değere atanmadıkça ilk değer olarak 0'a atanırlar.

 Bir tanıtıcının 4 faaliyet alanı vardır. Bunlar: fonksiyon faaliyet alanı,dosya faaliyet

alanı,blok faaliyet alanı ve fonksiyon prototipi faaliyet alanıdır.

 Etiketler, fonksiyon faaliyet alanına sahip tek tanıtıcıdır. Etiketler, bulundukları

fonksiyon içinde her yerde kullanılabilirler ancak fonksiyon gövdesi dışında

kullanılamazlar.

 Herhangi bir fonksiyonun dışında bildirilmiş tanıtıcılar, dosya faaliyet alanına

sahiptir.Bu tarzda bir tanıtıcı, bildirildiği yerden dosyanın sonuna kadar tüm

fonksiyonlar tarafından bilinir.

 Bir blok içinde bildirilmiş tanıtıcılar, blok faaliyet alanına sahiptir.Blok faaliyet alanı

bloğu sonlandıran küme parantezine( } ) ulaşıldığında sona erer

 Fonksiyonun başında bildirilen yerel değişkenler, fonksiyon parametreleri gibi blok

faaliyet alanına sahiptir ve fonksiyon için yerel değişkenler olarak bilinirler.

 Herhangi bir blok, değişken bildirimleri içerebilir.Bloklar yuvalandığında ve dıştaki

bloktaki tanıtıcı içteki bloktaki tanıtıcıyla aynı isme sahip olduğunda,dış bloktaki

tanıtıcı iç blok sona erene kadar saklanır.

181

 Fonksiyon prototipi faaliyet alanına sahip tek tanıtıcı, fonksiyon prototipinin

parametre listesinde kullanılan tanıtıcılardır.Fonksiyon prototipinde kullanılan

tanıtıcılar, programın herhangi bir yerinde karışıklık olmadan yeniden kullanılabilir.

 Bir yineleme fonksiyonu (recursive function), kendi kendini doğrudan ya da bir başka

fonksiyon içinden çağıran fonksiyondur

 Eğer yinelemeli bir fonksiyon temel bir durumla çağrılırsa, fonksiyon bir sonuç geri

döndürür. Eğer yinelemeli bir fonksiyon daha karmaşık bir problemle çağrılırsa

fonksiyon problemi iki kavramsal parçaya ayırır : Fonksiyonun nasıl yapacağını

bildiği parça ve fonksiyonun nasıl yapacağını bilmediği parça. Bu yeni problem

orijinal probleme benzediğinden fonksiyon bu küçük problem üzerinde çalışmak için

yeni bir kopyasını çağırır.

 Yinelemeden çıkılabilmesi için , her seferinde fonksiyon kendini problemin biraz daha

basit versiyonuyla çağırır, bu basit ve daha basit problemlerin dizisi en sonunda temel

duruma ulaşmalıdır. Bu noktada fonksiyon temel durumu tanır, fonksiyonun bir

önceki kopyasına bir sonuç aktarır ve sonuçların döndürüldüğü bir dizi, fonksiyonun

orijinal çağrısının en son sonucu döndürmesine kadar yukarıya doğru devam eder.

 ANSI standardı, çoğu operatörün(+ operatörü de dahil olmak üzere) operandlarının

değerlendirilme sıralarını belirtmemiştir. C 'in bir çok operatörü arasından standart

yalnızca 4 operatörün operandlarını değerlendirme sıralarını belirlemiştir. Bunlar &&,

|| , virgül operatörü(,) ve ?: operatörleridir. Bunlardan ilk üçü ikili operatörlerdir ve

operandlarının soldan sağa doğru değerlendirilmesi garanti altına alınmıştır. Son

operatör ise C 'in üçlü tek operatörüdür. Bu operatörün en soldaki operandı her zaman

ilk olarak ele alınır; eğer en soldaki operand sıfır harici bir değer olarak

hesaplanırsa,ortadaki operand hesaplanır ve en son operand ihmal edilir;eğer en

soldaki ifade 0 olarak hesaplanırsa en sağdaki operand ele alınır ve ortadaki operand

ihmal edilir.

 Yineleme ve tekrar bir kontrol yapısına dayanır: tekrar bir döngü yapısı,yineleme bir

seçim yapısı kullanır.

 Tekrar ve yinelemenin ikisi de döngü içerir. Tekrar özellikle döngü yapısını

kullanırken, yineleme döngüyü fonksiyon çağrılarının tekrarında kullanır.

 Tekrar ve yinelemenin ikisi de bir sonlandırma testi içerirler: yineleme temel bir

durumla karşılaşıldığında, tekrar döngü devam koşulu yanlış hale geldiğinde sona erer.

 Tekrar ve yinelemenin ikisi de sonsuz olabilir : Sonsuz bir döngü eğer döngü devam

şartı asla yanlış hale gelmiyorsa;sonsuz yineleme, yineleme adımı problemi temel

duruma yaklaştıracak biçimde indirgemiyorsa oluşur.

 Yineleme, mekanizmayı sürekli çağırarak fonksiyon çağrılarının artmasına sebep olur.

Bu işlemci zamanı ve hafızada fazladan yük demektir.

ÇEVRİLEN TERİMLER

abstraction....................................... özetleme

automatic storage............................ otomatik depolama

automatic variable.......................... otomatik değişken

auto storage class specifier............ auto depolama sınıfı belirteci

base case in recursion.................... yinelemede temel durum

block scope................................... blok faaliyet alanı

call by reference......................... referansa göre çağırma

call by value................................. değere göre çağırma

caller........................................... çağırıcı

182

calling function.......................... çağırıcı fonksiyon

file scope ................................. dosya faaliyet alanı

function call............................. fonksiyon çağrısı

function declaration................ fonksiyon bildirimi

function definition................. fonksiyon tanımı

function prototype................. fonksiyonun ilk hali/prototipi

global variable..................... global değişken (genel değişken olarak da bilinir)

header file................................. öncü dosya

information hiding.................... bilgi saklama/gizleme

linkage.................................... bağlama

local variable....................... yerek değişken

optimizing compiler............. derleyiciyi en iyi hale getirmek

randomize............................... rassallaştırmak

recursion................................ yineleme

return value............................ geri dönüş değeri

simulation.............................. simülasyon/benzetme

GENEL PROGRAMLAMA HATALARI

5.1 Matematik kütüphane fonksiyonlarını kullanırken, matematik öncü dosyasını eklemeyi

unutmak garip sonuçlara yol açabilir.

5.2 Fonksiyon tanımlamalarında geri dönüş değerini unutmak, eğer fonksiyonun ilk hali

(prototipi) int ipinden başka bir geri dönüş tipi ile belirtilmişse yazım hatası oluşturur.

5.3 Bir değer ile dönmesi beklenen bir fonksiyonun geri dönüş değerinin belirtilmemesi

beklenmeyen hatalara yol açabilir.ANSI standardı, bu ihmalin sonuçlarını belirlememiştir.

5.4 Geri dönüş tipi void olarak bildirilmiş bir fonksiyonun bir değer geri döndürmesi bir

yazım hatasıdır.

5.5 Aynı tipte fonksiyon parametrelerini double x, double y yerine double x,y olarak

bildirmek. double x, y biçiminde parametre bildirmek, y parametresinin tipinin int

olmasına sebep olur. Çünkü belirtilmeyen parametre tipi otomatik olarak int tipinde

varsayılır.

5.6 Parametre listesini yazdığımız parantezlerin dışına noktalı virgül koymak yazım

hatasıdır.

5.7 Bir fonksiyon parametresini daha sonradan fonksiyon içinde yerel bir değişken olarak

kullanmak bir yazım hatasıdır.

5.8 Bir fonksiyon içinde başka bir fonksiyon tanımlamak yazım hatasıdır.

5.9 Fonksiyon prototipinin sonuna noktalı virgül koymamak bir yazım hatasıdır.

5.10 Dönüşüm hiyerarşisinde yüksek bir veri tipi daha düşük bir veri tipine dönüştürülür ise

verinin değeri değişebilir.

5.11 Fonksiyon prototipinin unutulması, eğer fonksiyonun geri dönüş tipi int değilse ve

fonksiyon tanımı fonksiyon çağrısından daha sonra bulunmuyorsa yazım hatalarına

sebep olur. Aksi takdirde, fonksiyon prototipini unutmak çalışma zamanlı ya da

beklenmeyen hatalara yol açabilir.

5.12 Rasgele sayılar üretirken rand yerine srand kullanmak.

5.13 Bir tanıtıcı için birden çok depolama sınıfı belirteci kullanmak. Bir değişkene yalnızca

bir depolama sınıfı belirteci uygulanabilir.

5.14 Programcı dış bloktaki tanıtıcının iç blok çalışırken aktif olmasını isterken, yanlışlıkla iç

blokta kullandığı tanıtıcı ismiyle dış blokta kullandığı tanıtıcı isminin aynı olması.

5.15 İhtiyaç duyulmasına rağmen bir yinelemeli fonksiyondan değer geri döndürmeyi

unutmak.

183

5.16 Temel durumu dahil etmemek ya da yineleme adımını temel duruma ulaşmayacak yanlış

bir biçimde yazmak, neticede hafızayı yoran sonsuz yineleme yaratır.Bu, yinelemeli

olmayan bir çözümde sonsuz döngü problemiyle eşdeğerdir.Sonsuz yineleme

beklenmeyen bir giriş yapıldığında da oluşabilir.

5.17 &&,||,?: ve virgül(,) operatörleri dışındaki operatörlerin operandlarını değerlendirme

sıralarına bağımlı olarak yazılan programlar hatalara sebep olabilir.Çünkü derleyiciler

operandları programcının beklediği gibi ele almayabilir.

5.18 Yanlışlıkla, kendi kendini doğrudan ya da başka bir fonksiyon içinden çağıran,

yinelemeli olmayan bir fonksiyona sahip olmak.

İYİ PROGRAMLAMA ALIŞTIRMALARI

5.1 ANSI C standart kütüphanesi içindeki fonksiyonları dikkatlice inceleyin.

5.2 Bir programda matematik kütüphanesi fonksiyonları kullanıyorsak, programımızın başına

#include <math.h> önişlemci komutunu yazarak matematik öncü(header) dosyasını

programımıza katmalıyız.

5.3 Programın okunurluğunu arttırmak ve fonksiyonları ayırmak için fonksiyon tanımlarından

önce bir satır boşluk bırakmak.

5.4 Geri dönüş tipi ihmal edildiğinde derleyici, geri dönüş tipini int olarak belirlese de her

zaman geri dönüş tipini belirleyiniz. Ancak, main fonksiyonunun geri dönüş tipi normal

olarak ihmal edilebilir

5.5 Parametre listesindeki tüm parametrelerin tipini ,belirtilmeyenler otomatik olarak int

tipinde kullanılacak olsa da mutlaka belirtiniz.

5.6 Yanlış olmasa da, fonksiyona aktarılan argümanlarla bu argümanların yerine kullanılacak

parametrelerin aynı isimde olmamasına özen gösteriniz.Bu, belirsizlikten kurtulmamızı

sağlar.

5.7 Anlamlı fonksiyon isimleri ve anlamlı parametre isimleri kullanmak programları daha

okunur yapar ve yorumların çok fazla kullanılmasını engeller.

5.8 C’nin kontrol yeteneklerinden faydalanabilmek için tüm fonksiyonların ilk hallerini

(prototiplerini) programa dahil etmeliyiz. Uygun kütüphanelerdeki öncü dosyalardan,

standart kütüphane fonksiyonlarının ilk hallerini(prototiplerini) elde etmek için #include

önişlemci komutlarını kullanın. Ayrıca siz ve/ve ya arkadaşlarınızın kullandığı fonksiyon

prototiplerini içeren öncü dosyaları elde etmek içinde #include kullanın.

5.9 Parametre isimleri, belgeleme amaçlı olarak fonksiyon prototipleri içinde yazılabilir.

Derleyici bu isimleri ihmal eder.

5.10Yalnızca belli bir fonksiyonda kullanılan değişkenler, o fonksiyon içinde yerel olarak

bildirilmelidir.

5.11Dış faaliyet alanlarında isimleri gizleyen değişken isimlerinden kaçının.Bu bir

programda aynı tanıtıcı bir kez daha kullanmayarak sağlanabilir.

TAŞINIRLIK İPUÇLARI

5.1 ANSI C standart kütüphanesi içindeki fonksiyonları kullanmak daha taşınır programlar

yazmamıza yardımcı olur.

5.2 &&,||,?: ve virgül(,) operatörleri dışındaki operatörlerin operandlarını değerlendirme

sıralarına bağımlı olarak yazılan programlar farklı sistemlerde ve farklı derleyicilerle

farklı bir şekilde çalışabilirler.

184

PERFORMANS İPUÇLARI

5.1 Otomatik depolama hafızayı korumak için kullanılır.Çünkü otomatik değişkenler

yalnızca ihtiyaç duyulduklarında varolurlar.Bunlar bildirildikleri fonksiyon

çalıştırıldığında yaratılır, fonksiyonun çalıştırılması sona erdiğinde yok edilirler.

5.2 register depolama sınıfı belirteci, otomatik değişken bildiriminden önceye yerleştirilerek

derleyiciye değişkeni bilgisayarın yüksek hızlı donanım yazmaçlarından birine

yerleştirmesi önerilebilir.

5.3 Genellikle register bildirimleri gereksizdir. Bugünkü derleyicilerin bir çoğu sıklıkla

kullanılan değişkenleri tanıyıp, programcının register bildirimi yapmasına gerek

kalmadan değişkeni yazmaçlardan birinin içine koymaya karar verir.

5.4 Çağrıların üssel bir biçimde arttığı Fibonnacci tarzında yinelemeli programlardan

kaçınınız.

5.5 Performansın önemli olduğu durumlarda yinelemeden kaçının.Yineleme çağrıları fazladan

vakit ve hafıza gerektirir.

5.6 Fonksiyonların yoğun bir biçimde kullanıldığı programlar, fonksiyonların yer almadığı tek

parça programlarla karşılaştırıldığında, çok fazla sayıda fonksiyon çağrısı yapacaktır ve bu

da bilgisayarın işlemcisinin zamanını çok fazla alır.Ancak tek parça programları

yazmak,test etmek, hatalarını ayıklamak ve geliştirmek oldukça zordur.

YAZILIM MÜHENDİSLİĞİ GÖZLEMLERİ

5.1 Tekerleği yeniden icat etmekten kaçının.Mümkün olduğunda yeni fonksiyonlar yazmak

yerine ANSI C standart kütüphanesi içindeki fonksiyonları kullanın.Bu, program

geliştirme zamanını azaltacaktır.

5.2 Birden fazla fonksiyon kullanılan programlarda, main fonksiyonu programın esas

görevini yerine getiren fonksiyonların çağırıcısı olarak kullanılmalıdır.

5.3 Her fonksiyon, iyi olarak tanımlanmış tek bir işi yapacak şekilde sınırlandırılmalıdır ve

fonksiyon ismi, fonksiyonun görevini etkili bir biçimde açıklamalıdır.Bu, özetlemeyi ve

yazılımın yeniden kullanılabilirliğini sağlatır.

5.4 Eğer fonksiyonun görevini açıklayacak etkili bir isim bulamıyorsanız muhtemelen

yazdığınız fonksiyon birden fazla görevi yerine getirmeye çalışmaktadır.Bu tarzda

fonksiyonları daha küçük fonksiyonlara bölmek en iyi yoldur.

5.5 Bir fonksiyon genellikle bir sayfadan daha uzun olmamalıdır.Hatta en iyisi yarım

sayfadan uzun olmamalıdır.Küçük fonksiyonlar yazılımın yeniden kullanılabilmesini

sağlatır.

5.6 Programlar, küçük fonksiyonların bir araya getirilmesiyle yazılmalıdır.Bu, programların

daha kolay yazılması,değiştirilmesi ve hatalarının giderilmesini sağlar.

5.7 Çok fazla sayıda parametreye ihtiyaç duyan fonksiyonlar birden fazla görevi yerine

getiriyor olabilir. Böyle fonksiyonları ayrı görevleri gerçekleştiren daha küçük

fonksiyonlara bölmek gerekir. Fonksiyonun başlığı mümkünse bir satıra sığmalıdır.

5.8 Fonksiyonun ilk hali(prototipi) ,fonksiyonun başlığı ve fonksiyon çağrısı argüman ve

parametre sayısı,tipi ve sırasıyla, geri dönüş değerinin tipi bakımından uyumlu olmalıdır.

5.9 Herhangi bir fonksiyon tanımı dışına yerleştirilmiş fonksiyon prototipi, dosyada fonksiyon

prototipinin yazıldığı yerden itibaren fonksiyonun tüm çağrılarında geçerli olur.Eğer

fonksiyon prototipi fonksiyon tanımının içinde yer alırsa sadece o fonksiyon içinden

yapılan çağrılara uygulanır.

5.10 Otomatik depolama, en az yetki prensibinin bir başka örneğidir. Değişkenler neden

hafızada depolansın ve neden gerçekte ihtiyaç duyulmamalarına rağmen

erişilebilsin?Bir programın makine dili versiyonunda veri, hesaplamalar ve diğer

185

işlemler için genellikle yazmaçlara ( register ) yüklenir.

5.11 Bir değişkeni yerel değil de global olarak bildirmek, bir değişkene erişmemesi gereken

bir fonksiyonun değişkeni yanlışlıkla değiştirmesi gibi istenmeyen yan etkilere sebep

olabilir.Genelde bazı belirli ve çok özel durumlar hariç(14.Ünitede anlatıldığı gibi) global

değişkenlerin kullanımından kaçınılmalıdır.

5.12inelemeli olarak çözülen her problem tekrarlı bir biçimde çözülebilir. Yineleme yaklaşımı

genelde problemi daha iyi yansıttığı ve daha kolay anlaşılan ve hataları kolay ayıklanan

programlar yazılmasını sağlattığı için, tekrar yaklaşımına göre tercih edilebilir.

Yinelemeli çözümleri seçmenin başka bir sebebi de tekrarlı çözümün kolaylıkla

bulunamayışıdır.

5.13Programları düzgün, hiyerarşik bir düzende fonksiyonlardan oluşturmak iyi yazılım

mühendisliğini destekler.Ancak bunun da bir bedeli vardır.

ÇÖZÜMLÜ ALIŞTIRMALAR

5.1 Aşağıdaki boşlukları doldurunuz.

a) C’de bir program modülüne \_\_\_\_\_\_\_\_ denir.

b) Fonksiyon \_\_\_\_\_\_\_\_ ile çağrılır.

c) Sadece bildirildiği fonksiyon içerisinde kullanılan değişkene \_\_\_\_\_\_\_\_ değişken

denir.

d) Daha önceden çağrılmış bir fonksiyonun içersindeki \_\_\_\_\_\_\_\_ ifadesi, fonksiyonun

çağrıldığı yere bir değer gönderir.

e) \_\_\_\_\_\_\_\_ anahtar kelimesi, bir fonksiyonun başlığında kullanıldığında bir değer

göndermeyeceği ya da hiç bir parametre içermediği anlamına gelir.

f) Bir tanıtıcının\_\_\_\_\_\_\_\_, o tanıtıcının program içerisinde kullanılabilceği bölümü

gösterir.

g) Çağrılmış bir fonksiyonda kontrolü çağırıcıya göndermenin üç yolu \_\_\_\_\_\_\_\_,

\_\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_\_dır.

h) \_\_\_\_\_\_\_\_, derleyicinin argüman sayısını, sırasını ve türünü kontrol etmesini sağlar.

i) \_\_\_\_\_\_\_\_ fonksiyonu, rasgele sayı üretmeye yarar.

j) \_\_\_\_\_\_\_\_ fonksiyonu, rasgele bir programı rassallaştırmak için bir sayı üretmede

kullanılır.

k) Depolama sınıfı belirteçleri \_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_\_ dır.

l) Bir blok içerisinde ya da fonksiyonun parametre listesinde bildirilen değişkenler aksi

belirtilmediği takdirde \_\_\_\_\_\_\_\_ depolama sınıfından kabul edilirler.

m) \_\_\_\_\_\_\_\_ depolama sınıf belirteci, derleyiciye bir değişkeni bilgisayarın bir

yazmaçlarında saklamasını söyler.

n) Herhangi bir bloğun veya fonksiyonun dışında bildirilen değişkene \_\_\_\_\_\_\_\_ denir.

o) Bir fonksiyonda bildirilen yerel bir değişkenin değerini fonksiyonun çağrıldığı yerler

arasında da koruması için \_\_\_\_\_\_\_\_ depolama sınıfı belirteci ile bildirilmelidir.

p) Bir tanıtıcının muhtemel dört faaliyet alanı \_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_ ve

\_\_\_\_\_\_\_\_dir.

q) Kendi kendini, doğrudan ya da dolaylı olarak çağıran fonksiyona \_\_\_\_\_\_\_\_ fonksiyon

denir.

186

r) Tipik bir yineleme fonksiyonunun iki bileşeni vardır. Biri \_\_\_\_\_\_\_\_ durumu kontrol

ederek yinelemenin sonlandılıp sonlandırılmayacağına karar verir. Diğeri ise, bir çok

zor problemin kolayca çözülebilmesi için yineleme olayının getçekleştirilmesini

sağlar.

5.2 Aşağıdaki programda, şıklarda verilen değişkenlerin faaliyet alanlarını bulunuz.

a) main fonksiyonundaki x değişkeni

b) kup fonksiyonundaki y değişkeni

c) kup fonksiyonu

d) main fonksiyonu

e) kup fonksiyon prototipi

f) kup fonksiyon prototipindeki y tanıtıcısı

1 #include <stdio.h>

2 int kup(int y);

3

4 int main( )

5 {

6 int x;

7

8 for(x = 1; x <= 10; x++)

9 prinft ("%d\n", cube( x ) );

10 return 0;

11 }

12

13 int kup(int y)

14 {

15 return y \* y \* y;

16 }

5.3 Şekil 5.2’deki matematik fonksiyonlarının, gösterdiği sonuçları verip vermediğini kontrol

eden bir program yazınız.

5.4 Aşağıdaki fonksiyonlar için fonksiyon başlıklarını yazın.

a) kenar1 ve kenar2 isminde, double tipinde iki argümanı alan ve bir double tipinde

sayı döndüren hipotenus fonksiyonu.

b) x, y, z adında üç tamsayı alan ve bir tamsayı döndüren enKucuk fonksiyonu.

c) Hiç bir argümanı olmayan ve hiç bir şey döndüren secim fonksiyonu.(Not: Bu tarz

fonksiyonlar genellikle kullanıcıya ilgili seçenekleri göstermek için kullanılır.)

d) sayi tamsayı argümanını alan ve bir ondalıklı sayı döndüren tamsayidanOndaliga

fonksiyonu.

5.5 Aşağıdaki fonksiyonların prototiplerini yazınız.

a) Alıştırma 5.4a’deki fonksiyon.

b) Alıştırma 5.4b’deki fonksiyon.

c) Alıştırma 5.4c’deki fonksiyon.

d) Alıştırma 5.4d’deki fonksiyon.

187

5.6 Aşağıdakiler için birer bildirim yazınız.

a) İlk değeri 0 olan ve bir yazmaç içerisinde saklanacak olan sayici değişkeni.

b) Fonksiyon çağrılmaları arasında da değerini koruyan ondalıklı sayı değişkeni,

sonDeger

c) Faaliyet alanı, bildirildiği dosyanın geri kalanına kısıtlanmış dış tamsayı değişkeni

sayi

5.7 Aşağıdaki program parçacıklarındaki hatayı bulun ve nasıl düzeltileceğini

açıklayın(Alıştırma 5.50 ‘yi de dikkate alın)

a) int g(void) {

printf ("g Fonksiyonun içi\n");

int h(void) {

printf ("h fonksiyonun içi\n);

}

}

b) int toplam(int x, int y) {

int sonuc;

sonuc = x + y;

}

c) int toplam(int n) {

if (n == 0)

return 0;

else

n + toplam(n-1);

}

d) void f(float a); {

float a;

printf ("%f", a);

}

e) void carpim(void) {

int a, b, c, sonuc;

printf ("Üç tamsayı girin.: ");

scanf("%d%d%d", &a, &b, &c);

sonuc = a \* b \* c;

printf ("Sonuc %d", sonuc);

return sonuc;

}

Çözümler

188

5.1 a) fonksiyon b) fonksiyon çağırma c) yerel değişken d) return e) void f) faaliyet alanı g)

return ya da return deyim; yada sol küme parantezinin kullanılması h) fonksiyon prototipi i)

rand j) srand k) auto, register, extern, static l) otomatik m) register n) dış o) static f)

fonksiyon faaliyet alanı, dosya faaliyet alanı, blok faaliyet alanı, fonksiyon prototip faaliyet

alanı, q) yineleme r) temel

5.2 a) blok faaliyet alanı b) blok faaliyet alanı c) dosya faaliyet alanı d) dosya faaliyet alanı e)

dosya faaliyet alanı f) fonksiyon prototipi faaliyet alanı

5.3

1/\*Matematik kütüphanesi fonksiyonları testi \*/

2 #include <stdio.h>

3 #include <math.h>

4

5 int main()

6 {

7 printf( “sqrt(%.1f) = %.1f\n”, 900.0, sqrt(900.0));

8 printf(“ sqrt(%.1f) = %.1f\n”, 9.0, sqrt(9.0));

9 printf(“exp(%.1f) = %f\n”, 1.0, exp(1.0));

10 printf(“exp(%.1f) = %f\n”, 2.0, exp(2.0));

11 printf(“log(%f) = %.1f”, 2.718282, log(2.718282));

12 printf(“log(%f) = %.1f”, 7.389056, log(7.389056));

13 printf(“log10(%.1f) = %.1f\n”, 1.0, log10(1.0));

14 printf(“log10(%.1f) = %.1f\n”, 10.0, log10(10.0));

15 printf(“log10(%.1f) = %.1f\n”, 100.0, log10(100.0));

16 printf(“fabs(%.1f) = %.1f\n”, 13.5, fabs(13.5));

17 printf(“fabs(%.1f) = %.1f\n”, 0.0, fabs(0.0));

18 printf(“fabs(%.1f) = %.1f\n”, -13.5, fabs(-13.5));

19 printf(“ceil(%.1f) = %.1f\n”, 9.2, ceil(9.2));

20 printf(“ceil(%.1f) = %.1f\n”, -9.8, ceil(-9.8));

21 printf(“floor(%.1f) = %.1f\n”, 9.2, floor(9.2));

22 printf(“floor(%.1f) = %.1f\n”, -9.8, floor(-9.8));

23 printf(“pow(%.1f, %.1f) = %.1f\n”, 2.0, 7.0,

24 pow(2.0, 7.0));

25 printf(“pow(%.1f, %.1f) = %.1f\n”, 9.0, 0.5,

26 pow(9.0, 0.5));

27 printf(“fmod(%.3f / %.3f) = %.13\n”,

28 13.675, 2.333, fmod(13.675, 2.333));

29 printf(“sin(%.1f) = %.1f\n”, 0.0, sin(0.0));

30 printf(“cos(%.1f) = %.1f\n”, 0.0, sin(0.0));

31 printf(“tan(%.1f) = %.1f\n”, 0.0, sin(0.0));

32 return 0;

33 }

sqrt(900.0) = 30.0

sqrt(9.0) = 3.0

189

exp(1.0) = 2.718282

exp(2.0) = 7.389056

log(2.718282) = 1.0

log(7.389056) = 2.0

log10(1.0) = 0.0

log10(10.0) = 1.0

log10(100.0) = 2.0

fabs(13.5) = 13.5

fabs(0.0) = 0.0

fabs(-13.5) = 13.5

ceil(9.2) = 10.0

ceil(-9.8) = -9.0

floor(9.2) = 9.0

floor(-9.8) = -10.0

pow(2.0, 7.0) = 128.0

pow(9.0, 0.5) = 3.0

fmod(13.675 / 2.333) = 2.010

fsin(0.0) = 0.0

fcos(0.0) = 1.0

ftan(0.0) = 0.0

5.4

a) double hipotenus(double kenar1, double kenar2)

b) int enKucuk(int x, int y, int z)

c) void komut(void)

d) float tamsayidanOndaliga(int sayi)

5.5

a) double hipotenus(double, double);

b) int enKucuk(int, int, int);

c) void komut(void);

d) float tamsayidanOndaliga(int);

5.6

a) register int sayac = 0;

b) static float sonDeger;

c) static int sayi;

Not: Bu herhangi bir fonksiyonun tanımlanmasının dışında olmalıdır.

5.7

a) Hata: h fonksiyonu g fonksiyonunun içinde tanımlanmıştır.

Düzeltme: h fonksiyonunun tanımlanmasının g fonksiyonunun dışında yapılması

b) Hata :fonksiyonun bir tamsayı döndürmesi gerekirdi.

Düzeltme: sonuc değişkeninin silinmesi ve fonksiyona onun yerine

return x + y

konulması

c) Hata: n + toplam(n - 1) sonucu fonksiyondan döndürülmemiş. toplam 'la doğru

olmayan bir sonuç geri döner.

Düzeltme : else içerisine

190

return n + toplam( n – 1 )

yazılmalı

d) Hata: parametre listesinin sağ parantezinden sonraki ; ve a parametresinin fonksiyon

tanımlanmasında tekrar tanımlanması

Düzeltme: parametre listesinin sağındaki ; ve float a bildiriminin silinmesi

e) Hata: fonksiyonun istenen bir değeri geri döndürmemesi. Düzeltme: return ifadesinin

silinmesi

ALIŞTIRMALAR

5.8 Aşağıdaki ifadeler sonucunda x 'in alacağı değerleri bulunuz.

a) x = fabs(7.5)

b) x = floor(7.5)

c) x = fabs(0.0)

e) x = ceil(0.0)

f) x = ceil (-6.4)

g) x = ceil(-fabs(-8+floor(-5.5)))

5.9 Bir park yeri işletmesi, 3 saate kadar yapılan parklar için minimum 2.00$ almaktadır. 3

saatin geçilmesi halinde her saat için 0.5$ almaktadır. Maksimum alınan para ise 24 saatlik

park edilmesi halinde 10.00$'dır. 24 saatten daha uzun hiç bir aracın park etmediğini kabul

ediniz. Dün bu park yerine park etmiş olan üç müşterinin ödemelerini hesaplayan ve yazdıran

bir programı yazınız.(Kullanıcı, her müşterinin park saatini girmeli ve çıktı düzgün bir çizelge

şeklinde olmalıdır ve dünün toplam gelirini hesaplamalıdır. Program ucretHesapla adı altına

bir fonksiyonda her müşteri için ücreti hesaplamalı ve çıktı aşağıdaki gibi olmalıdır.)

Araba Saat Ücret

1 1.5 2.00

2 4.0 2.50

3 24.0 10.00

TOPLAM 29.5 14.50

5.10 floor fonksiyonu bir ondalıklı sayıyı en yakın tamsayıya yuvarlar.

y = floor(x + .5);

ifadesi x'i en yakın tam sayıya yuvarlar ve y'ye atar. Kullanıcıdan bir kaç sayı alan ve

yukarıdaki ifadeyle bu sayıları yuvarlayan hem orijinal sayıyı hem de yuvarlanmış sayıyı

ekrana yazdıran bir program yazınız.

5.11 floor fonksiyonu, bir sayıyı istenilen ondalık seviyede yuvarlamada da kullanılır.

y = floor ( x \* 10 + .5) / 10;

x 'i onluk pozisyonda yuvarlar (ondalık kısmın sağdan birinci pozisyonu)

y = floor ( x \* 100 + .5) / 100;

191

x 'i yüzlük pozisyonda yuvarlar (ondalık kısmın sağdan ikinci pozisyonu)

x 'i aşağıdaki fonksiyonlarla yuvarlayan bir program yazınız.

a) tamsayiyayuvarla(sayi)

b) onlugayuvarla(sayi)

c) yuzlugeyuvarla(sayi)

d) binligeyuvarla(sayi)

Girilen her sayı için program ekrana sayının orijinalini, en yakın tamsayıya, en yakın onluğa,

yüzlüğe, binliğe yuvarlanmış halini yazdırsın.

5.12 Aşağıdaki soruların her birine cevap verin.

a) Rasgele sayı üretmek ne demektir?

b) rand fonksiyonu nenden şans oyunları programlarında kullanışlıdır.

c) Neden srand ile rasgele bir program üretilir. Hangi şartlar altında rasgele üretmek için

kullanılmaz.

d) Neden bazen rand fonksiyonu ile üretilen değerler derecelendirilmeli ya da

kaydırılmalıdırlar?

e) Neden gerçek-dünyadaki olayların bilgisayar simülasyonlarının yapılması önemli bir

tekniktir?

5.13 Aşağıdaki n sınırları içerisinde rasgele sayı üreten ifadeleri yazınız.

a) 1  n  2

b) 1  n  100

c) 0  n  9

d) 1000  n  1012

e) –1  n  1

f) –3  n  11

5.14 Aşağıdaki her bir küme için, o kümeden rasgele bir sayı seçen ve ekrana yazdıran

fonksiyonu yazınız.

a) 2,4,6,8

b) 3,5,7,9,11

c) 6,10,14,18,22

5.15 hipotenus isminde, iki kenarı verilen bir dik üçgenin hipotenüsünü hesaplayan

fonksiyonu tanımlayınız. Bu fonksiyonu aşağıda kenarları verilen üçgenlerin hipotenüslerini

bulmak için bir programda kullanınız. Fonksiyon double türünde iki argüman almalı ve

hipotenüsü double türünde döndürmelidir.

Üçgen 1. Kenar 2.Kenar

1 3.0 4.0

2 5.0 12.0

3 8.0 15.0

5.16 tamsayikuvveti(taban, us) seklinde

192

tabanus

döndüren bir fonksiyon yazınız.

Örneğin, tamsayikuvveti(3,4) = 3 \* 3 \* 3 \* 3. us değişkenini pozitif ve sıfır olmayan bir

tamsayı, taban değişkeninin de tamsayı olduğunu kabul ediniz. Fonksiyon, hesaplamaları

kontrol için for yapısı kullanabilir. Herhangi bir matematik kütüphanesi fonksiyonunu

kullanmayınız.

5.17 kat isminde iki tamsayının ikincisinin, birincisinin tam katı olup olmadığına karar veren

bir fonksiyon yazınız. Fonksiyon iki tamsayı argümanı almalı ve ikinci tamsayı birincinin tam

katıysa 1(doğru) değilse 0(yanlış) döndürmelidir. Bu fonksiyonu iki sayı girişi yapılan bir

programda kullanınız.

5.18 Bir kaç tamsayı girişi yapılan ve tamsayıları birer birer, cift adı verilen bir fonksiyona

gönderen bir program yazınız. Bu fonksiyon gelen sayının çift sayı olup olmadığına karar

vermeli. Fonksiyon bir tamsayı argümanı almalı ve sayı çift sayı ise 1 değilse 0

döndürmelidir.

5.19 Ekranın soluna dayalı olarak yıldız karakterlerinden oluşan ve kenarı, kenar tamsayı

değişkeniyle alınan bir fonksiyon yazınız. Örneğin kenar, 4 ise fonksiyon ekrana aşağıda ki

deseni yazmalıdır.

\*\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*

5.20 5.19’da ki fonksiyonu birde karenin için dolduracak karakteri kullanıcıdan alacak şekilde

değiştirin. Parametre ismi icinidoldur olsun. Eğer kenar, 5 ve icinidoldur, # ise fonksiyon

aşağıdaki deseni yazmalıdır.

#####

#####

#####

#####

#####

5.21 5.19’daki ve 5.20’deki tekniklere göre çeşitli şekilleri ekrana çizen bir programı yazınız.

5.22 Aşağıdakileri gerçekleştiren bir program yazınız.

a) a tamsayısı b tamsayısına bölündüğünde, bölümün tamsayı kısmını hesaplasın.

b) c tamsayısı d tamsayısına bölündüğüne, kalan tamsayıyı hesaplasın

c) a) ve b) şıklarında geliştirdiğiniz program parçacıklarını kullanarak 1 ile 32767

arasında bir tamsayı alan bir fonksiyon yazın. Fonksiyon sayıyı basamaklarına ayırsın ve

her iki basamak arasında iki boşluk olacak şekilde ekrana yazdırsın. Örneğin 4562 sayısı

193

4 5 6 2

şeklinde yazdırılmalı.

5.23 Saati, üç argümanla (saat,dakika ve saniye) alan ve saat 12’den girilen saate kadar olan

zamanı saniye cinsinden hesaplayıp döndüren bir program yazınız. Daha sonra bu fonksiyonu

kullanarak girilen iki saat arasındaki farkı 12'lik saat dilimine göre hesaplayan bir program

yazınız.

5.24 Aşağıdaki tamsayı fonksiyonlarını yazınız.

a) derece fonksiyonu, Fahrenheit olarak girilen bir sıcaklığı derece olan geri göndersin.

b) fahrenhayt fonksiyonu derece olarak girilen bir sıcaklığı Fahrenheit olarak geri

göndersin.

c) Bu fonksiyonları kullanarak 0-100 arasındaki derece cinsinden sıcaklıkları Fahrenheita,

2-212 arasındaki Fahrenheit cinsinden sıcaklıkları dereceye çevirip ekrana bir çizelge

yazdıran programı yazınız. Çıktı düzgün çizelge biçiminde olsun ve az sayıda satır

kullanarak okunurluğu artırın.

5.25 float türündeki 3 sayının en küçüğünü döndüren bir fonksiyon yazınız.

5.26 Eğer bir sayının kendisi hariç, bütün çarpanlarının toplamı yine o sayıya eşitse bu sayıya

MÜKEMMEL SAYI denir. Örneğin, 6 bir mükemmel sayıdır. Çünkü 6 = 1 + 2 + 3.

mukemmel isminde, sayi parametresinin mükemmel bir sayı olup olmadığını tespit eden bir

fonksiyon yazınız. Bu fonksiyonu 1-1000 arasındaki tamsayılardan mükemmel olanlarını

bulmak için bir program içinde kullanınız. Program sayının mükemmel olduğunu göstermek

için mükemmel sayınının çarpanlarını ekrana yazdırsın. 1000’den daha büyük sayıları test

ederek bilgisayarınızın gücünü deneyebilirsiniz.

5.27 Bir tamsayı sadece kendisine ve 1'e bölünüyorsa bu tamsayıya asal sayı denir. Örneğin 2,

3, 5 ve 7 asal sayılardır ama 4,6,8 ve 9 asal değildir.

a) Bir tamsayının asal sayı olup olmadığına karar veren bir fonksiyon yazınız.

b) Bu fonksiyonu kullanarak 1-1000 arasındaki bütün asal sayıları bulan bir program

yazınız.

c) Bir sayının asal olup olmadığını bulmada n/2’ in çarpanlarının üst sınırı olduğunu

düşünebilirsiniz ama aslına kare kök n üst sınırdır. Neden? Programı iki şekilde de

yazıp çalıştırın ve performans farkını not edin.

5.28 Bir tamsayı değeri alan ve bu sayıyı basamaklarını tersten yazıp döndüren bir fonksiyon

yazınız. Örneğin verilen sayı 7631 ise fonksiyon 1367 geri göndermeli.

5.29 Ortak Bölenlerin En büyüğü (OBEB) iki tamsayının en büyük ortak bölenidir. Girilen iki

tamsayının OBEB'ini bulan bir obeb fonksiyonu yazınız.

194

5.30 siniflandir isminde bir fonksiyon yazın. Bu fonksiyon bir öğrencinin ortalama notunu

alsın ve bu not 90-100 arasında ise 4, 80-89 arasında ise 3, 70-79 arasında ise 2, 60-69

arasında ise 1 ve 60'ın altında ise 0 döndürsün.

5.31 Yazı, tura atan bir fonksiyon yazınız. Paranın her atılışında ekrana yazı veya tura yazsın.

Program 100 kez yazı tura atsın ve sonuçları ekrana yazdırsın. Program paraAt isminde

argüman almayan bir fonksiyon çağırsın ve yazı için 0, tura için 1 döndürsün. (Not: Eğer

program gerçekçi bir hesap yapıyorsa sonuçlar toplam atışın yarısına yakın olmalıdır yani 50

yazı ve 50 tura)

5.32 Bilgisayarların eğitimde sürekli artan bir rolü vardır. Bir ilkokul öğrencisine çarpma

işleminde yardımcı olacak bir program yazınız. rand fonksiyonunu kullanarak rasgele 2 adet

1 basamaklı sayı üretin ve ekrana şu şekilde bir soru yazdırın:

4 kere 7 kaçtır ?

Daha sonra öğrenci cevap versin. Program cevabı kontrol etsin. Eğer cevap doğruysa "Çok

güzel" yazdırsın ve yeni bir soru sorsun. Eğer cevap yanlışsa "lütfen tekrar deneyin."

yazdırsın ve aynı soruyu öğrenci doğru cevap verene kadar sorsun.

5.33 Bilgisayarın eğitimde kullanılmasına bilgisayar destekli eğitim denir. Buradaki en

önemli problemlerden biri öğrencinin isteksizliğidir. Bu, bilgisayarın öğrenci ile kurduğu

diyalogla engellenebilir. Alıştırma 5.32'yi doğru ve yanlış cevaplarda farklı yorumlar

yazdıracak şekilde tekrar yazınız

Doğru cevap karşılıkları

Çok güzel

Mükemmel

Aferin

Böyle devam et

Yanlış cevap karşılıkları

Hayır. Lütfen tekrar deneyin

Yanlış. Lütfen bir daha deneyin.

Pes etmeyin.

Hayır. Denemeye devam edin

5.34 Daha gelişmiş bilgisayar destekli eğitim programlarında öğrencinin performansı

önemlidir. Yani, yeni bir üniteye geçiş, öğrencinin önceki ünitelerde başarısıyla ilgilidir.

Alıştırma 5.33'teki programı öğrencinin doğru ve yanlış cevaplarını sayacak biçimde

değiştirin. Öğrenci 10 cevap verdikten sonra program öğrencinin doğru cevap yüzdesini

hesaplamalı ve %75 in altında ise "öğretmeninden yardım al" yazdırarak programdan

çıkmalı.

5.35 "Tuttuğum Sayıyı Tahmin Et" oynatan bir program yazınız. Program 1-1000 arasında

rasgele bir tamsayı üretsin ve ekrana

1-1000 arasında bir sayı tuttum.

195

Tahmin edebilir misin?

Lütfen ilk tahminini gir.:

yazdırsın. Kullanıcı ilk tahminini girdikten sonra program aşağıdaki ifadelerden biriyle cevap

versin :

1. Mükemmel. bildiniz !

Tekrar oynamak ister misiniz (E ya da H) ?

2.Çok küçük. Tekrar deneyin.

3.Çok büyük. Tekrar deneyin.

Eğer oyuncunun tahmini yanlışsa program doğru cevap verilene kadar döngü içinde

kalmalıdır. Program çok küçük yada çok büyük yazarak oyuncuya yardım etmelidir. Not: Bu

problemdeki arama tekniğine ikili arama denir. Bunun hakkında daha fazla bilgiyi bir sonraki

problemde bulabilirsiniz.

5.36 Alıştırma 5.35’deki programı oyuncunun tahminlerini sayacak şekilde değiştirin.

Eğer bu sayı 10 ya da daha küçükse "Siz sırrı biliyorsunuz ya da şanslısınız." Eğer bu sayı

10 a eşitse "Aha! Siz sırrı biliyorsunuz.." Eğer bu sayı 10'dan büyükse "Daha iyisini

yapabilirsiniz." yazdırın. Neden 10 tahminden daha az sayıda bir tahminle sonuç

bulunabilir?. İyi tahminlerle oyuncu sayıların yarısını eleyebilir. Şimdi nasıl 1-1000 arasında

tutulan sayının 10 yada daha az tahminle bulunabileceğini gösterin.

4 kuvvet(taban,us) şeklinde bir yinelenen fonksiyon yazın. Fonksiyon

tabanus

geri göndersin. Örneğin kuvvet(3,4)=3\*3\*3\*3. Us değişkeninin, 1'e eşit yada daha büyük bir

tamsayı olduğunu kabul edin. İpucu: Yineleme basamağı

tabanus = taban \* taban(us - 1)

şeklinde olmalı ve döngüden us, 1’e eşit olduğunda çıkılmalıdır. çünkü

taban1

= taban ’dır.

5.38 Fibonacci serisi

0,1,1,2,3,5,8,13,......

şeklinde 0 ve 1 ile başlar ve her terim kendinden önceki iki terimin toplamına eşittir. a)

yineleme fonksiyonu olmayan ve n. fibonacci sayısını hesaplayan bir fibonacci(n) fonksiyonu

yazın. b) sisteminizde yazabileceğiniz en büyük fibonacci sayısını yazdırın. (a) şıkkında

yazdığınız programda int yerine double türünü kullanın ve programın çok yüksek bir değer

yüzünden hata yapana kadar çalışmasına izin verin.

196

5.39 (Hanoi’nin kuleleri) Bütün bilgisayar bilimcileri mutlaka bazı klasik problemlerle

uğraşmışlardır. Bunların en ünlüsü Hanoi'nin kuleleridir. Bir efsaneye göre uzak doğudaki

din adamları bir grup diski bir çubuktan diğerine taşımaya çalışmaktadırlar. İlk grupta 64 disk

vardır ve çubuğa an altta en büyük disk olmak üzere büyükten küçüğe doğru dizilmişlerdir.

Din adamları bu diskleri ilk çubuktan diğerine, her seferinde yalnız bir disk taşımak ve küçük

diskin üzerine hiç bir zaman büyük disk koymamak koşuluyla taşımak zorundadırlar.

Üçüncü bir çubuk ise diskleri geçici olarak taşımak için kullanılmaktadır. Din adamları bu işi

bitirdiğinde dünyanın sonu gelecektir.

Din adamlarının diskleri birinci çubuktan üçüncü çubuğa taşıyacaklarını kabul edelim ve her

diskin transferi için bir algoritma yazalım.

Eğer bu probleme klasik yöntemlerle yaklaşacak olursak diskleri taşımada hemen ümitsizliğe

düşeriz ama probleme yineleme mantığıyla yaklaşacak olursak problem daha çözülebilir bir

hale gelecektir. n tane diski taşımak n-1 tane diski taşımak gibi aşağıdaki gibi düşünülebilir.

1. n - 1 diski 1. çubuktan 2. çubuğa, 3 çubuğu geçici olarak kullanarak taşı

2. en son(en büyük) diski 1. çubuktan 3. çubuğa taşı

3. n-1 diski 2. çubuktan 3. çubuğa 1. çubuğu geçici olarak kullanarak taşı.

İşlem, son görev olan n = 1 diskide taşındığında biter. Bu görev ise geçici bir çubuk

kullanılmadan başarılır.

Hanoi'nin kuleleri problemini çözen bir program yazınız. Yineleme fonksiyonunu dört

parametre ile kullanınız.

1. Taşınacak disk sayısı

2. Disklerin ilk bulunduğu çubuk

3. Disklerin taşınacağı çubuk

4. Disklerin taşınmasında kullanılacak geçici çubuk

Programınız taşınacak diskin bulunduğu çubuğu ve diskin taşınacağı çubuğu ekrana

yazdırmalı.

Örneğin 1. çubuktan 3. çubuğa 3 diskin taşınması aşağıdaki gibi olmalıdır.

1 -> 3 (Bu ifade 1. çubuktan 3.çubuğa 1 diskin taşınması anlamına gelir.)

1 -> 2

3 -> 2

1 -> 3

2 -> 1

2 -> 3

1 -> 3

5.41 (Yineleme fonksiyonunun görselleştirilmesi) yinelemeyi çalışırken görmek ilginç

olabilir. Şekil 5.14’teki faktöriyel fonksiyonunu, yerel değişkenini ve yinelemeyi çağıran

parametresini ekrana yazdıracak şekilde değiştirin. Yineleme fonksiyonu her çağrıldığında,

çıktıları aynı satırda gösterin ve kullanıcının yineleme fonksiyonunun nasıl çalıştığını

anlayabileceği bir şekilde bu çıktıları düzenleyin. Bu tür uygulamaları diğer yineleme

fonksiyonu kullanılan örneklerede uygulayabilirsiniz.

197

5.42 x ve y tam sayılarının ortak bölenlerinin en büyüğü (OBEB), her ikisini de tam bölen tam

sayıların en büyüğüdür. x ve y sayılarının OBEB'ini bulan ver döndüren bir obeb yineleme

fonksiyonu yazınız. x ve y ‘nin OBEB'i yineleme fonksiyonunda şu şekilde ifade edilmelidir:

Eğer y sıfır ise, obeb(x, y) = x, eğer y, sıfıra eşit değilse obeb(x, y) = obeb(y, x % y)

5.43 main fonksiyonu kendi kendini çağırabilir mi? main fonksiyonunu içeren bir program

yazın. Programınız static olarak bildirilen ve ilk değeri 1 olan sayac değişkenini içersin. Bu

değişken main fonksiyonu her çağrıldığında 1 artırlısın. Programı çalıştırınca ne oldu?

5.44 Alıştırma 5.32 den 5.34’e kadar olan bilgisayar destekli eğitim programları bir ilkokul

öğrencisine çarpmayı öğretiyordu. Bu alıştırma ile bu programı biraz daha geliştireceğiz.

a) Programı kullanıcının bir seviye belirleyebileceği şekilde değiştirin. 1. seviye

seçildiğinde öğrenciye 1 basamaklı sayılarla işlemler, 2. seviye seçildiğinde ise

öğrenciye 2 basamaklı sayılarla işlemler sorulsun.

b) Programı öğrencinin istediği aritmetik işlemleri çalışabileceği şekilde değiştirin. 1.

seçenek toplama işlemleri, 2.seçenek çıkarma işlemi, 3.seçenek çarpma, 4. seçenekte

bölme işlemleri ve 5. seçenekte karışık işlemler yaptırılsın.

5.45 mesafe isminde, verilen iki noktanın, (x1, y1) ve (x2, y2), arasındaki mesafeyi bulan bir

fonksiyon yazınız. Kullanacağınız bütün sayılar ve fonksiyonun döndüreceği değer float

türünde olsun.

5.46 aşağıdaki program ne yapar?

1 /\* ex05\_46.c \*/

2 #include <stdio.h>

3

4 int main( )

5 {

6 int c;

7

8 if ( ( c = getchar( ) ) != EOF ) {

9 main( );

10 printf ("%c",c);

11 }

12

13 return 0;

14 }

5.47 Aşağıdaki program ne yapar?

1 /\* ex05\_47.c \*/

2 #include <stdio.h>

3

4 int gizem(int, int);

5

6 int main( )

198

7 {

8 int x,y;

9

10 printf ("İki tamsayı girin.:");

11 scanf("%d%d" &x, &y);

12 printf ("Sonuç %d\n", gizem(x,y));

13 return 0;

14 }

15

16 /\* b parametresi, sonsuz yineleme olmaması için

17 pozitif olmak zorunda \*/

18 int gizem(int a, int b)

19 {

20 if (b == 1)

21 return a;

22 else

23 return a + gizem(a, b-1);

24 }

5.48 Alıştırma 5.47’deki programın ne yaptığını öğrendikten sonra ikinci argümanın negatif

olma olasılığını ortadan kaldıracak şekilde programı değiştiriniz.

5.49 Şekil 5.2’deki matematik kütüphane fonksiyonlarından mümkün olduğu kadar çoğunu

test eden bir program yazınız. Değişik argüman değerleri için bu fonksiyonların döndürdüğü

değerleri yazdırarak inceleyiniz.

5.50 Aşağıdaki program parçalarındaki hataları bulun ve nasıl düzeltileceğini açıklayın.

a) float kup(float); /\* fonksiyon prototipi \*/

...

kup(float sayi) /\* fonksiyon tanımlanması \*/

{

return sayi \* sayi \* sayi;

}

b) register auto int x = 7;

c) int rasgeleSayi = srand ( );

d) float y = 123.45678;

int x;

x = y;

printf ("%f\n", (float) x);

e) double kare(double sayi)

{

double sayi;

return sayi \* sayi;

}

f) int toplam(int n)

{

199

if (n == 0)

return 0;

else

return n + sum(n);

}

5.51 Şekil 5.10’daki barbut programını, bahis içerecek şeklide değiştiriniz. Programın barbut

oyununu bir kez oynatan bölümünü bir fonksiyon haline getiriniz. bakiye değişkeninin ilk

değerini 1000 dolar olarak atayınız. Ekrana kullanıcının bahis girmesini söyleyen bir ifade

yazdırınız. Girilen bahis’in bakiye’den küçük ya da eşit olup olmadığını bir while döngüsü

ile kontrol ediniz. Eğer girilen bahis uygun değilse, oyuncu uygun bahis değeri girene dek

bahis sorma işlemini tekrarlayınız. Doğru bir bahis girildiğinide, barbut oyununu bir kez

çalıştırınız. Eğer oyuncu kazanırsa, bakiye’yi bahis kadar artırınız ve yeni bakiyeyi ekrana

yazdırınız. Eğer oyuncu kaybederse, bakiyeyi, bahis kadar azaltınız, yeni bakiyeyi ekrana

yazdırınız ve bekiyenin sıfır olup olmadığını kontrol ediniz. Eğer sıfır ise ekrana “Üzgünüm

bütün paranızı kaybettiniz” yazdırınız. Program çalıştıkça ekrana çeşitli mesajlar yazdırırarak

programın, oyuncuyla sohbet etmesini sağlayınız. Örneğin “ Züğürt olmak üzeresiniz” ya da

“Hadi, bir kez daha deneyin!” ya da “Çok kazandınız” gibi...

200

DİZİLER

AMAÇLAR

 Dizi veri yapısını tanıtmak

 Dizilerin değerleri depolama, sıralama ve listeleri arama ile değer tabloları

oluşturmada kullanımlarını anlamak.

 Bir dizinin nasıl bildirileceğini, bir diziye nasıl ilk değer atanacağını ve dizideki

bağımsız elemanların nasıl çağrılacaklarını anlamak.

 Dizileri fonksiyonlara geçirebilmek.

 Temel sıralama tekniklerini anlamak.

 Çok boyutlu dizileri bildirebilmek ve kullanabilmek.

BAŞLIKLAR

6.1 Giriş

6.2 Diziler

6.3 Dizileri bildirmek

6.4 Dizileri kullanan örnekler

6.5 Dizileri fonksiyonlara geçirmek

6.6 Dizileri sıralamak

6.7 Örnekler: Ortalama,Mod ve Medyanı diziler kullanarak hesaplamak

6.8 Dizilerde arama yapmak

6.9 Çok boyutlu diziler

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Performans İpuçları\*

Taşınırlık İpuçları\*Yazılım Mühendisliği Gözlemleri\*Çözümlü Alıştırmalar\* Çözümler\*

Alıştırmalar

6.1 GİRİŞ

Bu ünite, önemli bir konu olan veri yapılarına bir giriş olacaktır. Diziler , birbirleriyle ilişkili

ve aynı tipte verileri içeren veri yapılarıdır. 10. Ünitede C’nin büyük olasılıkla farklı tiplerden

oluşan, birbirleriyle bağlantılı veri yapı biçimi olan struct (yapı) gösterimini tartışacağız.

Diziler ve yapılar statik yapılardır ve programın çalışması süresince hep aynı boyutta kalırlar.

(Bazen de otomatik depolama sınıfında olabilirler ve böylece tanımlandıkları blokların içine

giriş ve çıkış esnasında, yaratılıp yok edilebilirler) 12. Ünitede listeler, sıralar, yığınlar ve

ağaçlar gibi programın çalışması esnasında büyüyüp, küçülebilen dinamik veri yapılarını da

anlatacağız.

6.2 DİZİLER

Bir dizi, aynı isme ve aynı tipe sahip olmaları sebebiyle birbirleriyle ilişkili olan hafıza

konumlarının bir grubudur. Bir dizinin içindeki bir elemanı ya da konumu belirtmek için o

dizinin adını ve elemanın dizi içindeki pozisyonunu belirtmeliyiz. Şekil 6.1, c isminde bir

tamsayı dizisi göstermektedir. Bu dizinin 12 elemanı vardır. Bu elemanlardan herhangi biri,

201

dizinin ismi ve belirlenen elemanın pozisyonu köşeli parantez ( [ ] ) içinde belirtilerek

çağrılabilir. Bir dizinin ilk elemanı her zaman sıfırıncı elemandır. Bu sebepten, c dizisinin ilk

elemanı c [o] ile gösterilir, ikinci elemanı c[1] , yedinci elemanı c[6] ve genel olarak i ‘ninci

elemanı c [i - 1] şeklinde gösterilir. Dizi isimleri, diğer değişkenlerde olduğu gibi yalnızca

harf, rakam ve altçizgi karakterlerini içerebilir. Dizi isimleri rakam karakterleriyle

başlayamaz.

Bir dizinin ismi (bu dizinin her elemanının isminin aynı olduğuna dikkat edin)

Dizideki elemanların pozisyonunu belirten sayı

Şekil 6.1 12 elemanlı dizi

Köşeli parantez içinde bulunan sayı, belirteç (subscipt) olarak adlandırılır. Bir belirteç, ya bir

tamsayı ya da bir tamsayı deyimi olmalıdır. Eğer program belirteç olarak deyim kullanıyorsa ,

deyim belirtecin değerine karar vermek için hesaplanır. Örneğin, eğer a = 5 ve b = 6 ise

c [ a + b ] + = 2;

ifadesi c[11] elemanına 2 ekler. Dikkat edilirse, belirteçle birlikte dizi ismi bir sol taraf

değeridir. Bu sebepten, atamaların sol tarafında kullanılabilir.

Şimdi, c dizisini (Şekil 6.1.) daha yakından inceleyelim. Dizinin ismi c‘dir. Dizinin 12

elemanı c[0] , c[1] , c[2] , ...... c[11] şeklinde gösterilmiştir. c[0] içinde tutulan değer –45 ,

c[1] içinde tutulan değer 6 , c[2] içinde tutulan değer 0, c[7] içinde tutulan değer 62 ve c[11]

-45

6

78

1

6453

62

-3

72

0

0

1543

-89

c[1]

c[2]

c[3]

c[4]

c[5]

c[6]

c[7]

c[8]

c[9]

c[10]

c[11]

c[0]

202

içinde tutulan değer 78‘dir. Bu dizinin ilk üç elemanının içinde tutulan değerlerin toplamını

yazdırmak isteseydik;

printf (“%d”, c[0] + c[1] + c[2]);

yazacaktık. Bu dizinin yedinci elemanının değerini ikiye bölüp, oluşan sonucu x değişkenine

atasaydık;

x = c[6] / 2;

yazacaktık.

Genel Programlama Hataları 6.1

“Dizinin yedinci elemanı” ile “yedinci dizi elemanı” arasındaki farkı anlamak

önemlidir.Dizi belirteçleri sıfırdan başladığı için “dizinin yedinci elemanı” 6

belirtecine sahiptir. ”yedinci dizi elemanı” ise 7 belirtecine sahiptir ve aslında

dizinin sekizinci elemanıdır. Bu “bir eksik”(off-by-one) hatalarının

kaynağıdır.

Dizi belirteçlerini içine alan köşeli parantezler, C‘de bir operatör olarak kullanılırlar ve

parantezlerle aynı seviyede önceliğe sahiptirler. Şekil 6.2, bu ana kadar incelediğimiz

operatörlerin önceliğini ve işleyiş sıralarını göstermektedir. Öncelik sırası yukarıdan aşağıya

gidildikçe azalmaktadır.

Operatörler İşleyiş Tip

( ) [ ] soldan sağa en yüksek

++ -- ! (tip) sağdan sola tekli

\* / % soldan sağa multiplicative

+ - soldan sağa additive

< <= > >= soldan sağa karşılaştırma

= = != soldan sağa eşitlik

&& soldan sağa ve

|| soldan sağa veya

?: sağdan sola koşullu

= += -= \*= /= sağdan sola atama

, soldan sağa virgül

Şekil 6.2 Operatör öncelikleri

6.3 DİZİLERİN BİLDİRİLMELERİ

Diziler hafızada bir yer kaplarlar. Programcı, her elemanın tipini ve dizide kaç eleman

kullanacağını belirterek bilgisayarın en uygun hafızayı ayırmasını sağlatır. Mesela, bilgisayara

12 elemana sahip bir tamsayı dizisini

int c[12];

ile bildiririz. Aynı anda birden fazla dizi bildirimi yapabiliriz. 100 elemana sahip bir b

tamsayı dizisi ile 27 elemana sahip bir x tamsayı dizisini aynı anda bildirmek istersek

203

int b[ 100 ] , x [ 27 ] ;

yazarız.

Diziler başka veri tipleri içermek üzere de bildirilebilirler. Örneğin, char tipte bir dizi

karakter stringlerini depolamakta kullanılabilir. Karakter stringleri ile dizilerin benzerlikleri 8.

ünitede anlatılacaktır. Göstericiler ve diziler arasındaki ilişki ise 7.ünitede gösterilecektir.

6.4.DİZİLERİ KULLANAN ÖRNEKLER

Şekil 6.3’ teki program, for döngü yapısını yapısı kullanarak 10 elemana sahip bir dizinin tüm

elemanlarını 0’a atamaktadır ve diziyi çizelge biçiminde yazdırmaktadır.

1 /\*Şekil 6.3:fig06\_03.c

2 bir diziye ilk değer vermek\*/

3 # include <stdio.h>

4

5 int main ( )

6 {

7 int n[10] , i ;

8

9 for (i=0;i<=9;i++) /\*diziye değer ata\*/

10 n[i]=0;

11

12 printf(“%s%13s\n”,”Eleman”,”Değer”);

13 for(i=0;i<=9;i++) /\*diziyi yazdır\*/

14 printf(“%7d%13d\n”,i,n[i]);

15

16 return 0;

17 }

Eleman Değer

0 0

1 0

2 0

3 0

4 0

5 0

6 0

7 0

8 0

9 0

Şekil 6.3 Bir dizinin elemanlarına sıfır değerlerini atamak.

Şekil 6.3’te, ilk printf ifadesi ile (12.satır) for yapısı arasındaki yakın ilgiden dolayı boşluk

bırakmadığımıza dikkat ediniz. Bu durumda printf ifadesi, for yapısıyla yazdırılan sütunların

204

başlıklarını yazdırmaktadır. Programcılar, for yapısıyla ilgili olan printf ifadesi arasındaki

boşluğu genelde çıkarırlar.

Bir dizinin elemanları dizi bildirimi yapılırken, bildirimden sonra , eşittir işareti ve küme

parantezleri içinde virgülle ayrılmış atama değerleriyle ( initializers ) ilk değerlere atanabilir.

Şekil 6.4, bir tamsayı dizisine 10 değerle ilk değer atamakta ( 7.satır ) ve diziyi çizelge

biçiminde yazdırmaktadır.

1 /\*Şekil 6.4:fig06\_04.c

2 Diziye bildirim sırasına ilk değerler verme\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int n[10]={32, 27, 64, 18, 95, 14, 90, 70, 60, 37};

8 int i;

9

10 printf (“%s %13s\n”,”Eleman”,”Değer”);

11

12 for (i = 0; i <= 9 ;i++1)

13 printf (“%6d%14d\n”,i,n[i]);

14

15 return 0;

16 }

Eleman Değer

0 32

1 27

2 64

3 18

4 95

5 14

6 90

7 70

8 60

9 37

Şekil 6.4 Dizinin elemanlarına bildirim sırasında ilk değer atamak

Eğer dizideki elemanların sayısından daha az sayıda atama değeri varsa, kalan elemanların

hepsi 0 değerine atanır. Örneğin, Şekil 6.3’deki n dizisinin tüm elemanları

int n[10] = { 0 };

ifadesiyle 0 değerine atanabilirdi. Bu ifade tarzı ilk elemanı sıfıra atayacak, geriye kalan 9

elemanda, eleman sayısından daha az atama değeri olduğu için sıfıra atanacaktı. Dizilerin

otomatik olarak 0 ilk değerine atanmadıklarını hatırlamak önemlidir. Programcı, kalan

elemanların otomatik olarak 0’a atanmasını sağlamak için en azından ilk değeri 0’a

205

atamalıdır. Dizi elemanlarına ilk değer olarak 0 vermek için kullanılan bu yöntem, static

diziler için derleme zamanında ve otomatik diziler için çalışma zamanında uygulanır.

int n[5]={ 32, 27, 64,18, 95, 14};

Şeklinde bir dizi bildirimi ise yazım hatası oluşturacaktır çünkü 5 dizi elemanı ve 6 atama

değeri vardır.

Genel Programlama Hataları 6.2

Elemanlarına ilk değer verilmesi gerekilen bir dizinin elemanlarına ilk değer vermeyi unutmak.

Genel Programlama Hataları 6.3

Diziye ilk değer atanacakken dizi elemanından daha çok sayıda atama değeri kullanmak bir yazım

hatasıdır.

Eğer atama değerleri ile yapılan bir bildirimde dizinin boyutu belirtilmezse, dizinin eleman

sayısı atama listesindeki eleman sayısı olacaktır. Örneğin,

int n[ ] = { 1, 2, 3, 4, 5 };

5 elemanlı bir dizi oluşturur.

Şekil 6.5’teki program, 10 elemana sahip s dizisine ilk değer olarak 2,4,6,......20 değerlerini

vermektedir. Bu değerler döngü sayıcısını 2 ile çarpıp, çarpıma 2 eklenerek oluşturulmaktadır.

#define önişlemci komutu bu programda tanıtılmıştır. 5. satırdaki

#define BOYUT 10

10 değerine sahip, BOYUT isminde bir sembolik bir sabit yaratır. Sembolik sabitler, C

önişlemcisi tarafından derleme esnasında yerdeğiştirme metniyle (replacement text)

değiştirilecek olan tanıtıcılardır. Program önişlemeye sokulduğunda, BOYUT sembolik

sabitiyle karşılaşılan her yerde BOYUT, değiştirme metni olan 10 ile değiştirilecektir. Şekil

6.5’te ilk for döngüsü ( 11.satır ), #define komutundaki BOYUT sabiti 10’dan 1000’e

değiştirilerek 1000 elemanlı bir diziyi doldurabilirdi. Eğer BOYUT sembolik sabiti

kullanılmasaydı, 1000 elemanlı bir diziyi idare etmek için programda üç yerde değişiklik

yapmak zorunda kalacaktık. Programlar büyüdükçe bu teknik, açık programlar yazmak için

daha kullanışlı hale gelecektir.

1 /\*Şekil 6.5:fig06\_05.c

2 s dizisinin elemanlarına

3 2’den 20’ye kadar olan çift tamsayıları atamak\*/

4 # include <stdio.h>

5 #define BOYUT 10

6

7 int main ( )

8 {

9 int s[BOYUT], j ;

10

206

11 for (j=0 ; j <= BOYUT - 1; j++) /\*Değerleri hesapla\*/

12 s[j] = 2 + 2 \* j ;

13

14 printf (“ %s %13s \n”,“Eleman”,“Değer”);

15

16 for (j = 0; j <= BOYUT - 1 ; j++) /\*Değerleri yazdır\*/

17 printf ("%7d %13d\n”, j, s[j] );

18

19 return 0;

20 }

Eleman Değer

0 2

1 4

2 6

3 8

4 10

5 12

6 14

7 16

8 18

9 20

Şekil 6.5 Bir dizinin elemanlarına yerleştirilecek değerleri oluşturmak

Genel Programlama Hataları 6.4

#define ve # include önişlemci komutlarını noktalı virgül ile sonlandırmak.

Önişlemci komutlarının, C ifadeleri olmadığını hatırlayınız.

Eğer az önceki #define önişlemci komutu noktalı virgül ile sonlandırılsaydı, BOYUT

sembolik sabitiyle karşılaşılan her yerde önişlemci tarafından BOYUT yerine 10; metni

yerleştirilecekti. Bu, derleme esnasında yazım hatalarına ya da çalışma esnasında mantık

hatalarına neden olur. Önişlemcinin yalnızca bir metin yöneticisi olduğunu hatırlayınız.

Genel Programlama Hataları 6.5

Sembolik bir sabite, çalıştırılabilir bir ifade içinde değer atamak. Sembolik sabit bir değişken

değildir. Derleyici tarafından, çalışma zamanında değerleri tutan değişkenler gibi sembolik sabitlere

de hafızada yer ayrılmaz.

Yazılım Mühendisliği Gözlemleri 6.1

Her dizinin boyutunu sembolik sabitlerle belirtmek programı daha

ölçülendirilebilir yapar.

İyi Programlama Alıştırmaları 6.1

Sembolik sabitler için yalnızca büyük harfler kullanın. Bu, sembolik sabitlerin

program içinde göze çarpmasını sağlayarak, programcıya bunların değişken

olmadıklarını hatırlatacaktır.

207

Şekil 6.6, 12 elemanlı a tamsayı dizisinin içindeki değerleri toplamaktadır. for döngüsünün

gövdesi (13.satır) toplamayı yapmaktadır.

1 /\*Şekil 6.6:fig06\_06.c

2 Dizi elemanlarının toplamlarını hesaplamak\*/

3 #include <stdio.h>

4 #define BOYUT 12

5

6 int main( )

7 {

8 int a[ BOYUT ]={1, 3, 5, 4, 7, 2, 99,

9 16, 45, 67, 89, 45};

10 int i, toplam=0;

11

12 for (i = 0;i <= BOYUT - 1; i++)

13 toplam += a[i];

14

15 printf (“Dizideki elemanların toplamı %d dir.\n”,toplam);

16 return 0;

17 }

Dizideki elemanların toplamı 383 dir.

Şekil 6.6 Dizi elemanlarının toplamlarını hesaplamak

Bir sonraki örneğimiz ise dizileri bir araştırmada toplanacak verilerin özetini yapmak için

kullanmaktadır. Aşağıdaki problemi inceleyiniz;

40 öğrenciye kafeteryadaki yiyeceklerin kalitesine 1’den 10’a kadar bir not vermeleri

(1 çok kötü ve 10 mükemmel anlamındadır) söylenmiştir. 40 yanıtı bir diziye

yerleştirin ve oyların özetini yapın.

Bu örnek tipik bir dizi uygulamasıdır.(bakınız şekil 6.7) Her tipte cevabın (1’den 10’a kadar )

sayısını özetlemek istiyoruz. cevaplar dizisi ( 10.satır ) öğrencilerin cevaplarından oluşan 40

elemanlı bir dizidir. 11 elemana sahip frekans dizisiyle ( 9.satır ) her cevaptan kaç adet

olduğunu sayacağız. frekans[0]’ı önemsemeyeceğiz çünkü 1 yanıtı için frekans[0] yerine

frekans[1]’i arttırmak daha mantıklıdır. Bu sayede her yanıtı frekans dizisindeki belirteç gibi

doğrudan kullanabileceğiz.

1 /\* Şekil 6.7: fig06\_07.c

2 Öğrenci Oylama Programı \*/

3 #include <stdio.h>

4 #define CEVAP\_BOYUTU 40

5 #define FREKANS\_BOYUTU 11

6

7 int main( )

8 {

9 int ogrenciCevabi, oylama, frekans[ FREKANS\_BOYUTU ] = { 0 };

10 int cevaplar[ CEVAP\_BOYUTU ] =

11 { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,

208

12 1, 6, 3, 8, 6, 10, 3, 8, 2, 7,

13 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,

14 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };

15

16 for(ogrenciCevabi= 0;ogrenciCevabi<=CEVAP\_BOYUTU - 1;ogrenciCevabi ++ )

17 ++frekans[ cevaplar [ogrenciCevabi ] ];

18

19 printf( "%s%17s\n", "Oylama", "Frekans" );

20

21 for ( oylama = 1; oylama<= FREKANS\_BOYUTU - 1; oylama++ )

22 printf( "%6d%17d\n", oylama, frekans[ oylama] );

23

24 return 0;

25 }

Oylama Frekans

1 2

2 2

3 2

4 2

5 5

6 11

7 5

8 7

9 1

10 3

Şekil 6.7 Basit bir oylama analiz programı

İyi Programlama Alıştırmaları 6.2

Programın açıklığı için çaba gösterin. Kimi zaman, hafızanın ya da işlemci zamanının etkili

kullanılması daha açık programlar yazmak için feda edilebilir.

Performans İpuçları 6.1

Kimi zaman performans hususları açıklık hususlarından daha önemlidir.

for döngüsü (16.Satır), cevaplar dizisinden cevapları alarak frekans dizisi içindeki 10

sayıcıdan uygun olanını ( frekans[1]’den frekans [10]’a kadar) arttırır. Döngüdeki kilit ifade

17.satırdadır ;

++ frekans [ cevaplar [ ogrenciCevabi ] ] ;

Bu ifade, cevaplar[ ogrencicevabi ] değerine göre uygun frekans sayıcısını arttırır. Örneğin,

ogrenciCevabi sayıcı değişkeni 0 olduğunda cevaplar[ ogrencicevabi ] 1’dir. Bu yüzden,

++frekans [ cevaplar [ ogrenciCevabi ] ] ; gerçekte ilk dizi elemanını arttıran

++frekans [ 1 ] ;

209

olarak gösterilebilir.

ogrenciCevabi 1 olduğunda, cevaplar [ ogrenciCevabi ] 2’dir. Bu yüzden, ++ frekans [

cevaplar [ ogrenciCevabi ] ] ; ikinci dizi elemanını arttıran

++frekans[2];

biçiminde gösterilebilir.

ogrenciCevabi 2 olduğunda, cevaplar [ ogrenciCevabi ] 6’dır. Bu yüzden, ++ frekans [

cevaplar [ ogrenciCevabi ] ] ; altıncı dizi elemanını arttıran

++frekans[6];

biçiminde gösterilebilir. Araştırmadaki yanıtların sayısı ne olursa olsun, sonuçları özetlemek

için yalnızca 11 elemanlı bir diziye ( ilk elemanı ihmal edilerek ) ihtiyaç duyulmaktadır. Eğer

veri 13 gibi geçersiz değerler içeriyorsa, program frekans [ 13 ]’e 1 eklemeye çalışacaktır.Bu,

dizinin sınırları dışında olacaktır. C, bilgisayarın var olmayan bir elemanı kullanmasını

engelleyecek herhangi bir dizi sınırı kontrolüne sahip değildir. Bu sebepten, çalışmakta olan

bir program uyarı vermeden dizinin dışına çıkabilir. Programcı, bütün dizi kullanımlarının

dizi sınırları içinde kalacağından emin olmalıdır.

Genel Programlama Hataları 6.6

Dizi sınırları dışındaki bir elemanı kullanmak

İyi Programlama Alıştırmaları 6.3

Dizi boyunca döngü kullanırken dizi belirteci asla 0’ın altına inmemeli ve her zaman dizideki toplam

eleman sayısından az olmalıdır (büyüklük-1). Döngü devam şartının bu aralığın dışındaki elemanlara

ulaşılmasını engellediğinden emin olun.

İyi Programlama Alıştırmaları 6.4

for yapısında en yüksek dizi belirtecini kullanarak , “bir eksik” hatalarını ortadan kaldırmak

İyi Programlama Alıştırmaları 6.5

Programlar, hatalı bilginin programın hesaplarını etkilememesi için tüm giriş değerlerinin

doğruluğunu onaylamalıdır.

Performans İpuçları 6.2

Dizi sınırlarının dışındaki elemanları kullanmanın yaratacağı hatalar (genelde ciddi hatalardır)

sistemden sisteme farklılık gösterir.

Bir sonraki örneğimiz diziden sayıları okumakta ve bilgiyi çizgi grafik biçiminde

göstermektedir; sayı yazdırıldıktan sonra, sayının yanında o kadar sayıda yıldız karakteri \*

yazdırılmaktadır. Yuvalı for yapısı çizgileri çizmektedir. Çizgiyi sonlandırmak için kullanılan

printf ( “\n” ) kullanımına dikkat ediniz.

1 /\* Şekil 6.8: fig06\_08.c

2 Çizgi grafik yazdırma programı \*/

210

3 #include <stdio.h>

4 #define BOYUT 10

5

6 int main( )

7 {

8 int n[ BOYUT ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };

9 int i, j;

10

11 printf( "%s%13s%17s\n", "Eleman", "Değer", "Grafik" );

12

13 for ( i = 0; i <= SIZE - 1; i++ ) {

14 printf( "%7d%13d ", i, n[ i ]) ;

15

16 for ( j = 1; j <= n[ i ]; j++ ) /\* bir satır yaz \*/

17 printf( "%c", '\*' );

18

19 printf( "\n" );

20 }

21

22 return 0;

23 }

Eleman Değer Grafik

0 19 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1 3 \*\*\*

2 15 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3 7 \*\*\*\*\*\*\*

4 11 \*\*\*\*\*\*\*\*\*\*\*

5 9 \*\*\*\*\*\*\*\*\*

6 13 \*\*\*\*\*\*\*\*\*\*\*\*\*

7 5 \*\*\*\*\*

8 17 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9 1 \*

Şekil 6.8 Çizgi grafik çizen program

5.ünitede, zar atma problemini yazmak için daha şık bir yöntem göstereceğimizi söylemiştik.

Problem, 6 yüzlü bir zarı 6000 kez atacak ve rasgele sayı üreticisinin gerçekten de rasgele

sayılar üretip üretmediğini test edecekti. Bu programın dizilerle yapılmış hali şekil 6.9’da

gösterilmiştir.

1 /\* Şekil 6.9: fig06\_09.c

2 6000 kez zar atma programı \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5 #include <time.h>

6 #define BOYUT 7

7

211

8 int main( )

9 {

10 int yuz,zarAt, frekans[ BOYUT ] = { 0 };

11

12 srand( time( NULL ) );

13

14 for ( zarAt = 1; zarAt <= 6000; zarAt ++ ) {

15 yuz = rand() % 6 + 1;

16 ++frekans[ yuz ]; /\* Şekil 5.8 20.satırdaki switch’in yerine geçer \*/

17 }

18

19 printf( "%s%17s\n", "Yüz", "Frekans" );

20

21 for ( yuz = 1; yuz<= BOYUT- 1; yuz++ )

22 printf( "%3d%17d\n", yuz, frekans [ yuz ] );

23

24 return 0;

25 }

Yüz Frekans

1 1037

2 987

3 1013

4 1028

5 952

6 983

Şekil 6.9 switch yerine dizileri kullanan zar atma programı

Bu noktaya kadar sadece tamsayı dizilerini tartıştık. Ancak diziler her tipte verileri tutabilir.

Şimdi karakter dizilerinde dize [Not: string terimi bilişim sözlüğünde dize olarak

açıklanmıştır, ancak önemli olan bu terimin karşılığını kavramaktır.Bu kitapta hem

string hem de dize kelimelerini kullanmayı uygun buluyoruz] depolamayı anlatacağız. Şu

ana dek sahip olduğumuz tek string işleme yeteneği printf ile bir string yazdırmaktı.

”merhaba” gibi bir string aslında, C’de bağımsız karakterlerden oluşmuş static bir dizidir.

Karakter dizileri kendilerine has bir çok özelliğe sahiptir. Bir karakter dizisi, bir string

kullanılarak ilk değerlere atanabilir. Örneğin,

char string1[ ]=”birinci”;

biçimindeki bir bildirim, string1 dizisine “birinci” stringi içindeki bağımsız karakterleri

atamaktadır. Az önceki bildirimde string1 dizisinin boyutu, derleyiciye bağımlı bir biçimde

stringin uzunluğuyla belirlenir.

“birinci” stringinin 7 karakter ile stringi sonlandıran ve null karakter adı verilen özel bir

karakter içerdiğini bilmek önemlidir. Bu sebepten, string1 dizisi aslında 8 eleman

içermektedir. null karakterin, karakter sabiti olarak gösterimi ‘\0’ biçimindedir. C’de tüm

212

stringler bu karakter ile sonlanır. Bir stringi temsil eden diziler her zaman, stringin içindeki

karakter sayısı ve sonlandırıcı null karakteri tutabilecek kadar geniş bildirilmelidir.

Karakter dizilerine, atama listesindeki bağımsız karakter sabitleriyle de ilk değerler atanabilir.

Az önceki bildirim

char string1 [ ] = { ’b’ , ’i’ , ’r’ ,’i’ ,’n’ , ’c’ ,’i ’};

ile denktir.

Bir string aslında karakterlerden oluşan bir dizi olduğundan, string içindeki bağımsız

karakterlere dizi belirteci gösterimiyle erişebiliriz. Örneğin, string1[0] ‘b’ karakteri ve

string1[3] ‘i’ karakteridir.

Bir karakter dizisine scanf ve %s dönüşüm belirtecini kullanarak klavyeden okuyacağımız bir

stringi alabiliriz. Örneğin,

char string2[20];

bildirimi 19 karakter ve sonlandırıcı null karakteri tutabilecek bir karakter dizisi yaratır.

scanf(“%s”,string2);

ifadesi, klavyeden bir string okur ve okuduğu stringi string2 içine yazar. Dizi isminin scanf

fonksiyonuna diğer değişkenler için kullanılan & ile geçirilmediğine dikkat ediniz. & , bir

değişkenin hafızadaki konumunu scanf fonksiyonuna bildirir ve böylece o konuma bir değer

depolanabilir. Kısım 6.5’te dizilerin fonksiyonlara geçirilişini anlatacağız. Bir dizi isminin,

dizinin başlangıç adresi olduğunu ve bu sebepten & kullanmanın gereksiz olduğunu

göreceğiz.

Programcı, stringin yazılacağı dizinin, kullanıcının klavyeden yazacağı herhangi bir stringi

tutabilmesini garanti altına almalıdır. scanf fonksiyonu, ilk boşluk karakteri girilene dek

klavyeden karakter okumaya devam eder; dizinin büyüklüğüne dikkat etmez. Bu sebepten,

scanf dizinin sonundan öteye de yazabilir.

Genel Programlama Hataları 6.7

scanf ile klavyeden yazılan stringi tutabilecek kadar geniş olmayan bir karakter dizisi kullanmak, veri

kaybına ya da çalışma zamanlı diğer hatalara sebep olabilir.

Bir stringi temsil eden karakter dizisi, printf ve %s dönüşüm belirteciyle yazdırılabilir.

string2 dizisi

printf (“%s\n”,string2);

ile yazdırılabilir.

printf’in scanf gibi karakter dizisinin büyüklüğünü önemsemediğine dikkat ediniz.Stringin

karakterleri, sonlandırıcı null karakterle karşılaşılıncaya dek yazdırılır.

213

Şekil 6.10, bir karakter dizisine string ile ilk değer verilmesini , karakter dizisinin bir string

olarak yazdırılmasını ve stringin bağımsız karakterlerine erişilmesini göstermektedir.

Şekil 6.10, for yapısını (16.satır) string1 dizisi boyunca ilerlemek ve %c dönüşüm

belirteciyle karakterleri birer boşluk bırakarak yazdırmak için kullanmaktadır. for yapısındaki

koşul (string1 != ‘\0’ ), string içinde sonlandırıcı null karakterle karşılaşılmadığı sürece

doğrudur.

1 /\* Şekil 6.10: fig06\_10.c

2 Karakter dizilerini string gibi ele almak \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char string1[ 20 ], string2[] = "string literal";

8 int i;

9

10 printf("Bir string girin: ");

11 scanf( "%s", string1 );

12 printf( "string1: %s\nstring2: %s\n"

13 "karakterler arasında boşlukla string1:\n",

14 string1, string2 );

15

16 for ( i = 0; string1[ i ] != '\0'; i++ )

17 printf( "%c ", string1[ i ] );

18

19 printf( "\n" );

20 return 0;

21 }

Bir string girin: Herkese merhaba

string1: Merhaba

string2: string literal

karakterler arasında boşlukla string1:

M e r h a b a

Şekil 6.10 Karakter dizilerini string gibi ele almak.

5.ünite, static depolama sınıfı belirtecini anlatmıştı . static bir yerel değişken, program

süresince var olur ancak yalnızca fonksiyon gövdesi içinde görülebilir. Yerel dizi

bildirimlerinde static kullanarak, dizinin fonksiyon her çağrıldığında yeniden yaratılmasını ve

fonksiyondan her çıkıldığında dizinin yok edilmesini engelleyebiliriz. Bu, büyük diziler

içeren fonksiyonların sıklıkla çağrıldığı programların çalışma zamanını kısaltır.

214

Performans İpuçları 6.3

Faaliyet alanına sıklıkla girip çıkan ve otomatik diziler içeren

fonksiyonlarda,diziyi static yaparak fonksiyonun her çağrısında dizinin

yeniden yaratılmasını engelleyin.

static olarak bildirilen diziler, derleme zamanında yalnızca bir kere otomatik olarak ilk

değerlere atanır. Eğer static bir dizi programcı tarafından özellikle ilk değerlere atanmamışsa,

dizinin elemanları derleyici tarafından 0’a atanacaktır.

Şekil 6.11, statikDiziIlk fonksiyonunu (20.satır) ve fonksiyon içinde static olarak bildirilmiş

bir yerel dizi ile otomatikDiziIlk fonksiyonu (37.satır) ve fonksiyon içinde bildirilmiş

otomatik bir yerel dizinin kullanımlarını göstermektedir. statikDiziIlk fonksiyonu iki kez

çağrılmıştır (11 ve 14.satırlar). Fonksiyon içindeki static yerel dizi ise derleyici tarafından 0’a

atanmıştır. Fonksiyon diziyi yazdırmakta, her elemana 5 eklemekte ve diziyi bir kez daha

yazdırmaktadır. Fonksiyon ikinci kez çağrıldığında static dizi, ilk fonksiyon çağrısında

depolanan değerleri içermektedir. otomatikDiziIlk fonksiyonu da 2 kez çağrılmıştır (12 ve

15.satırlar). Fonksiyon içindeki otomatik yerel dizinin elemanları 1,2 ve 3 değerlerine

atanmıştır. Fonksiyon diziyi yazdırmakta, her elemana 5 eklemekte ve diziyi yeniden

yazdırmaktadır. Fonksiyon ikinci kez çağrıldığında dizi elemanları 1,2 ve 3’e yeniden

atanmışlardır çünkü dizi otomatik depolama sürecine sahiptir.

Genel Programlama Hataları 6.8

static olarak bildirilmiş bir dizinin elemanlarının, içinde bildirildiği fonksiyonun her çağrılışında 0’a

atandığını düşünmek.

1 /\* Şekil 6.11: fig06\_11.c

2 Statik dizilere 0 ilk değeri atandı \*/

3 #include <stdio.h>

4

5 void statikDiziIlk ( void );

6 void otomatikDiziIlk( void );

7

8 int main( )

9 {

10 printf( "Her fonksiyona ilk çağrı:\n" );

11 statikDiziIlk ( );

12 otomatikDiziIlk ( );

13 printf( "\nHer fonksiyona ikinci çağrı:\n" );

14 statikDiziIlk ( );

15 otomatikDiziIlk ( );

16 return 0;

17 }

18

19 /\* a statik yerel diziyi kanıtlayan fonksiyon \*/

20 void statikDiziIlk ( void )

21 {

22 static int a[ 3 ];

23 int i;

215

24

25 printf( "\nstatikDiziIlk’e girerken değerler:\n" );

26

27 for ( i = 0; i <= 2; i++ )

28 printf( "dizi1[%d] = %d ", i, a[ i ] );

29

30 printf( "\nstatikDiziIlk’den çıkarken değerler:\n" );

31

32 for ( i = 0; i <= 2; i++ )

33 printf( "dizi1[%d] = %d ", i, a[ i ] += 5 );

34 }

35

36 /\* otomatik yerel diziyi kanıtlayacak fonksiyon \*/

37 void otomatikDiziIlk ( void )

38 {

39 int a[ 3 ] = { 1, 2, 3 }, i;

40

41 printf( "\n\notomatikDiziIlk’e girerken değerler:\n" );

42

43 for ( i = 0; i <= 2; i++ )

44 printf("dizi1 [ %d ] = %d ", i, a[ i ] );

45

46 printf( "\notomatikDiziIlk’den çıkarken değerler:\n" );

47

48 for ( i = 0; i <= 2; i++ )

49 printf( "dizi1[ %d ] = %d ", i, a[ i ] += 5 );

50 }

Her fonksiyona ilk çağrı:

statikDiziIlk’e girerken değerler:

dizi1[0] = 0 dizi1[1] = 0 dizi1[2] = 0

statikDiziIlk’den çıkarken değerler:

dizi1[0] = 5 dizi1[1] = 5 dizi1[2] = 5

otomatikDiziIlk’e girerken değerler:

dizi1[0] = 1 dizi1[1] = 2 dizi1[2] = 3

otomatikDiziIlk’den çıkarken değerler:

dizi1[0] = 6 dizi1[1] = 7 dizi1[2] = 8

Her fonksiyona ikinci çağrı:

statikDiziIlk’e girerken değerler:

dizi1[0] = 5 dizi1[1] = 5 dizi1[2] = 5

statikDiziIlk’den çıkarken değerler:

dizi1[0] = 10 dizi1[1] = 10 dizi1[2] = 10

otomatikDiziIlk’e girerken değerler:

dizi1[0] = 1 dizi1[1] = 2 dizi1[2] = 3

otomatikDiziIlk’den çıkarken değerler:

dizi1[0] = 6 dizi1[1] = 7 dizi1[2] = 8

216

Şekil 6.11 Statik dizilere programcı tarafından ilk değerleri atanmazsa, otomatik olarak 0

atanır.

6.5 DİZİLERİ FONKSİYONLARA GEÇİRMEK

Bir dizi argümanını fonksiyona geçirebilmek için, dizinin ismini parantez kullanmadan

belirtmeliyiz. Örneğin, eğer saatliksicaklik dizisi

int saatliksicaklik [24];

olarak bildirilmişse

diziyiayarla ( saatliksicaklik,24);

fonksiyon çağrısı, saatliksicaklik dizisini ve boyutunu diziyiayarla fonksiyonuna geçirir. Bir

diziyi fonksiyona geçirirken, fonksiyonun dizinin belirli sayıdaki elemanını işleyebilmesi için,

dizinin boyutu da geçirilir.

C, dizileri fonksiyonlara otomatik olarak referansa göre çağırma yöntemiyle geçirir ; çağrılan

fonksiyonlar, çağırıcının orijinal dizilerindeki elemanların değerlerini değiştirebilir. Dizinin

ismi, gerçekte dizinin ilk elemanının adresidir! Dizinin başlangıç adresi geçirildiğinden,

çağrılan fonksiyon dizinin nerede tutulduğunu kesin olarak bilir. Bu sebepten, çağrılan

fonksiyon, fonksiyon gövdesinde dizinin elemanlarını değiştirirken gerçek dizinin

elemanlarını orijinal hafıza konumlarında değiştirmektedir.

Şekil 6.12, dizi isminin gerçektende dizinin ilk elemanının başlangıç adresini olduğunu %p

dönüşüm belirteci kullanarak ve dizi, &dizi[0] ve &dizi değerlerini yazdırarak

göstermektedir. %p dönüşüm belirteci, adresleri yazdırmak için kullanılan özel bir belirteçtir.

%p dönüşüm belirteci genellikle adresleri onaltılık sistemde yazdırır. Onaltılık sistemlerde

(heksadecimal) sayılar, 0’dan 9’a kadar rakamları ve A’dan F’ye kadar harfleri içermektedir.

Bunlar, genellikle çok büyük tamsayıları kısaltarak gösterebilmek için kullanılırlar. Ekler E

ikilik ( binary ), sekizlik ( octal ) ve onaltılık ( heksadecimal ) sayı sistemleri arasındaki

ilişkiyi göstermektedir. Çıktı, dizi ve &dizi[0] için aynı değeri (bu örnekte 0065FDF0)

göstermektedir. Bu programın çıktısı her sistemde farklı olabilir ancak adresler her zaman

özdeş çıkacaktır.

Performans İpuçları 6.4

Dizileri referansa göre çağırma, performans açısından oldukça mantıklıdır.

Eğer diziler değere göre çağırma ile geçirilmiş olsaydı, her elemanının kopyası

geçirilecekti. Büyük ve sıklıkla çağrılan dizilerde bu oldukça fazla vakit

alacaktı ve fazladan hafızaya ihtiyaç duyacaktı.

Yazılım Mühendisliği Gözlemleri 6.2

10.Ünitede açıklayacağımız basit bir numarayla dizileri değere göre geçirmek

mümkün olabilir.

Dizinin tümü referansa göre geçirilse de dizinin bağımsız elemanları basit değişkenlerde

olduğu gibi değere göre çağrılarla geçirilirler. Bu tarzda basit ve tek veri parçalarına skaler

nicelikler denir. Bir dizinin elemanını fonksiyona geçirmek için, dizinin ismini ve elemanın

217

dizideki belirtecini fonksiyon çağrısında argüman olarak kullanmak gerekir. 7.Ünitede, skaler

nicelikler için (bağımsız değişken ve dizi elemanları gibi) referansa göre çağrıların nasıl

yapılacağını göstereceğiz.

1 /\* Şekil 6.12: fig06\_12.c

2 Bir dizinin ismi &dizi[ 0 ] ile aynıdır. \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char dizi[ 5 ];

8

9 printf( " dizi = %p\n &dizi[0] = %p\n"

10 " &dizi= %p\n",

11 dizi, &dizi[ 0 ], &dizi);

12 return 0;

13 }

Dizi = 0065FDF0

&dizi[0] = 0065FDF0

&dizi = 0065FDF0

Şekil 6.12 Dizinin ismi ile dizinin ilk elemanının adresi aynıdır.

Bir fonksiyonun, fonksiyon çağrısı esnasında diziyi alabilmesi için, fonksiyonun parametre

listesinin alınacak diziyi belirlemesi gerekir. Örneğin, diziyiayarla fonksiyonunun başlığı

void diziyiayarla(int b[ ] , int boyut)

biçiminde yazılabilir. Böylece, diziyiayarla fonksiyonunun b parametresinde tamsayılardan

oluşan bir dizi ve boyut parametresinde de dizi elemanlarının sayısını almayı beklediği

belirtilir. Dizinin büyüklüğünü köşeli parantezler içine yazmaya gerek yoktur. Eğer yazılırsa,

derleyici bu sayının sıfırdan büyük olup olmadığını kontrol eder ve daha sonra da ihmal eder.

Negatif bir büyüklük belirtmek hata oluşmasına sebep olur. Diziler otomatik olarak referansa

göre geçirildiklerinden, çağrılan fonksiyon dizi ismi olarak b kullandığında aslında

çağırıcıdaki diziyi belirtmektedir (az önceki çağrıda saatliksicaklik). 7.ünitede, fonksiyonun

bir dizi aldığını belirten farklı gösterimler de tanıtacağız. İleride göreceğimiz gibi, bu

gösterimler C’de dizi ve göstericiler arasındaki yakın ilişkiye dayanmaktadır.

diziyiayarla fonksiyonunun garip görünen prototipini inceleyelim:

void diziyiayarla( int [ ] , int );

Bu prototip

void diziyiayarla ( int herhangiBirDiziIsmi [ ] , int herhangiBirDegisken);

218

biçiminde olabilirdi. Ancak 5.ünitede öğrendiğimiz gibi, C derleyicisi prototipler içindeki

değişken isimlerini ihmal eder.

İyi Programlama Alıştırmaları 6.6

Bazı programcılar programı daha açık hale getirmek için, fonksiyon prototipinde değişken isimleri

kullanırlar.Derleyici bunları ihmal eder.

Prototipin, derleyiciye argüman sayısını ve her argümanının tipini ( argümanların

çalıştırılacakları sırada ) söylediğini hatırlayınız.

Şekil 6.13, tüm diziyi geçirmekle, yalnızca bir dizi elemanı geçirmek arasındaki farkı

göstermektedir. Program, tamsayı dizisi a’nın ilk 5 elemanını yazdırmaktadır. ( 17 ve 18.satır)

Daha sonra a dizisi ve dizinin boyutu, a’nın elemanlarının her birinin 2 ile çarpıldığı,

diziyiAyarla fonksiyonuna (34.satırda tanımlanmıştır) geçirilmiştir. Şimdi program a[3]’ün

değerini yazdırmakta ve a[3]’ü elemaniAyarla fonksiyonuna (42.satırda tanımlanmıştır)

geçirmektedir. elemaniAyarla fonksiyonu, argümanını 2 ile çarpmakta ve yeni değerini

yazdırmaktadır. a[3] , main içinde yeniden yazdırıldığında a[3]’ün değiştirilmediğini çünkü

bağımsız dizi elemanlarının değere göre çağırma ile geçirildiğine dikkat ediniz.

1 /\* Şekil 6.13: fig06\_13.c

2 Dizileri ve dizi elemanlarını fonksiyonlara geçirme \*/

3 #include <stdio.h>

4 #define BOYUT 5

5

6 void diziyiAyarla( int [], int ); /\* garip gözükür \*/

7 void elemaniAyarla( int );

8

9 int main( )

10 {

11 int a[ BOYUT ] = { 0, 1, 2, 3, 4 }, i;

12

13 printf( "Bütün dizinin referansa göre çağrılarak geçmesinin "

14 "etkileri:\n\nOrijinal dizinin "

15 "değerleri:\n" );

16

17 for ( i = 0; i <= BOYUT - 1; i++ )

18 printf( "%3d", a[ i ] );

19

20 printf( "\n" );

21 diziyiAyarla ( a, BOYUT); /\* referansa göre çağrılarak geçti \*/

22 printf( "Ayarlanan dizinin değerleri:\n" );

23

24 for ( i = 0; i <= BOYUT - 1; i++ )

25 printf( "%3d", a[ i ] );

26

27 printf( "\n\n\nDizi elemanının değere göre çağrılarak "

28 "geçmesinin etkileri:\n\na[3] değeri %d\n", a[ 3 ] );

29 elemaniAyarla ( a[ 3 ] );

30 printf( "a[ 3 ] değeri %d\n", a[ 3 ] );

219

31 return 0;

32 }

33

34 void diziyiAyarla ( int b[], int boyut)

35 {

36 int j;

37

38 for ( j = 0; j <= boyut - 1; j++ )

39 b[ j ] \*= 2;

40 }

41

42 void elemaniAyarla ( int e )

43 {

44 printf( "elemaniAyarla da ki değer %d\n", e \*= 2 );

45 }

Bütün dizinin referansa göre çağrılarak geçmesinin etkileri

Orijinal dizinin değerleri:

0 1 2 3 4

Ayarlanan dizinin değerleri:

0 2 4 6 8

Dizi elemanının değere göre çağrılarak geçmesinin etkileri:

a[3] değeri 6

elemaniAyarla da ki değer 12

a[3] değeri 6

Şekil 6.13 Dizileri ve dizi elemanlarını fonksiyonlara geçirme

Programlarınızda, fonksiyonun diziyi değiştirmemesi gereken bazı durumlar olabilir. Diziler

referansa göre çağırma ile geçirildiğinden, bir dizideki değerlerin değiştirilmesini kontrol

etmek güçtür. C, const tip belirteciyle bir fonksiyon içinde dizinin elemanlarının değerlerinin

değiştirilmesini engeller. Bir dizi parametresinden önce const kullanılırsa, dizi elemanları

fonksiyon gövdesi içinde sabit hale gelir ve fonksiyon gövdesinde dizinin elemanlarını

değiştirmeye çalışmak, derleme anında hataya yol açar. Bu, programcıya, programını diziyi

değiştirmeyecek biçimde düzeltme imkanı verir. const belirteci, ANSI standardında açıkça

belirlenmiş olsa da C sistemlerinin bunu uygulama yetenekleri değişebilir.

Şekil 6.14, const belirtecinin kullanımını göstermektedir. diziyiAyarlamayiDene fonksiyonu

(16.satır), const int b[ ] parametresiyle tanımlanarak b dizisi sabit ve değiştirilemez hale

220

getirilmiştir. Programın çıktısı, derleyicinin hata mesajlarını göstermektedir. Hatalar sizin

sisteminizde farklı olabilir. Fonksiyonun, dizi elemanlarını değiştirmek için yaptığı üç

deneme de derleyici hatasıyla karşılaşmıştır. const belirteci 7.ünitede yeniden tartışılacaktır.

Yazılım Mühendisliği Gözlemleri 6.3

const tip belirteci, fonksiyon tanımında parametrelere uygulanarak, orijinal dizinin fonksiyon gövdesi

içinde değiştirilmesi engellenebilir. Bu, en az yetki prensibinin başka bir örneğidir. Fonksiyonlara,

gerçekten gerekmedikçe diziyi değiştirme yeteneği verilmemelidir.

1 /\* Şekil 6.14: fig06\_14.c

2 const tip belirtecinin dizilerde kullanımı \*/

3 #include <stdio.h>

4

5 void diziyiAyarlamayiDene ( const int [] );

6

7 int main( )

8 {

9 int a[] = { 10, 20, 30 };

10

11 diziyiAyarlamayiDene ( a );

12 printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );

13 return 0;

14 }

15

16 void diziyiAyarlamayiDene ( const int b[] )

17 {

18 b[ 0 ] /= 2; /\* hata \*/

19 b[ 1 ] /= 2; /\* hata \*/

20 b[ 2 ] /= 2; /\* hata \*/

21 }

Compiling...

Fig06\_14.c(18) :error C2166: l-value specifies const object

Fig06\_14.c(19) :error C2166: l-value specifies const object

Fig06\_14.c(20) :error C2166: l-value specifies const object

Şekil 6.14 const tip belirtecinin kullanımı

221

6.6 DİZİLERİ SIRALAMAK

Veri sıralamak (örneğin, verileri artan ya da azalan bir sırada yerleştirmek) en önemli

bilgisayar uygulamalarından biridir. Bir banka, bütün çekleri hesap numarasına göre

sıralayarak her ay sonunda bankanın hesaplarını yapmaktadır. Telefon şirketleri, abonelerini

soyadı ve adlarına göre sıralayarak onlara ulaşmayı kolaylaştırmaktadır. Hemen hemen tüm

organizasyonlar, verileri ve bazı durumlarda da oldukça büyük verileri sıralamak zorundadır.

Veri sıralama, bilgisayar bilimlerinde yoğun araştırmalara konu olan ilgi çekici bir konudur.

Bu ünitede belki de en basit sıralama yöntemlerini tartışacağız. Alıştırmalarda ve 12.ünitede

daha yüksek performans sağlayan karmaşık yöntemlerden de bahsedeceğiz.

Performans İpuçları 6.5

Genellikle en basit algoritmalar, en zayıf olanlarıdır. Bunların özelliği

kolaylıkla yazılmaları,test edilmeleri ve hatalarının kolay ayıklanmasıdır.

Ancak daha karışık algoritmalara, maksimum performansı gerçekleştirmek

için ihtiyaç duyulur.

Şekil 6.5, 10 elemanlı a dizisinin(9.satır) elemanlarının değerlerini artan bir sırada

dizmektedir. Bu tekniğe kabarcık sıralama ( buble sort ) denir çünkü en küçük değer, dizinin

en üstüne bir su kabarcığı gibi çıkmakta, büyük değerler ise dizinin sonuna doğru

çökmektedir. Bu teknik dizide bir çok tur yaptıracaktır. Her turda eleman çiftleri karşılaştırılır.

Eğer bir çift artan sırada ise ( ya da değerleri eşitse), değerleri olduğu gibi bırakılır. Eğer çift

azalan bir sırada ise dizinin içinde değerleri yer değiştirilir.

1 /\* Şekil 6.15: fig06\_15.c

2 Bu program bir dizinin değerlerini

3 artan sıraya sokar \*/

4 #include <stdio.h>

5 #define BOYUT 10

6

7 int main( )

8 {

9 int a[ BOYUT ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

10 int i, tur, tut;

11

12 printf( "Veriler orjinal sırasında\n" );

13

14 for ( i = 0; i <= BOYUT - 1; i++ )

15 printf( "%4d", a[ i ] );

16

17 for ( tur = 1; tur <= BOYUT - 1; tur++ ) /\* turlar \*/

18

19 for ( i = 0; i <= BOYUT - 2; i++ ) /\* bir tur \*/

20

21 if ( a[ i ] > a[ i + 1 ] ) { /\* bir karşılaştırma \*/

22 tut = a[ i ]; /\* bir değiştirme \*/

23 a[ i ] = a[ i + 1 ];

24 a[ i + 1 ] = tut;

25 }

222

26

27 printf( "\nVeriler artan sıralamada \n" );

28

29 for ( i = 0; i <= BOYUT - 1; i++ )

30 printf( "%4d", a[ i ] );

31

32 printf( "\n" );

33

34 return 0;

35 }

Veriler orjinal sırasında

2 6 4 8 10 12 89 45 37

Veriler artan sıralamada

2 4 6 8 10 12 37 45 89

Şekil 6.15 Bir diziyi kabarcık sıralama ile düzenlemek.

Program önce a [0] ile a [1]’i, daha sonra a [1] ile a [2]’yi, daha sonra a [2] ile a [3]’ü

karşılaştırmakta ve bu şekilde a [8] ile a [9]’u karşılaştırana kadar devam etmektedir. On

eleman olmasına rağmen dokuz karşılaştırmanın yapıldığına dikkat ediniz. Karşılaştırmaların

yapılma yöntemi yüzünden büyük bir değer tek bir turda dizide aşağıya doğru bir çok

pozisyon ilerleyebilirken, küçük bir değer yukarıya doğru yalnızca tek bir pozisyon

ilerleyebilir. İlk turda, en büyük değerin dizinin en alttaki elemanı olan a[9]’a gitmesi garanti

altına alınmıştır. İkinci turda, ikinci en büyük değerin a[8]’e gideceği garantidir. Dokuzuncu

turda, dokuzuncu en büyük değerin a[1]’e gitmesi garantidir. Bu, en küçük değeri a[0]’da

bırakır, böylece dizinin 10 elemanı olmasına rağmen diziyi sıralamak için yalnızca 9 geçiş

gereklidir.

Sıralama, yuvalı for yapıları (17 ile 25.satırlar arası) sayesinde yapılmaktadır. Eğer bir yer

değiştirme gerekliyse bu, aşağıdaki üç atama sayesinde yapılır:

tut = a [i];

a [i] = a [i+1];

a [i+1] = tut;

Bu atamalardaki tut değişkeni, değerlerin yeri değiştirilirken değerlerden birini geçici olarak

tutabilmek için kullanılır. Yer değiştirme aşağıdaki gibi yalnızca iki atama yapılarak

gerçekleştirilemez.

a [i] = a [i+1];

a [i+1] = a [i];

Örneğin, eğer a [ i ]’nin değeri 7 ve a [ i+1 ]’in değeri 5 olsaydı, ilk atamadan sonra her iki

değerde 5 olacak ve 7 değeri kaybolacaktı. Bu sebepten, tut değişkeni gibi fazladan bir

değişkene ihtiyaç duyarız.

Kabarcık sıralamanın en önemli özelliği yazılmasının kolay oluşudur. Buna rağmen kabarcık

sıralama yavaş çalışır. Bu, büyük diziler sıralanırken gözle görünür hale gelir. Alıştırmalarda

223

kabarcık sıralamanın daha etkili versiyonlarını araştıracağız. Şu ana kadar kabarcık

sıralamadan daha etkili yöntemler geliştirilmiştir. Bunlardan bir kaçını ileride inceleyeceğiz.

Daha teknik kurslar sıralama ve arama konularını daha detaylı bir şekilde incelemektedir.

6.7 ÖRNEKLER:ORTALAMA, MOD ve MEDYANI DİZİLER

KULLANARAK HESAPLAMAK

Şimdi daha büyük bir örnek inceleyeceğiz. Bilgisayarlar genellikle, araştırma ve oylama

sonuçlarını derlemek ve analiz etmek için kullanılırlar. Şekil 6.16, ilk değer olarak

araştırmaya verilmiş 99 (BOYUT sembolik sabiti ile gösterilmiştir) yanıtı kullanan cevap

dizisini kullanmaktadır. Her yanıt 1’den 9’a kadar bir rakamdır. Bu program 99 değerin mod,

medyan ve ortasını hesaplamaktadır.

1 /\* Şekil 6.16: fig06\_16.c

2 Bu program araştırma ve analiz yapar.

3 orta, medyan, ve mod hesaplar. \*/

4 #include <stdio.h>

5 #define BOYUT 99

6

7 void orta( const int [ ] );

8 void medyan( int [ ] );

9 void mod( int [ ], const int [ ] ) ;

10 void kabarcikSiralama( int [ ] );

11 void diziyiYazdir ( const int [ ] );

12

13 int main( )

14 {

15 int frekans[ 10 ] = { 0 };

16 int cevap [BOYUT] =

17 { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,

18 7, 8, 9, 5, 9, 8, 7, 8, 7, 8,

19 6, 7, 8, 9, 3, 9, 8, 7, 8, 7,

20 7, 8, 9, 8, 9, 8, 9, 7, 8, 9,

21 6, 7, 8, 7, 8, 7, 9, 8, 9, 2,

22 7, 8, 9, 8, 9, 8, 9, 7, 5, 3,

23 5, 6, 7, 2, 5, 3, 9, 4, 6, 4,

24 7, 8, 9, 6, 8, 7, 8, 9, 7, 8,

25 7, 4, 4, 2, 5, 3, 8, 7, 5, 6,

26 4, 5, 6, 1, 6, 5, 7, 8, 7 };

27

28 orta ( cevap );

29 medyan (cevap );

30 mod (frekans, cevap );

224

31 return 0;

32 }

33

34 void orta ( const int cevap [ ] )

35 {

36 int j, toplam = 0;

37

38 printf( "%s\n%s\n%s\n", "\*\*\*\*\*\*\*\*", " orta ", "\*\*\*\*\*\*\*\*" );

39

40 for ( j = 0; j <= BOYUT - 1; j++ )

41 toplam += cevap [ j ];

42

43 printf( "Orta veri nesnelerinin ortalama değeridir.\n"

44 "Ortalama bütün veri nesnelerinin\n"

45 "toplamının veri nesnelerinin sayısına( %d )\n"

46 "bölümüdür. Bu veriler için\n"

47 "ortalama: %d / %d = %.4f\n\n",

48 BOYUT, toplam, BOYUT, ( double ) toplam / BOYUT);

49 }

50

51 void medyan ( int cevap[ ] )

52 {

53 printf( "\n%s\n%s\n%s\n%s",

54 "\*\*\*\*\*\*\*\*", " medyan ", "\*\*\*\*\*\*\*\*",

55 "Sıralanmamış Dizi " );

56

57 diziyiYazdir( cevap );

58 kabarcikSiralama ( cevap );

59 printf( "\n\nSıralanmış dizi " );

60 diziyiYazdir( cevap );

61 printf( "\n\n Sıralanmış %d elemanlık dizide\n"

62 "medyan %d.elemandır\n"

63 "Bu çalıştırılmada medyan %d\n\n",

64 BOYUT, BOYUT/ 2, cevap[ BOYUT / 2 ] );

65 }

66

67 void mod( int frek[], const int cevap [] )

68 {

69 int puan, j, h, enBuyuk = 0, modDegeri = 0;

70

71 printf( "\n%s\n%s\n%s\n",

72 "\*\*\*\*\*\*\*\*", " Mod", "\*\*\*\*\*\*\*\*" );

73

74 for (puan = 1; puan <= 9; puan ++ )

75 frek[ puan ] = 0;

76

77 for ( j = 0; j <= BOYUT - 1; j++ )

78 ++frek [ cevap[ j ] ];

79

80 printf( "%s%11s%19s\n\n%54s\n%54s\n\n",

225

81 " Cevap ", " Frekans ", "Histogram",

82 "1 1 2 2", "5 0 5 0 5" );

83

84 for (puan = 1; puan <= 9; puan ++ ) {

85 printf( "%8d%11d ", puan, frek[puan] );

86

87 if ( frek[puan] > enBuyuk) {

88 enBuyuk = frek[puan];

89 modDegeri = puan;

90 }

91

92 for ( h = 1; h <= frek[puan]; h++ )

93 printf( "\*" );

94

95 printf( "\n" );

96 }

97

98 printf( "Mod en sık rastlanan değerdir\n"

99 "Bu çalıştırılmada mod %d dir."

100 " %d kez rastlanmıştır\n", modDegeri, enBuyuk);

101 }

102

103 void kabarcikSiralama( int a[] )

104 {

105 int tur, j, tut;

106

107 for ( tur = 1; tur <= BOYUT - 1; tur++ )

108

109 for ( j = 0; j <= BOYUT - 2; j++ )

110

111 if ( a[ j ] > a[ j + 1 ] ) {

112 tut = a[ j ];

113 a[ j ] = a[ j + 1 ];

114 a[ j + 1 ] = tut;

115 }

116 }

117

118 void diziyiYazdir( const int a[] )

119 {

120 int j;

121

122 for ( j = 0; j <= BOYUT - 1; j++ ) {

123

124 if ( j % 20 == 0 )

125 printf( "\n" );

126

127 printf( "%2d", a[ j ] );

128 }

129 }

Şekil 6.16 Araştırma Analiz Programı

226

\*\*\*\*\*\*\*\*

Orta

\*\*\*\*\*\*\*\*

Orta veri nesnelerinin ortalama değeridir.

Ortalama bütün veri nesnelerinin

toplamının veri nesnelerinin sayısına( 99 )

bölümüdür. Bu veriler için

ortalama: 681 / 99 = 6.8788

\*\*\*\*\*\*\*\*

Medyan

\*\*\*\*\*\*\*\*

Sıralanmamış Dizi

6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7, 8,

6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8, 9,

6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5, 3,

5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7, 8,

7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

Sıralanmış dizi

1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5

5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7

7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8

8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8

9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

Sıralanmış 99 elemanlık dizide

medyan 49.elemandır.s

Bu çalıştırılmada medyan 7

\*\*\*\*\*\*\*\*

Mode

\*\*\*\*\*\*\*\*

227

Cevap Frekans Histogram

1 1 2 2

5 0 5 0 5

1 1 \*

2 3 \*\*\*

3 4 \*\*\*\*

4 5 \*\*\*\*\*

5 8 \*\*\*\*\*\*\*\*

6 9 \*\*\*\*\*\*\*\*\*

7 23 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8 27 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9 19 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Mod en sık rastlanan değerdir\n"

Bu çalıştırılmada mod 8 dir."

27 kez rastlanmıştır

Şekil 6.17 Araştırma Analiz Programının örnek bir çalıştırılması

Orta, 99 değerin aritmetik ortalamasıdır. orta fonksiyonu ( 34.satır ), ortayı 99 değeri

toplayıp, toplamı 99’a bölerek hesaplamaktadır.

Medyan, ortadaki değerdir. medyan fonksiyonu (51.satır) medyanı bulabilmek için

kabarciksiralama fonksiyonunu (103.satırda tanımlanmıştır) çağırmaktadır. Böylece,

yanıtlardan oluşan diziyi artan bir sırada dizdirip, sıralanmış diziden ortadaki elemanı

cevap[BOYUT/2] ile seçmektedir. Çift sayıda eleman olduğunda, medyan ortadaki iki

elemanın aritmetik ortalaması hesaplanarak bulunmalıdır. Ancak medyan fonksiyonu şu anda

bu yeteneğe sahip değildir. diziyiyazdir fonksiyonu (118.satır), cevap dizisini yazdırmak için

çağrılmıştır.

Mod, 99 yanıt arasından en çok karşılaşılanıdır. mod fonksiyonu (67.satır) mod değerine, her

tipte yanıtın sayısını sayarak ve daha sonra da saydıkları arasında en büyük olanı seçerek

karar vermektedir. mod fonksiyonunun bu versiyonu bir düğümü çözememektedir

(Alıştırmalar 6.14’e bakınız). mod fonksiyonu, mod değerine karar verebilmesine yardımcı

olması için çizgi grafik de kullanmaktadır. Şekil 6.17, bu programın örnek bir çıktısını

göstermektedir. Bu örnek, dizi problemlerinde, dizileri fonksiyonlara geçirmek de dahil olmak

üzere bir çok genel işlemi içermektedir.

6.8 DİZİLERDE ARAMA YAPMAK

Programcı sıklıkla, dizilerde tutulan büyük miktarlarda veri ile çalışacaktır. Bir dizinin, belli

bir arama değerine eşit olan bir değer içerip içermediğine karar vermek gerekebilir. Dizinin

belirli bir elemanını bulma sürecine arama denir. Bu kısımda iki arama tekniğini (basit lineer

arama ve daha karmaşık olan ikili arama teknikleri) anlatacağız. Bu ünitenin sonundaki

alıştırma 6.34 ve 6.35’te bu iki tekniğin yinelemeli versiyonlarını soracağız.

Lineer arama (Şekil 6.18), dizinin her elemanını arama değeriyle karşılaştırmaktadır. Dizi

herhangi bir şekilde sıralanmadığından değer ilk ya da son elemanda bulunabilir. Bu sebepten,

program ortalama olarak, arama değeriyle dizinin elemanlarının yarısını karşılaştırmalıdır.

228

1 /\* Şekil 6.18: fig06\_18.c

2 Dizide lineer arama yapmak \*/

3 #include <stdio.h>

4 #define BOYUT 100

5

6 int lineerArama( const int [], int, int );

7

8 int main( )

9 {

10 int a[ BOYUT ], x, aramaDegeri, eleman;

11

12 for ( x = 0; x <= BOYUT - 1; x++ ) /\* veri oluştur \*/

13 a[ x ] = 2 \* x;

14

15 printf( "Arama değeri tamsayısını gir:\n" );

16 scanf( "%d", & aramaDegeri);

17 eleman = lineerArama( a, aramaDegeri, BOYUT);

18

19 if ( eleman != -1 )

20 printf( "Bu değer, eleman %d de bulundu\n", eleman );

21 else

22 printf( "Bu değer bulunamadı\n" );

23

24 return 0;

25 }

26

27 int lineerArama( const int dizi[], int anahtar, int boyut )

28 {

29 int n;

30

31 for ( n = 0; n <= boyut - 1; ++n )

32 if ( dizi[ n ] == anahtar )

33 return n;

34

35 return -1;

36 }

Arama değeri tamsayısını gir:

36

Değer,eleman 18 de bulundu

Arama değeri tamsayısını gir:

37

Değer bulunamadı

229

Şekil 6.18 Dizide lineer arama yapmak

Lineer arama , küçük ya da sıralanmamış dizilerde iyi bir şekilde çalışır. Ancak büyük diziler

için lineer arama yetersiz kalmaktadır. Eğer dizi sıralanmışsa, daha hızlı olan ikili arama

tekniği kullanılabilir.

İkili arama tekniği, her karşılaştırmadan sonra sıralanmış bir dizideki elemanların yarısını

elemektedir. Algoritma, dizinin ortadaki elemanını bulmakta ve bu elemanın değerini arama

değeriyle karşılaştırmaktadır. Eğer ikisi eşitse, arama değeri bulunmuştur ve o elemanın dizi

belirteci geri döndürülür. Eğer bu iki değer eşit değilse, problem dizinin yarısını aramaya

indirgenmiştir. Eğer arama değeri dizinin ortadaki elemanından daha küçükse, dizinin ilk

yarısı aranır. Aksi takdirde ise dizinin ikinci yarısı aranır. Eğer arama değeri belirlenen alt

dizide de (orijinal dizinin bir parçası) bulunamazsa, algoritma orijinal dizinin dörtte birinde

tekrarlanır. Arama, alt dizilerden birinin ortadaki elemanıyla arama değeri eşit olunca ya da

alt dizi arama değerine eşit olmayan tek bir elemana sahip oluncaya dek (yani arama değeri

bulunamayıncaya dek) devam eder.

En kötü durumda ikili arama, 1024 elemanlı bir dizide arama yaparken en fazla 10

karşılaştırma yapacaktır.1024’ü sürekli olarak ikiye bölmek 512,256,128,64,32,16,8,4,2 ve 1

değerlerini verir. 1024 ( 2

10 ) sayısı 1 değerini elde etmek için ikiye 10 kez bölünmüştür. 2’ye

bölmek, ikili aramada bir karşılaştırma yapmak ile eşdeğerdir. 1048576 ( 2

20 ) elemanlı bir

dizide arama değerini bulmak en fazla 20 karşılaştırma gerektirmektedir. Bir milyar eleman

içeren bir dizide arama değerini bulmak için en fazla 30 karşılaştırma yapılmalıdır. Bu, arama

değerini bulabilmek için ortalama olarak dizinin elemanlarının yarısıyla karşılaştırma yapan

lineer aramaya göre performansta olağanüstü bir artış demektir. Bir milyar elemana sahip bir

dizide, bu fark 500 milyon karşılaştırma ile 30 karşılaştırma arasındadır! Bir dizi için

yapılacak en fazla karşılaştırma, 2’nin dizideki eleman sayısından büyük ilk üssü ile

bulunabilir.

Şekil 6.19, ikiliArama fonksiyonunun tekrarlı versiyonunu göstermektedir. Fonksiyon

(32.satırda tanımlanmıştır) 4 argüman almaktadır. Bunlar ; b tamsayı dizisi, bir tamsayı olan

aramaDegeri, dizinin en düşük belirtecini gösteren enAlt ve dizinin en büyük belirtecini

gösteren enUst olarak belirlenmiştir. Eğer arama değeri bir alt dizinin ortadaki elemanıyla

eşleşmezse, enAlt ya da enUst argümanı daha küçük bir alt dizide aramanın devam

edebilmesi için değiştirilir. Eğer arama değeri ortadaki elemandan küçükse, enUst belirteci

orta-1 olacak biçimde değiştirilir ve arama, enAlt belirteci ile orta-1 arasındaki elemanlarda

devam ettirilir. Eğer arama değeri ortadaki elemandan daha büyükse, enAlt belirteci orta+1

olacak hale getirilir ve arama, orta+1 ile enUst arasındaki elemanlarda devam ettirilir.

Program 15 elemanlı bir dizi kullanmaktadır. 2’nin dizi elemanı sayısından büyük ilk üssü 16

(24

) olduğundan arama değerini bulmak için en fazla 4 arama yapmak gerekmektedir.

Program, baslikYazdir fonksiyonu (54.satır) ile dizi belirteçlerini yazdırmakta ve

satirYazdir fonksiyonu (73.satır) ile ikili arama sürecindeki her alt diziyi yazdırmaktadır. Her

alt dizideki orta eleman, arama değeriyle karşılaştırılan değeri göstermek için bir yıldız

karakteri (\*) ile belirtilmiştir.

1 /\* Şekil 6.19: fig06\_19.c

2 Bir dizide ikili arama \*/

3 #include <stdio.h>

4 #define BOYUT 15

5

230

6 int ikiliArama( const int [ ], int, int, int );

7 void baslikYazdir( void );

8 void satirYazdir( const int [], int, int, int );

9

10 int main( )

11 {

12 int a[BOYUT], i, anahtar, sonuc;

13

14 for ( i = 0; i <= BOYUT - 1; i++ )

15 a[ i ] = 2 \* i;

16

17 printf( "0 ile 28 arasında bir sayı giriniz: " );

18 scanf( "%d", & anahtar );

19

20 baslikYazdir ( );

21 sonuc = ikiliArama ( a, anahtar, 0, BOYUT - 1 );

22

23 if (sonuc!= -1 )

24 printf( "\n%d, dizi elemanı %d içinde bulundu\n", anahtar, sonuc);

25 else

26 printf( "\n%d bulunamadı\n", anahtar);

27

28 return 0;

29 }

30

31 int ikiliArama ( const int b[], int aramaDegeri, int enAlt, int enUst)

32 {

33 int orta;

34

35 while (enAlt<= enUst) {

36 orta = ( enAlt + enUst ) / 2;

37

38 satirYazdir ( b, enAlt, orta, enUst);

39

40 if (aramaDegeri == b[orta] )

41 return orta;

42 else if (aramaDegeri < b[orta] )

43 enUst = orta - 1;

44 else

45 enAlt = orta + 1;

46 }

47

48 return -1; /\* aramaDegeri bulunamadı \*/

49 }

50

51 /\* Çıkış için bir başlık yazdır \*/

52 void baslikYazdir ( void )

53 {

54 int i;

55

231

56 printf( "\nBelirteçler:\n" );

57

58 for ( i = 0; i <= BOYUT - 1; i++ )

59 printf( "%3d ", i );

60

61 printf( "\n" );

62

63 for ( i = 1; i <= 4 \* BOYUT; i++ )

64 printf( "-" );

65

66 printf( "\n" );

67 }

68

69 /\* Dizinin işlem yapılan kısmını gösteren bir

70 satır çıktı yazdır. \*/

71 void satirYazdir ( const int b[], int enAlt, int orta, int enUst)

72 {

73 int i;

74

75 for ( i = 0; i <= BOYUT - 1; i++ )

76 if ( i < enAlt || i > enUst)

77 printf( " " );

78 else if ( i == orta)

79 printf( "%3d\*", b[ i ] ); /\* ortadaki değeri işaretle \*/

80 else

81 printf( "%3d ", b[ i ] );

82

83 printf( "\n" );

84 }

0 ile 28 arasında bir sayı giriniz: 25

Belirteçler:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 2 4 6 8 10 12 14\* 16 18 20 22 24 26 28

16 18 20 22\* 24 26 28

24 26\* 28

24\*

25 bulunamadı

0 ile 28 arasında bir sayı giriniz: 8

Belirteçler:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 2 4 6 8 10 12 14\* 16 18 20 22 24 26 28

0 2 4 6\* 8 10 12

232

8 10\* 12

8\*

8, dizi elemanı 4 içinde bulundu

0 ile 28 arasında bir sayı giriniz: 8

Belirteçler:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 2 4 6 8 10 12 14\* 16 18 20 22 24 26 28

0 2 4 6\* 8 10 12

6, dizi elemanı 3 içinde bulundu

Şekil 6.19 Sıralı bir dizi için ikili arama

6.9 ÇOK BOYUTLU DİZİLER

C’de diziler çok boyutlu olabilir. Çok boyutlu dizilerin genel kullanımı, satırlar ve sütunlar

biçiminde düzenlenmiş değerler içeren tabloları göstermektir. Bir tablo elemanını

belirleyebilmek için iki belirteç kullanmalıyız: İlk belirteç (genellikle) elemanın satırını ve

ikinci belirteç (genellikle) elemanın sütununu belirler. Belirli bir elemanı tanımlayabilmek

için iki belirteç kullanan tablolar ya da diziler iki-boyutlu diziler olarak adlandırılır. Çok

boyutlu dizilerin iki belirteçten daha fazla belirtece sahip olabileceğine dikkat ediniz. ANSI

standardı bir ANSI-C sisteminin en az 12 dizi belirtecini desteklemesi gerektiğini belirtmiştir.

Şekil 6.20, iki boyutlu olan bir a dizisini göstermektedir. Dizi 3 satır ve 4 sütun içermektedir,

bu sebepten 3’e-4 dizi olarak da adlandırılır.Genelde m satırlı ve n sütunlu bir dizi m’e-n dizi

olarak adlandırılır.

a [0] [0] a [0] [1] a [0] [2] a [0] [3]

a [1] [0] a [1] [1] a [1] [2] a [1] [3]

a [2] [0] a [2] [1] a [2] [2] a [2] [3]

Şekil 6.20 3 satır ve 4 sütuna sahip iki boyutlu bir dizi

Sütun Sütun Sütun Sütun

0 1 2 3

Satır 0

Satır 1

Satır 2

Sütun belirteci

Satır belirteci

Dizi ismi

233

a dizisi içindeki her eleman, Şekil 6.20’de a[i][j] biçiminde bir eleman ismi ile

adlandırılmıştır. Burada, a dizinin ismi, i ve j ise a içindeki her elemanı kendine has bir

biçimde belirleyen belirteçlerdir. İlk satırdaki tüm eleman isimlerinin ilk belirteç olarak 0’a ve

dördüncü sütundaki elemanların isimlerinin 3 belirtecine sahip olduklarına dikkat ediniz.

Genel Programlama Hataları 6.9

İki boyutlu bir diziyi a[x][y] yerine a[x,y] biçiminde belirlemek.

Çok boyutlu bir diziye tek belirteçli dizilerde olduğu gibi bildirim esnasında değerler

atanabilir. Örneğin, iki boyutlu bir dizi

int b[2][2]={ {1, 2}, {3, 4} };

biçiminde bildirilip, değerlere atanabilir. Değerler parantezler içinde satırlara göre

gruplandırılmıştır. Bu sebepten, 1 ve 2 b[0][0] ve b[0][1], 3 ve 4 ise b[1][0] ve b[1][1]’e

atanmaktadır. Eğer bir satır için yeterince atama değeri yoksa, satırda kalan diğer elemanlar

0’a atanır. Bu sebepten,

int b[2][2]={ {1}, {3, 4} };

bildirimi b[0][0]’a 1, b[0][1]’e 0, b[1][0]’a 3 ve b[1][1]’e 4 atayacaktır.

Şekil 6.21, iki boyutlu dizilerde bildirim esnasında atama yapmayı göstermektedir. Program, 2

satır ve 3 sütunlu (her birinde 6 eleman olan) 3 adet dizi bildirmektedir. dizi1 bildirilirken

atama listesinde 3’er elemanlı 2 adet liste kullanılmıştır. İlk liste, dizinin ilk satır elemanlarını

1, 2 ve 3 değerlerine, ikinci liste ise dizinin ikinci satırını 4, 5 ve 6 değerlerine atamaktadır.

Eğer dizi1 bildiriminde listedeki parantezler kaldırılırsa, derleyici ilk satırın elemanlarını ve

daha sonrada ikinci satırın elemanlarını atayacaktır. dizi2 bildiriminde (10.satır) 5 atama

değeri bulunmaktadır. Atama değerleri önce ilk satıra daha sonra da ikinci satıra

atanmaktadır. Özel olarak bir atama değerine sahip olmayan elemanlar otomatik olarak 0’a

atanacaktır. Bu yüzden, dizi2[1][2] 0’a atanmıştır. dizi3 bildirimi (11.satır) iki liste içinde 3

atama değeri içermektedir. İlk liste, ilk satırın ilk iki elemanını özel olarak 1 ve 2’ye

atamaktadır. Üçüncü eleman ise 0’a atanmıştır. İkinci satır için olan liste ise ilk elemanı 4’e

atamaktadır. Kalan iki eleman 0’a atanmıştır.

1 /\* Şekil 6.21: fig06\_21.c

2 Çok boyutlu dizilere ilk değer atanması \*/

3 #include <stdio.h>

4

5 void diziyiYazdir( const int [ ][ 3 ] );

6

7 int main( )

8 {

9 int dizi1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } },

10 dizi2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 },

11 dizi3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

12

13 printf( "Satır satır dizi1’in elemanları:\n" );

14 diziyiYazdir ( dizi1 );

15

16 printf( " Satır satır dizi2’in elemanları:\n" );

234

17 diziyiYazdir ( dizi2 );

18

19 printf( " Satır satır dizi3’in elemanları:\n" );

20 diziyiYazdir ( dizi3 );

21

22 return 0;

23 }

24

25 void diziyiYazdir ( const int a[ ][ 3 ] )

26 {

27 int i, j;

28

29 for ( i = 0; i <= 1; i++ ) {

30

31 for ( j = 0; j <= 2; j++ )

32 printf( "%d ", a[ i ][ j ] );

33

34 printf( "\n" );

35 }

36 }

Satır satır dizi1’in elemanları:

1 2 3

4 5 6

Satır satır dizi2’in elemanları

1 2 3

4 5 0

Satır satır dizi3’in elemanları

1 2 0

4 0 0

Şekil 6.21 Çok boyutlu dizilere atama yapmak

Program, her dizinin elemanlarını yazdırmak için diziyiYazdir fonksiyonunu (25.satırda

tanımlanmıştır) çağırmaktadır. Fonksiyon tanımının, dizi parametresini const int a[ ][3]

olarak belirlediğine dikkat ediniz. Tek belirteçli bir dizi fonksiyon argümanı olarak

kullanıldığında, fonksiyonun parametre listesindeki dizinin parantezleri içini boş

bırakıyorduk. Çok boyutlu dizilerde ilk belirtece gerek yoktur ancak sonraki diğer tüm

belirteçler belirtilmelidir. Derleyici, bu belirteçleri çok boyutlu dizi elemanlarının hafıza

konumlarını belirlemek için kullanır. Tüm dizi elemanları, dizinin belirteç sayısı ne olursa

olsun, hafızada ard arda tutulur. İki boyutlu dizilerde hafızaya önce ilk satır daha sonra ise

ikinci satır yazılır.

Belirteç değerlerini parametre bildirimlerinde belirtmek, derleyicinin fonksiyona dizideki bir

elemanı nasıl bulacağını söylemesini sağlatır. İki boyutlu bir dizide her satır aslında tek

belirteçli bir dizidir. Belli bir satırdaki bir elemanın konumunu belirlemek için derleyici her

satırda kaç eleman bulunduğunu bilmek zorundadır. Böylece dizi elemanına ulaşırken uygun

sayıda hafıza konumunu atlayabilir. Bu sebepten, örneğimizde a[1][2]’ye ulaşırken derleyici

ikinci satıra ulaşabilmek için ilk satırın üç elemanını atlaması gerektiğini bilir. Daha sonra

derleyici o satırın üçüncü elemanına ulaşır.

235

Çoğu dizi işlemi for döngü yapısını kullanır. Örneğin, aşağıdaki yapı Şekil 6.20’deki a

dizisinin 3.satırındaki tüm elemanları 0’a atar.

for(sutun = 0; sutun <= 3; sutun++)

a[2][sutun]=0;

Üçüncü satırı belirledik, bu sebepten ilk belirtecin her zaman 2 olacağını biliyoruz. (0 ilk satır

ve 1 ikinci satırdır) for döngüsü yalnızca ikinci belirteci değiştirmektedir (yani sütun

belirtecini) Aşağıdaki ifadeler az önceki for yapısına denktir.

a[2][0] = 0;

a[2][1] = 0;

a[2][2] = 0;

Aşağıdaki yuvalı for yapısı, a dizisi içindeki tüm elemanların toplamını hesaplamaktadır.

toplam = 0;

for(satir = 0; satir <= 2; satir++)

for(sutun = 0; sutun <= 3; sutun++)

toplam += a [satir][sutun];

for yapısı, dizinin elemanlarını satır satır toplamaktadır. Dıştaki for yapısı satir’ı 0 yaparak

başlamaktadır. Böylece içteki for yapısıyla ilk satırın elemanları toplanabilir. Dıştaki for

yapısı satir’ı 1’e arttırmakta böylece ikinci satırın elemanları toplatılmaktadır. Daha sonra

dıştaki for yapısı satir’ı 2’ye arttırmakta böylece üçüncü satırın elemanları toplatılmaktadır.

Sonuç, yuvalı for yapısından çıkıldıktan sonra yazdırılmaktadır.

Şekil 6.22, üçe dörtlük bir ogrenciNotlari dizisindeki diğer genel işlemleri for yapısı

kullanarak yapmaktadır. Dizinin her satırı bir öğrenciyi ve her sütunu öğrencinin dönem

boyunca girdiği 4 sınavdan birinin sonucunu göstermektedir. Dizideki işlemler dört fonksiyon

tarafından yapılmaktadır. minimum fonksiyonu (satır 35), herhangi bir öğrenci tarafından

dönem boyunca alınan en düşük notu belirlemektedir. maksimum fonksiyonu herhangi bir

öğrenci tarafından dönem boyunca alınan en yüksek notu belirlemektedir. ortalama (satır 63)

fonksiyonu herhangi bir öğrencinin dönem ortalamasını hesaplamaktadır. diziyiYazdir

fonksiyonu (satır 74) iki boyutlu diziyi çizelge biçiminde yazdırmaktadır.

1 /\* Şekil 6.22: fig06\_22.c

2 İki boyutlu dizi kullanan örnek \*/

3 #include <stdio.h>

4 #define OGRENCILER 3

5 #define SINAVLAR 4

6

7 int minimum( const int [ ][SINAVLAR], int, int );

8 int maksimum( const int [ ][SINAVLAR], int, int );

9 double ortalama( const int [ ], int );

10 void diziyiYazdir( const int [ ][SINAVLAR], int, int );

11

12 int main( )

13 {

14 int ogrenci;

15 const int ogrenciNotlari [OGRENCILER][ SINAVLAR] =

236

16 { { 77, 68, 86, 73 },

17 { 96, 87, 89, 78 },

18 { 70, 90, 86, 81 } };

19

20 printf( "Dizi:\n" );

21 diziyiYazdir (ogrenciNotlari, OGRENCILER, SINAVLAR);

22 printf( "\n\nEn Düşük Not: %d\nEn Yüksek Not: %d\n",

23 minimum(ogrenciNotlari, OGRENCILER, SINAVLAR),

24 maksimum (ogrenciNotlari, OGRENCILER, SINAVLAR) );

25

26 for (ogrenci = 0; ogrenci <= OGRENCILER - 1; ogrenci ++ )

27 printf( "Öğrenci %d için ortalama not %.2f\n",

28 ogrenci,

29 ortalama (ogrenciNotlari [ogrenci], SINAVLAR) );

30

31 return 0;

32 }

33

34 /\* Minimum notu bul \*/

35 int minimum( const int notlar[][SINAVLAR],

36 int talebeler, int testler)

37 {

38 int i, j, dusukNot = 100;

39

40 for ( i = 0; i <= talebeler - 1; i++ )

41 for ( j = 0; j <= testler - 1; j++ )

42 if ( notlar[ i ][ j ] < dusukNot)

43 dusukNot = notlar[ i ][ j ];

44

45 return dusukNot;

46 }

47

48 /\* maksimum notu bul \*/

49 int maksimum( const int notlar[][SINAVLAR],

50 int talebeler, int testler )

51 {

52 int i, j, yuksekNot = 0;

53

54 for ( i = 0; i <= talebeler- 1; i++ )

55 for ( j = 0; j <= testler - 1; j++ )

56 if ( notlar[ i ][ j ] > yuksekNot)

57 yuksekNot = notlar[ i ][ j ];

58

59 return yuksekNot;

60 }

61

62 /\* Belirli bir sınavın ortalama notunun hesaplanması \*/

63 double ortalama( const int notlarinKumesi [], int testler )

64 {

65 int i, toplam = 0;

237

66

67 for ( i = 0; i <= testler - 1; i++ )

68 toplam += notlarinKumesi [ i ];

69

70 return ( double ) toplam / testler;

71 }

72

73 /\* Diziyi yazdır \*/

74 void diziyiYazdir ( const int notlar[][SINAVLAR],

75 int talebeler, int testler)

76 {

77 int i, j;

78

79 printf( " [0] [1] [2] [3]" );

80

81 for ( i = 0; i <= talebeler - 1; i++ ) {

82 printf( "\n ogrenciNotlari [%d] ", i );

83

84 for ( j = 0; j <= testler - 1; j++ )

85 printf( "%-5d", notlar[ i ][ j ] );

86 }

87 }

Dizi:

[0] [1] [2] [3]

ogrenciNotlari[0] 77 68 86 73

ogrenciNotlari[1] 96 87 89 78

ogrenciNotlari[2] 70 90 86 81

En Düşük Not: 68

En Yüksek Not: 96

Öğrenci 0 için ortalama not 76.00

Öğrenci 1 için ortalama not 87.00

Öğrenci 2 için ortalama not 81.00

Şekil 6.22 İki boyutlu dizilerin kullanan örnek

minimum, maksimum ve diziyiYazdir fonksiyonları 3’er argüman almaktadır;

ogrenciNotlari dizisi (her fonksiyonda notlar olarak adlandırılmıştır), öğrenci sayısı (dizinin

satırları) ve sınav sayısı (dizinin sütunları). Her fonksiyon, notlar dizisi içinde yuvalı for

yapıları sayesinde ilerlemektedir. Aşağıdaki yuvalı for yapısı, minimum fonksiyonu

tanımında yer almaktadır.

for( i=0 ; i<= talebe – 1 ; i++)

for(j=0; j<= testler –1 ;j++)

if(notlar[i][j]<dusukNot)

dusukNot=notlar[i][j];

238

Dıştaki for yapısı i ’yi (satır belirteci) 0 yaparak başlamakta, böylece ilk satırın elemanlarının

içteki for yapısının gövdesindeki dusukNot değişkeni ile karşılaştırılmasını sağlamaktadır.

İçteki for yapısı o andaki satırda yer alan dört not boyunca, her notu dusukNot ile

karşılaştırmaktadır. Eğer not dusukNot değişkeninin değerinden daha küçükse, dusukNot o

nota atanmaktadır. Daha sonra, dıştaki for yapısı satır belirtecini arttırarak 1 yapmaktadır.

Artık ikinci satırın elemanları dusukNot değişkeni ile karşılaştırılacaktır. Daha sonra, dıştaki

for yapısı satır sayısını bir arttırarak 2 yapmaktadır ve böylece üçüncü satırın elemanları

dusukNot değişkeniyle karşılaştırılmaktadır. Yuvalı yapıdan çıkıldığında dusukNot, iki

boyutlu dizideki en küçük notu tutmaktadır. maksimum fonksiyonu minimum fonksiyonuna

benzer bir biçimde çalışmaktadır.

ortalama fonksiyonu (satır 63) iki argüman alır ; notlarinKumesi olarak adlandırılan ve bir

öğrencinin test sonuçlarını tutan tek belirteçli bir dizi ile dizideki test sonuçlarının sayısı.

ortalama çağrıldığında fonksiyona ilk argüman ogrenciNotlari[ogrenci] geçirilir. Bu, iki

boyutlu dizinin bir satırının adresinin ortalama fonksiyonuna geçirilmesine sebep olur.

ogrenciNotlari[1] , dizinin ikinci satırının başlangıç adresidir. İki boyutlu bir dizinin, tek

belirteçli dizilerden oluşan bir dizi olduğunu ve tek belirteçli dizinin adının, dizinin hafızadaki

başlangıç adresi olduğunu hatırlayınız. ortalama fonksiyonu, dizi elemanlarının toplamını

hesaplar, toplamı test sonuçları sayısına böler ve sonucu ondalıklı bir sonuç olarak döndürür.

ÖZET

 C, değer listelerini dizilerde tutar. Bir dizi, aynı isme ve aynı tipe sahip olmaları

sebebiyle birbirleriyle ilişkili olan hafıza konumlarının bir grubudur. Bir dizinin

içindeki bir elemanı ya da konumu belirtmek için o dizinin adını ve elemanın dizi

içindeki pozisyonunu belirtmeliyiz

 Bir belirteç ya bir tamsayı ya da bir tamsayı deyimi olmalıdır. Eğer program belirteç

olarak deyim kullanıyorsa , deyim belirtecin değerine karar vermek için hesaplanır.

“Dizinin yedinci elemanı” ile “yedinci dizi elemanı” arasındaki farkı anlamak

önemlidir. Dizi belirteçleri sıfırdan başladığı için “dizinin yedinci elemanı” 6

belirtecine sahiptir. ”yedinci dizi elemanı” ise 7 belirtecine sahiptir ve aslında dizinin

sekizinci elemanıdır. Bu, “bir eksik” (off-by-one) hatalarının kaynağıdır.

 Diziler hafızada bir yer kaplarlar. 100 elemana sahip bir b tamsayı dizisi ile 27

elemana sahip bir x tamsayı dizisini bildirmek için programcı

int b[ 100 ] , x [ 27 ] ; yazar.

 char tipte bir dizi karakter stringlerini depolamakta kullanılabilir.

 Dizi elemanlarına ilk değer atamak için üç yol vardır: bildirim esnasında, atama ile ve

girişten değer alarak.

 Eğer dizideki elemanların sayısından daha az sayıda atama değeri varsa, C kalan

elemanların hepsini otomatik olarak 0 değerine atar.

 C, bilgisayarın var olmayan bir elemanı kullanmasını engelleyecek herhangi bir dizi

sınırı kontrolüne sahip değildir.

 Bir karakter dizisi, bir string kullanılarak ilk değerlere atanabilir.

 C’de tüm stringler null karakterle sonlanır. null karakterin, karakter sabiti olarak

gösterimi ‘\0’ biçimindedir.

239

 Karakter dizilerine, atama listesindeki bağımsız karakter sabitleriyle ilk değerler

atanabilir.

 Bir string aslında karakterlerden oluşan bir dizi olduğundan, string içindeki bağımsız

karakterlere dizi belirteci gösterimiyle erişebiliriz

 Bir karakter dizisine scanf ve %s dönüşüm belirtecini kullanarak klavyeden

okuyacağımız bir stringi alabiliriz.

 Bir stringi temsil eden karakter dizisi, printf ve %s dönüşüm belirteciyle

yazdırılabilir.

 Yerel dizi bildirimlerinde static kullanarak, dizinin fonksiyon her çağrıldığında

yeniden yaratılmasını ve fonksiyondan her çıkıldığında dizinin yok edilmesini

engelleyebiliriz.

 static olarak bildirilen diziler, derleme zamanında yalnızca bir kere otomatik olarak

ilk değerlere atanır. Eğer static bir dizi programcı tarafından özellikle ilk değerlere

atanmamışsa, dizinin elemanları derleyici tarafından 0’a atanacaktır.

 Bir diziyi fonksiyona geçirebilmek için fonksiyona dizinin ismi geçirilir. Bir dizinin

tek bir elemanını fonksiyona geçirmek için, dizinin ismini ve elemanın dizideki

belirtecini (kare parantezler içine alınmış şekilde) fonksiyon çağrısında argüman

olarak kullanmak gerekir.

 C, dizileri fonksiyonlara otomatik olarak referansa göre çağırma yöntemiyle geçirir ;

çağrılan fonksiyonlar, çağırıcının orijinal dizilerindeki elemanların değerlerini

değiştirebilir. Dizinin ismi, gerçekte dizinin ilk elemanının adresidir! Dizinin

başlangıç adresi geçirildiğinden, çağrılan fonksiyon dizinin nerede tutulduğunu kesin

olarak bilir.

 %p dönüşüm belirteci, adresleri onaltılık sistemde yazdırır.

 C, const tip belirteciyle bir fonksiyon içinde dizinin elemanlarının değerlerinin

değiştirilmesini engeller. Bir dizi parametresinden önce const kullanılırsa ,dizi

elemanları fonksiyon gövdesi içinde sabit hale gelir ve fonksiyon gövdesinde dizinin

elemanlarını değiştirmeye çalışmak, derleme anında hataya yol açar.

 Bir dizi kabarcık sıralama tekniği ile sıralanabilir. Bu teknik dizide bir çok tur

yaptıracaktır.Her turda eleman çiftleri karşılaştırılır.Eğer bir çift artan sırada ise( ya da

değerleri eşitse),değerleri olduğu gibi bırakılır. Eğer çift azalan bir sırada ise dizinin

içinde değerleri yer değiştirilir. Küçük diziler için kabarcık sıralama uygun olabilir

ancak büyük dizilerle kullanıldığında diğer sıralama algoritmalarına göre yetersizdir.

 Lineer arama , dizinin her elemanını arama değeriyle karşılaştırmaktadır. Dizi

herhangi bir şekilde sıralanmadığından değer ilk ya da son elemanda bulunabilir. Bu

sebepten program ortalama olarak, arama değeriyle dizinin elemanlarının yarısını

karşılaştırmalıdır. Lineer arama , küçük ya da sıralanmamış dizilerde iyi bir şekilde

çalışır.

 İkili arama tekniği, her karşılaştırmadan sonra sıralanmış bir dizideki elemanların

yarısını elemektedir. Algoritma, dizinin ortadaki elemanını bulmakta ve bu elemanın

değerini arama değeriyle karşılaştırmaktadır. Eğer ikisi eşitse, arama değeri

bulunmuştur ve o elemanın dizi belirteci geri döndürülür. Eğer bu iki değer eşit

değilse, problem dizinin yarısını aramaya indirgenmiştir.

 En kötü durumda ikili arama, 1024 elemanlı bir dizide arama yaparken en fazla 10

karşılaştırma yapacaktır. 1048576(220) elemanlı bir dizide arama değerini bulmak en

fazla 20 karşılaştırma gerektirmektedir.Bir milyar eleman içeren bir dizide arama

değerini bulmak için en fazla 30 karşılaştırma yapılmalıdır

 Diziler satır ve sütün şeklinde düzenlenmiş bilgileri temsil etmek için kullanılabilirler.

Bir tablo elemanını belirleyebilmek için iki belirteç kullanılır: İlk belirteç (genellikle)

240

elemanın satırını ve ikinci belirteç (genellikle) elemanın sütununu belirler.Belirli bir

elemanı tanımlayabilmek için iki belirteç kullanan tablolar ya da diziler, iki-boyutlu

diziler olarak adlandırılır.

 C standardı ,bir sistemin en az 12 dizi belirtecini desteklemesi gerektiğini

belirtmektedir.

 Çok boyutlu bir diziye, bildirim esnasında atama listeleri kullanılarak değerler

atanabilir.

 Tek belirteçli bir dizi fonksiyon argümanı olarak kullanıldığında, fonksiyonun

parametre listesindeki dizinin parantezlerinin içi boş olarak kullanılır.Çok boyutlu

dizilerde, ilk belirtece gerek yoktur ancak sonraki diğer tüm belirteçler belirtilmelidir.

Derleyici, bu belirteçleri çok boyutlu dizi elemanlarının hafıza konumlarını belirlemek

için kullanır.

 Tek belirteçli bir dizi alan bir fonksiyona, iki boyutlu bir dizinin bir sırası

geçirilecekse, fonksiyona dizinin ismi ve ilk belirteç geçirilmelidir.

ÇEVRİLEN TERİMLER

array initializer list........................................dizi atama listesi

bar chart.................................................... çizgi grafik

bounds checking....................................... sınır kontrolü

bubble sort............................................... kabarcık sıralama

declare an array...................................... bir dizi bildirmek

linear search........................................... lineer arama (doğrusal arama da denir)

mean...................................................... orta

median.................................................. medyan

mode..................................................... mod

multiple-subscripted array.................. çok boyutlu dizi

null character.............................. null karakter

off-by-one error.................................. bir eksik hatası

replacement text................................ yerdeğiştirme metni

scalar............................................... skaler

search key....................................... arama anahtarı

sorting............................................ sıralama

string............................................. dize/ string

subscript....................................... belirteç [not: bilişim sözlüğünde bu terimin

karşılığı alt simge olarak verilmiş ancak dizilerle

kullanıldığında dizinin elemanını belirtmekte

kullanıldığından belirteç demeyi tercih ettik]

GENEL PROGRAMLAMA HATALARI

241

6.1“Dizinin yedinci elemanı” ile “yedinci dizi elemanı” arasındaki farkı anlamak

önemlidir.Dizi belirteçleri sıfırdan başladığı için “dizinin yedinci elemanı” 6

belirtecine sahiptir.”yedinci dizi elemanı” ise 7 belirtecine sahiptir ve aslında dizinin

sekizinci elemanıdır.Bu, “bir eksik”(off-by-one) hatalarının kaynağıdır.

6.2 Elemanlarına ilk değer verilmesi gerekilen bir dizinin elemanlarına ilk değer vermeyi

unutmak.

6.3 Diziye ilk değer atanacakken, dizi elemanından daha çok sayıda atama değeri kullanmak

bir yazım hatasıdır.

6.4 #define ve # include önişlemci komutlarını noktalı virgül ile sonlandırmak.Önişlemci

komutlarının C ifadeleri olmadığını hatırlayınız.

6.5 Sembolik bir sabite, çalıştırılabilir bir ifade içinde değer atamak. Sembolik sabit bir

değişken değildir. Derleyici tarafından,çalışma zamanında değerleri tutan değişkenler

gibi,sembolik sabitlere de hafızada yer ayrılmaz.

6.6 Dizi sınırları dışındaki bir elemanı kullanmak.

6.7 scanf ile klavyeden yazılan stringi tutabilecek kadar geniş olmayan bir karakter dizisi

kullanmak, veri kaybına ya da çalışma zamanlı diğer hatalara sebep olabilir.

6.8 static olarak bildirilmiş bir dizinin elemanlarının, içinde bildirildiği fonksiyonun her

çağrılışında 0’a atandığını düşünmek.

6.9 İki boyutlu bir diziyi a[x][y] yerine a[x,y] biçiminde belirlemek.

İYİ PROGRAMLAMA ALIŞTIRMALARI

6.1 Sembolik sabitler için yalnızca büyük harfler kullanın.Bu, sembolik sabitlerin

program içinde göze çarpmasını sağlayarak,programcıya bunların değişken

olmadıklarını hatırlatacaktır.

6.2 Programın açıklığı için çaba gösterin.Kimi zaman hafızanın ya da işlemci zamanının etkili

kullanılması,daha açık programlar yazmak için feda edilebilir.

6.3 Dizi boyunca döngü kullanırken, dizi belirteci asla 0’ın altına inmemeli ve her zaman

dizideki toplam eleman sayısından az olmalıdır(büyüklük-1).Döngü devam şartının bu

aralığın dışındaki elemanlara ulaşılmasını engellediğinden emin olun.

6.4 for yapısında en yüksek dizi belirtecini kullanarak “bir eksik” hatalarını ortadan

kaldırmak

6.5 Programlar, hatalı bilginin programın hesaplarını etkilememesi için tüm giriş

değerlerinin

doğruluğunu onaylamalıdır.

6.6 Bazı programcılar programı daha açık hale getirmek için, fonksiyon prototipinde değişken

isimleri kullanırlar.Derleyici bunları ihmal eder.

PERFORMANS İPUÇLARI

6.1 Kimi zaman, performans hususları açıklık hususlarından daha önemlidir.

6.2 Dizi sınırlarının dışındaki elemanları kullanmanın yaratacağı hatalar(genelde ciddi

hatalardır) sistemden sisteme farklılık gösterir.

6.3 Faaliyet alanına sıklıkla girip çıkan ve otomatik diziler içeren fonksiyonlarda,diziyi

static yaparak fonksiyonun her çağrısında dizinin yeniden yaratılmasını engelleyin.

6.4 Dizileri referansa göre çağırma, performans açısından oldukça mantıklıdır.Eğer

diziler değere göre çağırma ile geçirilmiş olsaydı, her elemanının kopyası

242

geçirilecekti.Büyük ve sıklıkla çağrılan dizilerde bu, oldukça fazla vakit alacaktı ve

fazladan hafızaya ihtiyaç duyacaktı.

6.5 Genellikle en basit algoritmalar, en zayıf olanlarıdır.Bunların özelliği kolaylıkla

yazılmaları,test edilmeleri ve hatalarının kolay ayıklanmasıdır.Ancak daha karışık

algoritmalara, maksimum performansı gerçekleştirmek için ihtiyaç duyulur.

YAZILIM MÜHENDİSLİĞİ GÖZLEMLERİ

6.1 Her dizinin boyutunu sembolik sabitlerle belirtmek, programı daha ölçülendirilebilir

yapar.

6.2 10.Ünitede açıklayacağımız basit bir numarayla, dizileri değere göre geçirmek

mümkün olabilir.

6.3 const tip belirteci, fonksiyon tanımında parametrelere uygulanarak, orijinal dizinin

fonksiyon gövdesi içinde değiştirilmesi engellenebilir.Bu, en az yetki prensibinin

başka bir örneğidir.Fonksiyonlara gerçekten gerekmedikçe diziyi değiştirme

yeteneği verilmemelidir.

Çözümlü Alıştırmalar

6.1 Aşağıdakileri cevaplayınız.

a) Değerlerin listeleri ve tabloları \_\_\_\_\_\_\_ de saklanır.

b) Bir dizinin elemanlarının birbiriyle ilişkisi, aynı \_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ sahip

olmalarıdır.

c) Bir dizinin, bir elemanını göstermek için kullanılan sayıya o dizinin \_\_\_\_\_\_\_ denir.

d) \_\_\_\_\_\_\_ dizinin uzunluğunu belirler. Programın ölçeklenmesi bakımından önemlidir.

e) Bir dizinin elemanlarının belli bir sıraya yerleştirilmesine dizininin \_\_\_\_\_\_\_ denir.

f) Bir dizinin belli bir değeri içerip içermediğini tarama işlemine dizide \_\_\_\_\_\_\_ denir.

g) İki belirteç içeren dizilere \_\_\_\_\_\_\_ diziler denir.

6.2 Aşağıdakilerin doğru veya yanlış olduğunu belirtiniz. Yanlış olanların neden yanlış

olduğunu açıklayınız.

a) Bir dizi, değişik türdeki değerleri saklayabilir.

b) Bir dizi belirteci, float türünde olabilir.

c) Bir dizinin elemanlarına verilen ilk değerler dizinin eleman sayısından daha küçükse,

ilk değeri atanmamış olan elemanlara C otomatik olarak ilk değer listesindeki en son

değeri atar.

d) Diziye atanan ilk değerlerin eleman sayısının, dizinin eleman sayısından daha büyük

olması bir hatadır.

e) Tek başına bir dizi elemanı bir fonksiyona gönderildiğinde ve burada değiştirildiği

zaman, fonksiyonun çağırıldığı yerde de artık bu yeni değere sahip olacaktır.

243

6.3 oran isminde bir diziyi göz önüne alarak aşağıdaki soruları cevaplayınız.

a) 10 yerine kullanılacak, BOYUT isminde bir sembolik sabit bildiriniz.

b) Bu diziyi, BOYUT eleman sayısında, float türünde ve bütün ilk değerleri 0 olacak

şekilde bildiriniz.

c) Dizinin başlangıçtan itibaren 4. elemanını yazınız.

d) Dizideki eleman 4’ ü yazınız.

e) Eleman 9’a 1.667’ yi atınız.

f) Dizinin 7.elemanına 3.333’ ü atınız.

g) Eleman 6 ve 9’u iki basamak duyarlıkta yazdırınız. Çıktı nasıl olur?

h) Dizinin bütün elemanlarını bir for döngüsü ile ekrana yazdırınız. Kontrol değişkeni

olarak x tamsayı değişkenini kullanınız. Çıktı nasıl olur?

6.4 tablo isminde bir diziyi göz önünde bulundurarak aşağıdaki soruları cevaplayınız.

a) Diziyi 3 satır ve 3 sütun içerecek şekilde ve BOYUT sembolik değişkeni de 3 olacak

şekilde bildiriniz.

b) Bu dizinin kaç elemanı vardır?

c) Dizinin elemanlarına, belirteçleri toplamını ilk değer olarak atayacak bir for döngüsü

yazınız. x ve y tamsayı değişkenlerinin kontrol değişkenleri olarak daha önce

bildirildiğini kabul ediniz.

d) tablo dizisinin içerdiği bütün değerleri yazdırınız. Dizinin ilk değerinin

int tablo[BOYUT][BOYUT] = {{1, 8}, {2, 4, 6}, {5}};

şeklinde atandığını ve x ve y kontrol değişkenlerinin daha

önceden bildirildiğini kabul ediniz.

6.5 Aşağıdaki program parçalarındaki hataları bulunuz ve düzeltiniz.

a) #define BOYUT 100;

b) BOYUT = 10;

c) int b[10] = {0}, i; olduğunu kabul edin.

for ( i = 0; i <= 10; i++)

b[i] = 1;

d) #include <stdio.h>;

e) int a[2][2] = {{1, 2}, {3, 4}}; olduğunu kabul edin.

a[1, 1] = 5;

Çözümler

6.1 a) diziler b) isim, tip c) belirteci d) sembolik sabit e) sıralanması f) arama g) iki boyutlu

6.2

a) Yanlış. Bir dizi sadece aynı türdeki değerleri içerebilir.

b) Yanlış. Bir dizi belirteci mutlaka bir tamsayı yada tamsayı belirten bir ifade olmalıdır.

c) Yanlış. C geri kalan elemanların ilk değerlerine otomatik olarak 0 atar.

d) Doğru.

244

e) Bir dizinin tek bir elemanı, değere göre çağırılarak fonksiyona geçer. Eğer dizinin

bütünü fonksiyona gönderilirse yapılan değişiklikler dizinin orijinalini de etkiler.

6.3

a) #define BOYUT 10

b) float oran[BOYUT] = {0};

c) oran[3]

d) oran[4]

e) oran[9] = 1.667;

f) oran[9] = 3.333;

g) printf (“%.2f %.2f\n”, oran[6], oran[9]);

çıktı: 3.33 1.67

h) for (x = 0; x <= BOYUT – 1; x++)

printf (“oran[%d] = %f\n”, x, oran[x]);

Çıktı:

Oran[0] = 0.000000

Oran[1] = 0.000000

Oran[2] = 0.000000

Oran[3] = 0.000000

Oran[4] = 0.000000

Oran[5] = 0.000000

Oran[6] = 3.333000

Oran[7] = 0.000000

Oran[8] = 0.000000

Oran[9] = 1.667000

6.4

a) int tablo[BOYUT][BOYUT]

b) 9 tane elemanı vardır.

c) for (x = 0; x <= BOYUT – 1; x++)

for (y = 0; y <= BOYUT – 1; y++)

tablo[x][y] = x + y;

d) for (x = 0; x <= BOYUT – 1; x++)

for (y = 0; y <= BOYUT – 1; y++)

printf (“tablo[%d][%d] = %d\n”, x, y, tablo[x][y]);

Çıktı:

Tablo[0][0] = 1

Tablo[0][1] = 8

Tablo[0][2] = 0

Tablo[1][0] = 2

Tablo[1][1] = 4

Tablo[1][2] = 6

Tablo[2][0] = 5

Tablo[2][1] = 0

Tablo[2][2] = 0

6.5

a) Hata: #define ön işlemci komutundan sonra noktalı virgül kullanılması.

Düzeltme: Noktalı virgülün silinmesi.

b) Hata: Atama operatörü ile sembolik bir sabite değer atanmaya çalışılması

245

Düzeltme: Atama operatörü olmaksızın, #define BOYUT 10 şeklinde sembolik sabite

değer atanması.

c) Hata: Dizinin içermediği bir elemanın kullanılmaya çalışılması (b[10])

Düzeltme: Kontrol değişkeninin son değerinin 9 yapılması.

d) Hata: #include önişlemci komutundan sonra noktalı virgül kullanılması.

Düzeltme: Noktalı virgülün silinmesi.

e) Hata: Dizi belirteçlerinin yanlış kullanılması.

Düzeltme: Bu ifadenin a[1] [1] = 5 şeklinde değiştirilmesi.

ALIŞTIRMALAR

6.6 Aşağıdaki boşlukları doldurunuz.

a) C, değerlerin listesini \_\_\_\_\_\_ saklar.

b) Bir dizinin elemanlarının birbirleriyle ilişkisi \_\_\_\_\_\_ sayesindedir.

c) Bir dizinin bir elemanını göstermek için köşeli parantez içinde kullanılan sayıya

\_\_\_\_\_\_ denir.

d) p dizinin 5 elemanının isimleri \_\_\_\_\_\_, \_\_\_\_\_\_, \_\_\_\_\_\_, \_\_\_\_\_\_ ve \_\_\_\_\_\_ dir.

e) Bir dizinin bir elemanının içeriğine o elemanın \_\_\_\_\_\_ denir.

f) Bir diziye isim verilmesi, türünün belirlenmesi, kaç eleman içereceğinin

belirtilmesine o dizinin \_\_\_\_\_\_ denir.

g) Bir dizinin elemanlarının artan ya da azalan bir sırayla yerleştirilmesine \_\_\_\_\_\_ denir.

h) İki boyutlu bir dizide, ilk belirteç elemanın \_\_\_\_\_\_, ikincisi ise \_\_\_\_\_\_ belirtir.

i) Bir m’ e n dizi \_\_\_\_\_\_ satır, \_\_\_\_\_\_ sütun ve \_\_\_\_\_\_ eleman içerir.

j) d dizisinin, 3.satır ve 5. sütunundaki eleman \_\_\_\_\_\_dir.

6.7 Aşağıdakilerin hangilerinin doğru, hangilerinin yanlış olduğuna karar verin. Yanlış

olanların neden yanlış olduğunu açıklayın.

a) Bir dizinin belli bir konumuyla ya da elemanıyla işlem yapmak için dizinin ismi ve

elemanın değeri kullanılır.

b) Dizinin bildirilmesi sırasında o dizi için hafızada yer ayrılır.

c) Bir p tamsayı dizisine 100 adet yer ayrılması için programcı

p[100];

bildirimini yapmalıdır.

d) 15 elemanlı bir dizinin elemanlarına 0 ilk değerini atamak için mutlaka bir for döngüsü

kullanılmalıdır.

e) İki boyutlu bir dizinin elemanlarını toplamak için yuvalı for ifadesi kullanılmalıdır.

f) 1, 2, 5, 6, 7, 7 kümesi için orta,medyan ve mod değerleri sırasıyla; 5, 6 ve 7’dir

6.8 Aşağıdakileri gerçekleştirecek C ifadelerini yazınız.

a) f dizisinin 7. elemanının değerini yazdırınız.

b) Tek belirteçli, float tipindeki b dizisindeki eleman 4’e bir değer girdisi yaptırınız.

c) Tek belirteçli, integer tipindeki g dizisinin 5 elemanına da 8 ilk değerini verin.

d) float tipindeki c dizisinin 100 elemanını toplayın.

e) a dizisini, b dizisinin ilk kısmına kopyalayın. float a[11] ve b[34] ; tanımlamasının

yapıldığını kabul edin.

246

f) 99 elemanlık float tipindeki bir dizide saklanan en büyük ve en küçük değerleri bulup

ekrana yazdırın.

6.9 2’ye 5’lik bir t tamsayı dizisini göz önünde bulundurarak

a) t’ yi tanımlayınız.

b) t’ nin kaç satırı vardır?

c) t’ nin kaç sütunu vardır?

d) t’ nin kaç elemanı vardır?

e) t’ nin 2. satırındaki elemanlarının isimlerini yazınız.

f) t’ nin kaç satırı vardır?

g) t’ nin 1. satır ve 2. sütun elemanını 0 yapan ifadeyi yazınız.

h) t’ nin bütün elemanlarını 0 yapan C ifadelerini yazınız. Döngü kullanmayınız.

i) t’ nin bütün elemanlarını 0 yapan bir yuvalı for döngüsü yazınız.

j) t’ nin elemanlarına klavyeden giriş yaptıracak ifadeyi yazınız.

k) t dizisinin içerdiği en küçük değeri bulup ekrana yazdıracak C ifadelerini yazınız.

l) t dizisinin ilk satır elemanlarını ekrana yazdıran bir program yazınız.

m) t’ nin dördüncü sütunundaki elemanları toplayan bir program yazınız.

n) t dizisini düzgün bir çizelge şeklinde ekrana yazdıran C ifadelerini yazınız. Sütun

belirteçlerini değerlerin üzerine başlık olarak, satır belirteçlerini de değerlerin soluna

satırları gösterecek şekilde yazdırın.

6.10 Bu problemi çözmek için tek belirteçli bir dizi kullanın. Bir şirket, satıcı olarak

çalıştırdığı elemanlarına ücretlerini komisyona dayalı olarak ödemektedir. Satıcılar

haftalık sabit 200$ ve o haftaki brüt satışlarından %9 alırlar. Örneğin brüt 3000$’ lık satış

yapan bir satıcı, 200$ ve 3000$’ın yüzde dokuzunu alır. Yani, toplam 470$ alır. Aşağıdaki

sınırlar içersinde kaç satıcının maaş aldığını hesaplayan bir program (sayıcılardan oluşan

bir dizi kullanın) yazınız. (Satıcıların maaşlarının tamsayı olduğunu kabul edin.)

6.11 Şekil 6.15’deki kabarcık sıralama algoritması büyük diziler için pek uygun değildir.

Bu programın performansını artırmak için aşağıdaki değişiklikleri yapınız.

a) İlk turdan sonra dizideki en büyük sayı, dizinin en büyük numaralı konumuna

geleceği kesindir. 2. turdan sonra ikinci en büyük sayıda yerini alır. Bu yüzden, her

turda dokuz karşılaştırma yapmak yerine programı 2.turda 8 kez, 3 turda 7 kez

karşılaştırma yapacak şekilde değiştiriniz.

b) Dizideki sayıların tümü ya da büyük bir kısmı zaten sıralanmış olabilir. O zaman

daha azı mümkünden neden dokuz kez dönecek bir döngü kullanılsın? Her tur

sonunda doğru sıralamanın yapılıp yapılmadığını kontrol ediniz. Eğer dizi

elemanları arasında bir yer değişikliği olmadıysa dizi doğru sırasına kavuşmuştur.

Dizi elemanları arasında bir yer değişikliği olmuşsa döngünün en az bir kez daha

dönmesi gerekmektedir.

6.12 Aşağıdaki tek belirteçli dizi işlemlerini yapan C ifadelerini yazınız.

a) 10 elemanı olan sayaclar tamsayı dizisinin ilk değerlerini 0 olarak atayınız.

b) bonus tamsayı dizisinin her elemanına 1 ekleyiniz.

c) float tipindeki aylikSicakliklar dizisine klavyeden 12 tamsayı girdiriniz.

d) enIyiSkorlar dizisinin 5 değerini ekrana sütun halinde ekrana yazdırınız.

247

6.13 Aşağıdaki ifadelerdeki hatayı ya da hataları bulunuz.

a) char str[5]; tanımlamasının yapıldığını kabul edin.

scanf(“%s”, str); /\* Kullanıcı hello girer \*/

b) int a[3]; tanımlamasının yapıldığını kabul edin.

printf (“$d %d %d\n”, a[1], a[2], a[3]);

c) float f[3] = {1.1, 10.01, 100.001, 1000.0001};

d) double d[2][10]; tanımlamasının yapıldığını kabul edin.

d[1, 9] = 2.345;

6.14 Şekil 6.16’daki programda yer alan mod fonksiyonunu, çift sayıda eleman varken doğru

bir şekilde hesaplayacak (bir düğümü çözecek) hale getirin.

6.15 Tek belirteçli bir dizi kullanarak şu problemi çözün: Kullanıcı 10’la 100 arasında 20 sayı

girsin. Her sayı klavyeden alındığında, eğer daha önce aynı sayı girilmediyse bu sayı ekrana

yazdırılsın. Programınız, en kötü ihtimalle 20 farklı sayı girilme ihtimalini de desteklesin ve

mümkün olan en küçük diziyi kullanın.

6.16 3’ e 5’lik, iki boyutlu satislar isimli diziyle aşağıdaki işlemler yapıldığında sırayla hangi

elemanları sıfır olur?

for (satir = 0; satir <= 2; satir++)

for (sutun = 0; sutun <=4; sutun++)

satislar[satir][sutun] = 0;

6.17 Aşağıdaki program ne yapar?

#include <stdio.h>

#define BOYUT 10

int buNedir(int [ ], int);

main( )

{

int toplam, a[BOYUT] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

toplam = buNedir(b, BOYUT)

printf (“Dizi değerlerinin hepsi %d\n”, toplam);

return 0;

}

int buNedir(int b[ ], int boyut)

{

if (boyut == 1)

return b[0];

else

return b[boyut – 1] + buNedir(b, boyut – 1);

}

6.18 Aşağıdaki program ne yapar?

#include <stdio.h>

248

1 2 3 4 5 6

#define BOYUT 10

void birFonksiyon(int [], int);

main()

{

int a[BOYUT] = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37};

printf(“Dizideki değerler: \n”);

bitFonksiyon(a, BOYUT);

printf(“\n”);

return 0;

}

void birFonksiyon(int b[], int boyut)

{

if (boyut > 0) {

birFonksiyon(&b[1], boyut – 1);

printf(“%d “, b[0]);

}

}

6.19 İki adet zar atan bir program yazınız. Her iki zarıda atmak için rand fonksiyonun

kullanın. Daha sonra bu iki sayıyı toplatın. Not: Her zar atışının sonucu 1-6 arasında

olacağı için, iki zarın toplamını da 2-12 arasında olacaktır. Toplamın 7 gelmesi en büyük

olasılığa, 2 ve 12 gelmesi de en küçük olasılıklara sahiptir.Şekil 6.23, mümkün olan 36

olasılığı göstermektedir. Programınız iki zarı 36000 kez atsın ve tek belirteçli bir dizi

kullanarak bu zar toplamlarını diziye sırasıyla yazın. Sonuçlar çizelge şeklinde ekrana

yazdırın. Sonuçların doğru olup olmadığını da kontrol edin. İki zarın toplamının 7

olmasının 6 yolu vardır. Yani atılan zarlarda elde edilen toplam sonuçların 6’da biri 7

olmalıdır.

2 3 4 5 6 7

3 4 5 6 7 8

4 5 6 7 8 9

5 6 7 8 9 10

6 7 8 9 10 11

7 8 9 10 11 12

Şekil 6.23 İki zarı birlikte atmanın 36 muhtemel sonucu

6.20 Barbut oyununu 1000 kez oynatan bir program yazınız ve aşağıdaki soruları

cevaplayınız.

a) İlk zarlar atıldığında, ikinci zarlar atıldığında,.....20. zarlar atıldığında ve 20.

zarlardan sonraki atışlarda kaçar oyun kazanıldı?

1

2

3

4

5

6

249

b) İlk zarlar atıldığında, ikinci zarlar atıldığında,.....20. zarlar atıldığında ve 20.

zarlardan sonraki atışlarda kaçar oyun kaybedildi?

c) Barbutta kazanma şansı nedir? (Barbut neden çok popüler bir gazino

oyunudur?)

d) Barbut oyunun oynanması ortalama ne kadar sürer?

e) Oyunun uzaması kazanma şansını artırır mı?

6.21 (Havayolları rezervasyon Sistemi) Küçük bir havayolları şirketi, rezervasyon kayıtlarını

tutmak için yeni bir bilgisayar almıştır. Patron, size yeni sistemi C’ de programlamanızı

söylemiştir. Havayolları bir uçağa sahiptir ve sizden bu uçağın bütün uçuşlarının

rezervasyon kayıtlarını tutacak bir program istenmiştir.

Programınız aşağıdaki menüyü içermelidir.

“Sigara içilen bölüm” için 1’e basın.

“Sigara içilmeyen bölüm” için 2’ye basın.

Eğer kullanıcı 1’e basarsa, programınız sigara içilen bölümden bir koltuğun

rezervasyonunu yapmalıdır.(1-5. koltuklar.) Eğer kullanıcı 2’ye basarsa sigara içilmeyen

bölümün koltuklarından birinin rezervasyonu yapılmalıdır.(6-10. koltuklar). Daha sonra

programınız, müşterinin hangi bölümden ve kaç numaralı koltuğa rezervasyon yaptırdığını

göstermelidir.

Programınızda tek belirteçli bir dizi kullanınız. Dizi elemanlarının ilk değerlerini, koltukların

boş olduğunu göstermek için 0 yapınız. Bir koltuğun rezervasyonu yapıldığında ilgili dizi

elemanını 1 yapınız.

Programınız kesinlikle aynı koltuğa iki rezervasyon yapmamalı. Eğer sigara içilen bölüm

dolduysa müşterinin sigara içilmeyen bölümden rezervasyon yapmak isteyip istemediğini

sormalı(tam tersi durum içinde aynı işlemi yapmalı) . Eğer cevap evet ise uygun rezervasyon

yapılmalı. Eğer hayırsa “Diğer uçuş 3 saat sonra” yazmalı.

6.22 Kişisel kullanıcılar arasında oldukça popüler olan Logo dili, kaplumbağa grafiklerini

de popüler yapmıştır. Bir C programı kontrolünde, mekanik bir kaplumbağanın bir odada

yürüdüğünü düşünün. Kaplumbağa yukarı ya da aşağı pozisyonda olmak üzere bir kalem

tutmaktadır. Kalem yukarı pozisyonda ise, kaplumbağa yürürken, yürüdüğü yolu çizmektedir.

Kalem aşağı pozisyondayken ise hiçbir şey çizmeden serbestçe yürür. Bu problemle

kaplumbağanın hareketlerini gerçekleyeceksiniz.

50’ye 50’lik yer isminde ve ilk değerleri 0 olan bir dizi kullanın. Emirleri, bu emirleri içeren

bir diziden okuyun. Kalem aşağı veya yukarı pozisyondayken, kaplumbağanın her hareketini

yani o andaki konumunu saklayın. Kaplumbağa her zaman 0’a 0’dan başlasın. Kaplumbağaya

verilecek komutlar aşağıdadır.

Emir Anlamı

1 Kalem yukarı

2 Kalem aşağı

3 Sağa dön

4 Sola dön

250

5,10 ileri 10 adim at

(ya da 10’dan büyük bir sayı)

6 50’ye 50’lik diziyi yazdır.

9 Verinin sonu

Kaplumbağanın ortalarda bir yerde olduğunu kabul ederseniz, aşağıdaki program 12’ye

12’lik bir kare çizer.

2

5,12

3

5,12

3

5,12

3

5,12

1

6

9

Kaplumbağa, kalem aşağı pozisyondayken hareket ettiğinde yer dizisinin uygun

elemanlarını 1 yapın, 6 emri(yazdır) verildiğinde eğer dizinin uygun elemanlarında 1

varsa yıldız karakteri ya da sizin seçeceğiniz bir karakteri yazdırın. Dizinin bakılan

elemanında 0 varsa boşluk yazdırın. Bahsedilen bu işleri yapacak kapasitede olan bir

kaplumbağa grafik programı yazınız. Değişik şekiller çizebilmek için programınızı

geliştirin ve yeni komutlar ekleyin.

6.23 (Atın turu) Satrançta en ilginç bulmacalardan biri Euler’in bulduğu atın turu

bulmacasıdır. Soru şöyledir: At, satranç tahtasındaki bütün kareleri, yani 64 kareyi

birden her kareye sadece ama sadece bir kez uğrayarak gezebilir mi? Şimdi soruyu daha

ayrıntılı bir şekilde inceleyelim.

At, L şeklinde hareket eder ( Bir yöne doğru iki kare ve sonra buna dik olacak

şekilde yana bir kare). Yani, satranç tahtasının ortasında bir yerde

bulunan at, şekil 6.24’de gösterildiği gibi 8 farklı hamle(0’dan 7’ye kadar

numaralandırılmıştır) yapabilir.

a) 8’e 8 lik bir satranç tahtasını kağıda çizin ve öncelikle soruyu elinizle

çözmeye çalışın. İlk gittiğiniz kareye 1, ikincisine 2, üçüncüsüne 3 vb.

diyerek devam edin. Bütün turunuzun 64 hamle olacağını unutmayın. Ne

kadar zamanda çözebileceğinizi düşünüyorsunuz? Çözdüyseniz ne kadar

zaman aldı?

0 1 2 3 4 5 7 7

2 1

251

3 0

K

4 7

5 6

Şekil 6.24

b) Şimdi, atın satranç tahtasında hamleler yapacağı bir program yazalım.

Satranç tahtası 8’e 8’lik tahta isimli bir dizi ile gösterilsin. Bütün karelerin

ilk değerleri 0 olsun. Her sekiz hamleyi de, hamlenin yatay ve dikey

bileşenleri cinsinden göstereceğiz. Örneğin, Şekil 6.24’de gösterilen 0

hamlesi, iki kare sağa doğru yatay ve bir kare yukarı doğru dikeydir. 2

hamlesi ise bir kare sola doğru yatay ve 2 kare yukarı doğru dikeydir. Sola

yatay hamleler ve yukarı dikey hamleler negatif sayılarla gösterilmişlerdir.

Sekiz harekette, yatay ve dikey isminde iki adet tek belirteçli diziyle ifade

edilebilirler.

yatay[0] = 2

yatay[1] = 1

yatay[2] = -1

yatay[3] = -2

yatay[4] = -2

yatay[5] = -1

yatay[6] = 1

yatay[7] = 2

dikey[0] = -1

dikey[1] = -2

dikey[2] = -2

dikey[3] = -1

dikey[4] = 1

dikey[5] = 2

dikey[6] = 2

dikey[7] = 1

bulunulanSutun ve bulunulanSatir ise atın o anda hangi satır ve sütunda bulunduğunu

göstermektedir. Hamle yapmak ise 0 ile 7 arasında değer alan hamle değişkeni ile ayarlanır.

Programınız aşağıdaki ifadeleri kullanacaktır.

bulunulanSatir += dikey[hamle];

bulunulanSutun += yatay[hamle];

252

Programınızda 1’den 64’e kadar sayan bir sayacı sürekli saklayın ve atın her hamle yaptığı

karede sayacı kaydedin. Her potansiyel harekette atın o kareye daha önce gelip gelmediğini

test edin. Tabi ki her hareketi, atın satranç tahtasından dışarı çıkmaması için de test edin.

Şimdi ata hamleler yaptıracak şekilde programı yazı ve çalıştırın. At kaç hamle yaptı ?

c) Atın turu programını tamamlamadan önce muhtemelen kafanızda bazı fikirler belirmiştir.

Şimdi bir strateji geliştireceğiz. Stratejiler başarıyı garanti etmez ama dikkatli bir şekilde

hazırlanırsa başarı şansı oldukça yüksektir. Kenara yakın olan karelerin ortadaki karelerden

daha çok sorun çıkaracağını sizde görmüş olabilirsiniz, bununla beraber köşelerdeki kareler

en büyük sorunu oluşturacaktır.

Düşünecek olursanız, en çok sorun çıkaracak olan karelere ilk başta gitmek ve diğer karelere

daha sonra gitmek başarı şansını artıracaktır.

Bir “erişilebilirlik stratejisi” geliştireceğiz. Bu stratejiyle karelerin erişilebilirliklerini

belirlemeliyiz ve atı (atın L hareketlerini) erişilebilirliği en az olan karelere hamle

yaptırmalıyız. İki boyutlu, erisilebilirlik adında bir dizi tanımlayalım. Bu dizi ilgili kareye

kaç kareden erişilebildiğini göstersin. Boş bir satranç tahtasında ortadaki karelere 8, köşedeki

karelere ise 2 kare erişebilmektedir. Diğer karelere 3,4 ve 6 kare aşağıdaki gibi

erişebilecektedir.

2 3 4 4 4 4 3 2

3 4 6 6 6 6 4 3

4 6 8 8 8 8 6 4

4 6 8 8 8 8 6 4

4 6 8 8 8 8 6 4

4 6 8 8 8 8 6 4

3 4 6 6 6 6 4 3

2 3 4 4 4 4 3 2

Şimdi atın turu programını erişilebilirlik stratejisini uygulayarak tekrar yazınız. At her zaman

hamlelerini erişilebilirlik derecesi en küçük olan karelere yapmalıdır. Eşitlik olduğu durumda

her hangi bir kareye gidebilir. Hamlelere herhangi bir köşeden başlanılabilir. (Not: At, satranç

tahtasında hamleler yaptıkça, programınız, erişilebilirlik numaralarını azaltmalı. Bu yolla, her

hangi bir zamanda, müsait olan her karenin erişilebilirlik numarası ona ulaşabilecek olan

karelerin sayısına eşit olmalıdır. Programınızın bu sürümünü çalıştırın. Bütün tur yapılabildi

mi? Şimdi programı, her köşeden 64 tur yapılacak şekilde değiştirin. Bütün tur yapılabildi

mi?

d) İki karenin eşit erişilebilirliği olma durumunda hangi kare seçildikten sonra daha

az erişilebilirliği olan karenin geleceğini hesaplayarak ona göre atı hareket ettirecek

bir program yazınız.

6.24 (Atın turu : Kaba kuvvet yaklaşımları) Alıştırma 6.24’de atın turu problemine

erişilebilirlik stratejisi ile verimli bir şekilde bir çözüm getirdik.

253

Bilgisayarların gücü arttıkça bir çok problemi basit algoritmalarla çözmek mümkün

olmuştur. Problem çözmedeki yaklaşıma “kaba kuvvet” denir.

a) Rasgele sayılar üreterek atın satranç tahtasında yürümesini (tabi ki L hareketleri

ile) sağlayın. Programınız ata bir tur attırsın ve ekrana bunu yazdırsın. At ne

kadar turuna devam etti?

b) Büyük ihtimalle bir önceki programınız kısa bir tur atacak. Şimdi programa

1000 tur attırın ve bir adet tek belirteçli dizide her turun uzunluğunu saklayın.

1000 tur bittiğinde ekrana her turun ne kadar sürdüğünü bir çizelge ile yazdırın.

En iyi sonuç hangisi?

c) Bir önceki program kayda değer bazı turlar ortaya çıkaracaktır ama hiç biri

bütün bir tur olmayacaktır. Şimdi programınızı bütün bir turu tamamlamadan

programdan çıkmayacak şekilde değiştirin.(Dikkat: Bu tarz bir program güçlü

bir bilgisayarda saatlerce çalışabilir) Her tur denemesinin uzunluğunu yine bir

dizide saklayın. Doğru tur, kaç başarısız turdan sonra bulundu? Ne kadar zaman

sürdü?

d) Kaba kuvvet sürümü ile erişilebilirlik stratejisi sürümlerini karşılaştırın. Hangisi

problemin daha dikkatli çalışılmasını gerektirir? Hangi algoritmanın

uygulanması daha zordur? Erişilebilirlik stratejisi ile kesin sonuca ulaşmak

mümkün müdür? Kaba kuvvet yaklaşımı ile kesin sonuca ulaşmak mümkün

müdür? Kaba kuvvet yöntemini tartışınız.

6.25(Sekiz Vezir) Bir başka satranç problemi de sekiz vezir problemidir. Basitçe, bir satranç

tahtasına sekiz veziri birbirlerini tehdit etmeyecekleri şekilde yerleştirmek mümkün

müdür? Yani, herhangi bir satırda veya sütunda veya çaprazda iki vezir aynı anda

bulunmayacaktır. Alıştırma 6.24’deki düşünme yöntemini kullanarak bu problemi

çözmek için bir strateji geliştirin. Programınızı çalıştırın. (İpucu: Bir kareye bir vezir

yerleştirildiğinde elenen boş karelere bir sayısal değer verilmesi mümkündür. Örneğin

her 4 köşeye de 22 değeri şekil 6.25 de görüldüğü üzere atanabilir.

64 kareye de gerekli değerler atandıktan şu şekilde bir strateji geliştirilebilir: Her vezir,

eleme numarası en az olan kareye yerleştirilmelidir. Neden sezgilerimiz bize bu

stratejiyi uygulamamızı söylüyor?

\* \* \* \* \* \* \* \*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

\* \*

Şekil 6.25 Üst köşeye bir vezir konularak 22 kare elenmiştir.

6.26 (Sekiz vezir: Kaba kuvvet yaklaşımı) Bu problemde Alıştırma 6.26’daki problem için

çeşitli kaba kuvvet yaklaşımları geliştireceğiz.

a) Alıştırma 6.25’de olduğu gibi bir rasgele sayı üretme yolu ile sekiz vezir problemini

çözünüz.

254

b) Daha detaylı bir teknik kullanınız (örneğin vezirlerin satranç tahtasındaki bütün

kombinasyonlarını deneyin.)

c) Neden detaylı kaba kuvvet tekniği atın turu problemini çözmede kullanmak için

desteklenmez?

d) Rasgele kaba kuvvet ve detaylı kaba kuvvet yaklaşımlarını genel olarak

karşılaştırın.

6.27 (kopya eleme) 12.ünitede, yüksek hızlı ikili arama ağacını inceleyeceğiz. İkili arama

ağacının bir özelliği de değerler ağaca eklenirken birbirinin aynısı (kopyası) olan

değerlerin ihmal edilmesidir. 1 ile 20 arasında rasgele değerler üreten bir program

yazınız. Program birbirinin kopyası olmayan tüm değerleri bir dizide saklamalıdır. Bu

görevi yerine getirmek için en küçük diziyi kullanın.

6.28 (Atın turu: Kapalı tur testi) Atın turunda bütün tur, at her kareye sadece ama sadece bir

kez uğrayarak 64 başarılı hamle yaptıktan sonra bitmektedir. Kapalı tur ise atın başladığı

pozisyondan bir pozisyon sonrasında bütün turunu bitirmesiyle gerçekleşir. Alıştırma

6.24’de yazdığınız atın turu programını tur bittikten sonra o turun kapalı bir tur olup

olmadığını test edeceği şekilde değiştiriniz.

6.29 (Eratosthenes eleği) Sadece kendisine ve bire tam olarak bölünebilen sayılara asal sayılar

denir. Eratosthenes eleği, bir asal sayıları bulma metodudur. Şu şekilde çalışır:

1) Bütün elemanlarının ilk değerleri 1(doğru) olan bir dizi oluşturun. Belirteçleri asal

sayı olan dizi elemanlarının değeri 1, diğerleri ise 0 olacaktır.

2) Belirteci 2 olan elemandan başlayarak (belirteci 1 olan eleman mutlaka asal

olmalıdır.) , değeri 1 olan bir eleman her zaman bulunabilir. Dizinin kalanında

döngü ile ilerleyin ve belirteci, değeri 1 olan bir elemanın belirtecinin katı olan dizi

elemanlarına 0 atayın. Belirteci 2 olan eleman için 2’nin ilerisinde 2’nin katı olan

bütün dizi elemanlarının (belirteci 4, 6, 8, 10 vs.) değeri 0 olmalıdır. Belirteci 3 olan

eleman için 3’ün ilerisinde 3’ün katı olan bütün dizi elemanlarının(belirteci 6, 9, 12,

15 vs.) değeri 0 olmalıdır.

Bu işlem bittiğinde belirtec bilgisi asal olan elemanların değerleri 1 olarak kalır. Bu

belirteçler ekrana yazdırılabilir.1000 elemanı olan bir diziyi kullanarak 1 den 999 a

kadar olan sayıların içinde asal olanlarını bulan bir program yazınız. Eleman 0’ı

ihmal ediniz.

6.30 (Kova sıralaması) Kova sıralaması, tek belirteçli ve sıralanacak pozitif tamsayılar içeren

bir dizi ve çift belirteçleri , satır belirteçleri 0’dan 9 a kadar olan tamsayılarla belirtilen,

sütun bilgileri ise 1 den n-1 e kadar olan tamsayılarla belirtilen bir dizi ile başlar. n

dizideki sıralanacak değerlerin sayısıdır. kovaSiralama adında, argüman olarak bir

tamsayı dizisi ve bu dizinin uzunluğunu alan bir fonksiyon yazınız.

Algoritma aşağıdaki gibi olmalıdır :

1) Tek belirteçli dizide döngü kur ve bu dizinin her değerini kova dizinin bir satırına

birler basamağına bakarak yerleştirin. Örneğin 97, satır 72’ye , 3 ise satır 3’e, 100

ise satır 0’a yerleştirilmelidir.

2) Kova dizisinde bir döngü kurarak elemanları orijinal dizilerinde yerleştirin.

Yukarıdaki değerlerin tek belirteçli dizide 100, 3, 97 şekilde sıralanırlar.

255

3) Bu işlemi dizi değerlerinin bütün basamakları(onlar, yüzler, binler vs.) esas alınarak

gerçekleştirin ve en son basamaktan sonra durun.

Dizini döngüden ikinci kez geçtiğinde, 100 satır 0’a, 3 satır 0’a ve 97’de satır 9’a

yerleşmiştir. Tek belirteçli dizide ise sıralama 100, 3, 97 şeklindedir. Üçüncü geçişte ise

100 satır 1’e, 3 satır 0’a 97’de satır 0’a (3’den sonra) yerleşmiştir. En soldaki basamak

esas alınarak işlem yapıldıktan sonrada sıralama biter. Kova sıralaması bütün değerler

çift belirteçli dizinin satır 0’ına yazıldığında sıralamanın bittiğini anlar.

Çift belirteçli dizi, sıralanacak tam sayıları içeren tek belirteçli diziyi içeren dizinin 10

katı boyutundadır. Bu sıralama tekniği kabarcık tekniğinden daha iyi bir performansa

sahiptir ama daha fazla depolama alanına ihtiyaç duyar. Kabarcık metodu sıralanacak

veri türü için ek olarak sadece bir hafıza alanına gerek duyar. Kova sıralaması alan –

zaman değişimine bir örnektir. Daha fazla hafıza kullanır ama performansı daha iyidir.

Kova sıralamasının bu sürümü bütün veriyi her turda tekrar orijinal diziye

kopyalamaktadır. Bir diğer olasılık ise ikinci bir çift belirteçli dizi oluşturarak

sıralanacak değerlerin bu iki dizi arasında taşınmasını sağlamaktır. Sıralama, sıralanacak

bütün değerlerin her hangi bir dizinin satır 0’ına taşındığında bitecektir. Satır 0,

sıralanmış diziyi içerir.

YİNELEME ALIŞTIRMALARI

6.31 (Seçmeli sıralama) Seçmeli sıralama, dizideki en küçük elemanı arar. En küçük eleman

bulunduğunda bu eleman dizinin ilk elemanı ile yer değiştirir. Bu işlem dizinin ikinci

elemanından başlayan alt dizi ile devam eder. Her turda bir eleman yerine yerleştirilir. Bu

sıralama kabarcık metoduyla benzer işlem kapasitesine gerek duyar ; n elemanlı bir dizi

için n-1 tur yapılmalıdır. Her alt dizide en küçük elemanın bulunması için n-1

karşılaştırma yapılmalıdır. Bir eleman içeren dizide işlem gördüğünde sıralama biter.

secmeliSiralama isminde bu algoritmayı uygulayacak bir yineleme fonksiyonu yazınız.

6.32(Palindromlar) Tersten ve düzden okunuşu aynı olan stringlere palindrom denir.

Örneğin “kabak” yada “iki radar iki”. testPalindrome adında, dizide saklanan stringin

bir palindrome olması durumunda 1, aksi halde 0 geri gönderen bir yineleme fonksiyonu

yazınız. Fonksiyon, boşlukları ve noktalama işaretlerini ihmal etmelidir.

6.33 (Lineer Arama) Şekil 6. 18’deki programı dizide bir lineer arama yapacak lineerArama

fonksiyonunu içerecek şekilde değiştiriniz. Fonksiyon argüman olarak bir tamsayı dizisi

ve bu dizinin uzunluğunu almalı. Eğer arama anahtarı bulunursa dizinin belirteci,

bulunamazsa –1 döndürmelidir.

6.34 (İkili Arama) Şekil 6. 19’deki programı dizide bir ikilik arama yapacak ikilikArama

fonksiyonunu içerecek şekilde değiştiriniz. Fonksiyon argüman olarak bir tamsayı dizisi

ve bu dizinin uzunluğunu almalıdır. Eğer arama anahtarı bulunursa dizinin belirteci,

bulunamazsa –1 dödürmelidir.

6.35 (Sekiz vezir) Alıştırma 6.26’da yaptığınız sekiz vezir programını, problemi yineleme

metoduyla çözecek şekilde değiştiriniz.

256

6.36 (Bir dizinin yazılması) diziYazdir isminde, argüman olarak bir dizi ve bu dizinin eleman

sayısını alan ve bir şey döndürmeyen bir yineleme fonksiyonu yazınız. Fonksiyon 0

boyutunda bir dizi aldığında çalışmayı bırakmalı ve geri dönmelidir.

6.37 (Bir stringin ters yazdırılması) tersString isminde, argüman olarak bir karakter dizisi

alan ve bir şey dödürmeyen bir yineleme fonksiyonu yazınız. Fonksiyon stringin

sonundaki NULL karakterini aldığında çalışmayı bırakmalı ver geri dönmelidir.

6.38 (Bir dizinin en küçük elemanının bulunması) enKucukYineleme isminde argüman

olarak bir tam sayı dizisi ve bu dizinin eleman sayısını alan ve dizinin en küçük elemanını

geri gönderen bir yineleme fonksiyonu yazınız. Fonksiyon, bir elemanlık bir dizi

aldığında çalışmayı bırakmalı ve geri dönmelidir.

257

GÖSTERİCİLER

AMAÇLAR

 Göstericileri kullanabilmek

 Referansa göre çağırma ile fonksiyonlara argüman geçirmede göstericileri

kullanabilmek

 Göstericiler, diziler ve stringler arasındaki yakın ilişkiyi anlamak

 Fonksiyonlarda gösterici kullanımını anlamak

 String dizilerini bilidirebilmek ve kullanabilmek.

BAŞLIKLAR

7.1 GİRİŞ

7.2 GÖSTERİCİ DEĞİŞKENLERİ BİLDİRMEK VE GÖSTERİCİ

DEĞİŞKENLERİNE ATAMA YAPMAK

7.3 GÖSTERİCİ OPERATÖRLERİ

7.4 FONKSİYONLARI REFERANSA GÖRE ÇAĞIRMAK

7.5 const BELİRTECİNİ GÖSTERİCİLERLE KULLANMAK

7.6 REFERANSA GÖRE ÇAĞIRMA KULLANAN KABARCIK SIRALAMA

7.7 GÖSTERİCİ İFADELERİ VE GÖSTERİCİ ARİTMETİĞİ

7.8 GÖSTERİCİLER VE DİZİLER ARASINDAKİ İLİŞKİ

7.9 GÖSTERİCİ DİZİLERİ

7.10 ÖRNEK:KART KARMA VE DAĞITMA

7.11 FONKSİYONLARI GÖSTEREN GÖSTERİCİLER

7.1 GİRİŞ

Bu ünitede C programlama dilinin en güçlü özelliklerinden biri olan göstericileri anlatacağız.

Göstericiler, C’nin yönetilmesi en zor yetenekleri arasındadır. Göstericiler, programların

referansa göre çağırma yapmasını ve bağlı listeler, sıralar, yığınlar ve ağaçlar gibi büyüyüp

küçülebilen dinamik veri yapılarının oluşturulmasıyla, yönetilmesini sağlatır. Bu ünite, temel

gösterici kavramlarını açıklamaktadır. 10. ünite göstericilerin yapılarla kullanımını

incelemektedir. 12. ünite dinamik hafıza yönetme tekniklerini tanıtmakta ve dinamik veri

yapıları oluşturma ve yönetme ile ilgili örnekler vermektedir.

7.2 GÖSTERİCİ DEĞİŞKENLERİ BİLDİRMEK VE GÖSTERİCİ

DEĞİŞKENLERİNE ATAMA YAPMAK

Göstericiler, değer olarak hafıza adreslerini içeren değişkenlerdir. Normalde bir değişken

doğrudan kesin bir değeri içerir. Göstericiler ise kesin bir değeri tutan değişkenlerin adresini

içerir. Bu bağlamda, bir değişken ismi bir değeri doğrudan belirlerken, göstericiler değeri

258

dolaylı yoldan belirtir (Şekil 7.1). Bir değeri gösterici ile belirtmek dolaylama yapmak

( indirection ) olarak adlandırılır.

Göstericiler de diğer değişkenler gibi kullanılmadan önce bildirilmelidir.

int \*sayiciPtr, sayici;

biçiminde bir bildirim, sayiciPtr değişkenini int \* (bir tamsayıyı gösteren gösterici) tipinde

bildirir. Bu bildirim “sayiciPtr bir int göstericisidir” ya da “sayiciPtr , tamsayı tipinde bir

nesneyi gösterir” biçiminde okunur. Ayrıca, sayici değişkeni de tamsayı olarak bildirilmiştir.

Bildirim içindeki \* yalnızca sayiciPtr’ye uygulanır. Bildirimlerde \* bu biçimde

kullanıldığında, bildirilen değişkenin gösterici olduğunu belirtir. Göstericiler herhangi bir veri

tipindeki nesneleri göstermek için bildirilebilirler.

Şekil 7.1 Bir değişkeni doğrudan ve dolaylı olarak belirlemek

Genel Programlama Hataları 7.1

İçerik ( indirection ) operatörü (\*), bildirimde bütün değişken isimlerine

dağıtılmaz.Her gösterici isimden önce \* kullanılarak bildirilmelidir.

İyi Programlama Alıştırmaları 7.1

Gösterici değişkenlerinin isimlerinde ptr harflerini kullanarak, bu değişkenlerin gösterici olduklarını

ve dikkatlice ele alınması gerektiklerini daha belirgin hale getirmek.

Göstericilere, bildirimlerde ya da atama ifadelerinde değer atanmalıdır. Bir göstericiye 0,

NULL ya da bir adres atanabilir. NULL değerine sahip bir gösterici hiçbir şeyi göstermez.

NULL, <stdio.h> ve diğer bir çok öncü dosyada tanımlanmış bir sembolik sabittir. Bir

göstericiye 0 atamak, NULL atamakla eşdeğerdir ancak NULL tercih edilir. 0 atandığında

öncelikle uygun tipte bir göstericiye çevrilir. 0 değeri bir göstericiye doğrudan atanabilen tek

tamsayı değeridir. Göstericilere bir değişkeninin adresini atamak Kısım 7.3’te anlatılacaktır.

İyi Programlama Alıştırmaları 7.2

Beklenmeyen sonuçlarla karşılaşmamak için göstericilere ilk değer atamak.

7.3 GÖSTERİCİ OPERATÖRLERİ

& ya da adres operatörü, operandının adresini döndüren bir tekli operatördür. Örneğin ,

int y = 5;

int \*yPtr;

sayiciPtr sayici

sayici

7

7

sayici, değeri 7 olan değişkeni

doğrudan gösteriyor

sayiciPtr ise değeri 7 olan değişkeni

dolaylı bir biçimde belirtiyor.

259

bildirimlerini ele aldığımızda

yPtr =&y;

ifadesi, y değişkeninin adresini yPtr gösterici değişkenine atar. Buna, yPtr değişkeni y’yi

göstermektedir denir. Şekil 7.2, az önceki atama çalıştırıldıktan sonra hafızanın şematik

tasvirini göstermektedir.

y

Şekil 7.2 Hafızadaki bir tamsayı değişkenini gösteren bir göstericinin grafik ile tasviri.

Şekil 7.3, tamsayı değişkeni y’nin hafızada 600000 konumunda ve gösterici değişkeni

yPtr‘nin 500000 konumunda tutulduğu düşünülerek, göstericinin hafızadaki tasvirini

göstermektedir. Adres operatörünün operandı bir değer olmalıdır; adres operatörü sabitlere,

deyimlere ya da register depolama sınıfıyla bildirilmiş değişkenlere uygulanamaz.

\* operatörü ya da genellikle söylendiği biçimde içerik operatörü, operandının(yani

göstericinin) gösterdiği nesnenin değerini döndürür. Örneğin,

printf ( “%d”,\*yPtr );

ifadesi, y değişkeninin değerini yani 5’i yazdırır. Bu şekilde bir \* kullanımına, göstericinin

gösterdiği nesneye erişmek denir.

Genel Programlama Hataları 7.2

Uygun bir biçimde ilk değerlere atanmamış ya da belli bir hafıza konumunu

göstermek için atanmamış göstericilerin gösterdiği nesneye ulaşmaya

çalışmak. Bu, çalışma zamanlı ölümcül hatalara ya da önemli bir veriyi

yanlışlıkla değiştirerek programın yanlış sonuçlar üretmesine sebep olur.

Şekil 7.4, gösterici operatörlerinin kullanılışını göstermektedir. %p dönüşüm belirteci, printf

ile kullanıldığında hafıza konumunu onaltılık sayı sisteminde yazdırır. (bakınız Ekler E)

Programın çıktısında a ve aPtr değerlerinin eş olduğuna dikkat ediniz. Bu, aPtr değişkenine

gerçektende a değişkeninin adresinin atandığını kanıtlar. & ve \* operatörleri birbirlerinin

eşleniğidir. Bu yüzdende, aPtr’ye herhangi bir sırada ard arda uygulandıklarında aynı

sonuçlar yazdırılır. Şekil 7.5, şu ana kadar gösterilen operatörlerin öncelik sıralarını

listelemektedir.

5

yPtr

yPtr

260

Şekil 7.3 y ve yptr’nin hafızadaki gösterimleri

1 /\* Şekil 7.4: fig07\_04.c

2 & ve \* operatörlerini kullanmak \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int a; /\* a bir tamsayıdır \*/

8 int \*aPtr; /\* aPtr bir tamsayıyı gösteren bir göstericidir \*/

9

10 a = 7;

11 aPtr = &a; /\* aPtr ‘ye a’nın adresi atandı \*/

12

13 printf( "a’nın adresi %p"

14 "\naPtr değişkeninin değeri %p", &a, aPtr );

15

16 printf( "\n\na’nın değeri %d"

17 "\n\*aPtr nin değeri %d", a, \*aPtr );

18

19 printf( "\n\n\* ve & birbirlerinin"

20 "eşleniğidir.\n&\*aPtr = %p"

21 "\n\*&aPtr = %p\n", &\*aPtr, \*&aPtr );

22

23 return 0;

24 }

a’nın adresi 0012FF88

aPtr değişkeninin değeri 0012FF88

a’nın değeri 7

\*aPtr nin değeri 7

\* ve & birbirlerinin eşleniğidir.

\*&aPtr = 0012FF88

&\*aPtr = 0012FF88

Şekil 7.4 & ve \* operatörleri

OPERATÖR İŞLEYİŞ TİP

( ) [ ] soldan sağa en yüksek

+ - ++ -- ! \* & (tip) sağdan sola tekli

\* / % soldan sağa multiplicative

+ - soldan sağa additive

< <= > >= soldan sağa karşılaştırma

= = != soldan sağa eşitlik

500000 600000

y

600000 5

261

&& soldan sağa mantıksal ve

|| soldan sağa mantıksal veya

?: sağdan sola koşullu

= + = -= \*= /= %= sağdan sola atama

, soldan sağa virgül

Şekil 7.5 Operatör öncelikleri

7.4 FONKSİYONLARI REFERANSA GÖRE ÇAĞIRMAK

Bir fonksiyona argüman geçirmenin iki yolu vardır : değere göre ve referansa göre çağırma. C’de tüm

fonksiyon çağrıları değere göre çağırma ile yapılır. 5. ünitede gördüğümüz gibi return, çağrılan

fonksiyondan çağırıcıya bir değer döndürmek için (ya da bir değer döndürmeden çağrılan

fonksiyondan kontrolü geri almak için) kullanılabilir. Bir çok fonksiyon, çağırıcıdaki bir ya da birden

çok değişkeni değiştirebilme yeteneğine veya değere göre çağırmanın yükünden (bu nesnenin bir

kopyasının oluşturulmasını gerektirir) kurtulmak için büyük bir veri nesnesini gösteren göstericiyi

geçirmeye ihtiyaç duyarlar. Bu amaçlar için C, referansa göre çağırma yeteneklerini sunar.

C’de programcılar göstericileri ve içerik operatörünü referansa göre çağrıları

gerçekleştirmek için kullanırlar. Argümanları değiştirilecek bir fonksiyon çağrılırken, argümanların

adresleri geçirilir. Bu, genellikle değeri değiştirilecek değişkene adres operatörü (&) uygulanarak

gerçekleştirilir. 6. ünitede gördüğümüz gibi diziler & operatörü kullanılarak geçirilmezler çünkü C,

dizinin hafızadaki konumunun başlangıç adresini otomatik olarak geçirir. (dizinin ismi, &diziIsmi[0]

ile eşdeğerdedir.) Değişkenin adresi fonksiyona geçirildiğinde, içerik operatörü (\*) fonksiyonla

birlikte çağırıcının hafızasındaki o konumunda bulunan değeri değiştirmek için kullanılabilir.

Şekil 7.6 ve 7.7, bir tamsayının küpünü hesaplayan bir fonksiyonun iki çeşidini (degereGoreKup ve

referansaGoreKup) göstermektedir. Şekil 7.6, değere göre çağırma kullanarak sayi değişkenini

degereGoreKup fonksiyonuna geçirmektedir (satır 12) degereGoreKup fonksiyonu, argümanının

küpünü almakta ve yeni değeri return ifadesi ile main fonksiyonuna geçirmektedir. Yeni değer, main

içinde sayi değişkenine atanmaktadır.

Şekil 7.7, referansa göre çağırma kullanarak sayi değişkeninin adresini referansaGoreKup

fonksiyonuna geçirmektedir (satır 14). referansaGoreKup fonksiyonu, argüman olarak int gösteren

bir gösterici olan nPtr’yi almaktadır. Fonksiyon göstericinin gösterdiği nesneye erişir ve nPtr’nin

gösterdiği değerin küpünü alır. Bu, main içindeki sayi’nin değerini değiştirir. Şekil 7.8 ve 7.9’da Şekil

7.6 ve Şekil 7.7’deki programlar grafiklerle incelenmiştir.

Genel Programlama Hataları 7.3

Göstericinin gösterdiği değeri elde etmek gerektiğinde göstericinin gösterdiği nesneye erişmemek.

1 /\* Şekil 7.6: fig07\_06.c

2 Değere göre çağırma kullanarak bir değerin küpünü almak \*/

3 #include <stdio.h>

4

5 int degereGoreKup ( int ); /\* prototip \*/

6

7 int main( )

8 {

9 int sayi = 5;

10

11 printf( "Sayının esas değeri %d", sayi);

12 sayi = degereGoreKup ( sayi );

13 printf( "\nSayının yeni değeri %d\n", sayi);

14

262

15 return 0;

16 }

17

18 int degereGoreKup( int n )

19 {

20 return n \* n \* n; /\* yerel değişken n’in küpünü al\*/

21 }

Sayının esas değeri 5

Sayının yeni değeri 125

Şekil 7.6 Değere göre çağırma kullanarak bir değerin küpünü almak

1 /\* Şekil 7.7: fig07\_07.c

2 Referansa göre çağırma kullanarak bir değerin küpünü almak \*/

3

4 #include <stdio.h>

5

6 void referansaGoreKup ( int \* ); /\* prototip \*/

7

8 int main( )

9 {

10 int sayi = 5;

11

12 printf( "Sayının esas değeri %d", sayi);

13 referansaGoreKup( &sayi);

14 printf( "\nSayının yeni değeri %d\n", sayi);

15

16 return 0;

17 }

18

19 void referansaGoreKup( int \*nPtr )

20 {

21 \*nPtr = \*nPtr \* \*nPtr \* \*nPtr; /\* main’deki sayi’nın küpü alındı \*/

22 }

Sayının esas değeri 5

Sayının yeni değeri 125

Şekil 7.7 Referansa göre çağırma kullanarak bir değerin küpünü almak

main, degerGoreKup fonksiyonunu çağırmadan önce

main( ) sayi

{

int sayi = 5;

sayi = degereGoreKup(sayi);

}

5

int degereGoreKup(int n)

{

return n \* n \* n;

} n

263

degerGoreKup fonksiyonunu çağırıldığında

degerGoreKup fonksiyonunu n parametresinin küpünü aldıktan sonra

degerGoreKup fonksiyonundan main fonksiyonuna dönüldükten sonra

main sayi değişkenine atanma işlemini bitirdiği zaman

Şekil 7.8 Tipik bir değere göre çağırma işleminin analizi

referansaGoreKup fonksiyonunu referansa göre çağırılmadan önce

tanımlanmamış

main( ) sayi

{

int sayi = 5;

sayi = degereGoreKup(sayi);

}

5

int degereGoreKup(int n)

{

return n \* n \* n;

} n

5

main() sayi

{

int sayi = 5;

sayi = degereGoreKup(sayi);

}

5

int degereGoreKup(int n)

{

return n \* n \* n;

} 125 n

5

main() sayi

{

int sayi = 5;

sayi = degereGoreKup(sayi);

}

5

main() sayi

{

int sayi = 5;

sayi = degereGoreKup(sayi);

}

125

int degereGoreKup(int n)

{

return n \* n \* n;

} n

int degereGoreKup(int n)

{

return n \* n \* n;

} n

tanımlanmamış

main( ) sayi

{

int sayi = 5;

sayi = referensaGoreKup(&sayi);

}

5

int referansaGoreKup(int n)

{

\*nPtr = nPtr \* \*nPtr \* \*nPtr;

} nPtr

tanımlanmamış

tanımlanmamış

264

referansaGoreKup fonksiyonu referansa göre çağırıldıktan sonra ve \*nPtr’nin küpü alınmadan önce;

Şekil 7.9 Bir gösterici argümanıyla tipik bir referansa göre çağırma işleminin analizi.

Argüman olarak bir adres alan fonksiyonlar, adresi alabilmek için bir gösterici parametresi

bildirmelidir. Örneğin, referansaGoreKup fonksiyonunun başlığı (satır 19)

void referansaGoreKup(int \*nPtr)

biçimindedir. Başlık, referansaGoreKup fonksiyonunun argüman olarak bir tamsayı değişkeninin

adresini kullandığını, bu adresi yerel olarak nPtr içinde tuttuğunu ve bir değer geri döndürmediğini

belirtmektedir.

referansaGoreKup fonksiyonunun prototipi parantez içinde int \* içermektedir. Diğer değişken

tiplerinde olduğu gibi fonksiyon prototiplerinde göstericilerin isimlerinin belirtilmesine gerek yoktur.

Dokümantasyon amaçları için dahil edilen isimler, C derleyicisi tarafından ihmal edilir.

Tek belirteçli bir diziyi argüman olarak bekleyen bir fonksiyonun fonksiyon başlığında ve

prototipinde, referansaGoreKup fonksiyonunun parametre listesindeki gösterici yazımı kullanılabilir.

Derleyici bir gösterici alan fonksiyonla, tek belirteçli bir dizi alan fonksiyon arasında ayrım yapmaz.

Bu, fonksiyonun referansa göre çağırma yaptığında ne zaman bir dizi ya da tek bir değeri alacağını

bilmek zorunda olduğu anlamına gelir. Derleyici fonksiyon tanımında, int b[ ] biçiminde tek belirteçli

bir dizi gördüğünde bu parametreyi int \*b biçimine dönüştürür. İki biçimde birbirleriyle

değiştirilebilir.

İyi Programlama Alıştırmaları 7.3

Bir fonksiyona argüman geçirirken eğer çağırıcı özel olarak çağırdığı fonksiyonunun kendi

ortamındaki bir argüman değişkeninin değerini değiştirmesine gerek duymuyorsa değere göre

çağırmayı kullanınız. Bu, en az yetki prensibinin başka bir örneğidir.

7.5 const BELİRTECİNİ GÖSTERİCİLERLE KULLANMAK

main() sayi

{

int sayi = 5;

sayi = referensaGoreKup(&sayi);

}

5

int referansaGoreKup(int n)

{

\*nPtr = nPtr \* \*nPtr \* \*nPtr;

} nPtr

main() sayi

{

int sayi = 5;

sayi = referensaGoreKup(&sayi);

}

125

int referansaGoreKup(int n)

{

\*nPtr = nPtr \* \*nPtr \* \*nPtr;

} nPtr

265

const belirteci, programcının derleyiciye belirli bir değişkenin değerinin değiştirilmemesi

gerektiğinin bildirmesini sağlar. const belirteci C’nin ilk sürümlerinde yer almamaktaydı;

C’ye daha sonradan ANSI tarafından eklenmiştir.

Yazılım Mühendisliği Gözlemleri 7.1

const belirteci en az yetki prensibini zorla uygulatmak için kullanılabilir.En az yetki prensibini uygun

yazılımlar tasarlamak için kullanmak, hata ayıklama zamanını ve istenmeyen yan etkileri çok büyük

oranda azaltır ve programı daha basit geliştirilebilir bir hale getirir.

Taşınırlık İpuçları 7.1

const ANSI C’de açıkça belirtilmesine rağmen bazı C sistemleri bunu uygulayamaz.

Yıllar boyunca, C’nin eski sürümlerinde const bulunmadığı için, bu belirteci kullanmayan

oldukça fazla sayıda kod yazılmıştır. Bu sebepten, eski C kodlarında, yazılım mühendisliği

açısından oldukça büyük iyileştirmeler yapma şansı bulunmaktadır. Ayrıca şu anda ANSI C

kullanan bazı programcılar, programlamaya C’nin eski sürümlerinde başladıklarından,

programlarında const kullanmazlar. Bu programcılar, yazılım mühendisliği açısından bir çok

fırsatı kaçırmaktadırlar.

const belirtecini fonksiyon parametreleri ile kullanmada (ya da kullanmamada) 6 ihtimal

bulunmaktadır. Bunlardan ikisi, değere göre parametre geçirme ve dördü, referansa göre

parametre geçirme anında ortaya çıkar. Bu 6 ihtimalden birini nasıl seçeceksiniz? En az yetki

prensibi rehberiniz olacaktır. Fonksiyonlara her zaman, parametreleri ile yalnızca görevlerini

yerine getirmelerine yardımcı olacak kadar veriye ulaşma hakkı verin.

5.ünitede C’de bütün çağrıların değere göre yapıldığını anlattık. Fonksiyon çağrısındaki

argümanın bir kopyası oluşturuluyor ve fonksiyona geçiriliyordu. Eğer kopyanın değeri

fonksiyon içinde değiştirilirse, orijinal değer çağırıcı içinde değişmemiş bir halde

bulunabiliyordu. Çoğu durumda fonksiyonun görevini yerine getirebilmesi için fonksiyona

geçirilen değerin değiştirilmesi gerekir. Ancak bazı durumlarda, çağrılan fonksiyon orijinal

değerin kopyasını değiştirebilecek bile olsa değerin çağrılan fonksiyon içinde

değiştirilmemesi gerekebilir.

Tek belirteçli bir diziyi ve bu dizinin boyutunu argüman olarak kullanan ve diziyi yazdıran bir

fonksiyonu ele alalım. Böyle bir fonksiyon dizi boyunca ilerleyerek her dizi elemanını özel

olarak yazdırmalıdır. Dizinin boyutunun, programın yazdırma tamamlandığında

sonlandırılabilmesi için bilinmesi gerekmektedir. Dizi boyunca süren döngü, en yüksek

belirtece sahip elemandan sonra sona ermelidir. Dizinin boyutu fonksiyon gövdesi içinde

değişmemektedir.

Yazılım Mühendisliği Gözlemleri 7.2

Eğer bir değer geçirildiği fonksiyonun gövdesi içinde değişmiyorsa(ya da değişmemeliyse) bu değer

const olarak bildirilerek hata ile değiştirilmemesi garanti altına alınmalıdır.

Eğer const olarak bildirilmiş bir değer değiştirilmeye çalışılırsa, derleyici bunu yakalar ve

bilgisayara bağlı olarak hata ya da uyarı mesajı yayınlar.

Yazılım Mühendisliği Gözlemleri 7.3

Değere göre çağırma kullanıldığında çağrılan fonksiyon içinde yalnızca tek

bir değer değiştirilebilir. Bu değer, fonksiyonun geri dönüş değerinden

266

atanmalıdır.Çağrılan fonksiyon içinde birden çok değişkenin değerini

değiştirmek için mutlaka referansa göre çağırma kullanılmalıdır.

İyi Programlama Alıştırmaları 7.4

Bir fonksiyonu kullanmadan önce fonksiyonun kendisine geçirilen değerleri

değiştirip değiştiremeyeceğini anlamak için fonksiyon prototipini kontrol edin.

Genel Programlama Hataları 7.4

Bir fonksiyonun referansa göre çağrılar yapıldığında, argüman olarak

gösterici beklediğinden ve değere göre çağrılar yapıldığında argümanların

geçirildiğinden habersiz olmak. Bazı derleyiciler değer alırken, değerlere

gösterici gibi davranırlar ve değerlere göstericilerde olduğu gibi erişirler.

Çalışma esnasında genellikle hafıza erişimi sorunları ve segment hataları

oluşur. Diğer derleyiciler argüman ve parametreler arasındaki tip

farklılıklarını yakalar ve hata mesajları üretirler.

Bir göstericiyi fonksiyona geçirmenin dört yolu vardır. Bunlar ; sabit olmayan bir göstericiyi

sabit olmayan bir veriyle, sabit bir göstericiyi sabit olmayan bir veriyle, sabit olmayan bir

göstericiyi sabit bir veriyle ve sabit bir göstericiyi sabit bir veriyle kullanmaktır. Bu dört farklı

yöntem, en az yetki prensibini farklı seviyelerde uygular.

En üst seviyede veri erişimi, sabit olmayan bir veri ile sabit olmayan gösterici kullanma

durumunda sağlanır. Bu durumda veri, göstericinin gösterdiği nesneye erişilerek

değiştirilebilir ve gösterici başka bir veriyi göstermek üzere değiştirilebilir. Sabit olmayan bir

veriyi gösteren sabit olmayan gösterici bildirimleri const içermez. Bu tarzda bir gösterici,

string içindeki her karakteri işlemek (ve muhtemelen değiştirmek) için argüman olarak string

alan bir fonksiyonla kullanılabilir. Şekil 7.10’daki buyukHarfeCevir fonksiyonu, argümanını

sabit olmayan bir veriyi gösteren sabit olmayan gösterici , sPtr ( char \*sPtr ) , biçiminde

bildirmektedir. Fonksiyon string dizisinde (sPtr ile gösterilmektedir), gösterici aritmetiği

kullanarak bir anda bir karakter işlemektedir. C standart kütüphane fonksiyonu olan islower

(25.satırda çağrılmıştır), sPtr ile gösterilen adresin içeriğini test etmektedir.Eğer karakter a-z

aralığında ise islower fonksiyonu doğru bir değer döndürür ve C standart kütüphane

fonksiyonu olan toupper (satır 26), karakteri büyük harfe çevirmek için çağrılır.Aksi

takdirde, islower yanlış bir değer döndürür ve string içindeki bir sonraki karakter ele alınır.

1 /\* Şekil 7.10: fig07\_10.c

2 Sabit olmayan veriyi gösteren sabit olmayan gösterici kullanarak

3 küçük harfleri büyük harfe çevirmek.\*/

4

5 #include <stdio.h>

6 #include <ctype.h>

7

8 void buyukHarfeCevir( char \* );

9

10 int main( )

11 {

12 char string[ ] = "karakterler ve $32.98";

267

13

14 printf( "Çevrilmeden önceki string: %s", string );

15 buyukHarfeCevir( string );

16 printf( "\nÇevrildikten sonraki string: %s\n", string );

17

18 return 0;

19 }

20

21 void buyukHarfeCevir ( char \*sPtr )

22 {

23 while ( \*sPtr != '\0' ) {

24

25 if ( islower( \*sPtr ) )

26 \*sPtr = toupper( \*sPtr ); /\* büyük harfe çevir \*/

27

28 ++sPtr; /\* sPtr ile diğer karaktere geç \*/

29 }

30 }

Çevrilmeden önceki string: karakterler ve $32.98

Çevrildikten sonraki string: KARAKTERLER VE $32.98

Şekil 7.10 Bir stringi, sabit olmayan bir değer için sabit olmayan bir gösterici kullanarak

büyük harfe çevirmek.

Sabit bir veriyi gösteren sabit olmayan gösterici, her hangi bir anda değiştirilerek uygun tipte

bir veriyi gösterebilir ancak gösterdiği veri değiştirilemez. Bu tarzda bir gösterici, argüman

olarak aldığı dizinin elemanlarını değiştirmeden kullanan bir fonksiyonla birlikte

kullanılabilir. Örneğin, şekil 7.11’deki karakterleriyazdir fonksiyonu sPtr parametresini

const char \* tipinde bildirmektedir (satır 23). Bu bildirim “sPtr ,bir karakter sabitini gösteren

göstericidir” biçiminde okunur. Fonksiyonun gövdesi for yapısı kullanarak, null karakterle

karşılaşıncaya kadar string içindeki her karakteri yazdırmaktadır. Her karakter yazdırıldıktan

sonra sPtr göstericisi, string içindeki diğer karakteri göstermek için arttırılmaktadır.

1 /\* Şekil 7.11: fig07\_11.c

2 Sabit bir veriyi gösteren sabit olmayan bir gösterici kullanarak

3 bir stringin karakterlerini sırayla yazdırmak.\*/

4

5 #include <stdio.h>

6

7 void karakterleriYazdir ( const char \* );

8

9 int main( )

10 {

11 char string [ ] = "string karakterlerini yaz";

12

13 printf( "String:\n" );

14 karakterleriYazdir( string );

268

15 printf( "\n" );

16

17 return 0;

18 }

19

20 /\* karakterleriYazdir fonksiyonunda sPtr bir karakter sabitini

21 gösteren göstericidir. Karakterler sPtr kullanarak değiştirilemezler

22 (yani sptr, sadece okunabilir bir göstericidir). \*/

23 void karakterleriYazdir ( const char \*sPtr )

24 {

25 for ( ; \*sPtr != '\0'; sPtr++ ) /\* ilk değer ataması yok \*/

26 printf( "%c", \*sPtr );

27 }

String:

string karakterlerini yaz

Şekil 7.11 Sabit bir veri için sabit olmayan bir gösterici kullanarak bir stringin karakterlerini

sırayla yazdırmak.

Şekil 7.12, sabit bir veri için sabit olmayan gösterici (xPtr) alan bir fonksiyonun derlenmesi

esnasında Borland derleyicisinin verdiği hata mesajlarını göstermektedir. Bu fonksiyon, xPtr

ile gösterilen veriyi değiştirmeye çalışmaktadır (21.satır). Bu da bir derleme hatasına yol

açmaktadır.

1 /\* Şekil 7.12: fig07\_12.c

2 sabit bir veri için sabit olmayan gösterici

3 kullanarak değeri değiştirmeye çalışmak \*/

4

5 #include <stdio.h>

6

7 void f ( const int \* );

8

9 int main( )

10 {

11 int y;

12

13 f( &y ); /\* f doğru olmayan bir değişiklik yapmaya çalıştı \*/

14

15 return 0;

16 }

17

18 /\* f içinde xPtr,bir tamsayı sabitini gösteren göstericidir \*/

19 void f ( const int \*xPtr )

20 {

21 \*xPtr = 100; /\* const nesnesi değiştirilemez \*/

22 }

269

FIG07\_12.c:

Error E2024 FIG07\_12.c 21: Cannot modify a const object in function f

Warning W8057 FIG07\_12.c 22: Parameter ‘xPtr’ is never used in function f

\*\*\* 1 errors in Compile \*\*\*

Şekil 7.12 Sabit bir veri için sabit olmayan gösterici kullanarak değeri değiştirmeye çalışmak

Bildiğimiz gibi, diziler birbirleriyle ilişkili ve aynı tipteki verileri aynı isim altında tuttuğumuz

toplam veri tipleridir. 10. ünitede struct adı verilen (bazı programlama dillerinde kayıt da

denir) başka bir toplam veri tipini göreceğiz. Bir yapı (structure) birbirleriyle ilişkili ve farklı

tiplerdeki verileri (örneğin bir şirketteki elemanların bilgilerini) tutabilir. Argüman olarak bir

dizi ile çağrılan fonksiyonlara, diziler referansa göre çağırma ile otomatik bir biçimde

geçirilir. Ancak yapılar her zaman değere göre( tüm yapının bir kopyası oluşturularak)

geçirilir. Bu, yapı içindeki her verinin kopyalanması ve fonksiyon çağrısının tutulduğu yığın

içinde depolanmasını gerektirdiğinden çalışma zamanını arttırır. Bir yapıdaki veri fonksiyona

geçirileceğinde, sabit veriyi gösteren göstericileri kullanarak referansa göre çağırmanın

performansını ve değere göre çağırmanın güvenirliğini kullanabiliriz. Bir yapıyı gösteren

gösterici geçirildiğinde yalnızca yapının depolandığı adresin kopyası oluşturulur. 4 byte adres

kullanan bir makinede yapının muhtemel olarak yüzlerce ya da binlerce byte’ını kopyalamak

yerine yalnızca 4 byte kopyalanır.

Performans İpuçları 7.1

Yapılar gibi büyük verileri, sabit bir veriyi gösteren göstericilerle geçirerek referansa göre

çağırmanın performansını ve değere göre çağırmanın güvenliğini kullanın.

Göstericileri bu şekilde kullanmak zaman/mekan değişiminin (time/space trade off) bir

örneğidir. Eğer hafızada fazla yer yoksa ve çalışma verimi en önemli mesele ise göstericiler

kullanılmalıdır. Eğer hafıza sıkıntısı yaşanmıyorsa ve verimlilik temel mesele değilse veriler,

değere göre çağırma ile geçirilerek en az yetki prensibinin uygulanması sağlanmalıdır. Bazı

sistemlerin const belirtecini uygulayamadıkları düşünülürse, değerin değiştirilmesini

engellemek için değere göre çağırmanın en iyi yol olduğu görülecektir.

Sabit olmayan bir değişkeni gösteren sabit bir gösterici, her zaman aynı hafıza konumunu

gösterir ve bu konumdaki verinin değeri değiştirilebilir. Bu, dizi isimleri için aksi bir durum

belirtilmedikçe kullanılır. Dizi ismi, dizinin başlangıcını gösteren sabit bir göstericidir.

Dizideki bütün verilere dizi belirteçleri değiştirilerek ulaşılabilir ve veriler değiştirilebilir.

Sabit olmayan bir veriyi gösteren sabit göstericiler, dizi elemanlarına yalnızca dizi belirteçleri

sayesinde ulaşan dizileri argüman olarak alan fonksiyonlar tarafından kullanılır. const olarak

bildirilen göstericilere atamalar bildirim esnasında yapılmalıdır (eğer gösterici bir fonksiyon

parametresi ise fonksiyona geçirilen parametreye atanır ). Şekil 7.13, sabit bir göstericiyi

değiştirmeye çalışmaktadır. 11.satırda bildirilen ptr değişkeni int \* const tipindedir. Bu

bildirim “ptr, bir tamsayıyı gösteren sabit bir göstericidir” biçiminde okunur. Gösterici,

tamsayı değişkeni olan x’in adresine atanmıştır. Program y’nin adresini ptr’ye atamaya

çalışmaktadır ancak bir hata mesajı üretilmektedir.

1 /\* Şekil 7.13: fig07\_13.c

2 Sabit olmayan bir veriyi gösteren sabit

3 bir göstericiyi değiştirmeye çalışmak.\*/

4

270

5 #include <stdio.h>

6

7 int main( )

8 {

9 int x, y;

10

11 int \* const ptr = &x; /\* ptr, tamsayı gösteren sabit bir göstericidir

12 Bir tamsayı ptr ile değiştirilebilir

13 ama ptr hafızada

14 her zaman aynı yeri gösterir \*/

15 \*ptr = 7;

16 ptr = &y;

17

18 return 0;

19 }

FIG07\_13.c:

Error E2024 FIG07\_13.c 16: Cannot modifiy a const object in

function main

\*\*\* 1 errors in Compile \*\*\*

Şekil 7.13 Sabit olmayan bir veriyi gösteren sabit bir göstericiyi değiştirmeye çalışmak.

En az yetki prensibi, sabit bir veriyi gösteren sabit bir gösterici ile sağlanır. Bu türde bir

gösterici her zaman aynı hafıza konumunu gösterir ve bu hafıza konumundaki veri

değiştirilemez. Bu, diziyi dizi belirteçleri sayesinde inceleyen ancak dizideki değerleri

değiştirmeyen fonksiyonlara, dizileri geçirmek için kullanılır. Şekil 7.14, ptr değişkenini

const int \*const tipinde bildirmiştir (satır 10). Bu bildirim “ptr ,bir tamsayı sabitini gösteren

sabit bir göstericidir” biçiminde okunur. Şekil, ptr’nin gösterdiği veri değiştirilmeye

çalışıldığında ve gösterici değişkeninin tuttuğu adres değeri değiştirilmeye çalışıldığında

üretilen hataları göstermektedir.

1 /\* Şekil 7.14: fig07\_14.c

2 Sabit bir veriyi gösteren sabit bir göstericiyi

3 değiştirmeye çalışmak. \*/

4 #include <stdio.h>

5

6 int main( )

7 {

8 int x = 5, y;

9

10 const int \*const ptr = &x; /\* ptr sabit bir tamsayıyı gösteren

11 sabit bir göstericidir. ptr her zaman

12 aynı hafıza konumunu gösterir

13 ve o konumdaki tamsayı

14 değiştirilemez. \*/

15 printf( "%d\n", \*ptr );

271

16 \*ptr = 7;

17 ptr = &y;

18

19 return 0;

20 }

FIG07\_14.c:

Error E2024 FIG07\_14.c 16: cannot modify a const object in

function main

Error E2024 FIG07\_14.c 17: cannot modify a const object in

function main

\*\*\* 2 errors in Compile \*\*\*

Şekil 7.14 Sabit bir veriyi gösteren sabit bir göstericiyi değiştirmeye çalışmak.

7.6 REFERANSA GÖRE ÇAĞIRMA KULLANAN KABARCIK

SIRALAMA

Şekil 6.15’teki programı iki fonksiyon ( kabarciksiralama ve yerdegistir) kullanacak biçimde

değiştirelim. kabarciksiralama fonksiyonu diziyi sıralamaktadır ve dizi elemanlarını (dizi [ j ] ve dizi [

j+1 ] ) değiştirmek için yerdegistir fonksiyonunu çağırmaktadır (bakınız Şekil 7.15).

C’nin, fonksiyonlar arasında bilgi saklanmasını zorladığını hatırlayınız. Bu sebepten, yerdegistir

fonksiyonu kabarciksiralama içindeki dizi elemanlarına erişim hakkına sahip değildir.

kabarciksiralama fonksiyonu, yerdegistir fonksiyonunun yerleri değiştirilecek dizi elemanlarına

erişmesini istediğinden, değiştirilecek elemanları referansa göre çağırma kullanarak yerdegistir

fonksiyonuna geçirir. Bunu, yerleri değiştirilecek elemanların adreslerini yerdegistir fonksiyonuna

aktararak yapar. Dizilerin tamamı otomatik olarak referansa göre geçirilse de dizi elemanları

skalerdir ve her zaman olduğu gibi değere göre çağırma ile geçirilirler. Bu sebepten,

kabarciksiralama, yerdegistir fonksiyonunu çağırırken (satır 39) referansa göre çağırma etkisi

yaratabilmek için dizi elemanlarının her biriyle adres operatörünü(&), yerdegistir fonksiyonu çağrısı

içinde aşağıdaki biçimde kullanır.

yerdegistir ( &dizi [ j ], &dizi [ j+1] );

yerdegistir fonksiyonu, &dizi [ j]’yi gösterici değişkeni olan eleman1Ptr’ye alır. yerdegistir

fonksiyonunun, bilginin gizlenmesi yüzünden , dizi [ j ] ismini bilmeye hakkı yoktur ancak

\*eleman1Ptr’yi dizi [ j ] için eşanlamda kullanabilir. Bu sebepten yerdegistir,

\*eleman1Ptr’yi kullandığında aslında kabarciksiralama içindeki dizi [ j ]’yi kullanmış

olur. Benzer olarak yerdegistir, \*eleman2Ptr’yi kullandığında aslında kabarciksiralama

içindeki dizi[j+1]’i kullanmış olur. yerdegistir aşağıdaki ifadeleri kullanamasa da

temp=dizi[j];

dizi[j]=dizi[j+1];

dizi[j+1]=temp;

aynı etkiyi Şekil 7.15 ‘teki yerdegistir içinde

temp=\*eleman1Ptr;

\*eleman1Ptr =eleman2Ptr;

272

\*eleman2Ptr=temp;

ifadeleriyle elde edebilir.(satır44-satır 46)

1 /\* Şekil 7.15: fig07\_15.c

2 Bu program değerleri diziye koyar ve artan

3 sıralamada sıralar, ve dizinin son halini yazdırır. \*/

4 #include <stdio.h>

5 #define BOYUT 10

6 void kabarcikSiralama( int \*, const int );

7

8 int main( )

9 {

10

11 int a[ BOYUT ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

12 int i;

13

14 printf( "Veriler orijinal sırada\n" );

15

16 for ( i = 0; i < BOYUT; i++ )

17 printf( "%4d", a[ i ] );

18

19 kabarcikSiralama( a, BOYUT ); /\* diziyi sırala \*/

20 printf( "\nVeriler artan sırada\n" );

21

22 for ( i = 0; i < BOYUT; i++ )

23 printf( "%4d", a[ i ] );

24

25 printf( "\n" );

26

27 return 0;

28 }

29

30 void kabarcikSiralama( int \*dizi, const int boyut )

31 {

32 void yerDegistir( int \*, int \* );

33 int tur, j;

34 for ( tur = 0; tur < boyut - 1; tur++ )

35

36 for ( j = 0; j < boyut - 1; j++ )

37

38 if ( dizi[ j ] > dizi[ j + 1 ] )

39 yerDegistir( &dizi[ j ], &dizi[ j + 1 ] );

40 }

41

42 void yerDegistir( int \*eleman1Ptr, int \*eleman2Ptr )

43 {

44 int temp = \*eleman1Ptr;

45 \*eleman1Ptr = \*eleman2Ptr;

46 \*eleman2Ptr = temp;

273

47 }

Veriler orijinal sırada

2 6 4 8 10 12 89 68 45 37

Veriler artan sırada

2 4 6 8 10 12 37 45 68 89

Şekil 7.15 Referansa göre çağırma ile kabarcık sıralama

kabarciksiralama fonksiyonunun bir çok özelliğine dikkat edilmelidir. Fonksiyonun başlığı

(satır 30) dizi’yi int dizi[ ] yerine int \*dizi biçiminde bildirerek fonksiyonun tek belirteçli bir

diziyi argüman olarak kullanacağını belirtir. (bu iki gösterim birbiri yerine kullanılabilir)

boyut parametresi, en az yetki prensibini uygulamak için const olarak bildirilmiştir. boyut

parametresi, main içindeki değerin bir kopyasını almakta ve bu kopyayı değiştirmektedir.

Ancak bu kopyanın değiştirilmesi main içindeki değeri değiştirmemektedir.

kabarciksiralama görevini yerine getirmek için boyut değişkenini değiştirmek zorunda

değildir. Bu sebepten, boyut, const olarak bildirilerek değiştirilmemesi garantilenmiştir. Eğer

sıralama esnasında dizinin boyutu değiştirilirse, sıralama algoritması doğru bir biçimde

çalışmayabilir.

yerdegistir fonksiyonunun prototipi (satır 32), kabarciksiralama fonksiyonun gövdesi

içindedir çünkü bu fonksiyonu çağıran tek fonksiyon kabarciksiralama fonksiyonudur.

Prototipi kabarciksiralama fonksiyonunun içine yazmak, yerdegistir fonksiyonuna

yapılacak çağrıların kabarciksiralama fonksiyonu tarafından yapılanlarla sınırlı kalmasını

sağlatır. yerdegistir fonksiyonunu çağırmaya çalışan diğer fonksiyonların uygun bir

fonksiyon prototipine erişimleri bulunmadığından derleyici otomatik olarak bir prototip

oluşturacaktır. Bu, genelde fonksiyon başlığıyla fonksiyonun prototipinin uyuşmamasına

sebep olur (ve bir derleyici hatası oluşturur) çünkü derleyici prototipin geri dönüş değeri ve

parametrelerinin int tipte olduklarını varsayar.

Yazılım Mühendisliği Gözlemleri 7.4

Fonksiyon prototiplerini diğer fonksiyon tanımlamaları içine yerleştirmek,

uygun fonksiyon çağrılarının yalnızca prototiplerin yer aldığı fonksiyonlar

tarafından yapılabilmesine kısıtlayarak en az yetki prensibini zorlar.

kabarciksiralama fonksiyonunun, dizinin boyutunu parametre olarak aldığına dikkat ediniz.

Fonksiyon diziyi sıralayabilmek için dizinin boyutunu bilmek zorundadır. Bir dizi, fonksiyona

geçirildiğinde, dizinin ilk elemanının tutulduğu hafıza konumunun adresi otomatik olarak

fonksiyona geçirilir. Adres bilgisi dizinin boyutu hakkında herhangi bir bilgi içermediğinden,

programcı fonksiyona dizinin boyutunu geçirmek zorundadır.

Programda, kabarciksiralama fonksiyonu dizinin boyutunu açık bir biçimde geçirmiştir. Bu

yaklaşımın iki temel faydası vardır. Bunlar ; yazılımın yeniden kullanılabilirliği ve uygun

yazılım mühendisliğidir. Fonksiyonu dizinin boyutunu argüman biçiminde alacak şekilde

274

tanımlayarak, fonksiyonun her boyutta tek belirteçli tamsayı dizilerini sıralayan herhangi bir

programda kullanılabilmesini sağlamış olduk.

Yazılım Mühendisliği Gözlemleri 7.5

Bir diziyi fonksiyona geçirirken, dizinin boyutunu da geçirin. Bu, fonksiyonu

daha genel bir hale getirir. Genel fonksiyonlar çoğu programda sıklıkla

yeniden kullanılır.

Dizinin boyutunu tüm program tarafından erişilebilen bir global değişkende de tutabilirdik.

Bu daha verimli olurdu çünkü dizinin boyutunu fonksiyona geçirmek için boyutun kopyasının

oluşturulması gerekmezdi. Ancak tamsayı dizilerini sıralama yeteneğine ihtiyaç duyan diğer

programlar, aynı global değişkene sahip olmayabileceklerinden fonksiyon bu programlarda

kullanılamayacaktı.

Yazılım Mühendisliği Gözlemleri 7.6

Global değişkenler en az yetki prensibine uymazlar ve zayıf yazılım

mühendisliğinin bir örneğidirler.

Performans İpuçları 7.2

Bir dizinin boyutunu fonksiyona geçirmek zaman alır ve boyutun bir kopyası oluşturulacağından

fazladan hafızaya gerek duyar. Ancak global değişkenlere, her fonksiyon tarafından doğrudan

ulaşılabildiğinden fazladan zaman ve hafızaya ihtiyaç duymazlar.

Dizinin boyutu, fonksiyon içine doğrudan programlanabilir. Bu, fonksiyonun kullanımını belli

büyüklükte dizilerle kısıtlar ve yeniden kullanılabilirliğini büyük oranda azaltır. Yalnızca belli

büyüklükteki, tek belirteçli tamsayı dizilerini işleyen programlar bu fonksiyonu kullanabilir.

C, programın derlenmesi esnasında dizilerin ( ya da diğer veri tiplerinin) büyüklüklerini byte

olarak belirleyen, özel bir tekli operatör olan sizeof operatörüne sahiptir. Şekil 7.16’daki gibi

dizi isimlerine uygulandığında sizeof operatörü, dizideki toplam byte sayısını tamsayı

cinsinde geri döndürür. float tipte değişkenler hafızada normalde 4 byte olarak

tutulduklarından ve dizi1, 20 elemana sahip olacak biçimde bildirildiğinden dizi1 içinde 80

byte bulunmaktadır.

Dizideki eleman sayısı da derleme esnasında bulunabilir.Örneğin,

double gercek[22];

bildirimini ele alalım. double tipte değişkenler hafızada genellikle 8 byte olarak tutulurlar. Bu

sebepten, gercek dizisi 176 byte içermektedir. Dizideki eleman sayısına karar vermek için

aşağıdaki ifade kullanılabilir:

sizeof ( gercek ) / sizeof ( double );

Bu ifade, gercek dizisi içinde kaç byte olduğunu bulur ve bu değeri double tipte bir değişkeni

tutmak için hafızada gerekli olan byte sayısına böler.

1 /\* Şekil 7.16: fig07\_16.c

2 sizeof operatörü dizi isimlerine uygulandığında

3 dizideki byte sayısını döndürür.\*/

275

4 #include <stdio.h>

5

6 size\_t buyukluguBul( float \* );

7

8 int main( )

9 {

10 float dizi[ 20 ];

11

12 printf( "Dizinin byte uzunluğu: %d"

13 "\nbuyukluguBul ile döndürülen byte sayısı: %d\n",

14 sizeof( dizi), buyukluguBul( dizi ) );

15

16 return 0;

17 }

18

19 size\_t buyukluguBul( float \*ptr )

20 {

21 return sizeof( ptr );

22 }

Dizinin byte uzunluğu: 80

buyukluguBul ile döndürülen byte sayısı: 4

Şekil 7.16 sizeof operatörü dizi isimlerine uygulandığında dizideki byte sayısını döndürür.

buyukluguBul fonksiyonunun size\_t tipini döndürdüğüne dikkat ediniz. size\_t tipi, C

standardı tarafından sizeof operatörüyle döndürülen değerin unsigned ya da unsigned long

tipinde gösterimi olarak tanımlanmıştır.

1 /\* Şekil 7.17: fig07\_17.c

2 sizeof operatörü uygulaması \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char c;

8 short s;

9 int i;

10 long l;

11 float f;

12 double d;

13 long double ld;

14 int dizi[ 20 ], \*ptr = dizi;

15

16 printf( " sizeof c = %d"

17 "\tsizeof(char) = %d"

18 "\n sizeof s = %d"

19 "\tsizeof(short) = %d"

276

20 "\n sizeof i = %d"

21 "\tsizeof(int) = %d"

22 "\n sizeof l = %d"

23 "\tsizeof(long) = %d"

24 "\n sizeof f = %d"

25 "\tsizeof(float) = %d"

26 "\n sizeof d = %d"

27 "\tsizeof(double) = %d"

28 "\n sizeof ld = %d"

29 "\tsizeof(long double) = %d"

30 "\n sizeof dizi = %d"

31 "\n sizeof ptr = %d\n",

32 sizeof c, sizeof ( char ), sizeof s,

33 sizeof ( short ), sizeof i, sizeof ( int ),

34 sizeof l, sizeof ( long ), sizeof f,

35 sizeof ( float ), sizeof d, sizeof ( double ),

36 sizeof ld, sizeof ( long double ),

37 sizeof dizi, sizeof ptr );

38

39 return 0;

40 }

sizeof c = 1 sizeof(char) = 1

sizeof s = 2 sizeof(short) = 1

sizeof i = 4 sizeof(int) = 1

sizeof l = 4 sizeof(long) = 1

sizeof f = 4 sizeof(float) = 1

sizeof d = 8 sizeof(double) = 1

sizeof ld = 8 sizeof(long double) = 1

sizeof dizi = 80

sizeof ptr = 4

Şekil 7.17 Standart veri tiplerini büyüklüklerine karar vermek için sizeof kullanmak.

Taşınırlık İpuçları 7.2

Bir veri tipini depolamak için gerekli olan byte sayısı sistemler arasında farlılık gösterebilir. Veri

tiplerinin büyüklüğüne bağlı programlar yazarken ve bu programlar farklı sistemlerde çalışacaksa

veri tiplerini depolamak için gerekli olan byte sayısına sizeof kullanarak karar verin.

sizeof operatörü değişken isimlerine, sabitlere ve herhangi bir veri tipine uygulanabilir. Bir

değişken ismine (dizi olmayan) ya da sabite uygulandığında, belli tipteki değişkenin ya da

sabitin depolanabilmesi için gerekli byte sayısı döndürülür. Tip ismi operand olarak

kullanıldığında sizeof ile birlikte parantezlerin kullanılması gerekir. Bu parantezleri dahil

etmemek yazım hatasına sebep olur. Eğer değişken ismi operand olarak kullanılıyorsa,

parantezlere gerek yoktur.

277

7.7 GÖSTERİCİ İFADELERİ VE GÖSTERİCİ ARİTMETİĞİ

Göstericiler aritmetik ifadelerde, atama ifadelerinde ve karşılaştırma ifadelerinde geçerli

operandlardır. Ancak, bu ifadelerde normalde kullanılabilen operatörlerin bir kısmı

göstericilerle birlikte kullanılamaz. Bu kısım, göstericileri operand olarak kullanan

operatörleri ve bu operatörlerin nasıl kullanılacağını açıklamaktadır.

Göstericilerle yalnızca kısıtlı sayıda aritmetik işlem yapılabilir. Bir gösterici arttırılabilir (++)

ya da azaltılabilir ( -- ), bir tamsayı göstericiye eklenebilir (+ ya da += ), bir tamsayı

göstericiden çıkartılabilir (- ya da - =) ya da bir gösterici diğerinden çıkarılabilir.

int v[10] dizisi bildirilsin ve ilk elemanının hafızada 3000 konumunda tutulduğu bilinsin.

vPtr, v[0]’ı göstermek üzere atanmış olsun yani vPtr’nin değeri 3000 olsun. Şekil 7.18, bu

durumu 4-byte tamsayılar kullanan bir makine için göstermektedir. vPtr’nin v dizisini

gösterebilmesi için aşağıdaki ifadeler kullanılabilir.

vPtr=v;

vPtr=&v[0];

vPtr gösterici değişkeni

Şekil 7.18 v dizisi ve v dizisini gösteren vPtr gösterici değişkeni.

Taşınırlık İpuçları 7.3

Günümüzdeki bilgisayarların çoğu 2 ya da 4 byte tamsayılara sahiptir. Bazı

yeni makineler 8 byte tamsayılar kullanmaktadır. Gösterici aritmetiği

göstericinin gösterdiği nesnelerin boyutuna bağlı olduğundan, gösterici

aritmetiği makine bağımlıdır.

Geleneksel aritmetikte 3000+2 toplamı 3002 değerini oluşturur. Bu, gösterici aritmetiğinde

genellikle geçerli değildir. Bir göstericiye bir tamsayı eklendiğinde ya da çıkartıldığında,

göstericiler o tamsayı kadar arttırılıp azaltılmaz. Bunun yerine, göstericinin gösterdiği

nesnenin boyutuyla o tamsayı çarpılarak bulunan sonuç, göstericiye eklenir ya da göstericiden

çıkartılır. Byte sayısı nesnenin veri tipine bağlıdır. Örneğin,

v[0] v[1] v[2] v[3] v[4]

3000 3004 3008 3012 3016

278

vPtr +=2;

ifadesi tamsayıların hafızada 4 byte olarak tutulduğu düşünülürse 3008 ( 3000+2\*4 )

sonucunu oluşturacaktır. v dizisinde, vPtr artık v [ 2 ]’yi gösterecektir (Şekil 7.19). Eğer bir

tamsayı hafızada iki byte içinde tutuluyorsa, az önceki hesaplama hafıza konumu 3004

(3000+2\*2 ) sonucunu verecekti. Eğer dizi farlı bir veri tipinde olsaydı, az önceki ifade

göstericiyi o veri tipini depolayabilmek için gerekli olan byte sayısının iki katı kadar

arttıracaktı. Gösterici aritmetiği karakter dizilerinde uygulandığında, geleneksel aritmetikle

sonuçlar birbirini tutacaktır çünkü her karakter bir byte uzunluğundadır.

Eğer vPtr, v[4]’ü gösteren 3016’ya arttırılmış olsaydı,

vPtr - =4;

vPtr’yi 3000’e (dizinin başlangıcı) geri döndürecekti. Eğer bir gösterici bir ile arttırılır ya da

azaltılırsa arttırma ( ++ ) ve azaltma ( -- ) operatörleri kullanılabilir.

++vPtr;

vPtr++;

ifadelerinin ikisi de göstericiyi, dizideki bir sonraki konumu gösterecek biçimde arttırır.

--vPtr;

vPtr--;

ifadelerinin ikisi de göstericiyi, dizide bir önceki elemanı gösterecek biçimde azaltır.

vPtr gösterici değişkeni

Şekil 7.19 Gösterici aritmetiğinden sonra vPtr göstericisi

Gösterici değişkenleri birbirlerinden çıkartılabilir. Örneğin, eğer vPtr, 3000 konumunu ve

v2Ptr, 3008 konumunu gösteriyorsa,

x = v2Ptr-vPtr;

ifadesi x’e, vPtr’den v2Ptr’ye kadar olan dizi elemanı sayısını (bu durumda 2) atar. Gösterici

aritmetiği diziler ile kullanılmadığında anlamsızıdır. Aynı tipte iki eleman, bir dizide ard arda

gelen elemanlar olmadıkları sürece hafızada bitişik olarak tutulduklarını düşünemeyiz.

v[0] v[1] v[2] v[3] v[4]

3000 3004 3008 3012 3016

279

Genel Programlama Hataları 7.5

Dizi değerleri haricindeki değerlere gösterici aritmetiği uygulamak.

Genel Programlama Hataları 7.6

Aynı diziyi göstermeyen iki göstericiyi çıkartmak ya da karşılaştırmak.

Genel Programlama Hataları 7.7

Gösterici aritmetiği kullanırken dizi sınırlarını aşmak

Bir gösterici, eğer göstericiler aynı tipte ise başka bir göstericiye atanabilir. Aksi takdirde,

atamanın sağındaki göstericinin tipi, atamanın solunda yer alan göstericinin tipine dönüşüm

operatörü kullanılarak çevrilmelidir. Bu kural için tek istisna void gösteren göstericilerdir

(void\*). Bu göstericiler, her hangi bir gösterici tipini temsil edebilen genel göstericilerdir.

Bütün göstericiler void göstericiye atanabilir ve void gösterici her tipte göstericiye atanabilir.

Her iki durumda da dönüşüm operatörüne gerek yoktur.

void tipte bir göstericinin gösterdiği nesneye erişilemez. Örneğin, derleyici 4-byte tamsayı

kullanan bir makinede int tipinde bir göstericinin 4 byte’ı kullandığını bilir ancak void

gösterici, bilinmeyen veri tipindeki bir hafıza konumunu gösterdiğinden, derleyici göstericinin

gösterdiği byte sayısını kesin olarak bilemez. Derleyici, göstericinin gösterdiği nesneye

erişebilmek için göstericinin veri tipini bilmelidir.

Genel Programlama Hataları 7.8

İkisinden biri void\* tipte olmadıkça, farlı tiplerdeki göstericileri birbirine atamak yazım hatası

oluşturur.

Genel Programlama Hataları 7.9

void\* göstericilerin gösterdiği nenelere erişmeye çalışmak.

Göstericiler, eşitlik operatörleri ve koşullu operatörler ile karşılaştırılabilirler. Ancak böyle

bir karşılaştırma, göstericiler aynı dizinin elemanlarını göstermiyorsa anlamsızdır. Gösterici

karşılaştırmaları, göstericiler içindeki adresleri karşılaştırır. Örneğin, aynı diziyi gösteren iki

göstericinin karşılaştırılması, bir göstericinin dizide diğer göstericinin gösterdiği elemandan

daha yüksek numaralı elemanı gösterdiğini belirtebilir. Göstericilerin karşılaştırılmasında en

yaygın kullanım bir göstericinin NULL olup olmadığına karar vermektir.

7.8 GÖSTERİCİLER VE DİZİLER ARASINDAKİ İLİŞKİ

Diziler ve göstericiler, C’de özel bir biçimde ilişkilidirler ve birbirleri yerine hemen hemen

her yerde kullanılabilirler. Bir dizi ismi, sabit bir gösterici olarak düşünülebilir. Göstericiler,

dizilerin belirteçlerle gösterimi de dahil olmak üzere her işlemde kullanılabilir.

Performans İpuçları 7.3

Dizi belirteç gösterimi, derleme esnasında göstericilerle ifade edildiğinden, dizi

belirteci kullanan deyimleri göstericilerle yazmak derleme zamanını

azaltabilir.

280

İyi Programlama Alıştırmaları 7.5

Dizileri ele alırken, göstericiler yerine dizi belirteçlerini kullanın. Bu, derleme

zamanını biraz arttırsa da bu sayede muhtemelen daha açık programlar

yazmamızı sağlatır.

b[5] tamsayı dizisinin ve bPtr tamsayı göstericisinin bildirildiğini varsayalım. Bir belirteç

kullanmayan dizi isminin, dizinin ilk elemanını gösteren bir gösterici olduğunu bildiğimize

göre, bPtr ’yi b dizisinin ilk elemanına aşağıdaki ifade ile eşitleyebiliriz.

bPtr =b;

Bu ifade, dizinin ilk elemanının adresini aşağıdaki biçimde almakla eşdeğerdir.

bPtr = &b[0];

b[3] dizi elemanı, alternatif bir biçimde

\*(bPtr+3)

gösterici deyimi ile belirtilebilir.

Yukarıdaki deyimde 3, göstericinin offsetidir. Gösterici, dizinin başlangıç adresini

gösterirken, offset dizinin hangi elemanının kullanılacağını belirtir ve offset değeri dizi

belirteciyle eştir. Az önceki gösterime gösterici/offset gösterimi denir. Parantezler gereklidir

çünkü \* operatörünün önceliği + operatörünün önceliğinden yüksektir. Parantezler olmadan

yukarıdaki ifade, \*bPtr’ye 3 eklerdi. (yani, bPtr’nin dizinin başlangıcını gösterdiği

düşünülürse, b[0]‘a 3 eklenirdi.) Dizi elemanlarının, gösterici deyimleri ile belirtilebilmesi

gibi

&b[3]

adresi de

bPtr+3

biçimindeki gösterici deyimi ile belirtilebilir.

Dizinin kendisine de bir gösterici olarak davranılabilir ve göstericiği aritmetiğinde

kullanılabilir. Örneğin,

\*(b+3)

deyimi de b[3] dizi elemanını belirtir. Genelde belirteç kullanan tüm dizi deyimleri, gösterici

ve offset ile yazılabilir. Bu durumda, gösterici/offset gösterimi, dizinin ismini gösterici olarak

kullanır. Az önceki ifadenin, dizinin ismini değiştirmediğine dikkat ediniz. b, hala dizinin ilk

elemanını göstermektedir.

Göstericiler de diziler gibi belirteçlerle kullanılabilir. Örneğin,

bPtr[1]

deyimi, b[1] dizi elemanını belirtir. Buna, gösterici/belirteç gösterimi denir.

281

Dizi isminin, her zaman dizinin başlangıcını gösteren sabit bir gösterici olduğunu hatırlayınız.

Bu sebepten,

b+=3

deyimi geçersizdir çünkü dizi isminin değerini, gösterici aritmetiği kullanarak değiştirmeye

çalışmaktadır.

Genel Programlama Hataları 7.10

Bir dizi ismini, gösterici aritmetiği kullanarak değiştirmeye çalışmak yazım

hatasıdır.

Şekil 7.20, dizi elemanlarını belirtmek için kullandığımız 4 yöntemi (dizi belirteçleri, dizi

ismini gösterici olarak kullanan gösterici/offset metodu, göstericileri belirteçlerle kullanmak

ve bir gösterici kullanan gösterici/offset metodu) tamsayı dizisi olan b dizisinin dört

elemanını yazdırmak için kullanmaktadır.

1 /\* Şekil 7.20: fig07\_20.cpp

2 dizi elemanlarını belirlemenin dört metodu.\*/

3

4 #include <stdio.h>

5

6 int main( )

7 {

8 int b[ ] = { 10, 20, 30, 40 };

9 int \*bPtr = b; /\* bPtr b dizisini göstersin \*/

10 int i, offset;

11 printf( "b dizisi aşağıdaki metodla yazılmıştır:\n"

12 "Dizi belirteçleri yöntemi\n" );

13

14 for ( i = 0; i < 4; i++ )

15 printf( "b[ %d ] = %d\n", i, b[ i ] );

16

17

18 printf( "\nGösterici/offset yöntemi,\n"

19 "gösterici dizinin ismidir\n" );

20

21 for ( offset = 0; offset < 4; offset++ )

22 printf( "\*( b + %d ) = %d\n", offset, \*( b + offset ) );

23

24

25 printf( "\nGösterici belirteç yöntemi\n" );

26

27 for ( i = 0; i < 4; i++ )

28 printf( " bPtr[ %d ] = %d\n", i, bPtr[ i ] );

29

30 printf( "\nGösterici/offset yöntemi\n" );

31

32 for ( offset = 0; offset < 4; offset++ )

33 printf( "\*( bPtr + %d ) = %d\n", offset, \*( bPtr + offset ) );

282

34

35 return 0;

36 }

b dizisi aşağıdaki metodla yazılmıştır:

Dizi belirteçleri yöntemi

b[0] = 10

b[1] = 20

b[2] = 30

b[3] = 40

Gösterici/offset yöntemi

gösterici dizinin ismidir

\*(b + 0) = 10

\*(b + 1) = 20

\*(b + 2) = 30

\*(b + 3) = 40

Gösterici belirteç yöntemi

bPtr[0] = 10

bPtr[0] = 20

bPtr[0] = 30

bPtr[0] = 40

Gösterici/offset yöntemi

\*(bPtr + 0) = 10

\*(bPtr + 1) = 20

\*(bPtr + 1) = 30

\*(bPtr + 1) = 40

Şekil 7.20 dizi elemanlarını belirlemenin dört metodu.

Dizi ve göstericilerin birbirleri yerine kullanılabileceğini daha iyi göstermek için Şekil

7.21’deki program içinde string kopyalayan iki fonksiyona ( kopya1 ve kopya2 ) bakalım.

Her iki fonksiyonda, bir stringi ( muhtemelen bir karakter dizisi ) bir karakter dizisine

kopyalamaktadır. İki fonksiyonun da prototipine bakıldığında, fonksiyonlar eş

gözükmektedirler. Fonksiyonlar aynı görevi yerine getirmektedir ancak birbirlerinden farlı bir

biçimde yazılmışlardır.

1 /\* Şekil 7.21: fig07\_21.cpp

2 Bir stringi dizi gösterimi ve göstericileri

3 kullanarak kopyalamak.\*/

4 #include <stdio.h>

5

6 void kopya1( char \*, const char \* );

7 void kopya2( char \*, const char \* );

8

9 int main( )

283

10 {

11 char string1[ 10 ], \*string2 = "Merhaba",

12 string3[ 10 ], string4[] = "Güle güle";

13

14 kopya1( string1, string2 );

15 printf( "string1 = %s\n", string1 );

16

17 kopya2( string3, string4 );

18 printf( "string3 = %s\n", string3 );

19

20 return 0;

21 }

22

23 /\* s2 yi s1e dizi gösterimi ile kopyalama \*/

24 void kopya1( char \*s1, const char \*s2 )

25 {

26 int i;

27 for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )

28 ; /\* döngünün içinde bir şey yapma \*/

29 }

30

31 /\* s2 yi s1 e gösterici gösterimi ile kopyala \*/

32 void kopya2( char \*s1, const char \*s2 )

33 {

34 for ( ; ( \*s1 = \*s2 ) != '\0'; s1++, s2++ )

35 ; /\* döngünün içinde bir şey yapma \*/

36 }

string1 = Merhaba

string2 = Güle güle

Şekil 7.21 Bir stringi dizi gösterimi ve göstericileri kullanarak kopyalamak.

kopya1 fonksiyonu, s2 içindeki stringi s1 karakter dizisine kopyalamak için dizi belirteç

gösterimini kullanmaktadır. Fonksiyon, tamsayı sayıcı değişkeni olan i’yi, dizi belirteci olarak

kullanmak için bildirmektedir. for yapısı başlığı (27.satır), tüm kopyalama işlemini

yapmaktadır. for yapısının gövdesi boş ifadedir. Başlık, i’yi sıfır olarak atamış ve döngünün

her tekrarında bir arttırarak kullanmıştır. for yapısı içindeki koşul, s1[i]=s2[i], s2’den s1’e

kopyalama işlemini karakter karakter yapmaktadır. s2 içinde null karakterle karşılaşıldığında,

bu karakter s1‘e atanır ve döngü sona erer çünkü null karakterin tamsayı değeri 0’dır

(yanlıştır). Bir atama ifadesinin değerinin, sol argümana atanan değer olduğunu hatırlayınız.

kopya2 fonksiyonu, s2 içindeki stringi s1 karakter dizisine kopyalamak için göstericileri ve

gösterici dizilerini kullanmaktadır. for yapısının başlığı (satır34) yine tüm kopyalama işlemini

yapmaktadır. Başlıkta herhangi bir değişkene değer atanmamıştır. kopya1 fonksiyonunda

olduğu gibi (\*s1=\*s2) koşulu, kopyalama işlemini gerçekleştirmektedir. s2 göstericisinin

gösterdiği nesneye erişilmiş ve sonuçta oluşan karakter, s1 göstericisinin gösterdiği nesneye

atanmıştır. Koşul içindeki atamadan sonra, göstericiler sırasıyla s1 dizisindeki diğer karakteri

ve s2 stringi içindeki diğer karakteri gösterebilmeleri için bir ile arttırılmıştır. s2 içinde null

284

karakterle karşılaşıldığında, bu karakter s1 göstericisinin gösterdiği nesneye atanır ve döngü

sona erer.

kopya1 ve kopya2 fonksiyonlarının ilk argümanlarının, ikinci argüman içindeki stringi

tutabilecek kadar büyük bir dizi olması gerektiğine dikkat ediniz. Aksi takdirde, dizinin

parçası olmayan bir hafıza konumuna yazma yapılmaya çalışıldığında bir hata oluşabilir.

Ayrıca, her fonksiyonun ikinci parametresinin const char \* (sabit bir string) olarak

bildirildiğine dikkat ediniz. İki fonksiyonda da ikinci argüman, ilk argümana kopyalanmıştır.

Karakterler ikinci argümandan sırayla okunmuştur, ancak karakterler hiçbir zaman

değiştirilmemiştir. Bu sebepten, ikinci parametre sabit bir değeri göstermek üzere bildirilerek

en az yetki prensibi uygulanmıştır. İki fonksiyonun da ikinci argümanı değiştirmeleri

gerekmemektedir. Bu sebepten, ikisinin de bunu yapabilmeleri engellenmiştir.

7.9 GÖSTERİCİ DİZİLERİ

Diziler göstericiler içerebilir. Bu tipte veri yapılarının genel kullanımı, stringlerden oluşan bir

dizi oluşturmaktır. Bu tür dizilere string dizileri denir. Dizideki her girdi bir stringtir, ancak

C’de bir string, gerçekte ilk karakterini gösteren bir göstericidir. Bu yüzden, string

dizilerindeki her girdi, aslında stringin ilk karakterini gösteren bir göstericidir. Aşağıda, bir

desteyi temsil etmek için kullanılabilecek takim string dizisini ele alalım.

const char \*takim[4]={“kupa”,”karo”,”sinek”,” maça”};

Bildirimin takim[4] kısmı, 4 elemanlı bir diziyi belirtmektedir. Bildirimin char\* kısmı,

takim dizisinin her elemanının char gösteren tipte bir gösterici olduğunu belirtmektedir.

const belirteci, dizinin gösterici elemanları tarafından gösterilen stringlerin

değiştirilemeyeceğini belirtir. Diziye yerleştirilecek 4 değer “kupa”,”karo”,”sinek” ve

“maça” olarak verilmiştir. Bunların her biri, hafızada null karakterle sonlandırılmış stringler

olarak tutulurlar ve bu stringlerin uzunluğu tırnak içindeki karakter sayılarından birer karakter

daha fazladır. 4 stringin uzunluğu sırasıyla 5, 5, 6 ve 5 karakter uzunluğundadır. Bu stringler,

takim dizisi içine yerleştirilecekmiş gibi gözükse de aslında dizide yalnızca göstericiler

tutulmaktadır ( Şekil 7.22 ). Her gösterici, ilgili stringin ilk karakterini göstermektedir. Bu

sebepten, takim dizisi büyüklük olarak sabit olsa da her hangi bir uzunluktaki karakter

stringlerine erişimi sağlar. Bu esneklik, C’nin güçlü veri yapılandırma yeteneklerinin bir

örneğidir.

Takımlar, her satırın bir takımı ve her sütunun takım isminin harflerinden birini temsil ettiği

iki boyutlu bir diziye yerleştirilebilirdi. Bu şekildeki bir veri yapısı, her satır için sabit sayıda

sütuna sahip olmalıydı ve bu sayı en büyük stringi tutabilecek kadar geniş olmalıydı. Bu

sebepten, çoğu stringin en uzun stringten daha küçük olduğu durumlarda önemli miktarlarda

hafıza gereksiz yere harcanmış olacaktı. String dizilerini, bir sonraki kısımda iskambil

destelerini temsil etmek için kullanacağız.

‘K’ ‘u’ ‘p’ ‘a’ ‘\0’

‘K’ ‘a’ ‘r’ ‘o’ ‘\0’

‘S’ ‘i’ ‘n’ ‘e’ ‘k’ ‘\0’

takim[0]

takim[1]

takim[2]

takim[3]

285

Kupa 0

Karo 1

Sinek 2

Maça 3

As iki üç dört beş altı yedi sekiz dokuz on vale kız papaz

0 1 2 3 4 5 6 7 8 9 10 11 12

‘M’ ‘a’ ‘ç’ ‘a’ ‘\0’

Şekil 7.22 takim dizisinin grafik temsili.

7.10 ÖRNEK:KART KARMA VE DAĞITMA

Bu kısımda, kart karma ve dağıtma programı geliştirebilmek için rasgele sayılar üretmeyi

kullanacağız. Bu program, daha sonradan başka kart oyunlarında da kullanılabilir. Daha iyi

performans sağlayan problemler gösterebilmek için, burada bilerek en iyi kart karma ve

dağıtma algoritmalarını göstermedik. Alıştırmalarda ve 10.ünitede daha verimli algoritmalar

geliştireceğiz.

Yukarıdan aşağıya, adımsal iyileştirme yaklaşımını kullanarak 52 kartlık bir desteyi karan ve

her kartı dağıtan bir program geliştireceğiz. Yukarıdan aşağıya yaklaşımı, özellikle, büyük ve

ilk ünitelerde gördüğümüz problemlerden daha karmaşık problemlerde kullanışlıdır.

Kartların destesini temsil edebilmek için, 4’e 13’lük iki boyutlu bir dizi olan deste dizisini

kullanacağız (Şekil 4.23). Satırlar takımları göstermektedir. 0.satır kupa takımını, 1.satır karo

takımını, 2.satır sinek takımını ve 3.satır maça takımını göstermektedir. Sütunlar kartları

temsil etmektedir. 0- 9 sütunları sırasıyla As-10 kartlarını ve 10-12 sütunları sırasıyla vale, kız

ve papaz kartlarını temsil etmektedir. takim string dizisini, 4 takımı gösteren karakter

stringleriyle ve taraf string dizisini, 13 kartında değerini gösteren karakter stringleriyle

yüklemeliyiz.

Kartların destesini şu şekilde karabiliriz. Öncelikle, deste dizisi sıfırlanarak temizlenir.Daha

sonra rasgele olarak bir satir (0-3) ve bir sutun (0-12) seçilir. 1 sayısı, deste[satir][sutun]

elemanının içine yerleştirilerek, bu kartın karılmış desteden ilk olarak dağıtılacağını belirtilir.

Bu işlem, hangi kartların karılmış destede 2, 3, ..., 52. sırada olacağını belirlemek için, 2, 3, ...,

52 sayıları deste içine rasgele yerleştirilene dek sürer. deste dizisi kart sayılarıyla dolarken,

bir kartın iki kere seçilme yani, deste[satir][sutun] seçildiğinde bu elemanın sıfır olmama

ihtimali bulunmaktadır. Bu seçim ihmal edilir ve seçilmemiş kart bulunana dek diğer satir ve

sutunlar rasgele olarak seçilmeye devam edilir. En sonunda, 1’den 52’ye kadar olan sayılar

deste dizisindeki boşlukları dolduracaktır. Bu noktada deste tamamen karılmış olur.

286

deste[ 2 ][ 12 ] Sinek papazı gösterir

sinek papaz

Şekil 7.23 Bir deste kartın iki boyutlu dizilerle temsil edilişi.

Bu karma algoritması, eğer karılmış kartlar defalarca rasgele bir biçimde seçilmişlerse belirsiz

bir biçimde çalışabilir. Bu olay belirsiz erteleme olarak bilinir.Alıştırmalarda belirsiz erteleme

ihtimalini ortadan kaldıran daha iyi bir karma algoritması tartışacağız.

Performans İpuçları 7.4

Doğal bir biçimde ortaya çıkan algoritmalar, belirsiz erteleme gibi daha önceden belirlenemeyen

performans sorunları içerebilir. Belirsiz ertelemeden kaçınan algoritmaları arayın.

İlk kartı dağıtabilmek için dizide deste[satir][sutun]=1’i arayacağız. Bu, satir’ı 0-3 ve

sutun’u 0-12 arasında değiştiren yuvalı for yapısıyla gerçekleştirilir. Peki, dizinin her hangi

bir elemanının değerinin neye karşılık geldiğini nasıl bileceğiz? takim dizisi daha önceden 4

takımla yüklenmişti, öyleyse takım değerini elde edebilmek için takim[satir] karakter dizisini

yazdıracağız. Benzer bir biçimde, kartın değerini elde edebilmek için taraf[sutun] karakter

stringini yazdırırız. Bu bilgileri uygun sırada yazdırmak, her kartı “Karo As” ya da “Sinek

Sekiz” gibi ifade edebilmemizi sağlatır.

Şimdi yukarıdan aşağı adımsal iyileştirme metodunu kullanalım.En üstte basit olarak

52 kartı kar ve dağıt

yer alır. İlk iyileştirmemiz şu sonucu verir:

takim dizisine değerler ata.

taraf dizisine değerler ata

deste dizisine değerler ata

desteyi kar

52 kartı dağıt.

“desteyi kar” şu şekilde genişletilebilir:

52 kartın her biri için (for)

destenin rasgele seçilmiş boş bir alanına kart sayısını yerleştir.

“52 kartı dağıt” şu şekilde genişletilebilir:

52 kartın her biri için (for)

deste dizisinde kart sayısını bul ve kartın hangi desteye ait olduğunu ve değerini

yazdır.

Bu genişletmeleri bir araya getirmek ikinci iyileştirmemizin tümünü oluşturur:

287

takim dizisine değerler ata.

taraf dizisine değerler ata

deste dizisine değerler ata

52 kartın her biri için

destenin rasgele seçilmiş boş bir alanına kart sayısını yerleştir.

52 kartın her biri için

deste dizisinde kart sayısın bul ve kartın hangi desteye ait olduğunu ve değerini

yazdır.

“destenin rasgele seçilmiş boş bir alanına kart sayısını yerleştir” şu şekilde genişletilebilir:

Destedeki alanı rasgele seç

Destedeki bu alan daha önceden seçildiği sürece(while)

Destedeki alanı rasgele seçmeye devam et

Destede seçilen alana kart sayısını yerleştir.

“deste dizisinde kart sayısın bul ve kartın hangi desteye ait olduğunu ve değerini yazdır” şu

şekilde genişletilebilir:

Deste dizisindeki her alan için(for)

Eğer alan kart sayısı içeriyorsa(if)

Kartın hangi desteye ait olduğunu ve değerini yazdır.

Bu genişletmeleri birleştirmek üçüncü iyileştirmemizin tamamını oluşturur:

takim dizisine değerler ata.

taraf dizisine değerler ata

deste dizisine değerler ata

52 kartın her biri için

Destedeki alanı rast gele seç

Destedeki bu alan daha önceden seçildiği sürece

Destedeki alanı rasgele seçmeye devam et

Destede seçilen alana kart sayısını yerleştir.

52 kartın her biri için

Deste dizisindeki her alan için

Eğer alan kart sayısı içeriyorsa

Kartın hangi desteye ait olduğunu ve değerini yazdır.

Bu, iyileştirme sürecini tamamlar. Bu programın, algoritmanın karma ve dağıtma kısımlarını

birleştirerek her kartın desteye yerleştirildiğinde dağıtılması ile daha verimli hale geleceğine

dikkat ediniz. Programın, bu işlemleri ayrı ayrı yapmasını tercih ettik çünkü normalde kartlar

karıldıktan sonra dağıtılır ( karılırken değil )

Kart karma ve dağıtma programı Şekil 7.24’te ve bu programın örnek bir çıktısı Şekil 7.25’te

gösterilmiştir. printf çağrıları içinde karakter stringlerini yazdırmak için %s dönüşüm

288

belirtecinin kullanıldığına dikkat ediniz. printf çağrısındaki ilgili argüman, char gösteren bir

gösterici (ya da karakter dizisi) olmalıdır. dagıtma fonksiyonunda “%5s-%8s” biçim tarifi

(54.satır) sağa dayalı olarak 5 karakterlik bir alana karakter stringi ve sola dayalı olarak 8

karakterlik bir alana diğer karakter stringini yazdırmak için kullanılmıştır.%-8s içindeki eksi

işareti, stringin 8 alan genişliği içinde sola dayalı olacağını belirtir.

1 /\* Şekil 7.24: fig07\_24.c

2 Kart dağıtma Programı \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5 #include <time.h>

6

7 void kar( int [ ][ 13 ] );

8 void dagit( const int [ ][ 13 ], const char \*[ ], const char \*[ ] );

9

10 int main( )

11 {

12 const char \*takim[ 4 ] =

13 { "Kupa", "Karo", "Sinek", "Maça" };

14 const char \*taraf[ 13 ] =

15 { "As", "İki", "Üç", "Dört",

16 "Beş", "Altı", "Yedi", "Sekiz",

17 "Dokuz", "On", "Vale", "Kız", "Papaz" };

18 int deste[ 4 ][ 13 ] = { 0 };

19

20 srand ( time( 0 ) );

21

22 kar( deste );

23 dagit( deste, taraf, takim );

24

25 return 0;

26 }

27

28 void kar( int wDeste[ ][ 13 ] )

29 {

30 int satir, sutun, kart;

31

32 for ( kart = 1; kart <= 52; kart++ ) {

33 do {

34 satir = rand( ) % 4;

35 sutun = rand() % 13;

36 } while( wDeste [ satir ][ sutun ] != 0 );

37

38 wDeste[ satir ][ sutun] = kart;

39 }

40 }

41

42 void dagit( const int wDeste[ ][ 13 ], const char \*wTaraf [ ],

43 const char \*wTakim[ ] )

289

44 {

45 int kart, satir, sutun;

46

47 for ( kart = 1; kart <= 52; kart++ )

48

49 for ( satir = 0; satir <= 3; satir++ )

50

51 for ( sutun = 0; sutun <= 12; sutun++ )

52

53 if ( wDeste[ satir ][ sutun] == kart )

54 printf( "%5s of %-8s%c",

55 wTakim[ satir ], wTaraf[ sutun ],

56 kart % 2 == 0 ? '\n' : '\t' );

57 }

Şekil 7.24 Kart dağıtma Programı

Karo Sekiz Kupa As

Sinek Sekiz Sinek Beş

Kupa Yedi Karo İki

Sinek As Karo On

Maça İki Karo Altı

Maça Yedi Sinek İki

Sinek Vale Maça On

Kupa Papaz Karo Vale

Kupa Üç Karo Üç

Sinek Üç Sinek Dokuz

Kupa On Kupa İki

Sinek On Karo Yedi

Sinek Altı Maça Kız

Kupa Altı Maça Üç

Karo Dokuz Karo As

Maça Vale Sinek Beş

Karo Papaz Sinek Yedi

Maça Dokuz Kupa Dört

Maça Altı Maça Sekiz

Karo Kız Karo Beş

Maça As Kupa Dokuz

Sinek Papaz Kupa Beş

Maça Papaz Karo Dört

Kupa Kız Kupa Sekiz

Maça Dört Kupa Vale

Sinek Dört Sinek Kız

Şekil 7.25 Kart dağıtma programının örnek bir çıktısı

Dağıtma algoritmasında bir zayıflık vardır. Bir eşleme bulunduğunda, ilk denemede bile

bulunmuş olsa, içteki iki for yapısı deste’nin geriye kalan elemanlarını bir eşleme yapmak

için aramaya devam etmektedir. Alıştırmalarda ve 10.ünitedeki örneklerde bu kusuru

düzelteceğiz.

290

7.11 FONKSİYONLARI GÖSTEREN GÖSTERİCİLER

Bir fonksiyonu gösteren gösterici, fonksiyonun hafızadaki adresini tutar. 6.ünitede, dizi isminin,

gerçekte dizinin ilk elemanının hafızadaki adresi olduğunu görmüştük. Benzer olarak, bir fonksiyon

ismi, fonksiyonun görevini yapan kodun hafızadaki başlangıç adresidir. Fonksiyonları gösteren

göstericiler, fonksiyonlara geçirilebilir, fonksiyonlardan geri döndürülebilir ve fonksiyonları gösteren

diğer göstericilere atanabilir.

Fonksiyonları gösteren göstericileri açıklayabilmek için Şekil 7.15’teki kabarcık sıralama

programını değiştirerek Şekil 7.26’da yeniden gösterdik. Yeni programımız kabarcik,

yerdegistir, artan ve azalan fonksiyonlarını içermektedir. kabarcikSiralama fonksiyonu,

bir tamsayı dizisi ve bu dizinin büyüklüğü yanında fonksiyon gösteren (artan ya da azalan

fonksiyonlarından birini) bir göstericiyi argüman olarak almaktadır. Program, kullanıcının

sıralamanın artan mı yoksa azalan bir şekilde mi yapılacağını seçmesini isteyen, bir ileti

yazdırmaktadır. Eğer kullanıcı 1 girerse, artan fonksiyonunu gösteren bir gösterici kabarcik

fonksiyonuna geçirilerek, dizinin artan bir sırada sıralatılması sağlanır. Eğer kullanıcı 2

girerse, azalan fonksiyonunu gösteren bir gösterici kabarcik fonksiyonuna geçirilerek,

dizinin azalan bir sırada sıralatılması sağlanır. Programın çıktısı Şekil 7.27’de gösterilmiştir.

Aşağıdaki parametreler kabarcik fonksiyonunun ( satır 42 ) başlığında yer almaktadır.

int(\*karsilastir)(int,int)

Bu, kabarcik fonksiyonuna iki tamsayı parametresi alan ve bir tamsayı sonucu döndüren bir

fonksiyonu gösteren bir gösterici alacağını söyler. \*karsilastir etrafındaki parantezler

gereklidir, çünkü \* fonksiyon parametrelerini içine alan parantezlere göre daha düşük

önceliğe sahiptir.

Eğer parantezleri dahil etmeseydik, bildirim

int \*karsilastir(int,int)

biçiminde olacaktı ve iki tamsayı parametresi alan ve bir tamsayıyı gösteren bir gösterici

döndüren bir fonksiyon bildirilmiş olacaktı.

1 /\* Şekil 7.26: fig07\_26.cpp

2 fonksiyonları gösteren göstericiler kullanan çok amaçlı bir sıralama fonksiyonu \*/

3 #include <stdio.h>

4 #define BOYUT 10

5 void kabarcik ( int [ ], const int, int (\*)( int, int ) );

6 int artan( int, int );

7 int azalan( int, int );

8

9 int main( )

10 {

11

12 int secim,

13 sayici,

14 a[ BOYUT ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

15

16 printf( "Artan sıralama için 1 girin,\n"

17 "Azalan sıralama için 2 girin: " );

18 scanf( "%d", &secim);

19 printf( "\nVeriler orijinal sırasında \n" );

291

20

21 for ( sayici = 0; sayici < BOYUT; sayici++ )

22 printf( "%5d", a[ sayici] );

23

24 if ( secim == 1 ) {

25 kabarcik( a, BOYUT, artan);

26 printf(

"

\nVeriler artan sırada

\n" );

27

}

28 else

{

29 kabarcik( a, BOYUT, azalan);

30 printf( "

\nVeriler azalan sırada

\n" );

31

}

32

33 for ( sayici = 0; sayici < BOYUT; sayici++ )

34 printf( "%5d", a[ sayici ] );

35

36 printf( "

\n" );

37

38 return 0;

39

}

40

41 void kabarcik( int is[ ], const int boyut,

42 int (\*karsilastir)( int, int ) )

43

{

44 int tur, sayici;

45

46 void yerDegistir( int \*, int \* );

47

48 for ( tur = 1; tur < boyut; tur++ )

4950 for ( sayici = 0; sayici < boyut

- 1; sayici++ )

51

52 if ( (\*karsilastir)( is[ sayici ], is[ sayici + 1 ] ) )

53 yerDegistir( &is[ sayici ], &is[ sayici + 1 ] );

54

}

55

56 void yerDegistir( int \*eleman1Ptr, int \*eleman2Ptr )

57

{

58 int temp;

59

60 temp = \*eleman1Ptr;

61 \*eleman1Ptr = \*eleman2Ptr;

62 \*eleman2Ptr = temp;

63

}

64

65 int artan( int a, int b )

66

{

67 return b < a; /\* b, a dan küçükse yer değiştir\*/

68

}

69

292

70 int azalan( int a, int b )

71 {

72 return b > a; /\* b, a dan büyükse yer değiştir \*/

73 }

Şekil 7.26 Fonksiyonları gösteren göstericiler kullanan çok amaçlı bir sıralama fonksiyonu

Artan sıralama için 1 girin

Azalan sıralama için 2 girin: 1

Veriler orijinal sırasında

2 6 4 8 10 12 89 68 45 37

Veriler artan sırada

2 4 6 8 10 12 37 45 68 89

Artan sıralama için 1 girin

Azalan sıralama için 2 girin: 2

Veriler orijinal sırasında

2 6 4 8 10 12 89 68 45 37

Veriler artan sırada

89 68 45 37 12 10 8 6 4 2

Şekil 7.27 Şekil 7.26’daki kabarcık sıralama programının çıktıları.

kabarcik fonksiyonunun prototipindeki ilgili parametre(satır 5)

int(\*)(int,int)

biçimindedir. Yalnızca tiplerin kullanıldığına dikkat ediniz. Ancak programcı,

dokümantasyon amacıyla derleyici tarafından ihmal edilecek isimler ekleyebilir.

kabarcik fonksiyonuna geçirilen fonksiyon, if ifadesi içinden(52.satır) aşağıdaki biçimde

çağrılmıştır:

if ( ( \* karsilastir ) ( is [sayici] , is [sayici+1] ) )

Bir değişkenin değerine erişmek için değişkeni gösteren göstericileri kullanılması gibi, fonksiyonu

kullanmak içinde fonksiyonu gösteren göstericiler ele alınmalıdır.

Fonksiyon çağrısı, göstericinin gösterdiği nesneye erişilmeden aşağıdaki biçimde

yapılabilirdi.

if ( karsilastir ) ( is [sayici] , is [sayici+1] ) )

293

Burada gösterici, fonksiyon ismi gibi doğrudan kullanılmıştır. Bir fonksiyonu gösterici ile

çağırmak için ilk yöntemi tercih ederiz çünkü bu, fonksiyonu çağırmak için kullanılan

karsilastir’ın, bir fonksiyonu gösteren gösterici olduğunu vurgular. Bir fonksiyonu gösterici

ile çağırmak için kullanılan ikinci yöntemde, karsilastir sanki gerçek fonksiyonmuş gibi

gözükmektedir. Bu, karsilastir fonksiyonunun tanımını görmek isteyen ve bunun dosyada hiç

tanımlanmadığını fark eden bir kullanıcı için kafa karıştırıcı olacaktır.

Fonksiyonları gösteren göstericilerin en yaygın kullanımı, menüler sunan sistemlerdir.

Kullanıcının menüden (muhtemelen 1-5 arasında) bir seçim yapması istenir.Her seçim farklı

bir fonksiyon sayesinde gerçekleştirilir. Her fonksiyonu gösteren göstericiler, fonksiyonları

gösteren göstericiler dizisi içinde tutulur. Kullanıcının seçimi dizinin belirteci olarak kullanılır

ve dizideki gösterici fonksiyonu çağırır.

Şekil 7.28, fonksiyonları gösteren gösterici dizilerinin bildirilmesi ve kullanılmasıyla ilgili

genel bir örnek sunmaktadır. 3 fonksiyon tanımlanmıştır ( fonksiyon1, fonksiyon2 ve

fonksiyon3 ) Bu fonksiyonların her biri bir tamsayı argümanı almakta ve geriye hiçbir şey

döndürmemektedir. Bu fonksiyonları gösteren göstericiler, aşağıdaki gibi tanımlanmış f

dizisine (satır 10) depolanmıştır.

void ( \*f [3] )( int ) = { fonksiyon1,fonksiyon2,fonksiyon3};

Bu bildirim şu şekilde okunur: ”f ,argüman olarak int alan ve geriye değer döndürmeyen

fonksiyonları gösteren 3 göstericinin dizisidir”. Diziye bu 3 fonksiyonun isimleri atanmıştır.

Kullanıcı 0 ile 2 arasında bir değer girdiğinde bu değer, fonksiyonları gösteren göstericilerin

dizisinde belirteç olarak kullanılır. Fonksiyon çağrısı ( 17.satır ) şu şekilde yapılmıştır:

( \*f [secim] )( secim );

Fonksiyon çağrısında, f [secim], dizide secim konumunda bulunan göstericiyi seçer.

Göstericinin fonksiyonu çağırabilmesi için göstericinin içeriğine erişilir ve secim argüman

olarak fonksiyona geçirilir. Her fonksiyon argümanını ve kendi fonksiyonun ismini

yazdırarak, fonksiyon çağrısının doğru bir biçimde yapıldığını gösterir. Alıştırmalarda

menüler kullanan bir program geliştireceksiniz.

1 /\* Şekil 7.28: fig07\_28.cpp

2 Fonksiyonları gösteren göstericilerin dizisi \*/

3 #include <stdio.h>

4 void fonksiyon1( int );

5 void fonksiyon2( int );

6 void fonksiyon3( int );

7

8 int main( )

9 {

10 void (\*f[ 3 ])( int ) = { fonksiyon1, fonksiyon2, fonksiyon3 };

11 int secim;

12

13 printf( "0 ile 2 arasında bir sayı, ya da çıkış için 3 giriniz: " );

14 scanf( "%d", &secim );

15

16 while ( secim >= 0 && secim < 3 ) {

294

17 (\*f[ secim ])( secim );

18 printf( "0 ile 2 arasında bir sayı, ya da çıkış için 3 giriniz: ");

19 scanf( "%d", &secim );

20 }

21

22 printf( "Programın çalışması bitti\n" );

23

24 return 0;

25 }

26

27 void fonksiyon1( int a )

28 {

29 printf( "%d girdiniz"

30 " yani fonksiyon1 çağırıldı \n\n", a );

31 }

32

33 void fonksiyon2( int b )

34 {

35 printf( " %d girdiniz"

36 " fonksiyon2 çağırıldı\n\n", b );

37 }

38

39 void fonksiyon3( int c )

40 {

41 printf( "%d girdiniz"

42 " fonksiyon3 çağırıldı\n\n", c );

43 }

0 ile 2 arasında bir sayı, ya da çıkış için 3 giriniz: 0

0 girdiniz yani fonksiyon1 çağırıldı

0 ile 2 arasında bir sayı, ya da çıkış için 3 giriniz: 1

1 girdiniz yani fonksiyon2 çağırıldı

0 ile 2 arasında bir sayı, ya da çıkış için 3 giriniz:2

2 girdiniz yani fonksiyon3 çağırıldı

0 ile 2 arasında bir sayı, ya da çıkış için 3 giriniz: 3

Programın çalışması bitti.

Şekil 7.28 Fonksiyonları gösteren göstericilerin dizisi

295

ÖZET

 Göstericiler, değer olarak hafıza adreslerini içeren değişkenlerdir.

 Göstericiler kullanılmadan önce bildirilmelidir.

 int \*ptr

biçiminde bir bildirim, ptr değişkenini int tipinde bir nesneyi gösteren gösterici

olarak bildirir. Bu bildirim “ptr bir int göstericisidir” biçiminde okunur.

Bildirim içindeki \* bildirilen değişkenin gösterici olduğunu belirtir.

 Göstericilere ilk değer olarak üç değer : 0 , NULL ya da bir adres atanabilir. Bir

göstericiye 0 atamak, NULL atamakla eşdeğerdir.

 Bir göstericiye atanabilen tek tamsayı 0’dır.

 & ya da adres operatörü, operandının adresini döndüren bir tekli operatördür.

 Adres operatörünün operandı bir değer olmalıdır; adres operatörü sabitlere,deyimlere

ya da register depolama sınıfıyla bildirilmiş değişkenlere uygulanamaz.

 \* operatörü ya da genellikle söylendiği biçimde içerik operatörü, operandının(yani

göstericinin) gösterdiği nesnenin değerini döndürür. Bu şekilde bir \* kullanımına,

göstericinin gösterdiği nesneye erişmek denir.

 Argümanları değiştirilecek bir fonksiyon çağrılırken, argümanların adresleri geçirilir.

Değişkenin adresi fonksiyona geçirildiğinde, içerik operatörü (\*) fonksiyonla birlikte

çağırıcının hafızasındaki o konumunda bulunan değeri değiştirmek için kullanılabilir.

 Argüman olarak bir adres alan fonksiyonlar, adresi alabilmek için bir gösterici parametresi

bildirmelidir

 Fonksiyon prototiplerinde göstericilerin isimlerinin belirtilmesine gerek

yoktur.Dokümantasyon amaçları için dahil edilen isimler, C derleyicisi tarafından ihmal

edilir.

 const belirteci, programcının derleyiciye belirli bir değişkenin değerinin

değiştirilmemesi gerektiğinin bildirmesini sağlar.

 Eğer const olarak bildirilmiş bir değer değiştirilmeye çalışılırsa, derleyici bunu

yakalar ve bilgisayara bağlı olarak hata ya da uyarı mesajı yayınlar.

 Bir göstericiyi fonksiyona geçirmenin dört yolu vardır.Bunlar ; sabit olmayan bir

göstericiyi sabit olmayan bir veriyle, sabit bir göstericiyi sabit olmayan bir veriyle,

sabit olmayan bir göstericiyi sabit bir veriyle ve sabit bir göstericiyi sabit bir veriyle

kullanmaktır.

 Diziler referansa göre çağırma ile otomatik bir biçimde geçirilirler çünkü dizi isminin

değeri dizinin başlangıç adresidir.

 Bir dizinin tek bir elemanı referansa göre çağırma ile geçirileceğinde,belirlenen dizi

elemanının adresi geçirilir.

 C, programın derlenmesi esnasında dizilerin ( ya da diğer veri tiplerinin)

büyüklüklerini byte olarak belirleyen, özel bir tekli operatör olan sizeof operatörüne

sahiptir.

 Dizi isimlerine uygulandığında sizeof operatörü, dizideki toplam byte sayısını tamsayı

cinsinde geri döndürür.

 sizeof operatörü değişken isimlerine, sabitlere ve herhangi bir veri tipine

uygulanabilir.

 size\_t tipi, C standardı tarafından sizeof operatörüyle döndürülen değerin unsigned ya

da unsigned long tipinde gösterimi olarak tanımlanmıştır.

 Göstericilerle yalnızca kısıtlı sayıda aritmetik işlem yapılabilir. Bir gösterici

arttırılabilir (++) ya da azaltılabilir ( -- ), bir tamsayı göstericiye eklenebilir (+ ya da

296

+= ), bir tamsayı göstericiden çıkartılabilir (- ya da - =) ya da bir gösterici diğerinden

çıkarılabilir.

 Bir göstericiye bir tamsayı eklendiğinde ya da çıkartıldığında, göstericiler o tamsayı

kadar arttırılıp azaltılmaz. Bunun yerine, göstericinin gösterdiği nesnenin boyutuyla o

tamsayı çarpılarak bulunan sonuç, göstericiye eklenir ya da göstericiden çıkartılır.

 Gösterici aritmetiği diziler ile kullanılmadığında anlamsızıdır.Aynı tipte iki eleman,

bir dizide ard arda gelen elemanlar olmadıkları sürece hafızada bitişik olarak

tutulduklarını düşünemeyiz.

 Gösterici aritmetiği karakter dizilerinde uygulandığında, geleneksel aritmetikle

sonuçlar birbirini tutacaktır çünkü her karakter bir byte uzunluğundadır.

 Bir gösterici, eğer göstericiler aynı tipte ise başka bir göstericiye atanabilir.Aksi

takdirde, atamanın sağındaki göstericinin tipi, atamanın solunda yer alan göstericinin

tipine dönüşüm operatörü kullanılarak çevrilmelidir.Bu kural için tek istisna void

gösteren göstericilerdir (void\*). Bu göstericiler, her hangi bir gösterici tipini temsil

edebilen genel göstericilerdir. Bütün göstericiler void göstericiye atanabilir ve void

gösterici her tipte göstericiye atanabilir.Her iki durumda da dönüşüm operatörüne

gerek yoktur.

 void tipte bir göstericinin gösterdiği nesneye erişilemez.

 Göstericiler, eşitlik operatörleri ve koşullu operatörler ile karşılaştırılabilirler.Ancak

böyle bir karşılaştırma, göstericiler aynı dizinin elemanlarını göstermiyorsa

anlamsızdır.

 Göstericiler dizi isimlerinde olduğu gibi belirteçlerle kullanılabilir.

 Dizi ismi, dizinin ilk elemanını gösteren bir göstericidir.

 Gösterici/offset yönteminde offset, dizi belirteciyle eşdeğerdedir.

 Belirteçli tüm dizi deyimleri, bir gösterici ve dizi ismini gösterici olarak kullanan bir

offset ya da diziyi gösteren başka bir gösterici yardımıyla yazılabilir.

 Bir dizi ismi hafızada her zaman aynı konumu gösteren bir göstericidir.Dizi

isimleri,göstericilerde olduğu gibi değiştirilemez.

 Gösterici dizileri oluşturmak mümkündür.

 Fonksiyonları gösteren göstericiler kullanılabilir.

 Fonksiyonu gösteren gösterici, fonksiyonun kullandığı kodların bulunduğu hafıza

adresidir.

 Fonksiyonları gösteren göstericiler fonksiyonlara geçirilebilir,fonksiyonlardan geri

döndürülebilir,dizilerde saklanabilir ve başka göstericilere atanabilir.

 Fonksiyon göstericilerinin en yaygın kullanımı menü içeren programlardır.

ÇEVRİLEN TERİMLER

adress operator.................................... adres operatörü ( & )

array of pointers................................. göstericilerin dizisi

call by reference................................ referansa göre çağırma

call by value...................................... değere göre çağırma

character pointer............................... karakter göstericisi

constant pointer................................ sabit gösterici

dereference a pointer....................... göstericinin gösterdiği nesneye erişmek

dynamic memory allocation............. dinamik hafıza tahsisi

function pointer............................... fonksiyon göstericisi

indirection....................................... dolaylama

indirection operator........................ içerik operatörü ( \* )

297

linked list....................................... bağlı liste

offset............................................. offset (kayma olarak da bilinir)

pointer.......................................... gösterici

pointer arithmetic......................... gösterici aritmetiği

pointer expression....................... gösterici değeri

pointer to a function.................... bir fonksiyonu gösteren gösterici

GENEL PROGRAMLAMA HATALARI

7.1 İçerik ( indirection ) operatörü (\*), bildirimde bütün değişken isimlerine

dağıtılmaz.Her gösterici isimden önce \* kullanılarak bildirilmelidir.

7.2 Uygun bir biçimde ilk değerlere atanmamış ya da belli bir hafıza konumunu

göstermek için atanmamış göstericilerin gösterdiği nesneye ulaşmaya çalışmak.Bu,

çalışma zamanlı ölümcül hatalara ya da önemli bir veriyi yanlışlıkla değiştirerek

programın yanlış sonuçlar üretmesine sebep olur.

7.3 Göstericinin gösterdiği değeri elde etmek gerektiğinde göstericinin gösterdiği nesneye erişmemek.

7.4 Bir fonksiyonun referansa göre çağrılar yapıldığında, argüman olarak gösterici beklediğinden ve

değere göre çağrılar yapıldığında argümanların geçirildiğinden habersiz olmak.Bazı derleyiciler

değer alırken, değerlere gösterici gibi davranırlar ve değerlere göstericilerde olduğu gibi erişirler.

Çalışma esnasında genellikle hafıza erişimi sorunları ve segment hataları oluşur.Diğer derleyiciler

argüman ve parametreler arasındaki tip farklılıklarını yakalar ve hata mesajları üretirler.

7.5 Dizi değerleri haricindeki değerlere gösterici aritmetiği uygulamak.

7.6 Aynı diziyi göstermeyen iki göstericiyi çıkartmak ya da karşılaştırmak.

7.7 Gösterici aritmetiği kullanırken dizi sınırlarını aşmak

7.8 İkisinden biri void\* tipte olmadıkça, farlı tiplerdeki göstericileri birbirine atamak yazım

hatası oluşturur.

7.9 void\* göstericilerin gösterdiği nenelere erişmeye çalışmak.

7.10 Bir dizi ismini, gösterici aritmetiği kullanarak değiştirmeye çalışmak yazım

hatasıdır.

İYİ PROGRAMLAMA ALIŞTIRMALARI

7.1 Gösterici değişkenlerinin isimlerinde ptr harflerini kullanarak, bu değişkenlerin

gösterici olduklarını ve dikkatlice ele alınması gerektiklerini daha belirgin hale

getirmek.

7.2 Beklenmeyen sonuçlarla karşılaşmamak için göstericilere ilk değer atamak.

7.3 Bir fonksiyona argüman geçirirken eğer çağırıcı özel olarak çağırdığı fonksiyonunun

kendi ortamındaki bir argüman değişkeninin değerini değiştirmesine gerek duymuyorsa

değere göre çağırmayı kullanınız.Bu, en az yetki prensibinin başka bir örneğidir.

7.4 Bir fonksiyonu kullanmadan önce fonksiyonun kendisine geçirilen değerleri

değiştirip değiştiremeyeceğini anlamak için fonksiyon prototipini kontrol edin.

7.5 Dizileri ele alırken, göstericiler yerine dizi belirteçlerini kullanın.Bu, derleme

zamanını biraz arttırsa da bu sayede muhtemelen daha açık programlar yazmamızı

sağlatır.

PERFORMANS İPUÇLARI

7.1 Yapılar gibi büyük verileri, sabit bir veriyi gösteren göstericilerle geçirerek referansa göre

çağırmanın performansını ve değere göre çağırmanın güvenliğini kullanın.

298

7.2 Bir dizinin boyutunu fonksiyona geçirmek zaman alır ve boyutun bir kopyası

oluşturulacağından fazladan hafızaya gerek duyar. Ancak global değişkenlere, her fonksiyon

tarafından doğrudan ulaşılabildiğinden fazladan zaman ve hafızaya ihtiyaç duymazlar.

7.3 Dizi belirteç gösterimi, derleme esnasında göstericilerle ifade edildiğinden, dizi

belirteci kullanan deyimleri göstericilerle yazmak derleme zamanını azaltabilir.

7.4 Doğal bir biçimde ortaya çıkan algoritmalar, belirsiz erteleme gibi daha önceden

belirlenemeyen performans sorunları içerebilir.Belirsiz ertelemeden kaçınan algoritmaları

arayın.

TAŞINIRLIK İPUÇLARI

7.1 const ANSI C’de açıkça belirtilmesine rağmen bazı C sistemleri bunu uygulayamaz.

7.2 Bir veri tipini depolamak için gerekli olan byte sayısı sistemler arasında farlılık

gösterebilir.Veri tiplerinin büyüklüğüne bağlı programlar yazarken ve bu programlar

farklı sistemlerde çalışacaksa veri tiplerini depolamak için gerekli olan byte sayısına

sizeof kullanarak karar verin.

7.3 Günümüzdeki bilgisayarların çoğu 2 ya da 4 byte tamsayılara sahiptir.Bazı yeni

makineler 8 byte tamsayılar kullanmaktadır.Gösterici aritmetiği göstericinin gösterdiği

nesnelerin boyutuna bağlı olduğundan, gösterici aritmetiği makine bağımlıdır.

YAZILIM MÜHENDİSLİĞİ GÖZLEMLERİ

7.1 const belirteci en az yetki prensibini zorla uygulatmak için kullanılabilir.En az yetki

prensibini uygun yazılımlar tasarlamak için kullanmak, hata ayıklama zamanını ve

istenmeyen yan etkileri çok büyük oranda azaltır ve programı daha basit geliştirilebilir bir

hale getirir.

7.2 Eğer bir değer geçirildiği fonksiyonun gövdesi içinde değişmiyorsa(ya da

değişmemeliyse) bu değer const olarak bildirilerek hata ile değiştirilmemesi garanti altına

alınmalıdır.

7.3 Değere göre çağırma kullanıldığında çağrılan fonksiyon içinde yalnızca tek bir değer

değiştirilebilir.Bu değer, fonksiyonun geri dönüş değerinden atanmalıdır.Çağrılan

fonksiyon içinde birden çok değişkenin değerini değiştirmek için mutlaka referansa

göre çağırma kullanılmalıdır.

7.4 Fonksiyon prototiplerini diğer fonksiyon tanımlamaları içine yerleştirmek,uygun

fonksiyon çağrılarının yalnızca prototiplerin yer aldığı fonksiyonlar tarafından

yapılabilmesine kısıtlayarak en az yetki prensibini zorlar.

7.5 Bir diziyi fonksiyona geçirirken,dizinin boyutunu da geçirin.Bu fonksiyonu daha

genel bir hale getirir.Genel fonksiyonlar çoğu programda sıklıkla yeniden kullanılır.

7.6 Global değişkenler en az yetki prensibine uymazlar ve zayıf yazılım mühendisliğinin

bir örneğidirler.

ÇÖZÜMLÜ ALIŞTIRMALAR

7.1 Aşağıdakileri cevaplayınız.

a) Gösterici, başka bir değişkenin \_\_\_\_\_\_ değer olarak içeren değişkendir.

b) Bir göstericiye ilk değer olarak üç değer verilebilir. Bunlar \_\_\_\_\_\_, \_\_\_\_\_\_ ve

\_\_\_\_\_\_ dir.

c) Bir göstericiye atanabilecek tek tamsayı \_\_\_\_\_\_ dir.

299

7.2 Aşağıdakilerin hangilerinin doğru, hangilerinin yanlış olduğuna karar veriniz. Yanlış

olanların neden yanlış olduğunu açıklayınız.

a) & operatörü, sadece register depolama sınıfı olarak bildirilen sabitlere, deyimlere ya

da ifadelere atanabilir.

b) void olarak bildirilen göstericilerin gösterdiği nesnelere erişilebilir.

c) Değişik tipteki göstericiler, uygun tip dönüşümü yapılmadan birbirlerine atanamaz.

7.3 Aşağıdakileri cevaplayınız. Tek duyarlıklı, float tipinde sayıların dört byte uzunluğunda

saklandığını kabul ediniz. Bu dizinin hafızadaki başlangıç adresi 1002500 dir. Bu alıştırmanın

her sorusunda uygun olduğu sürece bir önceki sorunun cevabını kullanmak gerekli olabilir.

a) float tipinde, sayilar isminde 10 elemanı olan bir diziyi bildiriniz ve ilk değerlerini

0.0, 1.1, 2.2, ... 9.9 olarak veriniz. BOYUT sembolik sabitini 10 olarak tanımlayınız.

b) nPtr isminde bir göstericiyi float tipinde bir nesneyi gösterecek şekilde bildiriniz.

c) sayilar dizisini, dizinin belirteçlerinden yararlanarak ekrana yazdırınız. Bunun için,

daha önceden bildirilmiş olan ‘i’ kontrol değişkenini kullanan bir for döngüsü yazınız.

Her sayıyı, ondalık noktasının sağından bir pozisyon duyarlıkta yazdırınız.

d) sayilar dizisinin başlangıç adresini, nPtr göstericisine iki ayrı yoldan atayınız.

e) sayilar dizisinin elemanlarını, gösterici/offset gösterimi ve nPtr göstericisi

yardımıyla ekrana yazdırınız.

f) sayilar dizisinin elemanlarını, gösterici/offset gösterimi ve dizinin ismini gösterici

olarak kullanarak ekrana yazdırınız.

g) sayilar dizisini, nPtr göstericisini belirteçler ile kullanarak ekrana yazdırınız.

h) sayilar dizisinin belirteci 4 olan elemanını, dizi belirteci yöntemiyle belirtiniz.

Gösterici olarak dizinin ismini kullanan gösterici/offset yöntemiyle belirtiniz. nPtr

göstericisini belirteçle kullanarak belirtiniz ve nPtr ile gösterici/offset yöntemini

kullanarak belirtiniz.

i) nPtr göstericisinin, sayilar dizisinin başlangıç adresini gösterdiğini kabul ediniz. Bu

adreste hangi değer saklanmaktadır.

j) nPtr göstericisinin sayilar[5]’i gösterdiğini kabul ediniz. nPtr -= 4 hangi adresi

gösterir? Bu adreste saklanan değer nedir?

7.4 Aşağıdakileri gerçekleştirecek birer ifade yazınız. sayi1 ve sayi2 değişkenlerinin float

tipinde bildirildiğini ve sayi1 değişkeninin ilk değerinin 7.3 olarak atandığını kabul ediniz.

a) float tipinde bir nesne gösterecek şekilde fPtr değişkenini bildiriniz.

b) sayi1 değişkeninin adresini fPtr değişkenine atayınız.

c) fPtr değişkenin gösterdiği nesnenin içeriğini yazdırınız.

d) fPtr göstericisinin gösterdiği değeri sayi2 değişkenine atayınız.

e) sayi2 değerini ekrana yazdırınız.

f) %p dönüşüm belirtecini kullanarak sayi1 değişkeninin adresini yazdırınız.

g) %p dönüşüm belirtecini kullanarak fPtr değişkeninde saklanan adresi ekrana

yazdırınız. sayi1 ve sayi2 değişkenlerinin adresleri aynı mı?

7.5 Aşağıdakileri gerçekleştiren ifadeleri yazınız.

a) degistir fonksiyonu parametre olarak, float tipindeki x ve y değişkenlerini gösteren

göstericiler almaktadır.Bu fonksiyonun başlığını yazınız.

300

b) a şıkkındaki fonksiyonun prototipini yazınız.

c) hesapla adında, tamsayı döndüren ve x tamsayı parametresi ile poli fonksiyonunu

gösteren göstericiyi alan fonksiyonun başlığını yazınız. poli fonksiyonu, bir tamsayı

değişkeni alıp yine bir tamsayı döndürmektedir.

d) c şıkkındaki fonksiyonun prototipini yazınız.

7.6 Aşağıdaki program parçacıklarındaki yanlışlıkları bulunuz.

int zPtr; /\* zPtr, z dizisini gösterecektir. \*/

int \*aPtr = NULL;

void \*sPtr = NULL;

int sayi, i;

int z[5] = {1, 2, 3, 4, 5};

sPtr = z;

a) zPtr;

b) /\* dizinin ilk değerinin gösterici kullanılarak alınması \*/

sayi = zPtr;

c) /\* dizi elemanı 2’ yi( değer 3 ) sayi değişkenine ata\*/

number = \*zPtr[2];

d) /\* Bütün diziyi yazdır \*/

for( i = 0; i <= 5; i++)

printf(“%d “, zPtr[i]);

e) /\*sPtr nin gösterdiği değeri sayi değişkenine ata \*/

sayi = \*sPtr;

f) ++z;

ÇÖZÜMLER

7.1 a) adresini b) 0, NULL, bir adres c) 0

7.2

a) Yanlış. Adres operatörü sadece değişkenlere uygulanabilir, register depolama sınıfı

değişkenlerine uygulanamaz.

b) Yanlış. void tipte göstericilerin gösterdiği nesnelere erişilemez çünkü void tipte

göstericinin gösterdiği nesnenin tam olarak ne kadar hafıza kullandığı bilinemez.

c) Yanlış. void tipindeki göstericiler diğer tipteki göstericilere atanabilirler.

7.3

a) float sayilar[BOYUT] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};

b) float \* nPtr;

c) for (i = 0; i <= BOYUT – 1; i++)

printf (“%.1f “, sayilar[i]);

d) nPtr = sayilar;

nPtr = &sayilar[0];

e) for ( i = 0; i <= BOYUT – 1; i++)

printf 8”%.1f “, \*(nPtr + i));

f) for (i = 0; i <= BOYUT – 1; i++)

printf (“%.1f “, \*(sayilar + i));

g) for ( i = 0; i<= BOYUT – 1; i++)

printf (“&.1f “, nPtr[i]);

301

h) sayilar[4]

\*(sayilar + 4)

nPtr[4]

\*(nPtr + 4)

i) Adres: 1002500 + 8 \* 4 = 1002532. Değer: 8.8

j) sayilar[5] in adresi: 1002500 + 5 \* 4 = 1002520

nPtr -= 4 adresi 1002520 – 4 \* 4 = 1002504

O konumdaki değer 1.1

7.4

a) float \*fPtr;

b) fPtr = &sayi1;

c) printf(“\*fPtr nin değeri %f\n”, \*fPtr);

d) sayi2 = \*fPtr;

e) printf (“sayi2 değişkeninin değeri %f\n”, sayi2);

f) printf (“sayi1 değişkeninin adresi %p\n”, &sayi1);

g) printf (“fPtr değişkeninde saklanan adres %p\n”, fPtr);

Evet, değerler aynıdır.

7.5

a) void degistir(float \*x, floay \*y)

b) void degistir(float \*, float \*);

c) int hesapla(int x, int (\*poli) (int))

d) int hesapla(int, int (\*)(int));

7.6

a) Hata: zPtr değişkenin ilk değeri atanmamıştır.

Düzeltme: zPtr değişkenine zPtr = z; ifadesiyle ilk değer atanmalıdır.

b) Hata: Gösterici yanlış kullanılmıştır.

Düzeltme: İfadeyi sayi = \*zPtr şeklinde değiştirmek

c) Hata: zPtr[2] bir gösterici olmadığı için \* ile kullanılamaz.

Düzeltme: \*zPtr[2]’yi, zPtr[2] ile değiştirmek

d) Hata: Dizinin sınırları dışında kalan bir eleman yazdırılmak istenmiştir.

Düzeltme: for yapısındaki kontrol değişkeninin son değerinin 4 yapılması.

e) Hata: Bir void göstericinin gösterdiği nesneye ulaşılmaya çalışılmıştır.

Düzeltme: Göstericinin gösterdiği nesneye erişilebilmesi için, öncelikle tamsayı

değişkenine dönüştürülmesi lazımdır. sayi = \*(int \*) sPtr;

f) Hata: Dizi isminin gösterici aritmetiğiyle değiştirilmeye çalışılması.

Düzeltme: Dizi isminin yerine bir gösterici değişkeninin kullanılarak gösterici

aritmetiğinin uygulanması ya da istenilen dizi elemanı için dizi belirtecinin

kullanılması

ALIŞTIRMALAR

7.7 Aşağıdaki boşlukları doldurunuz.

a) \_\_\_\_\_ operatörü, operandının hafızada saklandığı yeri döndürür.

b) \_\_\_\_\_ operatörü ,operandının gösterdiği nesnenin içeriğini döndürür.

302

c) Dizi olmayan bir değişkeni, referansa göre çağırma tekniğine göre fonksiyona

yollarken değişkenin \_\_\_\_\_ geçirmelidir.

7.8 Aşağıdakilerden hangilerinin doğru, hangilerinin yanlış olduklarına karar veriniz. Yanlış

olanların neden yanlış olduklarını açıklayınız.

a) Farklı dizileri gösteren iki gösterici karşılaştırılamaz.

b) Bir dizinin ismi, o dizinin ilk elemanının göstericisi olduğu için, dizi isimleri gösterici

olarak kullanılabilir.

7.9 Aşağıdakileri cevaplarken unsigned int tipindeki sayıların 2 byte olarak saklandıklarını

ve dizinin başlangıç adresinin 1002500 olduğunu kabul ediniz.

a) degerler isminde, unsigned int tipinde, beş elemanı olan ve elemanlarının ilk

değerleri 2 ile 10 arasındaki çift sayılar olan diziyi bildirin ve BOYUT sembolik

sabitine 5 atayın.

b) vPtr isminde unsigned int tipinde bir gösterici bildiriniz.

c) degerler dizisinin elemanlarını dizi belirteç yöntemi ile ekrana yazdırınız. Daha

önceden bildirildiğini kabul edeceğiniz i değişkeni ile bir for yapısı kullanın.

d) degerler dizisinin başlangıç adresini vPtr göstericisine atayacak iki farklı ifade

yazınız.

e) degerler dizisinin elemanlarını gösterici/offset gösterimi ile ekrana yazdırınız.

f) degerler dizisinin elemanlarını gösterici/offset gösterimi ve gösterici olarak dizi

ismini kullanarak ekrana yazdırınız.

g) degerler dizisinin elemanlarını göstericiyle belirteç ile kullanarak ekrana yazdırınız.

h) degerler dizisindeki eleman 5’i, dizi belirteç yöntemiyle gösteriniz. Gösterici olarak

dizi ismi kullanan gösterici/offset gösterimiyle gösteriniz ve gösterici belirteci

gösterimi ile gösteriniz.

i) vPtr + 3 hangi adresi gösterir? Bu adreste saklanan değer nedir?

j) vPtr’nin degerler[4]’ü gösterdiğini kabul ediniz. vPtr -= 4 hangi adresi gösterir. Bu

adreste saklanan değer nedir?

7.10 Aşağıdakileri gerçekleştiren C ifadelerini yazınız. long integer tipinde deger1 ve deger2

isminde iki değişkenin bildirilmiş olduğunu ve deger1’ in ilk değerinin 200000 olduğunu

kabul ediniz.

a) lPtr yi long int tipinde bir nesneyi gösterecek şekilde bildiriniz.

b) deger1 değişkeninin adresini, lPtr göstericisine atayınız.

c) lPtr değişkeninin gösterdiği nesneyi yazdırınız.

d) lPtr değişkeninin gösterdiği nesneyi deger2 değişkenine atayınız.

e) deger2 değişkeninin içeriğini yazdırınız.

f) deger1 değişkeninin adresini yazdırınız.

g) lPtr değişkeninin içerdiği adresi yazdırınız.Yazdırılan adres deger1 değişkeninin

adresi ile aynı mı?

7.11 Aşağıdakileri gerçekleştirin.

a) sifir isminde ve long integer tipindeki buyukTamsayilar dizisini parametre olarak

alıp bir şey döndürmeyen fonksiyonun başlığını yazınız.

b) a şıkkındaki fonksiyonun prototipini yazınız.

303

0 1 2 3 4 5 6 7 8 9 10 11 12

0

1

2

3

c) birEkleVeTopla ismindeki fonksiyonun başlığını yazınız. Fonksiyon parametre

olarak bir birCokKucuk tamsayı dizisini almalı ve geriye bir şey döndürmemelidir.

d) c şıkkında yazdığınız fonksiyonun prototipini yazınız.

Not: Alıştırma 7.12’den Alıştırma 7.15’e kadar olan alıştırmalar oldukça uğraştırıcıdır. Bu

problemleri çözdüğünüzde en popüler kart oyunlarını kolayca programlayabileceksiniz.

7.12 Şekil 7.24 deki programı, fonksiyon 5 kartlık bir poker eli dağıtacak şekilde değiştiriniz.

Daha sonra aşağıdaki yeni fonksiyonları yazınız.

a) Elin bir çift içerip içermediğine karar versin.

b) Elin iki çift içerip içermediğini karar versin.

c) Elin, aynı takımdan üç kağıt içerip içermediğine karar versin(Örneğin 3 vale)

d) Elin, aynı takımdan dört kağıt içerip içermediğine karar versin(Örneğin 4 as)

7.13 Alıştırma 7.12’de yazdığınız fonksiyonlardan yararlanarak iki adet beş kartlık poker eli

dağıtan ve hangi elin daha iyi olduğuna karar veren bir program yazınız.

7.14 Alıştırma 7.13’deki programı değiştirerek, programın bir kart dağıtıcıyı canlandırmasını

sağlayınız. Bilgisayarın kartları beş kartta kapalı olacak şekilde dağıtılsın ve oyuncu bu

kartları göremesin. Daha sonra program bu eli değerlendirsin ve elin kalitesine göre 1, 2 ya da

3 kart isteyerek kart değişimi sağlansın ve eli tekrar değerlendirsin.(Dikkat: Bu zor bir

problemdir.)

7.15 Alıştırma 7.14’deki programı bilgisayarın kendi elini otomatik bir şekilde hazırlayacak

ve oyuncunun kendi eli için istediği kart değişikliklerini yapabileceği şekilde değiştiriniz.

Program iki elide değerlendirerek kimin kazandığına karar vermelidir. Şimdi hazırladığınız

programı kullanarak 20 oyun oynayın. Kim daha çok kazandı? Bu sonuçlara göre

programınızdaki gerekli değişiklikleri yapınız. Daha sonra 20 kez daha oynayın.

Değiştirdiğiniz program daha iyi oynadı mı?

7.16 Şekil 7.24’de kart karma ve dağıtma programında tanımsız bazı olasılıkları da içeren

(belirsiz erteleme) pek de güzel olmayan bir algoritma kullandık. Bu alıştırmada daha yüksek

performansta çalışan ve bahsettiğimiz tanımsız olasılıkları içermeyen bir program

yazacaksınız.

Şekil 7.24 deki programda şu değişiklikleri yapınız. kar fonksiyonunu dizide satır satır ve

sütun sütun döngü kuracak ve her elemanı ele alacak şekilde değiştiriniz. Her eleman, dizinin

rasgele seçilen bir elemanı ile yer değiştirmelidir.

Elde edilen diziyi, başarılı olunup olunmadığını anlayabilmek için ekrana yazdırınız. (şekil

7.30 da olduğu gibi). kar fonksiyonunun başarılı bir şekilde çalışması için bir kaç kez

çağırabilirsiniz.

Bu programdaki karma ve dağıtma algoritmasındaki yaklaşım, deste dizisinde kart1, kart2... aramaya

ihtiyaç duyar. Şekil 7.24 teki programı bir kart seçildikten sonra aynı kartın tekrar seçilmesini

engelleyecek şekilde değiştirin. 10.Ünitede her bir kart için tek bir operasyon gerektirecek bir

algoritma kullanacağız.

304

0 1 2 3 4 5 6 7 8 9 10 11 12

0

1

2

3

1 2 3 4 5 6 7 8 9 10 11 12 13

14 15 16 17 18 19 20 21 22 23 24 25 26

27 28 29 30 31 32 33 34 35 36 37 38 39

40 41 42 43 44 45 46 47 48 49 50 51 52

Şekil 7.29 Karılmamış deste dizisi

19 40 27 25 36 46 10 34 35 41 18 2 44

13 28 14 16 21 30 8 11 31 17 24 7 1

12 33 15 42 43 23 45 3 29 32 4 47 26

50 38 52 39 48 51 9 5 37 49 22 6 20

Şekil 7.30 Karılmış örnek bir deste dizisi

7.17 (Simulasyon: Kaplumbağa ve tavşan) Bu problemde kaplumbağa ve tavşanın klasik

yarışını canlandıracaksınız. Bunun için rasgele sayılar üreteceğiz.

Yarışmacılarımız, yarışmaya 70 karenin ilkinden başlayacaklardır. Her kare, yarış pistindeki

olası konumları temsil edecektir. Bitiş çizgisi kare 70’dedir. Kare 70’e ulaşan ilk yarışmacı

taze havuç ve marulla ödüllendirilecektir. Pist kaygan bir yapıda olduğu için yarışmacılar

konumlarını kaybedebilirler.

Her saniyede tıklayan bir saat kullanılmalı ve her tıklamada programınız aşağıdaki kurallara

göre yarışmacıların konumlarını bulmalıdır.

Hayvan Hareket tipi Zaman Yüzdesi Hareket

Kaplumbağa Hızlı yürümek %50 3 kare sağa

Kaymak %20 6 kare sola

Yavaş yürümek %30 1 kare sağa

Tavşan Uyku %20 Hareket yok

Büyük zıplama %20 9 kare sağa

Büyük kayma %10 12 kare sola

Küçük zıplama %30 1 kare sağa

Küçük kayma %20 2 kare sola

Hayvanların konumlarını saklamak için değişkenler kullanınız(Örneğin pozisyon numaraları

1-70) İki hayvanda pozisyon 1’den (başlangıç pozisyonu) yarışa başlasın. Eğer bir hayvan

kare 1’den sola kayarsa tekrar kare 1’e yerleştiriniz.

Tablodaki yüzdelerin oluşturulması için 1<= i <= 10 olmak üzere rasgele bir i tamsayısı

oluşturun. Kaplumbağanın hızlı yürümesi için 1 <= i <= 6, kayması için 6<=i <=7 ya da yavaş

yürümesi için 8 <= i <= 10 olmalıdır. Benzer bir tekniği de tavşan için uygulayınız.

Yarışa

305

CUVV !!!!

İŞTE BAŞLADILAR !!!

yazdırarak başlayınız.

Saatin her tıklamasında, (döngünün her tekrarında) 70 pozisyonluk çizgide K harfi kaplumbağayı, T

harfi ise tavşanı belirtsin. Yarışmacılar, yarış içersinde aynı pozisyonda bulunabilirler. Bu durumda

kaplumbağa tavşanı ısırır ve ekrana AH !!! yazdırılmalıdır. T, H ve AH !!! dışında kalan pozisyonlar

boşluk karakteriyle gösterilmelidir.

Her satır yazıldığında, herhangi bir hayvanın kare 70’e gelip gelmediğini kontrol ediniz. Eğer

bu kareye ulaşıldıysa kazananı yazdırınız. Eğer kaplumbağa kazanırsa, KAPLUMBAĞA

KAZANDI !!!, tavşan kazanırsa TAVŞAN KAZANDI !!! yazdırınız. Eğer her iki hayvanda

kazanırsa kaplumbağaya torpil yapabilirsiniz ya da berabere yazdırabilirsiniz. Eğer hiç biri

kazanamazsa döngüyü tekrar çalıştırınız. Programı çalıştırmadan önce bir kaç taraftar

bulursanız tezahüratlar sizi de şaşırtacaktır.

ÖZEL KISIM: KENDİ BİLGİSAYARINIZI OLUŞTURMA

Bundan sonraki bir kaç soruda yüksek seviyeli diller dünyasından ayrılıp bir bilgisayarın iç

yapısını inceleyeceğiz. Makine dili programlamayı tanıtacağız ve bir kaç makine dili

programı yazacağız. Bunun kabul edilebilir bir tecrübe olması için (simulasyon adı verilen

yazılım tekniğine dayanarak) ve programlarınızı çalıştırabilmeniz için bir bilgisayar

oluşturacağız.

7.18 (Makine dili Programlama) Simpletron adında bir bilgisayar yaratalım. Bilgisayarımız

basit ama oldukça güçlü bir bilgisayar. Simpletron sadece kendi anlayacağı dilde yazılan

programları çalıştırabilmektedir. Bu dil Simpletron makine dili ya da kısaca SMD’dir.

Simpletron bir akümülatör (Simpletron hesaplamalarda kullanacağı bilgiyi buraya koyar)

içermektedir. Simpletron word ile işlenir. Bir word, onluk sistemde dört basamaklı sayıdır,

örneğin +3364, -1293, +0007, -0001 vb. Simpletronun 100 word depolayabilecek bir hafızası

vardır ve tüm word’ler konum numaralarıyla ifade edilirler, 00, 01, ..., 99

Bir SMD programı çalıştırmadan önce, o programı hafızaya yüklemeliyiz. İlk komut(ya da

ifade) her zaman 00 konumundan başlar.

SMD’de yazılan her komut hafızada bir word yer kaplar. SMD komutlarının hepsinin artı

işaret ile başladığını düşünebiliriz ama bir verinin word’ü eksi ya da artı işaret ile başlayabilir.

Simpletron’un hafızasındaki her word, bir komutu yada bir program tarafından kullanılacak

veriyi ya da kullanılmamış (tanımlanmamış) bir hafıza alanını içerebilir. SMD komutunun ilk

iki basamağı hangi işlem yapılacağını gösteren işlem kodudur. Bu kodlar şekil 7.31’de

özetlenmiştir.

SMD komutunun son iki basamağı ise işlenecek bilginin hafıza adresini word olarak gösteren

operand’tır. Şimdi bir kaç SMD programını inceleyelim.

306

İşlem Kodu Anlamı

giriş/çıkış işlemleri:

#define OKU 10 Terminalden bir word bilgiyi oku ve hafızadaki

belli bir konuma yaz.

#define YAZ 11 Hafızadaki belli bir konumdan bir word bilgiyi,

terminale yaz.

yükle/sakla işlemleri

#define YUKLE 20 Akümülatörü, belli bir hafıza konumundan

alınan bir word bilgi ile yükle

#define SAKLA 21 Akümülatörden bir word bilgi al ve hafızanın

belli bir konumunda sakla.

Aritmetik işlemler

#define TOPLA 30 Akümülatördeki bir word bilgi ile belli bir

hafıza konumundaki bir word bilgiyi topla.

(sonucu akümülatörde sakla)

#define CIKART 31 Akümülatördeki bir word bilgiden belli bir

hafıza konumundaki bir word bilgiyi çıkart

(sonucu akümülatörde sakla)

#define BOL 32 Akümülatördeki bir word bilgiyi belli bir

hafıza konumundaki bir word bilgiye böl.

(sonucu akümülatörde sakla)

#define CARP 33 Akümülatördeki bir word bilgiyle belli bir

hafıza konumundaki bir word bilgiyi çarp.

(sonucu akümülatörde sakla)

kontrol işlemlerinin transferi

#define DALLAN 40 Hafızadaki belli bir konuma dallan

#define DALLANNEG 41 Akümülatör negatif ise hafızadaki belli bir

konuma dallan.

#define DALLANSIFIR 42 Akümülatör sıfır ise hafızadaki belli bir

konuma dallan.

#define BITIR 43 Program görevini tamamladı.

Şekil 7.31 Simpletron Makine Dili (SMD) işletim kodları

Örnek 1

Konum Sayı Komut

00 +1007 (Oku A)

01 +1008 (Oku B)

02 +2007 (Yukle A)

03 +3008 (Topla B)

04 +2109 (Sakla C)

05 +1109 (Yaz C)

06 +4300 (BITIR)

07 +0000 (A degisken)

08 +0000 (B degisken)

307

09 +0000 (Sonuc C)

Bu SMD programı klavyeden iki sayı okur ve bunların toplamlarını ekrana yazdırır. +1007 komutu

klavyeden ilk sayıyı okur ve 07(ilk değeri sıfırdır) konumuna yazar. Sonra +1008 diğer sayıyı 08

hafıza konumuna okur. +2007, yükle komutu ilk sayıyı akümülatöre alır ve +3008, ekle komutu ikinci

sayıyı akümülatördeki ilk sayı ile toplar. Bütün SMD aritmetik işlemlerin komutları, sonuçları

akümülatörde saklarlar. +2109, sakla komutu ile sonucu 09 hafıza konumunda saklar ve +1109 yaz

komutu ile sonuç ekrana yazdırılır (işaretli dört basamaklı onluk sistemde bir sayı olarak). +4300,

bitir komutu ile program sonlanır.

Örnek 2

Konum Sayı Komut

00 +1009 (Oku A)

01 +1010 (Oku B)

02 +2009 (Yukle A)

03 +3110 (Cikart B)

04 +4107 (Negatifse 07’ye dallan)

05 +1109 (Yaz A)

06 +4300 (BITIR)

07 +1100 (Yaz B)

08 +4300 (BITIR)

09 +0000 (A degisken)

09 +0000 (B degisken)

Bu SMD programı klavyeden iki sayı okur ve büyük olanını bulup ekrana yazdırır. +4107

komutu C’deki if gibi bir kontrolün koşullu transferidir. Şimdi aşağıdaki görevleri yapacak

SMD programlarını yazınız.

a) Bir nöbetçi kontrollü döngü ile 10 pozitif sayıyı okuyunuz ve bu sayıların

toplamını yazdırınız.

b) Sayıcı kontrollü bir döngü kullanarak yedi sayıyı okuyunuz ve bazısı pozitif, bazısı

negatif olan bu sayıların ortalamasını buldurunuz.

c) Bir kaç sayıyı okuyan ve en büyüğüne karar veren bir program yazınız. İlk okunan

sayı kaç sayı okunacağıdır.

7.19 (Bir bilgisayar simülasyonu) Biraz insafsız bir soru gibi gözükse de, bu problemde kendi

bilgisayarınızı oluşturacaksınız. Tabi ki, parçaları birleştirerek değil. Bunun için çok güçlü bir

teknik olan yazılıma dayalı simülasyon kullanarak, bir Simpletronun yazılım modelini

oluşturacağız. Simpletron simülasyonu bilgisayarınızı bir Simpletron’a dönüştürecektir ve

Alıştırma 7.18’de yazdığınız programları derleme, test etme ve hata ayıklama imkanınız

olacaktır.

Simpletron simülasyonunu çalıştığında aşağıdakileri ekrana yazdırarak başlamalıdır.

\*\*\* Simpletrona hoş geldiniz ! \*\*\*

\*\*\* Lütfen her seferinde bir komut ya da \*\*\*

\*\*\* (bir word veri) giriniz. Hafıza konumunu \*\*\*

\*\*\* ve soru işaretini(?) ekrana yazdıracağım \*\*\*

\*\*\* Siz daha sonra o konum için word’ü girin \*\*\*

\*\*\* Programınızı girmeyi sonlandırmak için –99999 \*\*\*

\*\*\* nöbetçi değerini giriniz.\*\*\*

308

Simpletronun hafızasını tek belirteçli, 100 elemanlı hafiza dizisi ile belirtiniz. Şimdi

simülasyonun çalıştığını farz edin. Alıştırma 7.18’ in 2. örneğini aşağıdaki gibi gireriz.

00 ? +1009

01 ? +1010

02 ? +2009

03 ? +3110

04 ? +4107

05 ? +1109

06 ? +4300

07 ? +1110

08 ? +4300

09 ? +0000

10 ? +0000

11 ? -99999

\*\*\* Programın yüklenmesi bitti \*\*\*

\*\*\* Program çalıştırılıyor \*\*\*

SMD programı hafiza dizisine yüklenmiştir. Şimdi Simpletron programınızı çalıştırır.

Programın çalışması 00 konumundaki komutla başlar ve C’de olduğu gibi, kontrol transferi

denk gelmediği sürece sırayla aşağıya doğru çalışır.

akumulator değişkenini, akümülatörü temsil edecek şekilde kullanınız.

komutSayici değişkeni ise çalıştırılacak komutun hafızadaki konumunu içermelidir.

isletimKodu değişkeni ise o anda yapılan işlemi göstermelidir(Komut word’ünün sol iki

basamağı).

operand değişkeni ise o anda işlenen komutun hangi hafıza konumuyla ilgili işlem

yapacağını göstermelidir. operand, komut word’ün sağdan ilk iki basamağıdır. Komutları

doğrudan hafızadan çalıştırmayınız. Çalıştırılacak bir sonraki komutu, hafızadan

komutRegister adı verilen bir değişkene atınız ve en soldaki iki basamağı alıp isletimKodu

değişkenine, en sağdaki iki basamağı alıp operand değişkenine yazınız.

Simpletron çalışmaya başladığında özel değişkenlerin ilk değerleri aşağıdaki gibi olmalıdır.

akümülatör +0000

komutSayici 00

komutRegister +0000

isletimKodu 00

operand 00

Şimdi, ilk SMD komutunun(00 konumundaki +1009) çalışmasını inceleyelim. Buna komut

işleme devri denir.

isletimKodu bize çalıştırılacak bir sonraki komutun hafıza konumunu gösterir. O hafiza

konumunun içeriğini aşağıdaki ifadeyle çekeriz.

komutRegister = hafiza[komutSayici];

işletim kodu ve operand, komutRegister’dan aşağıdaki gibi elde edilir.

309

isletimKodu = komutRegister / 100;

operand = komutRegister % 100;

Şimdi, Simpletron bu işletim kodunun hangi komuta karşılık geldiğini bulmalıdır. Bir switch

yapısı ile Simpletronun on iki işlemi bir birinden ayrılabilir.

switch yapısında bazı SMD komutları aşağıdaki gibi gösterilmiştir(Diğerlerini okuyucuya

bıraktık) :

oku: scanf(“%d”, &hafiza[operand]);

yukle: akumulator = hafiza[operand];

topla: akumulator += hafiza[operand];

Çeşitli dallanma komutları: Bunlardan kısaca bahsedeceğiz.

bitir: Bu komut

\*\*\* Simpletronun çalışması durduruldu \*\*\*

mesajını ve daha sonra bütün registerların ismini ve içeriğini yazdırmalıdır. Böyle bir çıktıya

bilgisayar çöplüğü (bilgisayar çöplüğü, eski bilgisayarların gittiği bir yer değildir) denir.

Bu çöplük çıktısı için örnek olabilecek bir çıktı Şekil 7.32’te gösterilmiştir. Bir Simpletron

programı çalıştırıldıktan sonra çöplük çıktısı, çıkıştan hemen sonraki komut değerlerini ve

verileri içerir.

Programımızın ilk komutu çalıştırmasını ele alalım. Bu komut 00 hafıza konumundaki +1009

komutudur. Daha öncede belirttiğimiz gibi, switch ifadesi aşağıdaki C ifadesiyle bunu

gerçekleştirir:

scanf(“%d”, &hafiza[operand]);

scanf ile kullanıcı bir giriş yapmadan önce, ekrana bir soru işareti (?) yazdırılarak

Simpletronun kullanıcının bir değer girmesini beklediği belirtilir. Kullanıcının girdiği değer

09 konumuna okunur.

Bu noktada, ilk komutun çalıştırılması bitmiştir. Artık Simpletron bir sonraki komut için

hazırlanmalıdır. Biraz önce işlenen komut bir kontrol transferi olmadığı için komutSayici bir

artırılmalıdır.

++komutSayici;

Bu ifadeyle bir önceki komutun işlenmesi tamamıyla biter. Tüm bu işlemler(komut işleme

devri) yeni bir komutun çalıştırılmasıyla tekrar yapılır.

Şimdi, dallanma komutlarını –kontrol transferi—inceleyeceğiz. İhtiyacımız olan tek şey,

komut sayıcıyı doğru olarak ayarlamaktır. (40) koşulsuz dallanma komutu switch tarafından

aşağıdaki şekilde gerçekleştirilir.

komutSayici = operand;

REGISTERLAR

akümülatör +0000

komutSayici 00

310

komuRegister +0000

isletimKodu 00

operand 00

HAFIZA:

0 1 2 3 4 5 6 7 8 9

0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

10 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

20 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

30 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

40 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

50 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

60 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

70 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

80 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

90 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

“Eğer akümülatör sıfırsa dallan” koşullu ifadesi aşağıdaki gibi gerçekleştirilir.

if (akumulator == 0)

komutSayici = operand;

Bu noktada artık Alıştırma 7.18’de yazdığınız bütün örnekleri çalıştırabilirsiniz. SMD’i yeni

özellikler ekleyerek süsleyebilir ve bunları simülasyonunuzda da içerebilirsiniz.

Simülasyonunuz çeşitli hataları da kontrol edebilir. Örneğin, program yükleme sürecinde

kullanıcının Simpletronun hafızasına girdiği her sayı –9999 ve +9999 arasında olmalıdır.

Programınız bir while döngüsü içerisinde girilen her değeri kontrol edebilir ve istenilen

aralıkta bir sayı girilmediğinde kullanıcının bu sayıyı tekrar girmesi sağlanabilir.

Programın çalışma süreci içerisinde, simülasyonunuz çeşitli ciddi hataları kontrol etmelidir.

Örneğin sıfıra bölme, yanlış işletim kodlarını girilmesi yada akümülatörün taşması gibi(+9999

ve –9999 aralığı dışında aritmetik işlemlerin yapılması). Bu ciddi hatalara ölümcül hatalar

denir. Ölümcül bir hata oluştuğunda programınız aşağıdaki gibi bir mesajı ekrana

yazdırmalıdır:

\*\*\* Sıfıra bölmeye çalışıldı \*\*\*

\*\*\* Simpletronun programı çalıştırması anormal sonlandı \*\*\*

Daha sonra, bir çöplük çıktısı ekrana yazdırılmalıdır. Bu kullanıcıya hatayı nerde yaptığını

anlamasında yardımcı olur.

7.20 Şekil 7.24’deki programı, kart karma ve dağıtma işlemlerinin tek bir fonksiyonda

gerçekleştirilebileceği şekilde değiştiriniz ( karVeDagit ) . Fonksiyonunuz Şekil 7.24’deki

kar fonksiyonunda olduğu gibi yuvalı bir döngü yapısı içermelidir.

311

7.21 Aşağıdaki program ne yapar? 1 /\* ex07\_21.c \*/ 2 #include <stdio.h> 34 void gizem1( char \*, const char \* ); 56 int main( ) 7 { 8 char string1[ 80 ], string2[ 80 ]; 9

10 printf( "İki string giriniz: " );

11 scanf( "%s%s" , string1, string2 );

12 gizem1( string1, string2 );

13 printf(" %s", string1 );

14

15 return 0;

16

}

17

18 void gizem1( char \*s1, const char \*s2 )

19

{

20 while ( \*s1 != '

\0' )

21 ++s1;

22

23 for ( ; \*s1 = \*s2; s1++, s2++ )

24 ; /\* boş ifade \*/

25

}

7.22 Aşağıdaki program ne yapar? 1 /\* ex07\_22.c \*/ 2 #include <stdio.h> 34 int gizem2( const char \* ); 56 int main( ) 7 { 8 char string[ 80 ]; 9

10 printf( " Bir string giriniz: ");

11 scanf( "%s", string );

12 printf( " %d

\n", gizem2( string ) );

13

14 return 0;

15

}

16

17 int gizem2( const char \*s )

18

{

19 int x;

20

21 for ( x = 0; \*s != '

\0'; s++ )

312

22 ++x;

23

24 return x;

25 }

7.23 Aşağıdaki program parçacıklarındaki hataları bulunuz ve bu hatalar düzeltilebilirlerse

nasıl düzeltileceklerini açıklayınız.

a) int \*sayi;

printf(“%d”\n”, \*sayi);

b) float \*gercekPtr;

long \*tamsayiPtr;

tamsayiPtr = gercekPtr;

c) int \* x, y;

x = y;

d) char s [ ] = “bu bir karakter dizisidir”

int sayici;

for ( ; \*s != ‘\0’; s++)

printf (“%c “, \*s);

e) short \*sayiPtr, sonuc;

void \*jenerikPtr = sayi;

sonuc = \*jenerikPtr + 7;

f) float x = 19.34;

float xPtr = &x;

printf (“%f\n”, xPtr);

g) char \*s;

printf (“%s\n”, s);

7.24 (Hızlı Sıralama) 6.Ünitenin örnek ve alıştırmalarında, kabarcık sıralama, kova ve seçim

sıralamalarını inceledik. Şimdi ise yinelemeli sıralamayı yani hızlı sıralamayı inceleyeceğiz.

Tek belirteçli bir dizi için algoritma aşağıdaki gibidir.

1) Yerleştirme Basamağı: Sıralanmamış dizideki ilk elemanı alın ve sıralı dizideki olması

gereken yere karar verin. Bu durum, elemanın solundaki bütün değerler bu elemandan

küçükse, sağındakiler ise elemandan büyükse gerçekleşir. Böylece artık yerinde olan

bir sayımız ve iki adet sıralanmamış alt dizimiz vardır.

2) Yineleme Basamağı: İlk basamağı, sıralanmamış her basamağa uygulayınız.

İlk basamak bir alt diziye uygulandığında yeni bir eleman doğru pozisyona gelirken iki adet

sıralanmamış sizi oluşur. Bir alt dizi tek eleman içerdiğinde mutlaka sıralanmış ve olması

gereken pozisyona gelmiş olmalıdır.

Algoritma yeterince basit görünüyor ama her alt dizinin ilk elemanının pozisyonuna nasıl

karar verilecek? Örnek olarak aşağıdaki değer kümesini ele alın. (Kalın yazılan değer,

yerleştirilecek değerdir ve sıralanmış dizide son pozisyona gelcektir.)

37 2 6 4 89 8 10 12 68 45

1) Dizinin en sağ elemanından başlayarak her elemanı 37 ile karşılaştırın. 37 den

daha küçüğünü bulana kadar devam edin. 37 den küçük ilk eleman 12 dir. 37 ile 12

nin yerlerini değiştirin. Yeni dizi

313

12 2 6 4 89 8 10 37 68 45

12 elemanı 37 ile henüz yer değiştiği için yatık yazılmıştır.

2) Dizinin en solundaki ama 12’den bir sonra gelen elemanı tekrar 37 ile

karşılaştırın.37 den daha büyük bir eleman bulanana kadar devam edin. 37 den

büyük ilk eleman 89 dur. Böylece 37 ve 89 da yer değiştirin. Yeni dizi:

12 2 6 4 37 8 10 89 68 45

3) Sağdan, 89’dan bir önceki eleman 37 ile karşılaştırın. 37’ den daha küçük bir

eleman bulunduğunda yer değişirler. 37 den küçük ilk eleman 10 dur. 37 ve 10 yer

değişirler. Yeni dizi:

12 2 6 4 10 8 37 89 68 45

4) Soldan, 10 dan hemen sonraki eleman 37 ile karşılaştırın. 37 den daha büyük bir

eleman arayın. 37 den daha büyük bir eleman olmadığı için 37’ nin doğru yere

geldiği anlaşılır.

Bu işlem sonucunda iki adet sırasız dizi oluşur. 37 den küçük elemanların oluşturduğu bir

dizi: 12, 2, 6, 4, 10, ve 37den büyük elemanların oluşturduğu bir dizi: 89, 68, 45. Sıralama

aynı mantıkla bu alt dizilere de uygulanır.

Yukarıdaki anlatılanlara göre hizliSiralama isminde bir yineleme fonksiyonu yazınız.

Fonksiyon argüman olarak bir tamsayı dizisi, bir başlangıç belirteci ve bir bitiş belirteci

almalı. yerlestir fonksiyonu hizliSiralama tarafından çağırılmalıdır.

7.25 (Labirent) Aşağıdaki diyezler(#) ve noktardan(.) oluşan iki boyutlu dizi bir labirenti

göstermektedir.

# # # # # # # # # # # #

# . . . # . . . . . . #

. . # . # . # # # # . #

# # # . # . . . . # . #

# . . . . # # # # . # .

# # # # . # . # . # . #

# . . # . # . # . # . #

# # . # . # . # . # . #

# . . . . . . . . # . #

# # # # # # . # # # . #

# . . . . . . . . . . #

# # # # # # # # # # # #

Diyezler (#) labirentin duvarları, noktalar(.) ise olası yol kareleridir.

Labirentten çıkışı bulmanın çok kolay bir algoritması vardır (Bir çıkış olduğunu kabul ediniz.)

Eğer bir çıkış yoksa başladığımız noktaya ger dönersiniz. Labirentin içinde sağ elinizi

sağınızdaki duvara koyun ve yürüyün. Elinizi asla duvardan çekmeyin. Eğer duvar sağa

dönerse, sizde dönün. Sonunda çıkışa ulaşacaksınız. Daha kısa bir yol olabilir ama bu şekilde

kesinlikle çıkışa ulaşacaksınız.

314

labirent isminde bir yineleme fonksiyonu yazınız. Fonksiyon argüman olarak labirenti temsil

eden 12’ye 12 karakterler dizisi ve labirentin başlangıcını noktasını almalıdır.

Ekrana, bu labirenti yazdırınız ve labirentte o anda bulunulan noktayı x ile gösteriniz ki

kullanıcı labirentin nasıl çözüldüğünü görebilsin.

7.26 (Rasgele Labirent Oluşturma) labirentYap isminde, argüman olarak 12’ye 12’lik iki

boyutlu karakter dizisini alan ve rasgele bir labirent oluşturan bir fonksiyon yazınız.

Fonksiyon tabi ki labirentin başlangıç ve bitiş konumlarının adreslerini içermeli. Alıştırma

7.25’de yazdığınız fonksiyonu bu labirentleri çözmede kullanınız.

7.27 (Farklı Boyutlarda Labirentler) Alıştırma 7.25 ve 7.26’da yazdığınız labirent ve

labirentYap fonksiyonlarını farklı en ve boylarda labirentler oluşturacak şekilde

genelleştiriniz.

7.28 Şekil 6.22’deki programı aşağıdaki menüdeki dört opsiyonu sunacak şekilde tekrar

yazınız.

Seçiminizi giriniz:

0 notlar dizisini yazdır

1 En düşük notu bul

2 En büyük notu bul

3 Tüm derslerdeki ortalamayı her öğrenci için bul

4 Programı sonlandır

Gösterici dizilerini fonksiyonlarda kullanırken karşımıza çıkan bir sınırlama bu göstericilerin

hepsinin aynı tipe sahip olması gerektiğidir. Fonksiyonunun argüman olarak aldığı ve

döndürdüğü göstericiler aynı tipte olmalıdır. Bu yüzden şekil 6.22’deki program,

fonksiyonların aynı tip parametre alacağı ve aynı tip döndüreceği şekilde değiştirilmelidir.

minimum ve maksimum fonksiyonlarını en küçük ve en büyük değerleri bulacak ama hiçbir

şey döndürmeyecek şekilde değiştiriniz. 3. seçenekte ise, Şekil 6.22’deki ortalama

fonksiyonunu her öğrencinin ( belli bir öğrenci değil ) ortalamasını bulacak şekilde

değiştiriniz. Bu fonksiyon bir şey döndürmemeli ve diziyiYazdir, minimum ve maksimum

parametrelerini almalıdır. Bu dört fonksiyona gönderilen göstericileri, notIslemleri dizisinde

saklayınız ve kullanıcının yaptığı seçimi dizi belirteci olarak kullanınız.

7.29 (Simpletron Simülasyonunda Değişiklikler) Alıştırma 7.19’da Simpletron Makine

Dilinde ( SMD ) yazılan programları çalıştıran bir yazılım simülasyonu yaptınız. Bu

alıştırmada Simpletron simülasyonunda değişiklikler yaparak simülasyonu geliştireceğiz.

Alıştırma 12.26 ve 12.27’de yükse seviyeli bir dilde ( bir tür BASIC ) yazılmış programları

Simpletron Makine Diline çevirecek bir derleyici yazacağız. Derleyici tarafından bazı

programların derlenebilmesi için aşağıdaki değişiklerin yapılması gerekmektedir.

315

a) Simpletronun daha geniş programları işleyebilmesi için hafızasını 1000 hafıza konumu

içerecek şekilde değiştiriniz.

b) Bir Simpletron Makine dili komutu daha ekleyerek simülasyonun mod işlemleri

yapabilmesini de sağlayınız.

c) Bir Simpletron Makine dili komutu daha ekleyerek simülasyonun üssel işlemleri

yapabilmesini de sağlayınız.

d) Simpletronun makine dilindeki komutları ifade etmek için tamsayılar yerine, on altılık

sistemde sayılar kullanılacak şekilde değişiklikler yapın.

e) Simülasyon programınızın yeni bir satır çıktısı verebilmesini sağlayın. Bunun için

Simpletron Makine diline bir komut daha eklenmelidir.

f) Simülasyon programınızın tamsayılara ek olarak ondalıklı sayılarla da işlem

yapabilmesini sağlayınız.

g) Simülasyon programınızı string işleyebilecek bir şekilde değiştiriniz. İpucu: Her

Simpletron wordü, iki basamaklı tamsayı içeren iki gruba ayrılabilir. İki basamaklı her

tamsayı bir karakterin ASCII olarak onluk sistemde eşitidir. Bir string girişi yaptıracak

ve bunu hafızanın belli bir konumunda saklayacak bir makine dili komutu ekleyiniz. O

konumdaki wordün ilk yarısı, stringin karakter sayıcısı (stringin uzunluğu) olacaktir.

Birbirini takip eden her yarım word, bir ASCII karakterinin, iki onluk basamakta

gösterilmesidir. Makine dili komutu, her karakteri ASCII eşitine çevirir ve bunu yarım

worde atar.

h) g şıkkındaki string çıktılarını işleyebilecek şekilde değiştiriniz. İpucu: Belli bir hafıza

konumundan başlayan bir stringi yazdıran bir makine dili komutu ekleyiniz. O

konumdaki ilk yarım word stringin karakter sayısını(stringin uzunluğunu)

göstermektedir. Birbirini takip eden her yarım word, bir ASCII karakterinin, iki onluk

basamakta gösterilmesidir. Makine dili komutu, stringin uzunluğunu kontrol etmeli ve

stringi, iki basamklı her sayıyı karakter eşitine çevirerek yazdırır.

7.30 Aşağıdaki program ne yapar?

1 /\* ex07\_30.c \*/

2 #include <stdio.h>

3

4 int gizem3( const char \*, const char \* );

5

6 int main()

7 {

8 char string1[ 80 ], string2[ 80 ];

9

10 printf( "İki string giriniz: " );

11 scanf( "%s%s", string1 , string2 );

12 printf( " Sonuç: %d\n",

13 gizem3( string1, string2 ) );

14

15 return 0;

16 }

17

18 int gizem3( const char \*s1, const char \*s2 )

19 {

20 for ( ; \*s1 != '\0' && \*s2 != '\0'; s1++, s2++ )

21

316

22 if ( \*s1 != \*s2 )

23 return 0;

24

25 return 1;

26 }

317

KARAKTER VE STRINGLER

AMAÇLAR

 Karakter kütüphanesindeki (ctype) fonksiyonları kullanabilmek

 Standart giriş/çıkış kütüphanesindeki ( stdio ) string ve karakter giriş/çıkış

fonksiyonlarını kullanabilmek

 Genel amaçlı kütüphanedeki (stdlib) string dönüşüm fonksiyonlarını kullanabilmek

 String kütüphanesindeki (string) string işleme fonksiyonlarını kullanabilmek

 Yazılımın yeniden kullanılabilirliğinde fonksiyon kütüphanelerinin gücünü anlamak

BAŞLIKLAR

8.1 GİRİŞ

8.2 STRING VE KARAKTERLERİN TEMELLERİ

8.3 KARAKTER KÜTÜPHANESİ

8.4 STRING DÖNÜŞÜM FONKSİYONLARI

8.5 STANDART GİRİŞ/ÇIKIŞ KÜTÜPHANE FONKSİYONLARI

8.6 STRING KÜTÜPHANESİNDEKİ STRING İŞLEME FONKSİYONLARI

8.7STRING KÜTÜPHANESİNDEKİ STRING KARŞILAŞTIRMA FONKSİYONLARI

8.8 STRING KÜTÜPHANESİNDEKİ ARAMA FONKSİYONLARI

8.9 STRING KÜTÜPHANESİNDEKİ HAFIZA FONKSİYONLARI

8.10 STRING KÜTÜPHANESİNDEKİ DİĞER FONKSİYONLAR

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Taşınırlık İpuçları\*

Çözümlü Alıştırmalar\*Cevaplar\*Alıştırmalar

8.1 GİRİŞ

Bu ünitede, string ve karakter işlemeyi kolaylaştıran C standart kütüphane fonksiyonlarını

tanıtacağız. Bu fonksiyonlar programların karakterleri, stringleri, metin satırlarını ve hafıza

bloklarını işleyebilmelerini sağlatır.

Bu ünitede editör, kelime işlemci, sayfa çizimi, bilgisayarlı daktilo sistemleri ve diğer metin

işleme yazılımlarını geliştirmek için kullanılan teknikler anlatılmaktadır. printf ve scanf gibi

formatlı giriş/çıkış fonksiyonları tarafından gerçekleştirilen metin işlemleri, bu ünitede

anlatılan fonksiyonlar tarafından yapılabilir.

8.2 STRING VE KARAKTERLERİN TEMELLERİ

Karakterler, kaynak programları oluşturan blokların temelidir. Her program, anlamlı olarak

gruplandığında, bilgisayar tarafından yerine getirilecek komutlar olarak algılanan karakter

dizilerinden oluşur. Bir program, karakter sabitleri içerebilir. Bir karakter sabiti, tek tırnak

içinde gösterilen bir int değeridir. Karakter sabitinin değeri, karakterin makinenin karakter

seti içindeki tamsayı değeridir. Örneğin, ‘z’, z’nin tamsayı değerini ve ‘\n’, yeni satırın

tamsayı değerini göstermektedir. Bir string, tek bir birim olarak ele alınan karakter serileridir.

318

Bir string harfler, rakamlar ve +, -, \*, / ve $ gibi özel karakterler içerebilir. String bilgileri ya

da string sabitleri, C’de çift tırnak içinde aşağıda gösterildiği biçimde yazılır:

“John Q. Doe” (bir isim)

“99 Main Street” (bir cadde ismi)

“Waltham,Massachusetts” (bir şehir ve eyalet)

“(201) 555-1212” (bir telefon numarası)

C’de bir string, null karakterle ( ‘\0’ ) sonlanan bir karakter dizisidir. Bir stringe, stringin ilk

karakterini gösteren bir gösterici ile erişilir. Bir stringin değeri, ilk karakterinin adresidir. Bu

sebepten, C’de bir string, bir göstericidir demek uygundur. Gerçekte bir string, stringin ilk

karakterini gösteren bir göstericidir. Bu anlamda, stringler dizilere benzemektedir, çünkü bir

dizi de ilk elemanını gösteren bir göstericidir.

Bir string, bildirimlerde bir karakter dizisine ya da char\* tipinde bir değişkene atanabilir.

char renk[ ]=”mavi”;

const char\* renkPtr =”mavi”;

bildirimlerinin ikisi de “mavi” stringini bir değişkene atamaktadır. İlk bildirim 5 elemanlı

renk dizisini yaratır. Bu dizinin elemanları ‘m’,’a’,’v’,’i’ ve ’\0’ karakterleridir. İkinci

bildirim, hafızada bir yerlerde bulunan “mavi” stringini gösteren bir gösterici değişkeni olan

renkPtr’yi yaratır.

Taşınırlık İpuçları 8.1

char \* tipinde bir değişkene string bilgisi atandığında bazı derleyiciler stringi, hafızada stringin

değiştirilemeyeceği bir konuma yerleştirir. Eğer string bilgisini değiştirmeye ihtiyaç duyacaksanız,

stringin tüm sistemlerde değiştirilebilmesini garanti altına almak için, stringi karakter dizisine

yerleştirin.

Az önceki dizi bildirimi

char renk[ ]={‘m’,’a’,’v’,’i’,’\0’};

biçiminde de yazılabilir.

Bir karakter dizisini string içerecek biçimde bildirirken, dizi, stringi ve stringi sonlandıran

null karakteri tutabilecek kadar geniş olmalıdır. Az önceki bildirim dizinin boyutuna, atama

listesindeki atama değerlerinin sayısına dayanarak, otomatik olarak karar vermektedir.

Genel Programlama Hataları 8.1

Bir karakter dizisinde, stringi sonlandıran null karakteri depolamak için gerekli

alanı ayırmamak

Genel Programlama Hataları 8.2

Sonlandırıcı null karakteri içermeyen bir stringi yazdırmak

319

İyi Programlama Alıştırmaları 8.1

Karakter stringlerini bir karakter dizisinde tutarken, dizinin depolanacak en büyük stringi

tutabilecek kadar geniş olduğundan emin olun. C, her uzunluktaki stringlerin

depolanmalarına izin verir. Eğer bir string depolanacağı karakter dizisinden daha uzunsa,

diziden sonraki karakterler, hafızada bir sonraki dizide yer alan verilerin üzerine

yazılacaktır.

Bir string, scanf kullanılarak diziye atanabilir. Örneğin, aşağıdaki ifade bir stringi, bir

karakter dizisi olan kelime[20] dizisine atamaktadır:

scanf(“%s”,kelime);

Kullanıcı tarafından girilen string, kelime içinde tutulur. ( kelime’nin bir dizi olduğuna, yani

bir gösterici olduğuna, bu sebepten de kelime argümanıyla & kullanılmadığına dikkat ediniz)

scanf fonksiyonu karakterleri boşluk, yeni satır ya da dosya sonu belirteciyle karşılaşıncaya

dek okur. Stringin, sonlandırıcı null karaktere yer bırakmak için 19 karakterden daha uzun

olmaması gerektiğine dikkat ediniz. Bir karakter dizisinin yazdırılabilmesi için, dizi bir

sonlandırıcı null karakter içermelidir.

Genel Programlama Hataları 8.3

Tek bir karakteri string olarak işlemek. Bir string bir

göstericidir.(muhtemelen oldukça büyük bir tamsayıdır) Bir karakter ise

küçük bir tamsayıdır.(ASCII değerler 0-255 arsındadır) Bir çok sistemde bu,

bir hataya yol açar çünkü düşük hafıza adresleri, işletim sisteminin kesme

istemleri gibi özel amaçlar için ayrılmıştır. Bu sebepten, erişim kısıtlamaları

oluşabilir.

Genel Programlama Hataları 8.4

Bir string beklenirken, fonksiyona argüman olarak bir karakter geçirmek.

Genel Programlama Hataları 8.5

Bir karakter beklenirken, fonksiyona argüman olarak string geçirmek.

8.3 KARAKTER KÜTÜPHANESİ

Karakter kütüphanesi, karakter verilerini işlemek ve test etmek için kullanışlı bir çok

fonksiyon içermektedir. Her fonksiyon argüman olarak, int ile temsil edilen bir karakter ya da

EOF alır. 4. ünitede anlattığımız gibi, karakterler genellikle tamsayı olarak ele alınır çünkü

C‘de bir karakter genellikle bir byte’lık bir tamsayıdır. EOF’in genellikle –1 değerine sahip

olduğunu ve bazı donanım mimarilerinin, negatif değerlerin char değişkenler içinde

depolanmasına izin vermediğine dikkat ediniz. Bu sebepten, karakter kütüphane fonksiyonları

karakterleri tamsayılar olarak ele alır. Şekil 8.1, karakter kütüphane fonksiyonlarını

özetlemektedir.

Prototip Fonksiyon Tanımı

320

int isdigit ( int c); c bir rakam ise doğru bir değer,değilse 0(yanlış) döndürür.

int isalpha (int c); c bir harf ise doğru bir değer,değilse 0 döndürür.

int iaslnum (int c); c bir harf ya da rakamsa doğru bir değer,değilse 0 döndürür.

int isxdigit (int c); c onaltılık sistemde bir rakam değeri karakteri ise doğru bir

değer,değilse 0 döndürür.(ikilik sistem,sekizlik sistem,onluk sistem ve

onaltılık sistem için daha detaylı bilgiyi Ekler E kısmında

bulabilirsiniz.)

int islower ( int c); c küçük bir harf ise doğru bir değer,değilse 0 döndürür.

int isupper (int c); c büyük bir harf ise doğru bir değer,değilse 0 döndürür.

int tolower ( int c); c bir büyük harf ise,tolower c ’yi küçük harfe çevirerek döndürür,

değilse tolower argümanı değiştirmeden döndürür

int toupper (int c); c küçük harf ise,toupper c ’yi büyük harfe çevirip döndürür,değilse

toupper argümanı değiştirmeden döndürür.

int isspace (int c); c yeni satır(‘\n’),boşluk(‘ ‘),form besleme(‘\f ’),satır başı(‘\r’),yatay

tab(‘\t’) ya da düşey tab(‘\v’) karakterlerinden biriyse doğru bir değer,

değilse 0 döndürür.

int iscntrl (int c); c bir kontrol değişkeni ise doğru bir değer,değilse 0 döndürür.

int ispunct (int c); c boşluk,rakam ya da harften başka bir yazdırma karakteri ise doğru bir

değer,değilse 0 döndürür.

int isprint (int c); c boşluk (‘ ‘) karakteri de dahil olmak üzere bir yazdırma karakteri ise

doğru bir değer,değilse 0 döndürür.

int isgraph (int c); c boşluk karakteri haricinde bir yazdırma değeri ise doğru bir

değer,değilse 0 döndürür.

Şekil 8.1 Karakter kütüphane fonksiyonlarının özeti

İyi Programlama Alıştırmaları 8.2

Karakter kütüphanesinden fonksiyonlar kullanırken <ctype.h> öncü dosyasını programınıza dahil

edin.

Şekil 8.2, isdigit, isalpha, isalnum ve isxdigit fonksiyonlarını göstermektedir. isdigit

fonksiyonu, argümanının bir rakam ( 0-9 ) olup olmadığına karar verir. isalpha, argümanının

büyük harf ( A-Z ) ya da küçük harf ( a-z ) olup olmadığına karar verir. isalnum fonksiyonu,

argümanının büyük harf, küçük harf ya da rakam olup olmadığına karar verir. isxdigit

fonksiyonu, argümanının onaltılık sistemde rakam değeri ( A-F,a-f,0-9 ) olup olmadığına

karar verir.

Şekil 8.2, her fonksiyonla ( ?: ) koşullu operatörünü kullanarak, test edilen her karakter için

operatörle birlikte kullanılan stringlerden hangisinin yazdırılacağına karar verir. Örneğin,

isdigit(‘8’) ? “8 bir rakamdır”: ”8 bir rakam değildir”

deyimi eğer ‘8’ rakam ise ( yani isdigit doğru bir değer döndürüyorsa), ”8 bir rakamdır ”

stringinin yazdırılacağını ve eğer ‘8’ rakam değilse (yani isdigit 0 değerini döndürüyorsa) “8

bir rakam değildir” stringinin yazdırılacağını belirtir.

1 /\* Şekil 8.2: fig08\_02.c

2 isdigit, isalpha, isalnum, ve isxdigit fonksiyonlarını kullanmak\*/

3 #include <stdio.h>

4 #include <ctype.h>

321

5

6 int main( )

7 {

8 printf( "%s\n%s\n%s\n\n", "isdigit için: ",

9 isdigit( '8' ) ? "8 bir rakamdır" : "8 bir rakam değildir ",

10 isdigit( '#' ) ? "# bir rakamdır " :

11 "# bir rakam değildir" );

12 printf( "%s\n%s\n%s\n%s\n%s\n\n",

13 "isalpha için:",

14 isalpha( 'A' ) ? "A bir harftir " : "A bir harf değildir",

15 isalpha( 'b' ) ? "b bir harftir " : "b bir harf değildir ",

16 isalpha( '&' ) ? "& bir harftir " : "& bir harf değildir",

17 isalpha( '4' ) ? "4 bir harftir " :

18 "4 bir harf değildir" );

19 printf( "%s\n%s\n%s\n%s\n\n",

20 “isalnum için:",

21 isalnum( 'A' ) ? "A bir rakam yada harftir" : "A bir rakam yada

22 harf değildir ",

23 isalnum( '8' ) ? "8 bir rakam yada harftir " : "8 bir rakam yada

24 harf değildir",

25 isalnum( '#' ) ? "# bir rakam yada harftir " : "# bir rakam yada

26 harf değildir" );

27 printf( "%s\n%s\n%s\n%s\n%s\n%s\n",

28 "isxdigit için:",

29 isxdigit( 'F' ) ? "F bir heksadesimal rakamdır" : "F bir heksadesimal

30 rakam değildir ",

31 isxdigit( 'J' ) ? "J bir heksadesimal rakamdır " : " J bir heksadesimal ",

32 "rakam değildir ",

33 isxdigit( '7' ) ? "7 bir heksadesimal rakamdır " : " 7 bir heksadesimal

34 rakam değildir",

35 isxdigit( '$' ) ? "$ bir heksadesimal rakamdır " : " $ bir heksadesimal

36 rakam değildir",

37 isxdigit( 'f' ) ? " f bir heksadesimal rakamdır " : " f bir heksadesimal

38 rakam değildir");

39 return 0;

40 }

isdigit için:

8 bir rakamdır

# bir rakam değildir

isalpha için:

A bir harftir

b bir harftir

& bir harf değildir

4 bir harf değildir

isxdigit için:

F bir heksadesimal rakamdır

322

J bir heksadesimal rakam değildir

7 bir heksadesimal rakamdır

$ bir heksadesimal rakam değildir

f bir heksadesimal rakamdır

Şekil 8.2 isdigit,isalpha,isalnum ve isxdigit fonksiyonlarının kullanılması

Şekil 8.3, islower, isupper, tolower ve toupper fonksiyonlarını göstermektedir. islower

fonksiyonu, argümanının küçük harf ( a-z ) olup olmadığına karar verir. isupper fonksiyonu,

argümanının büyük harf ( A-Z ) olup olmadığına karar verir. tolower fonksiyonu, büyük bir

harfi küçük harfe çevirir ve küçük harfi döndürür. Eğer argüman büyük harf değilse, tolower

argümanını değiştirmeden döndürür. toupper fonksiyonu, küçük bir harfi büyük harfe çevirir

ve büyük harfi geri döndürür. Eğer argüman küçük harf değilse, toupper argümanını

değiştirmeden döndürür.

1 /\* Şekil 8.3: fig08\_03.c

2 islower, isupper, tolower, toupper fonksiyonlarını kullanma\*/

3 #include <stdio.h>

4 #include <ctype.h>

5

6 int main( )

7 {

8 printf( "%s\n%s\n%s\n%s\n%s\n\n",

9 "islower için:",

10 islower( 'p' ) ? "p bir küçük harftir " : "p bir küçük

11 harf değildir “,

12 islower( 'P' ) ? "P bir küçük harftir " : "P bir küçük

13 harf değildir ",

14 islower( '5' ) ? "5 bir küçük harftir " : "5 bir küçük

15 harf değildir ",

16 islower( '!' ) ? "! bir küçük harftir " : "! bir küçük

17 harf değildir");

18 printf( "%s\n%s\n%s\n%s\n%s\n\n",

19 "isupper için:",

20 isupper( 'D' ) ? "D bir büyük harftir " : "D bir büyük

21 harf değildir”,

22 isupper( 'd' ) ? "d bir büyük harftir " : "d bir büyük

23 harf değildir”,

24 isupper( '8' ) ? "8 bir büyük harftir " : "8 bir büyük

25 harf değildir”,

26 isupper( '$' ) ? "$ bir büyük harftir " : "$ bir büyük

27 harf değildir" );

28 printf( "%s%c\n%s%c\n%s%c\n%s%c\n",

29 "u nun büyük harfi ", toupper( 'u' ),

30 "7 nin büyük harfi ", toupper( '7' ),

31 "$ ın büyük harfi ", toupper( '$' ),

32 "L nin küçük harfi ", tolower( 'L' ) );

323

33 return 0;

34 }

islower için:

p bir küçük harftir

P bir küçük harf değildir

5 bir küçük harf değildir

! bir küçük harf değildir

isupper için:

D bir büyük harftir

d bir büyük harf değildir

8 bir büyük harf değildir

$ bir büyük harf değildir

u nun büyük harfi U

7 nin büyük harfi 7

$ ın büyük harfi $

L nin küçük harfi l

Şekil 8.3 islower,isupper,tolower ve toupper fonksiyonlarını kullanmak

Şekil 8.4, isspace, iscntrl, ispunct, isprint ve isgraph fonksiyonlarını göstermektedir.

isspace fonksiyonu argümanının, boşluk (‘ ‘), form besleme ( ‘\f’ ), yeni satır ( ‘\n’ ), satır

başı ( ‘\r’ ), yatay tab ( ‘\t’ ) ya da düşey tab ( ‘\v’ ) karakterlerinden herhangi biri olup

olmadığına karar verir. iscntrl fonksiyonu argümanının, yatay tab ( ‘\t’ ), düşey tab ( ‘\v’ ),

form besleme ( ‘\f’ ), alarm(‘\a’), ters eğik çizgi (‘\b’), yeni satır ( ‘\n’ ) ya da satır başı ( ‘\r’ )

karakterlerinden herhangi biri olup olmadığına karar verir. ispunct fonksiyonu, argümanının

boşluk, rakam ya da harften farklı bir karakter (örneğin, $, #, (,), [,], {,}, ;, :, % gibi) olup

olmadığına karar verir. isprint fonksiyonu, argümanının ekranda gösterilebilecek

karakterlerden biri olup olmadığına ( boşluk karakteri de dahil olmak üzere) karar verir.

isgraph fonksiyonu, isprint ile aynı karakterleri test eder ancak boşluk karakteri test edilecek

karakterlere dahil değildir.

1 /\* Şekil 8.4: fig08\_04.c

2 isspace, iscntrl, ispunct, isprint, isgraph fonksiyonlarını kullanmak\*/

3 #include <stdio.h>

324

4 #include <ctype.h>

5

6 int main( )

7 {

8 printf( "%s\n%s%s\n%s%s\n%s\n\n",

9 "isspace için:",

10 "Yenisatır", isspace( '\n' ) ? " boşluk karakteridir" :

11 " boşluk karakteri değildir ",

12 "tab karakteri",isspace( '\t' ) ? " boşluk karakteridir " :

13 " boşluk karakteri değildir ",

14 isspace( '%' ) ? "% boşluk karakteridir" :

15 "% boşluk karakteri değildir " );

16 printf( "%s\n%s%s\n%s\n\n", "iscntrl için:",

17 "Yenisatır", iscntrl( '\n' ) ? " kontrol karakterdir " :

18 " kontrol karakteri değildir ",

19 iscntrl( '$' ) ? "$ kontrol karakterdir " :

20 "$ kontrol karakteri değildir ");

21 printf( "%s\n%s\n%s\n%s\n\n",

22 "ispunct için:",

23 ispunct( ';' ) ? "; noktalama işaretidir " :

24 "; noktalama işareti değildir ",

25 ispunct( 'Y' ) ? "Y noktalama işaretidir " :

26 "Y noktalama işareti değildir ",

27 ispunct( '#' ) ? "# noktalama işaretidir " :

28 "# noktalama işareti değildir ",);

29 printf( "%s\n%s\n%s%s\n\n", "isprint için:",

30 isprint( '$' ) ? "$ yazı karakterdir " :

31 " $ yazı karakteri değildir ",

32 "Alarm", isprint( '\a' ) ? " yazı karakterdir " :

33 " yazı karakteri değildir " );

34 printf( "%s\n%s\n%s%s\n", "isgraph için:",

35 isgraph( 'Q' ) ? "Q boşluktan farklı bir yazı karakteridir" :

36 "Q boşluktan farklı bir yazı karakteri değildir",

37 "Boşluk", isgraph( ' ' ) ? " boşluktan farklı bir yazı karakteridir " :

38 " boşluktan farklı bir yazı karakteri değildir" );

39 return 0;

40 }

isspace için:

Yenisatır boşluk karakteridir

tab karakteri boşluk karakteridir

% boşluk karakteri değildir

iscntrl için:

Yenisatır kontrol karakteridir

$ kontrol karakteri değildir

ispunct için:

; noktalama işaretidir

325

Y noktalama işareti değildir

# noktalama işaretidir

isprint için:

$ yazı karakteridir

Alarm yazı karakteri değildir

isgraph için:

Q boşluktan farklı bir yazı karakteridir

Boşluk boşluktan farklı bir yazı karakteri değildir

Şekil 8.4 isspace, iscntrl, ispunct, isprint ve isgraph fonksiyonlarını kullanmak.

8.4 STRING DÖNÜŞÜM FONKSİYONLARI

Bu kısım, genel amaçlı kütüphanedeki ( stdlib ) string dönüşüm fonksiyonlarını

göstermektedir. Bu fonksiyonlar, rakam stringlerini tamsayı ve ondalıklı sayı değerlerine

dönüştürür. Şekil 8.5, string dönüşüm fonksiyonlarını özetlemektedir. Fonksiyon

başlıklarında, nPtr değişkeninin const olarak bildirildiğine dikkat ediniz. (“nPtr, bir karakter

sabitini gösteren bir göstericidir” biçiminde okuyunuz) const, argümanın değerinin

değiştirilmeyeceğini belirtmektedir.

Fonksiyon Prototipi Fonksiyon tanımı

double atof (const char \*nPtr) ; nPtr stringini double’a dönüştürür.

int atoi (const char \*nPtr); nPtr stringini int’e dönüştürür.

long atol (const char \*nPtr); nPtr stringini long int’e dönüştürür.

double strtod (const char \*nPtr, char \*\* endPtr); nPtr stringini double’a dönüştürür.

lond strol (const char \*nPtr,char \*\*endPtr,int base); nptr stringini long’a dönüştürür.

unsigned long strtoul(const char \*nPtr,char \*\*endPtr,int base) nPtr stringini unsigned long’a

dönüştürür.

Şekil 8.5 Genel amaçlı kütüphanedeki string dönüşüm fonksiyonlarının özeti

İyi Programlama Alıştırmaları 8.3

Genel amaçlı kütüphaneden bir fonksiyon kullandığınızda <stdlib.h> öncü dosyasını programınıza

ekleyin.

atof fonksiyonu (Şekil 8.6), argümanını ( ondalıklı bir sayıyı temsil eden bir stringi) double

değere dönüştürür. Fonksiyon, double değeri döndürür. Eğer dönüştürülen değer temsil

edilemezse (örneğin,stringin ilk karakteri rakam değilse), atof fonksiyonunun davranışı

belirsizdir.

atoi fonksiyonu (Şekil 8.7), argümanını (bir tamsayıyı temsil eden rakamlar stringini) int bir

değere dönüştürür. Fonksiyon, int değeri geri döndürür. Eğer dönüştürülen değer temsil

edilemezse, atoi fonksiyonunun davranışı belirsizdir.

1 /\* Şekil 8.6: fig08\_06.c

2 atof \*/

3 #include <stdio.h>

326

4 #include <stdlib.h>

5

6 int main( )

7 {

8 double d;

9

10 d = atof( "99.0" );

11 printf( "%s%.3f\n%s%.3f\n",

12 " \"99.0\"stringi double’ tipine dönüştürüldü ", d,

13 "Dönüştüren sayının 2 ye bölümü ",

14 d / 2.0 );

15 return 0;

16 }

“99.0” stringi double tipine dönüştürüldü 99.000

Dönüştüren sayının 2 ye bölümü 49.500

Şekil 8.6 atof kullanmak

1 /\* Şekil 8.7: fig08\_07.c

2 atoi kullanmak\*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 int main( )

7 {

8 int i;

9

10 i = atoi( "2593" );

11 printf( "%s%d\n%s%d\n",

12 "\"2593\" stringi int tipine dönüştürüldü", i,

13 "Dönüştürülen sayı eksi 593: ", i - 593 );

14 return 0;

15 }

“2593” stringi int tipine dönüştürüldü

Dönüştürülen sayı eksi 593: 2000

Şekil 8.7 atoi kullanmak

atol fonksiyonu (Şekil 8.8), argümanını (long tamsayı temsil eden rakamlar stringini) long

değere dönüştürür. Fonksiyon, long değeri döndürür. Eğer dönüştürülen değer temsil

edilemezse, atol fonksiyonunun davranışı belirsizdir. Eğer int ve long’un ikisi de 4 byte

içinde depolanmışsa, atoi fonksiyonu ve atol fonksiyonu eş olarak çalışacaktır.

1 /\* Fig. 8.8: fig08\_08.c

327

2 atol kullanma\*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 int main( )

7 {

8 long l;

9

10 l = atol( "1000000" );

11 printf( "%s%ld\n%s%ld\n",

12 "\"1000000\" stringi long int tipine çevrildi ", l,

13 "Çevrilen değerin 2’ ye bölümü: ", l / 2 );

14 return 0;

15 }

“1000000” stringi long int tipine çevrildi 1000000

Çevrilen değerin 2’ye bölümü: 500000

Şekil 8.8 atol kullanmak

strtod fonksiyonu (Şekil 8.9), ondalıklı bir değeri temsil eden karakterleri double değere

dönüştürür. Fonksiyon iki argüman alır; bir string (char\*) ve bu stringi gösteren bir gösterici

(char\*\*). String, double’a dönüştürülecek karakterleri içerir. Gösterici, stringin

dönüştürülmüş kısmından sonraki ilk karakterin konumuna atanır.

Şekil 8.9’daki

d = strtod(string, &stringPtr);

ifadesi, d’nin stringten dönüştürülen double değere atandığını ve stringPtr’nin dönüştürülen

değerden (51.2) sonra, string içindeki ilk karakterin konumuna atandığını gösterir.

strtol fonksiyonu (Şekil 8.10), bir tamsayıyı temsil eden karakterleri long’a dönüştürür.

Fonksiyon üç argüman almaktadır ; bir string (char \*), stringi gösteren bir gösterici ve bir

tamsayı. String, dönüştürülecek karakter dizisini içermektedir. Gösterici, stringin

dönüştürülen kısmından sonraki ilk karakterin konumuna atanır. Tamsayı, dönüştürülen

değerin tabanını belirtir.

Şekil 8.10’daki

x = strtol(string, &kalanPtr,0);

ifadesi, x’in stringten dönüştürülen long değere atandığını gösterir. İkinci argüman, kalanPtr,

dönüşümden sonra stringin kalanına atanmıştır. İkinci argüman için NULL kullanmak,

stringin geri kalanının ihmal edilmesini sağlatır. Üçüncü argüman, 0, dönüştürülecek değerin

sekizlik (8 tabanı), onluk (10 tabanı) ya da onaltılık sistemde (16 tabanı) olabileceğini

gösterir. Taban 0 ya da 2-36 arsında herhangi bir değer olarak belirtilebilir. Sekizlik, onluk ve

onaltılık sistemler için detaylı bilgiyi, Ekler E kısmında bulabilirsiniz. 11 tabanından 36

328

tabanına kadar olan tamsayıların nümerik temsilleri, 10-35 değerleri için A-Z karakterlerini

kullanır. Örneğin, onaltılık sistemler 0-9 rakamlarını ve A-F karakterlerini içerebilir. 11

tabanındaki bir tamsayı, 0-9 rakamlarını ve A karakterini içerebilir. 24 tabanındaki bir

tamsayı, 0-9 rakamlarını ve A-N karakterlerini içerebilir. 36 tabanındaki bir tamsayı, 0-9

rakamlarını ve A-Z karakterlerini kullanır.

1 /\* Şekil 8.9: fig08\_09.c

2 strtod kullanma\*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 int main( )

7 {

8 double d;

9 const char \*string = "51.2% kabul edildi";

10 char \*stringPtr;

11

12 d = strtod( string, &stringPtr );

13 printf( "string: \"%s\"\n",

14 string );

15 printf( "double %.2f ve string \"%s\"e dönüştürüldü.\n",

16 d, stringPtr );

17

18 return 0;

19 }

string: “51.2% kabul edildi”

double 51.20 ve string “% kabul edildi” e dönüştürüldü.

Şekil 8.9 strtod kullanmak

1 /\* Şekil 8.10: fig08\_10.c

2 strtol kullanma\*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 int main( )

7 {

8 long x;

9 const char \*string = "-1234567abc";

10 char \*kalanPtr;

11

12 x = strtol( string, &kalanPtr, 0 );

13 printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",

14 "Orijinal string: ", string,

15 "Dönüştürülen değer: ", x,

16 "Orijinal stringden geriye kalan: ",

17 kalanPtr,

18 "Dönünüştürülen değer artı 567: ", x + 567 );

329

19 return 0;

20 }

Orijinal string: “-1234567abc”

Dönüştürülen değer: -1234567

Orijinal stringden geriye kalan: “abc”

Dönünüştürülen değer artı 567: -1234000

Şekil 8.10 strtol kullanmak

strtoul fonksiyonu (Şekil 8.11), unsigned long tamsayıları temsil eden karakterleri unsigned

long’a çevirir. Bu fonksiyon, strtol fonksiyonuyla eş bir biçimde çalışmaktadır.

Şekil 8.11’deki

x = srtroul(string, &kalanPtr,0);

ifadesi x’in string’ten dönüştürülen değere atandığını belirtir. İkinci argüman, &kalanPtr,

dönüşümden sonra string’in kalanına atanmıştır. Üçüncü argüman, 0, dönüştürülecek değerin

sekizlik, onluk ya da onaltılık sistemde olabileceğini belirtir.

1 /\* Fig. 8.11: fig08\_11.c

2 strtoul kullanma \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 int main( )

7 {

8 unsigned long x;

9 const char \*string = "1234567abc";

10 char \*kalanPtr;

11

12 x = strtoul( string, &kalanPtr, 0 );

13 printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",

14 " Orijinal string: ", string,

15 " Dönüştürülen değer: ", x,

16 " Orijinal stringden geriye kalan: ",

17 kalanPtr,

18 " Dönünüştürülen değer eksi 567: ", x - 567 );

19 return 0;

20 }

Orijinal string: “1234567abc”

Dönüştürülen değer: -1234567

Orijinal stringden geriye kalan: “abc”

Dönünüştürülen değer eksi 567: 1234000

Şekil 8.11 strtoul kullanmak

330

8.5 STANDART GİRİŞ/ÇIKIŞ KÜTÜPHANE FONKSİYONLARI

Bu kısım, standart giriş/çıkış kütüphanesindeki (stdio) karakter ve string verilerini ele alan

bazı fonksiyonları tanıtmaktadır. Şekil 8.12, standart giriş/çıkış kütüphanesindeki string

giriş/çıkış fonksiyonlarını özetlemektedir.

İyi Programlama Alıştırmaları 8.4

Standart giriş/çıkış kütüphanesindeki fonksiyonları kullanırken , <stdio.h> öncü dosyasını

programlarınıza ekleyin.

Fonksiyon Prototipi Fonksiyon tanımı

int getchar(void); Standart girişteki karakteri alır ve tamsayı olarak döndürür.

char \*gets(char \*s); Standart girişten aldığı karakterleri, yeni satır ya da dosya sonu

belirteciyle karşılaşıncaya dek s dizisine alır. Sonlandırıcı null

karakter diziye eklenir.

int putchar(int c); c içindeki karakteri yazdırır.

int puts(const char \*s); yani satır karakteri ile devam edilen bir stringi yazdırır.

int sprintf(char \*s,const char \*format,...);

printf ile denktir, ancak çıktılar ekran yerine s dizisine

gönderilir.

int sscanf (char \*s,const char \*format,...);

scanf ile denktir,ancak girdiler klavye yerine s dizisinden

okunur.

Şekil 8.12 standart giriş/çıkış kütüphanesindeki karakter ve string fonksiyonları

Şekil 8.13,gets ve putchar fonksiyonlarını kullanarak, standart giriş biriminden ( klavye) bir

metnin bir satırını okuyup daha sonra da satırdaki karakterleri yineleme kullanarak ters bir

sırada yazdırmaktadır. gets, standart giriş biriminden yeni satır ya da dosya sonu belirteciyle

karşılaşıncaya kadar aldığı karakterleri argümanına (char tipinde bir dizi) okur. Okuma sona

erdiğinde, dizinin sonuna null karakter (‘\0’) eklenir. putchar fonksiyonu, argümanı olan

karakteri yazdırır. Program, metnin satırını tersten yazdırmak için yinelemeli ters

fonksiyonunu çağırmaktadır. Eğer dizinin ters tarafından alınan ilk karakteri null karakter

(‘\0’) ise ters geri döner. Aksi takdirde ters, s[1] elemanıyla başlayan bir alt diziyle yeniden

çağrılır ve s[0] karakteri yineleme çağrısı sonlandığında putchar ile yazdırılır. if yapısının

else kısmındaki iki ifade, ters fonksiyonunun dizi içinde bir karakter yazdırılmadan önce

sonlandırıcı null karaktere kadar ilerlemesini sağlar. Yineleme çağrıları tamamlandığında

karakterler ters bir sırada yazdırılır.

1 /\* Şekil 8.13: fig08\_13.c

2 gets ve putchar kullanma \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char cumle[ 80 ];

8 void ters( const char \* const );

331

9

10 printf( "Metin girişi yapınız:\n" );

11 gets( cumle );

12

13 printf( "\nGirdiğiniz metin tersten yazıldığında:\n" );

14 ters( cumle);

15

16 return 0;

17 }

18 void ters( const char \* const sPtr )

19 {

20 if ( sPtr[ 0 ] == '\0' )

21 return;

22 else {

23 ters( &sPtr[ 1 ] );

24 putchar( sPtr[ 0 ] );

25 }

26 }

Metin girişi yapınız:

Karakterler ve Stringler

Girdiğiniz metin tersten yazıldığında:

relgnirtS ev relretkaraK

Metin girişi yapınız:

iki kabak iki

Girdiğiniz metin tersten yazıldığında:

iki kabak iki

Şekil 8.13 gets ve putchar kullanmak

Şekil 8.14, getchar ve puts fonksiyonlarını kullanarak standart giriş biriminden alınan

karakterleri cumle dizisi içine okuyup, dizideki karakterleri bir string olarak yazdırmaktadır.

getchar fonksiyonu, standart giriş biriminden bir karakter alıp bu karakterin tamsayı değerini

döndürür. puts fonksiyonu, argüman olarak bir string (char\*) alır ve stringi yazdırıp sonra da

yeni satıra geçer. Program, kullanıcı tarafından satır sonuna konan yeni satır karakteri

getchar fonksiyonu tarafından okununca, karakter okumayı durdurur. Diziyi bir string olarak

kullanabilmek için, cumle dizisinin sonuna null karakter eklenir. puts fonksiyonu cumle

içindeki stringi yazdırır.

1 /\* Şekil 8.14: fig08\_14.c

2 getchar ve puts kullanma\*/

3 #include <stdio.h>

4

332

5 int main( )

6 {

7 char c, cumle[ 80 ];

8 int i = 0;

9

10 puts( "Metin girişi yapınız: " );

11 while ( ( c = getchar( ) ) != '\n')

12 cumle[ i++ ] = c;

13

14 cumle[ i ] = '\0'; /\* stringin sonunu null ekle \*/

15 puts( "\nGirilen metin:" );

16 puts( cumle );

17 return 0;

18 }

Metin girişi yapınız:

Bu bir testtir.

Girilen metin:

Bu bir testtir.

Şekil 8.14 getchar ve puts kullanmak

Şekil 8.15, sprintf fonksiyonunu s dizisine ( bir karakter dizisidir ) formatlı bir biçimde veri

yazmak için kullanır. Fonksiyon, printf ile aynı dönüşüm belirteçlerini kullanır (tüm yazı

biçimlendirme özellikleri hakkında daha detaylı bilgi için 9.üniteye bakınız). Program bir int

ve bir double değeri alıp, biçimlendirerek s dizisine yazar. s dizisi, sprintf’in ilk argümanıdır.

Şekil 8.16, sscanf fonksiyonunu s karakter dizisinden biçimlendirilmiş verileri okumak için

kullanır. Fonksiyon, scanf ile aynı dönüşüm belirteçlerini kullanır. Program, s dizisinden bir

int ve bir double değer okur ve okuduğu değerleri sırasıyla x ve y değişkenleri içinde tutar. x

ve y değerleri daha sonra yazdırılır. s dizisi, sscanf ’in ilk argümanıdır.

1 /\* Şekil 8.15: fig08\_15.c

2 sprintf kullanma\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char s[ 80 ];

8 int x;

9 double y;

10

11 printf( " integer ve double tipte değer girin:\n" );

12 scanf( "%d%lf", &x, &y );

13 sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );

14 printf( "%s\n%s\n", "s dizisinde saklanan çıktı: ", s );

15 return 0;

333

16 }

integer ve double tipte değer girin girin:

288 87.375

s dizisinde saklanan çıktı:

integer: 298

double: 87.38

Şekil 8.15 sprintf kullanmak

1 /\* Şekil 8.16: fig08\_16.c

2 sscanf kullanma\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char s[ ] = "31298 87.375";

8 int x;

9 double y;

10

11 sscanf( s, "%d%lf", &x, &y );

12 printf( "%s\n%s%6d\n%s%8.3f\n",

13 "s karakter dizisinde saklanan değerler:",

14 "integer:", x, "double:", y );

15 return 0;

16 }

s karakter dizisinde saklanan değerler:

integer: 31298

double: 87.375

Şekil 8.16 sscanf kullanmak

8.6 STRING KÜTÜPHANESİNDEKİ STRING İŞLEME

FONKSİYONLARI

String kütüphanesi, string verilerini ele almak, stringleri karşılaştırmak, stringlerde karakterler

ya da başka stringler aramak, stringleri atomlara (stringi mantıklı parçalara bölmek) ayırmak

ve stringlerin uzunluğuna karar vermek gibi bir çok kullanışlı fonksiyon sunar. Bu kısım,

string kütüphanesindeki string işleme fonksiyonlarını ele almaktadır. Fonksiyonlar, Şekil

8.17’de özetlenmiştir. Her fonksiyon (strncpy hariç) , sonucunun sonuna null karakter ekler.

Fonksiyon Prototipi Fonksiyon Tanımı

char \*strcpy (char \*s1,const char \*s2) s2 stringini s1 dizisi içine kopyalar,s1’in

değeri döndürülür.

char \*strncpy(char \*s1,const char \*s2,size\_t n) s2 stringinin en fazla n karakterini s1

dizisi içine kopyalar.s1’in değeri

döndürülür.

334

char \*strcat(char \*s1,const char \*s2) s2 stringini s1 dizisine ekler.s2’nin ilk

karakteri s1 dizisinin null karakteri

üzerine yazılır.s1’in değeri döndürülür.

char \*strncat(char \*s1,const char \*s2,size\_t n) s2 stringinin en fazla n karakterini s1

dizisine keler.s2’nin ilk karakteri s1

dizisinin null karakteri üzerine

yazılır.s1’in değeri döndürülür.

Şekil 8.17 string kütüphanesindeki string işleme fonksiyonları

strncpy ve strncat fonksiyonlarının size\_t parametresini kullandığına dikkat ediniz. size\_t,

standart tarafından, sizeof operatörünün döndürdüğü değerin tipi olarak belirlenmiştir.

Taşınırlık İpuçları 8.2

size\_t tipi, sisteme bağımlı olarak, unsigned long ya da signed int tipinin eşanlamlısıdır.

İyi Programlama Alıştırmaları 8.5

String kütüphanesindeki fonksiyonları kullanırken,<string.h> öncü dosyasını eklemeyi

unutmayın.

strcpy fonksiyonu, ikinci argümanını(bir string) ilk argümanı (string ve stringle birlikte

kopyalanan sonlandırıcı null karakteri tutabilecek kadar geniş bir karakter dizisi) içine

kopyalar.strncpy fonksiyonu, strcpy fonksiyonu ile denktir. Ancak, strncpy farklı olarak

stringten diziye kopyalanacak karakterlerin sayısını belirlemektedir.strncpy fonksiyonunun

ikinci argümanına, sonlandırıcı null karakteri kopyalamak zorunda olmadığına dikkat ediniz.

Sonlandırıcı NULL karakter yalnızca, kopyalanacak karakterlerin sayısı dizinin

uzunluğundan en az bir büyükse eklenir. Örneğin,eğer “test” ikinci argümansa, null karakter

yalnızca strncpy’nin üçüncü argümanı en az 5 ise(“test” içinde 4 karakter artı null karakter)

eklenir.Eğer üçüncü argüman 5’ten büyükse,üçüncü argümanda belirtilen sayıya ulaşılana dek

diziye null karakter eklenmeye devam edilir.

Genel Programlama Hataları 8.6

strncpy ’nin üçüncü argümanı, ikinci argümandaki stringin uzunluğundan küçükse ya da ikinci

argümandaki stringin uzunluğuna eşitse, ilk argümana null karakter eklememek.

Şekil 8.18, strcpy kullanarak x dizisi içindeki stringi y dizisi içine kopyalamakta ve strncpy

kullanarak x dizisinin ilk 14 karakterini z dizisine eklemektedir. z dizisine null karakter

eklenmiştir çünkü programda strncpy fonksiyonuna yapılan çağrı, sonlandırıcı null karakteri

eklememektedir. ( Üçüncü argüman ikinci argümandaki stringin uzunluğundan küçüktür )

1 /\* Fig. 8.18: fig08\_18.c

2 strcpy ve strncpy kullanma\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 char x[ ] = "Doğum günün kutlu olsun";

9 char y[ 25 ], z[ 15 ];

10

335

11 printf( "%s%s\n%s%s\n",

12 "x dizisindeki string: ", x,

13 "y dizisindeki string: ", strcpy( y, x ) );

14

15 strncpy( z, x, 11);

16 z[ 11 ] = '\0';

17 printf( "z dizisindeki string: %s\n", z );

18 return 0;

19 }

x dizisindeki string: Doğum günün kutlu olsun

y dizisindeki string: Doğum günün kutlu olsun

z dizisindeki string: Doğum günün

Şekil 8.18 strcpy ve strncpy kullanmak

strcat fonksiyonu, ikinci argümanını (bir string) ilk argümanına (bir string içeren karakter

dizisi) eklemektedir. İkinci argümanın ilk karakteri, ilk argümandaki stringin sonlandırıcı null

karakterinin yerini alır. Programcı, ilk stringi tutmak için kullanılan dizinin, ilk stringi, ikinci

stringi ve ikinci stringten kopyalanan null karakteri tutabilecek kadar geniş olduğundan emin

olmalıdır. strncat, ikinci stringten ilk stringe belli sayıdaki karakteri ekler. Sonlandırıcı null

karakter sonuca otomatik olarak eklenir. Şekil 8.9 strcat ve strncat fonksiyonlarını

göstermektedir.

1 /\* Şekil 8.19: fig08\_19.c

2 strcat ve strncat kullanma\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 char s1[ 20 ] = "Mutlu ";

9 char s2[] = "Yeni Yıllar";

10 char s3[ 40 ] = "";

11

12 printf( "s1 = %s\ns2 = %s\n", s1, s2 );

13 printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );

14 printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );

15 printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );

16 return 0;

17 }

s1 = Mutlu

s2 = Yeni Yıllar

strcat(s1, s2) = “Mutlu Yeni Yıllar

strncat(s3, s1, 6) = Mutlu

336

strcat(s3, s1) = Mutlu Mutlu Yeni Yıllar

Şekil 8.19 strcat ve strncat kullanmak

8.7 STRING KÜTÜPHANESİNDEKİ STRING KARŞILAŞTIRMA

FONKSİYONLARI

Bu kısım, string kütüphanesindeki string karşılaştırma fonksiyonlarını, strcmp ve strncmp,

tanıtmaktadır. Fonksiyon prototipleri ve açıklamaları Şekil 8.20’de gösterilmiştir.

Fonksiyon Prototipi Fonksiyon tanımı

int strcmp(const char\*s1,const char \*s2);

s1 stringiyle s2 stringini karşılaştırır.Fonksiyon,

s1 s2’ye eşitse 0 , s1 s2’den küçükse 0’dan

küçük,s1 s2’den büyükse 0’dan büyük bir değer

döndürür.

int strcmp(const char\*s1,const char \*s2,size\_t n);

s1 stringinin n karakterine s2 stringiyle

karşılaştırır.Fonksiyon, s1 s2’ye eşitse 0 , s1

s2’den küçükse 0’dan küçük,s1 s2’den büyükse

0’dan büyük bir değer döndürür.

Şekil 8.20 string kütüphanesindeki string karşılaştırma fonksiyonları

Şekil 8.21, üç stringi strcmp ve strncmp fonksiyonlarını kullanarak karşılaştırmaktadır.

strcmp fonksiyonu, ilk argümanının karakterlerini ikinci argümanını karakterleriyle teker

teker karşılaştırır. Fonksiyon eğer stringler eşitse 0 değerini, eğer ilk string ikinci stringten

küçükse negatif bir değeri, eğer ilk string ikinci stringten büyükse pozitif bir değeri döndürür.

strncmp, strcmp’ye benzer ancak karşılaştırma yalnızca ilk stringteki n karakter için yapılır.

strncmp bir string içinde null karakterden sonra gelen karakterleri karşılaştırmaz. Program,

her fonksiyon çağrısından döndürülen tamsayı değerlerini yazdırmaktadır.

Genel Programlama Hataları 8.7

strcmp ve strncmp fonksiyonlarının, argümanları eşitken 1 döndüreceğini düşünmek. İki fonksiyonda

eşitlik için 0 değerini (C’nin yanlış bir değer olarak kullandığı değer) döndürür. Bu sebepten iki

stringin eşitliğini test ederken fonksiyonlardan döndürülen sonuçlar 0 ile karşılaştırılmalıdır.

1 /\* Şekil 8.21: fig08\_21.c

2 strcmp ve strncmp kullanma\*/

337

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*s1 = "Mutlu Yeni Yıllar ";

9 const char \*s2 = "Mutlu Yeni Yıllar ";

10 const char \*s3 = "Mutlu Tatiller”;

11

12 printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",

13 "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,

14 "strcmp(s1, s2) = ", strcmp( s1, s2 ),

15 "strcmp(s1, s3) = ", strcmp( s1, s3 ),

16 "strcmp(s3, s1) = ", strcmp( s3, s1 ) );

17

18 printf("%s%2d\n%s%2d\n%s%2d\n",

19 "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),

20 "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),

21 "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );

22 return 0;

23 }

s1 = Mutlu Yeni Yıllar

s2 = Mutlu Yeni Yıllar

s3 = Mutlu Tatiller

strcmp(s1, s2) = 0

strcmp(s1, s3) = 5

strcmp(s3, s1) = -5

strncmp(s1, s3, 6) = 0

strncmp(s1, s3, 7) = 5

strncmp(s3, s1, 7) = -5

Şekil 8.21 strcmp ve strncmp kullanmak

Bir stringin diğerinden “büyük” ya da “küçük” olmasının hangi manaya geldiğini anlamak

için, soyadlardan oluşan bir serinin alfabetik sıralanış sürecini ele alalım. Okuyucu kuşkusuz

“Jones” soyadının “Smith” soyadından önce geldiğini bilecektir, çünkü “Jones” soyadının ilk

harfi, “Smith” soyadının ilk harfine göre alfabede daha önce gelecektir. Fakat alfabe yalınızca

29 harfin listesi değildir. Alfabe karakterlerin sıralı listesidir. Her harf, listede belli bir

pozisyonda bulunur. ”Z” yalnızca listedeki bir harf değil listenin 29. harfidir.

Bilgisayar bir harfin diğerinden önce geleceğini nasıl bilmektedir? Her karakter, bilgisayarın

içinde nümerik kodlar olarak temsil edilir; bilgisayar iki stringi karşılaştırdığında aslında

stringler içindeki karakterlerin nümerik kodlarını karşılaştırmaktadır.

338

Taşınırlık İpuçları 8.3

Karakterleri temsil eden nümerik kodlar her bilgisayar için farklı olabilir.

Karakter gösterimlerini standartlaştırmak için çoğu üretici makinelerini iki popüler kodlama

şemasından birine göre tasarlamışlardır. Bu şemalardan ilki ASCII (American Standart Code

for Information Interchange) ve diğeri EBCDIC’dir (Extended Binary Coded Decimal

Interchange Code). Başka kodlama şemaları da vardır ancak bu ikisi daha popülerdir.

ASCII ve EBCDIC, karakter kodları ya da karakter kümeleri olarak adlandırılır. String ve

karakter işlemleri gerçekte karakterlerin kendileri yerine uygun kodların işlenmesini içerir.

Bu, C’de karakter ve küçük tamsayıların birbirleri yerine kullanılabilmesini açıklar. Bir

nümerik kodun diğerinden küçük ya da büyük olması anlamlı olacağından çeşitli karakterlerin

ya da stringlerin karakter kodları kullanılarak ilişkilendirilmesi mümkün hale gelecektir.

8.8 STRING KÜTÜPHANESİNDEKİ ARAMA FONKSİYONLARI

Bu kısım, string kütüphanesindeki yer alan ve stringlerin içinde karakter ve başka stringleri

aramak için kullanılan fonksiyonları tanıtmaktadır. Fonksiyonlar, Şekil 8.22’de özetlenmiştir.

strcspn ve strspn fonksiyonlarının size\_t döndürdüğüne dikkat ediniz.

Fonksiyon Prototipi Fonksiyon Tanımı

char \* strchr(const char \*s,int c); c’nin s stringi içindeki ilk konumunu belirler.Eğer c

bulunursa,c’yi gösteren bir gösterici döndürülür.Aksi

takdirde NULL gösterici döndürülür.

size\_t strcspn(const char \*s1,const char \*s2); s1 stringinde, s2 stringi içindeki

karakterlerden oluşmayan ilk kısmı bulur

ve bu kısmın uzunluğunu döndürür.

size\_t strspn(const char \*s1,const char \*s2); s1 stringi içinde, yalnızca s2 stringi

içindeki karakterlerden oluşan ilk kısmı

bulur ve bu kısmın uzunluğunu döndürür.

char \*strpbrk(const char \*s1,const char \*s2); s2 içindeki herhangi bir karakterin, s1

stringi içinde yer aldığı ilk konumu

bulur.Eğer s2 stringindeki bir karakter s1

içinde bulunursa,s1 içindeki karakteri

gösteren bir gösterici döndürür.Aksi

takdirde,NULL gösterici döndürür.

char \*strrchr(const char \*s,int c); s stringi içinde, c karakterinin en son

konumunu belirler.Eğer c bulunursa,s

stringi içindeki c ’yi gösteren bir gösterici

döndürülür.Aksi takdirde,NULL gösterici

döndürülür.

char \*strstr(const char \*s1,const char \*s2); s2 stringi içinde, s1 dizisinin son konumun

belirler.Eğer string bulunursa, s1 stringini

gösteren bir gösterici döndürülür.Aksi

takdirde,NULL gösterici döndürülür.

339

char \*strtok(const char \*s1,const char \*s2); Bir dizi strtok çağrısı s1 stringini,s2

içinde belirtilen karakterle ayrılmış

atomlara (bir satırdaki kelimeler gibi

mantıklı parçalara) ayırır.İlk çağrı ilk

argüman olarak s1 alırken,daha sonraki

çağrılar ilk argüman olarak NULL alır.Her

çağrıda o andaki atomu gösteren bir

gösterici döndürülür.Eğer fonksiyon

çağrıldığında daha fazla atom yoksa

NULL döndürülür.

Şekil 8.22 string kütüphanesindeki string arama fonksiyonları

strchr fonksiyonu, bir karakterin bir string içindeki ilk konumunu arar. Eğer karakter

bulunursa, strchr string içindeki karakteri gösteren bir gösterici döndürür. Aksi takdirde

strchr NULL döndürür. Şekil 8.23, strchr kullanarak “Bu bir testtir” stringi içinde ‘u’ ve

‘z’ karakterlerini aramaktadır.

1 /\* Şekil 8.23: fig08\_23.c

2 strchr kullanma \*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*string = "Bu bir testtir";

9 char character1 = 'u', character2 = 'z';

10

11 if ( strchr( string, character1 ) != NULL )

12 printf( "\'%c\' bu stringte bulundu: \"%s\".\n",

13 character1, string );

14 else

15 printf( "\'%c\' bu stringte bulunamadı: \"%s\".\n",

16 character1, string );

17

18 if ( strchr( string, character2 ) != NULL )

19 printf( "\'%c\' bu stringte bulundu: \"%s\".\n",

20 character2, string );

21 else

22 printf( "\'%c\' bu stringte bulunamadı: \"%s\".\n",

23 character2, string );

24 return 0;

25 }

‘u’ bu stringte bulundu: “Bu bir testtir”

‘z’ bu stringte bulunamadı: “Bu bir testtir”

340

Şekil 8.23 strchr kullanmak

strcspn fonksiyonu (Şekil 8.24), ilk argümanı olan stringin içinde ikinci argümanı olan

stringin içindeki karakterlerden oluşmayan ilk kısmı bulur ve bu kısmın uzunluğunu

döndürür.

strbrk fonksiyonu, ikinci argümanındaki stringte yer alan herhangi bir karakterin ilk

argümanındaki stringte ilk konumunu arar. Eğer ikinci argümandaki karakter bulunursa,

strbrk ilk argümandaki karakteri gösteren bir gösterici döndürür.Aksi takdirde, strbrk

fonksiyonu NULL döndürür. Şekil 8.25, string2 içindeki herhangi bir karakterin string1

içindeki ilk konumunu belirlemektedir.

1 /\* Şekil 8.24: fig08\_24.c

2 strcspn kullanma\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*string1 = "Değer 3.14159";

9 const char \*string2 = "1234567890";

10

11 printf( "%s%s\n%s%s\n\n%s\n%s%u",

12 "string1 = ", string1, "string2 = ", string2,

13 "string1’den string2’nin karakterleri",

14 "çıkarıldığında string1’in uzunluğu = ",

15 strcspn( string1, string2 ) );

16 return 0;

17 }

string1 = Değer 3.14159

string2 = 1234567890

string1’den string2’nin karakterleri

çıkarıldığında string1’in uzunluğu = 6

Şekil 8.24 strcspn kullanmak

1 /\* Şekil 8.25: fig08\_25.c

2 strpbrk kullanma \*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*string1 = "Bu bir testtir";

9 const char \*string2 = "dikkat";

341

10

11 printf( "\”\%s\”, %s ‘%c’ %s\n\”%s\” %s”,

12 string2 ,” stringindeki karakterlerden”,

13 \*strpbrk( string1, string2 ),

14 " karakteri”, string1, “de görünen ilk karakterdir.” );

15 return 0;

16 }

“dikkat” stringindeki karakterlerden

‘i’ karakteri string1’de görülen ilk karakterdir. string1=

“Bu bir testtir”

Şekil 8.25 strpbrk kullanmak

strrchr fonksiyonu, karakterin belirlenen string içindeki son konumunu bulur. Eğer karakter

bulunursa, strrchr string içindeki karakteri gösteren bir gösterici döndürür. Aksi takdirde,

strrchr NULL döndürür. Şekil 8.26, ‘z’ karakterinin “A zoooo asmnu ahgl wudhu zktr”

stringi içindeki son konumunu arayan bir program göstermektedir.

1 /\* Şekil 8.26: fig08\_26.c

2 strrchr kullanma \*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*string1 = " A zoooo asmnu ahgl wudhu zktr ";

9 int c = 'z';

10

11 printf( "’%c’%s\n\”%s\"\n",

12 c,"karakterinin son göründüğü yerden",

13 "itibaren string1: ",

14 strrchr( string1, c ) );

15 return 0;

16 }

‘z’ karakterinin son göründüğü yerden itibaren string1 :

“zktr”

Şekil 8.26 strrchr kullanmak

strspn fonksiyonu (Şekil 8.27), ilk argümanındaki stringin içinde yalnızca ikinci argümanı

olan stringin içindeki karakterlerden oluşan ilk kısmı bulur ve bu kısmın uzunluğunu

döndürür.

342

1 /\* Şekil 8.27: fig08\_27.c

2 strspn kullanma\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*string1 = "pi sayısı 3.14159";

9 const char \*string2 = "is paıy";

10

11 printf( "%s%s\n%s%s\n\n%s\n%s%u\n",

12 "string1 = ", string1, "string2 = ", string2,

13 "içerdiği string2 karakterleri bakımından ",

14 "string1’in uzunluğu = ",

15 strspn( string1, string2 ) );

16 return 0;

17 }

string1 = pi sayısı 3.14159

string2 = is paıy

içerdiği string2 karakterleri bakımından

string1’in uzunluğu = 10

Şekil 8.27 strspn kullanmak

strstr fonksiyonu, ikinci string argümanının, ilk string argümanında bulunduğu ilk konumu

arar. Eğer ikinci string ilk string içinde bulunursa, ilk argüman içindeki stringi gösteren bir

gösterici döndürülür. Şekil 8.28, strstr kullanarak “abcdefabcdef” stringi içinde “def”

stringini aramaktadır.

strtok fonksiyonu, bir stringi atomlarına ayırmak için kullanılır. Atom, boşluk ya da

noktalama karakterleri gibi sınırlayıcı bazı karakterlere kadar olan karakter serileridir.

Örneğin, bir satırda her kelime bir atom olarak ve kelimeleri ayıran boşluklar sınırlayıcı

karakter olarak düşünülebilir.

Bir stringi atomlarına ayırmak için (stringin birden fazla atom içerdiği düşünülürse) strtok

fonksiyonu birçok kez çağrılmalıdır. strtok için yapılan ilk çağrı iki argüman alır: Atomlarına

ayrılacak string ve atomları ayıran karakterleri içeren başka bir string.

Şekil 8.29’daki

atomPtr = strtok(string, “ ”);

ifadesi, atomPtr’ye string içindeki ilk atomu gösteren bir gösterici atar. strtok fonksiyonunun

ikinci argümanı (“ ”), string içindeki atomların boşluklarla ayrıldığını gösterir. strtok

fonksiyonu, string içinde sınırlayıcı karakter olmayan ilk karakteri arar. Bu, ilk atomun

başıdır. Fonksiyon daha sonra string içindeki diğer sınırlayıcı karakteri bulur ve bu karakteri

343

NULL ( ‘\0’ ) karakter ile değiştirir. Bu, o andaki atomu sonlandırır. strtok fonksiyonu, string

içindeki atomdan sonraki karakteri gösteren bir göstericiyi saklar ve o andaki atomu gösteren

bir gösterici döndürür.

1 /\* Şekil 8.28: fig08\_28.c

2 strstr kullanma\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*string1 = "abcdefabcdef";

9 const char \*string2 = "def";

10

11 printf( "%s%s\n%s%s\n\n%s\n%s%s\n",

12 "string1 = ", string1, "string2 = ", string2,

13 "string1’in içinde string2’nin karakterleriyle karşılaşıldığı ",

14 "ilk yerden itibaren string1: ",

15 strstr( string1, string2 ) );

16

17 return 0;

18 }

string1 = abcdefabcdef

string2 = def

string1’in içinde string2’nin karakterleriyle karşılaşıldığı

ilk yerden itibaren: defabcdef

Şekil 8.28 strstr kullanmak

1 /\* Şekil 8.29: fig08\_29.c

2 strtok kullanma\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 char string[ ] = "Bu 6 kelimeden oluşan bir cümledir";

9 char \*atomPtr;

10

11 printf( "%s\n%s\n\n%s\n",

12 "Kelimelerine ayrılacak string:", string,

13 "Kelimeler:" );

14

15 atomPtr = strtok( string, " " );

16

17 while ( atomPtr != NULL ) {

344

18 printf( "%s\n", atomPtr );

19 atomPtr = strtok( NULL, " " );

20 }

21

22 return 0;

23 }

Kelimelerine ayrılacak string:

Bu altı kelimeden oluşan bir cümledir

Kelimeler:

Bu

6

kelimeden

oluşan

bir

cümledir

Şekil 8.29 strok kullanımı

strtok için yapılan sonraki çağrılar, stringi atomlara bölmeye devam eder. Bu çağrılar ilk

argüman olarak NULL içerir. NULL argümanı, strtok fonksiyonuna yapılan çağrıların stringi

atomlarına ayırmaya, en son strtok fonksiyonunda saklanan konumdan devam edileceğini

belirtir. Eğer strtok çağrıldığında atom kalmamışsa, strtok NULL döndürür. Şekil 8.29,

strtok kullanarak “Bu 6 kelimeden oluşan bir cümledir” stringini atomlarına ayırır. Daha

sonra her atom ayrı ayrı yazdırılmaktadır. strtok, girilen stringi değiştirmektedir. Bu

sebepten, string ileride strtok çağrılarından sonra yeniden kullanılacaksa, stringin bir kopyası

oluşturulmalıdır.

8.9 STRING KÜTÜPHANESİNDEKİ HAFIZA FONKSİYONLARI

Bu kısımda tanıtılan string kütüphanesi fonksiyonları, hafıza bloklarını kullanma,

karşılaştırma ve arama işlemlerini gerçekleştirmektedir. Fonksiyonlar, hafıza bloklarına

karakter dizileri olarak davranmakta ve böylece veri bloklarını yönetebilmektedir. Şekil 8.30,

string kütüphanesindeki hafıza fonksiyonlarını özetlemektedir. Fonksiyon tanımlarındaki

“nesne” veri bloğu anlamına gelmektedir.

Fonksiyon Prototipi Fonksiyon Tanımı

void \* memcpy(void \*s1,const void \*s2,size\_t n); s2 ile gösterilen nesneden n

karakteri s1 ile gösterilen nesneye

kopyalar.Sonuçta, oluşan nesneyi

gösteren bir gösterici döndürülür.

void \*memmove(void \*s1,const void \*s2,size\_t n); s2 ile gösterilen nesneden n

karakteri s1 ile gösterilen nesneye

kopyalar.Kopyalama işlemi, s2 ile

gösterilen nesnedeki karakterler

önce geçici bir diziye kopyalanıp

345

daha sonra da bu geçici diziden s1

ile gösterilen nesneye

kopyalanıyormuş gibi yapılır.

Sonuçta, oluşan nesneyi gösteren

bir gösterici döndürülür.

int memcmp(const void \*s1,const void \*s2,size\_t n); s1 ve s2 ile gösterilen nesnelerin

ilk n karakterlerini karşılaştırır.

Fonksiyon, s1 s2’ye eşitse 0,s1

s2’den küçükse 0’dan küçük,s1

s2’den büyükse 0’dan büyük bir

değer döndürür.

void \*memchr(void \*s,int c,size\_t n); s ile gösterilen nesne içinde c ’in

(unsigned char’a dönüştürülür) ilk bulunduğu konumu belirler.Eğer c bulunursa, nesne

içindeki c ’in konumunu gösteren bir gösterici döndürülür.Aksi takdirde, NULL

döndürülür.

void \*memset(void \*s,int c,size\_t n); s ile gösterilen nesnenin ilk n

karakterine, c(unsigned int’e

dönüştürülür) kopyalar.Sonucu

gösteren bir gösterici döndürülür.

Şekil 8.30 string kütüphanesindeki hafıza fonksiyonları

Bu fonksiyonların gösterici parametreleri void\* olarak bildirilmiştir. 7.ünitede herhangi bir

tipteki bir göstericinin void\* tipte bir göstericiye ve void\* tipte bir göstericinin herhangi bir

tipteki göstericiye doğrudan atanabileceğini görmüştük. Bu sebepten, bu fonksiyonlar her

hangi bir tipte veriyi gösteren göstericileri kullanabilir. void\* göstericilerin gösterdiği

nesnelere erişilemediğinden, her fonksiyon işleyeceği byte (karakter) sayısını belirten bir

boyut argümanı alır. Kolaylık olması için bu kısımda örnekler karakter dizilerini (karakter

bloklarını) yönetmektedir.

memcpy fonksiyonu, ikinci argümanıyla belirtilen nesneden aldığı belli sayıdaki karakteri ilk

argümanıyla gösterilen nesneye kopyalar. Fonksiyon, her tipte nesne için gösterici alabilir.

Fonksiyonun sonucu, eğer iki nesne hafızada çakışıyorsa (aynı nesnenin kısımlarıysa)

belirsizdir. Şekil 8.31, memcpy kullanarak s2 dizisindeki stringi s1 dizisine kopyalar.

1 /\* Şekil 8.31: fig08\_31.c

2 memcpy kullanımı\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 char s1[ 16], s2[ ] = "Stringi kopyala";

9

10 memcpy( s1, s2, 16);

11 printf( "%s\n%s\"%s\"\n",

12 "s2, s1’e memcpy ile kopyalandıktan sonra",

13 "s1 in içeriği: ", s1 );

346

14 return 0;

15 }

s2, s1’e memcpy ile kopyalandıktan sonra

s1 in içeriği: “Stringi kopyala”

Şekil 8.31 memcpy kullanmak.

memmove fonksiyonu, memcpy fonksiyonu gibi ikinci argümanıyla belirtilen nesneden

aldığı belli sayıdaki karakteri, ilk argümanıyla gösterilen nesneye kopyalar. Kopyalama

işlemi, ikinci argümandaki byte’lar önce geçici bir karakter dizisine kopyalanıp daha sonra da

bu geçici diziden ilk argümana kopyalanıyormuş gibi yapılır. Bu, stringin bir kısmındaki

karakterlerin aynı stringin başka bir kısmına kopyalanabilmesini sağlar.

Genel Programlama Hataları 8.8

memmove haricinde kopyalama yapan diğer string yönetme fonksiyonları, kopyalama işlemi aynı

stringte yapıldığında tanımlanmamış sonuçlar verir.

Şekil 8.32, memmove kullanarak x dizisinin son 10 byte’ını x dizisinin ilk 10 byte’ına

kopyalar.

1 /\* Şekil 8.32: fig08\_32.c

2 memmove kullanımı\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 char x[ ] = "Evim Tatlı Evim";

9

10 printf( "%s%s\n",

11 "memmove’dan önce x dizisi içindeki string: ", x );

12 printf( "%s%s\n",

13 "memmove’dan sonra x dizisi içindeki string: ",

14 memmove( x, &x[ 5 ], 10 ) );

15

16 return 0;

17 }

memmove’dan önce x dizisi içindeki string: Evim Tatlı Evim

memmove’dan sonra x dizisi içindeki string: Tatlı Evim Evim

Şekil 8.32 memmove kullanmak

347

memcmp fonksiyonu (Şekil 8.33), ilk argümanındaki belli sayıda karakteri ikinci

argümanıyla karşılaştırır. Fonksiyon, ilk argüman ikinci argümandan büyükse 0’dan büyük bir

değer, ilk argüman ikinci argümana eşitse 0 ve ilk argüman ikinci argümandan küçükse 0’dan

küçük bir değer döndürür.

memchr fonksiyonu, bir byte’ın (unsigned char olarak temsil edilir) bir nesnenin belli

sayıdaki byte’ı içindeki ilk konumunu arar. Eğer byte bulunursa, nesne içindeki byte’ı

gösteren bir gösterici, byte bulunamazsa NULL gösterici döndürülür. Şekil 8.34, “Bu bir

stringtir” stringi içinde ‘r’ byte’ını (karakterini) arar.

1 /\* Şekil 8.33: fig08\_33.c

2 memcmp kullanımı\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 char s1[ ] = "ABCDEFG", s2[ ] = "ABCDXYZ";

9

10 printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",

11 "s1 = ", s1, "s2 = ", s2,

12 "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),

13 "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),

14 "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );

15 return 0;

16 }

s1 = ABCDEFG

s2 = ABCDXYZ

memcmp(s1, s2, 4) = 0

memcmp(s1, s2, 7) = -19

memcmp(s2, s1, 7) = 19

Şekil 8.33 memcmp kullanmak

1 /\* Şekil 8.34: fig08\_34.c

2 memchr kullanımı \*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*s = "Bu bir stringtir";

9

10 printf( "%s\'%c\'%s\"%s\"\n",

11 "s’ in ", 'r',

12 " karakterinden sonra kalanı ", memchr( s, 'r', 16 ) );

13 return 0;

14 }

348

s’ in r karakterinden sonra kalanı “r stringtir”

Şekil 8.34 memchr kullanmak

memset fonksiyonu, ikinci argümanındaki byte’ın değerini ilk argümanıyla gösterilen

nesnenin belli sayıdaki byte’ına kopyalar. Şekil 8.35, memset kullanarak string1’in ilk 7

byte’ına ‘b’ kopyalamaktadır.

1 /\* Şekil 8.35: fig08\_35.c

2 memset kullanımı\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 char string1[ 15 ] = "BBBBBBBBBBBBBB";

9

10 printf( "string1 = %s\n", string1 );

11 printf( "memset’ten sonra string1 = %s\n",

12 memset( string1, 'b', 7 ) );

13 return 0;

14 }

string1 = BBBBBBBBBBBBBB

memset’ten sonra string1 = bbbbbbbBBBBBBB

Şekil 8.35 memset kullanmak

8.10 STRING KÜTÜPHANESİNDEKİ DİĞER FONKSİYONLAR

String kütüphanesindeki son iki fonksiyon, strerror ve strlen fonksiyonlarıdır. Şekil 8.36,

strerror ve strlen fonksiyonlarını özetlemektedir.

Fonksiyon Prototipi Fonksiyon Tanımı

char \*strerror(int errornum); Sisteme bağımlı olmak üzere, hata sayısını bir

metin stringi haline dönüştürür.Stringi gösteren

bir gösterici döndürülür.

size\_t strlen(const char \*s); s stringinin uzunluğunu bulur.null karakterden

önceki karakterlerin sayısı döndürülür.

Şekil 8.36 String kütüphanesindeki diğer fonksiyonlar

strerror fonksiyonu, bir hata sayısı alır ve bir hata mesajı stringi yazdırır. Stringi gösteren bir

gösterici döndürülür.Şekil 8.37, strerror fonksiyonunu kullanmaktadır.

349

Taşınırlık İpuçları 8.4

strerror tarafından oluşturulan hata mesajı sisteme bağlıdır.

strlen fonksiyonu, argüman olarak bir string alır ve stringteki karakterlerin sayısını döndürür.

(sonlandırıcı NULL karakter uzunluğa dahil değildir) Şekil 8.38, strlen fonksiyonunu

kullanmaktadır.

1 /\* Şekil 8.37: fig08\_37.c

2 strerror kullanımı \*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 printf( "%s\n", strerror( 2 ) );

9 return 0;

10 }

No such file or directory

Şekil 8.37 strerror kullanmak

1 /\* Şekil. 8.38: fig08\_38.c

2 strlen kullanımı\*/

3 #include <stdio.h>

4 #include <string.h>

5

6 int main( )

7 {

8 const char \*string1 = "abcdefghijklmnopqrstuvwxyz";

9 const char \*string2 = "dört";

10 const char \*string3 = "Niğde";

11

12 printf("%s\"%s\" = %u\n%s\"%s\" = %u\n%s\"%s\" = %u\n",

13 "String1’in uzunluğu ", string1,

14 strlen( string1 ),

15 "String2’in uzunluğu ", string2,

16 strlen( string2 ),

17 "String3’ün uzunluğu ", string3,

18 strlen( string3 ) );

19

20 return 0;

21 }

350

String1’in uzunluğu "abcdefghijklmnopqrstuvwxyz" = 26

String2’in uzunluğu “dört” = 4

String3’ün uzunluğu “Niğde” = 5

Şekil 8.38 strlen kullanmak

ÖZET

 islower fonksiyonu, argümanının küçük harf (a-z) olup olmadığına karar verir.

 isupper fonksiyonu, argümanının büyük harf(A-Z) olup olmadığına karar verir.

 isdigit fonksiyonu, argümanının bir rakam(0-9)olup olmadığına karar verir.

 isalpha ,argümanının büyük harf(A-Z) ya da küçük harf (a-z) olup olmadığına karar

verir.

 isalnum fonksiyonu, argümanının büyük harf, küçük harf ya da rakam olup

olmadığına karar verir.

 isxdigit fonksiyonu, argümanının onaltılık sistemde rakam değeri (A-F, a-f, 0-9) olup

olmadığına karar verir.

 tolower fonksiyonu, büyük bir harfi küçük harfe çevirir ve küçük harfi döndürür.

 toupper fonksiyonu, küçük bir harfi büyük harfe çevirir ve büyük harfi geri döndürür.

isspace fonksiyonu argümanının, boşluk (‘ ‘), form besleme (‘\f’), yeni satır (‘\n’),

satır başı(‘\r’), yatay tab(‘\t’) ya da düşey tab(‘\v’) karakterlerinden herhangi biri olup

olmadığına karar verir

 iscntrl fonksiyonu argümanının, yatay tab(‘\t’), düşey tab(‘\v’), form besleme (‘\f’),

alarm(‘\a’), ters eğik çizgi (‘\b’), yeni satır(‘\n’) ya da satır başı(‘\r’) karakterlerinden

herhangi biri olup olmadığına karar verir.

 ispunct fonksiyonu, argümanının boşluk,rakam ya da harften farklı bir

karakter(örneğin,$,#,(,),[,],{,},;,:,% gibi) olup olmadığına karar verir

 isprint fonksiyonu, argümanının ekranda gösterilebilecek karakterlerden biri olup

olmadığına (boşluk karakteri de dahil olmak üzere) karar verir

 isgraph fonksiyonu, isprint ile aynı karakterleri test eder ancak boşluk karakteri test

edilecek karakterlere dahil değildir

 atof fonksiyonu , argümanını ( ondalıklı bir sayıyı temsil eden bir stringi) double

değere dönüştürür.

 atoi fonksiyonu, argümanını(bir tamsayıyı temsil eden rakamlar stringini)int bir

değere dönüştürür.

 atol fonksiyonu , argümanını (long tamsayı temsil eden rakamlar stringini)long değere

dönüştürür.

 strtod fonksiyonu, ondalıklı bir değeri temsil eden karakterleri double değere

dönüştürür. Fonksiyon iki argüman alır ;bir string (char\*) ve bu stringi gösteren bir

gösterici(char\*\*) .String, double ’a dönüştürülecek karakterleri içerir.Gösterici,

stringin dönüştürülmüş kısmından sonraki ilk karakterin konumuna atanır.

 strtol fonksiyonu, bir tamsayıyı temsil eden karakterleri long’a dönüştürür.Fonksiyon

üç argüman almaktadır ; bir string(char \*),stringi gösteren bir gösterici ve bir tamsayı.

String, dönüştürülecek karakter dizisini içermektedir.Gösterici, stringin dönüştürülen

kısmından sonraki ilk karakterin konumuna atanır.Tamsayı, dönüştürülen değerin

tabanını belirtir.

351

 strtoul fonksiyonu, unsigned long tamsayıları temsil eden karakterleri unsigned

long’a çevirir. Fonksiyon üç argüman almaktadır ; bir string(char \*),stringi gösteren

bir gösterici ve bir tamsayı. String, dönüştürülecek karakter dizisini

içermektedir.Gösterici, stringin dönüştürülen kısmından sonraki ilk karakterin

konumuna atanır.Tamsayı, dönüştürülen değerin tabanını belirtir.

 gets ,standart giriş biriminden(klavyeden) yeni satır ya da dosya sonu belirteciyle

karşılaşıncaya kadar aldığı karakterleri argümanına(char tipinde bir dizi)okur.Okuma

sona erdiğinde, dizinin sonuna null karakter(‘\0’) eklenir.

 putchar fonksiyonu, argümanı olan karakteri yazdırır

 getchar fonksiyonu, standart giriş biriminden bir karakter alıp bu karakterin tamsayı

değerini döndürür.Eğer dosya sonu belirteci girilmişse, getchar EOF döndürür.

 puts fonksiyonu, argüman olarak bir string(char\*) alır ve stringi yazdırıp sonra da

yeni satıra geçer.

 sprintf fonksiyonu, printf ile aynı dönüşüm belirteçlerini kullanarak char tipte bir

diziye formatlı bir biçimde veri yazmak için kullanır.

 sscanf fonksiyonu, scanf ile aynı dönüşüm belirteçlerini kullanarak bir diziden

biçimlendirilmiş verileri okumak için kullanır.

 strcpy fonksiyonu, ikinci argümanını(bir string) ilk argümanı içine(string ve stringle

birlikte kopyalanan sonlandırıcı null karakteri tutabilecek kadar geniş bir karakter

dizisi) kopyalar.

 strncpy fonksiyonu, strcpy fonksiyonu ile denktir. Ancak, strncpy farklı olarak

stringten diziye kopyalanacak karakterlerin sayısını belirlemektedir. Sonlandırıcı

NULL karakter yalnızca, kopyalanacak karakterlerin sayısı dizinin uzunluğundan en

az bir büyükse eklenir.

 strcat fonksiyonu, ikinci argümanını ilk argümanına eklemektedir.İkinci argümanın

ilk karakteri, ilk argümandaki stringin sonlandırıcı null karakterinin yerini alır.

Programcı, ilk stringi tutmak için kullanılan dizinin,ilk stringi,ikinci stringi ve ikinci

stringten kopyalanan null karakteri tutabilecek kadar geniş olduğundan emin

olmalıdır.

 strncat ,ikinci stringten ilk stringe belli sayıdaki karakteri ekler.Sonlandırıcı null

karakter sonuca otomatik olarak eklenir.

 strcmp fonksiyonu, ilk argümanının karakterlerini ikinci argümanını karakterleriyle

teker teker karşılaştırır.Fonksiyon eğer stringler eşitse 0 değerini,eğer ilk string ikinci

stringten küçükse negatif bir değeri,eğer ilk string ikinci stringten büyükse pozitif bir

değeri döndürür

 strncmp, strcmp’ ye benzer ancak karşılaştırma yalnızca belli sayıda karakter için

yapılır.Eğer bir stringteki karakter sayısı belirlenen karakter sayısından

küçükse,strncmp küçük stringteki null karakterle karşılaşıncaya kadar karşılaştırma

yapar.

 strchr fonksiyonu, bir karakterin bir string içindeki ilk konumunu arar. Eğer karakter

bulunursa, strchr string içindeki karakteri gösteren bir gösterici döndürür. Aksi

takdirde strchr NULL döndürür.

 strcspn fonksiyonu(Şekil 8.24), ilk argümanı olan stringin içinde ikinci argümanı olan

stringin içindeki karakterlerden oluşmayan ilk kısmı bulur ve bu kısmın uzunluğunu

döndürür.

 strbrk fonksiyonu, ikinci argümanındaki stringte yer alan herhangi bir karakterin ilk

argümanındaki stringte ilk konumunu arar.Eğer ikinci argümandaki karakter

bulunursa, strbrk ilk argümandaki karakteri gösteren bir gösterici döndürür.Aksi

takdirde, strbrk fonksiyonu NULL döndürür.

352

 strrchr fonksiyonu, karakterin belirlenen string içindeki son konumunu bulur. Eğer

karakter bulunursa, strrchr string içindeki karakteri gösteren bir gösterici döndürür.

Aksi takdirde, strrchr NULL döndürür

 strspn fonksiyonu, ilk argümanındaki stringin içinde yalnızca ikinci argümanı olan

stringin içindeki karakterlerden oluşan ilk kısmı bulur ve bu kısmın uzunluğunu

döndürür

 strstr fonksiyonu, ikinci string argümanının, ilk string argümanında bulunduğu ilk

konumu arar. Eğer ikinci string ilk string içinde bulunursa, ilk argüman içindeki

stringi gösteren bir gösterici döndürülür.

 strtok fonksiyonu için ard arda yapılan bir dizi çağrı , s1 stringini s2 stringi içinde

belirtilen karakterlerle ayrılmış atomlara ayırır.İlk çağrı s1’i ilk argüman olarak

kullanır ve sonradan yapılan çağrılar ilk argüman olarak NULL alarak, aynı stringi

atomlarına ayırmaya devam eder. Her çağrıda o andaki atomu gösteren bir gösterici

döndürülür.Eğer fonksiyon çağrıldığında daha fazla atom bulunamazsa, NULL

gösterici döndürülür.

 memcpy fonksiyonu, ikinci argümanıyla belirtilen nesneden aldığı belli sayıdaki

karakteri ilk argümanıyla gösterilen nesneye kopyalar. Fonksiyon, her tipte nesne için

gösterici alabilir.Göstericiler fonksiyona void olarak alınır ve fonksiyonun

kullanabilmesi için char göstericilere dönüştürülür.memcpy fonksiyonu nesnenin

byte’larını karakter olarak yönetir.

 memmove fonksiyonu, memcpy fonksiyonu gibi ikinci argümanıyla belirtilen

nesneden aldığı belli sayıdaki karakteri, ilk argümanıyla gösterilen nesneye kopyalar.

Kopyalama işlemi, ikinci argümandaki byte’lar önce geçici bir karakter dizisine

kopyalanıp daha sonra da bu geçici diziden ilk argümana kopyalanıyormuş gibi yapılır

 memcmp fonksiyonu, ilk argümanındaki belli sayıda karakteri ikinci argümanıyla

karşılaştırır.

 memchr fonksiyonu, bir byte’ın(unsigned char olarak temsil edilir) bir nesnenin belli

sayıdaki byte’ı içindeki ilk konumunu arar. Eğer byte bulunursa, nesne içindeki byte’ı

gösteren bir gösterici,byte bulunamazsa NULL gösterici döndürülür

 memset fonksiyonu, ikinci argümanını(unsigned char olarak kullanılır) ilk

argümanıyla gösterilen nesnenin belli sayıdaki byte’ına kopyalar

 strerror fonksiyonu, bir hata sayısı alır ve bir hata mesajı stringi yazdırır. Stringi

gösteren bir gösterici döndürülür.

 strlen fonksiyonu, argüman olarak bir string alır ve stringteki karakterlerin sayısını

döndürür. (sonlandırıcı NULL karakter uzunluğa dahil değildir)

ÇEVRİLEN TERİMLER

character constant..................................... karakter sabiti

character set................................................ karakter kümesi

general utilities library.............................. genel amaçlı kütüphane

literal......................................................... hazır bilgi

string......................................................... string / dize

token......................................................... atom

whitespace characters.............................. boşluk karakterleri

word processing...................................... kelime işleme

353

GENEL PROGRAMLAMA HATALARI

8.1 Bir karakter dizisinde, stringi sonlandıran null karakteri depolamak için gerekli alanı

ayırmamak

8.2 Sonlandırıcı null karakteri içermeyen bir stringi yazdırmak

8.3 Tek bir karakteri string olarak işlemek.Bir string bir göstericidir.(muhtemelen

oldukça büyük bir tamsayıdır) Bir karakter ise küçük bir tamsayıdır.(ASCII

değerler 0-255 arsındadır) Bir çok sistemde bu bir hataya yol açar çünkü düşük

hafıza adresleri, işletim sisteminin kesme istemleri gibi özel amaçlar için ayrılmıştır.

Bu sebepten, erişim kısıtlamaları oluşabilir.

8.4 Bir string beklenirken, fonksiyona argüman olarak bir karakter geçirmek.

8.5 Bir karakter beklenirken, fonksiyona argüman olarak string geçirmek.

8.6 strncpy’nin üçüncü argümanı, ikinci argümandaki stringin uzunluğundan küçükse ya da

ikinci argümandaki stringin uzunluğuna eşitse, ilk argümana null karakter eklememek.

8.7 strcmp ve strncmp fonksiyonlarının, argümanları eşitken 1 döndüreceğini düşünmek.İki

fonksiyonda eşitlik için 0 değerini(C’nin yanlış bir değer olarak kullandığı değer)

döndürür.Bu sebepten, iki stringin eşitliğini test ederken fonksiyonlardan döndürülen

sonuçlar 0 ile karşılaştırılmalıdır.

8.8 memmove haricinde kopyalama yapan diğer string yönetme fonksiyonları, kopyalama

işlemi aynı stringte yapıldığında tanımlanmamış sonuçlar verir.

İYİ PROGRAMLAMA ALIŞTIRMALARI

8.1 Karakter stringlerini bir karakter dizisinde tutarken,dizinin depolanacak en büyük stringi

tutabilecek kadar geniş olduğundan emin olun. C,her uzunluktaki stringlerin

depolanmalarına izin verir.Eğer bir string depolanacağı karakter dizisinden daha uzunsa,

diziden sonraki karakterler, hafızada bir sonraki dizide yer alan verilerin üzerine

yazılacaktır.

8.2 Karakter kütüphanesinden fonksiyonlar kullanırken, <ctype.h> öncü dosyasını

programınıza dahil edin.

8.3 Genel amaçlı kütüphaneden bir fonksiyon kullandığınızda, <stdlib.h> öncü dosyasını

programınıza ekleyin.

8.4 Standart giriş/çıkış kütüphanesindeki fonksiyonları kullanırken,<stdio.h> öncü dosyasını

programlarınıza ekleyin.

8.5 String kütüphanesindeki fonksiyonları kullanırken,<string.h> öncü dosyasını eklemeyi

unutmayın.

TAŞINIRLIK İPUÇLARI

8.1 char \* tipinde bir değişkene, string bilgisi atandığında bazı derleyiciler stringi hafızada

stringin değiştirilemeyeceği bir konuma yerleştirir.Eğer string bilgisini değiştirmeye

ihtiyaç duyacaksanız, stringin tüm sistemlerde değiştirilebilmesini garanti altına almak

için stringi karakter dizisine yerleştirin.

8.2 size\_t tipi, sisteme bağımlı olarak, unsigned long ya da signed int tipinin eşanlamlısıdır.

8.3 Karakterleri temsil eden nümerik kodlar her bilgisayar için farklı olabilir.

8.4 strerror tarafından oluşturulan hata mesajı sisteme bağlıdır.

354

ÇÖZÜMLÜ ALIŞTIRMALAR

8.1 Aşağıdakileri gerçekleştirecek birer ifade yazınız. c (karakter saklayacak), x, y ve z

değişkenlerinin int; d, e ve f değişkenlerinin float, ptr değişkeninin ise char\* türünde ve

s1[100], s2[200] dizilerinin bildirildiğini kabul ediniz.

a) c değişkeninde saklanan karakteri büyük harfe çeviriniz ve yine c’ ye atayınız.

b) c değişkenin içeriğinin bir rakam olup olmadığına karar veriniz. Şekil 8.2, 8.3 ve

8.4’teki koşullu operatörü kullanınız. “rakamdır.” yada “rakam değildir.” şeklinde

sonucu yazdırınız.

c) “1234567” stringini long tipine çevirip ekrana yazdırınız.

d) c değişkeninin içeriğinin bir kontrol karakteri olup olmadığına karar verin. Bir koşul

operatörü kullanarak ekrana “kontrol karakteridir.” ve ya “kontrol karakteri

değildir.” biçiminde yazdırınız.

e) Klavyeden s1 dizisine, kullanıcı “enter” tuşuna basıncaya kadar yazdığı metni scanf

kullanmadan girdiriniz.

f) s1 dizisinin içerisindeki metni printf kullanmadan ekrana yazdırınız.

g) c değişkeninin s1 içerisinde en son bulunduğu konumu ptr değişkenine atayınız.

h) c değişkenin değerini ekrana printf kullanmadan yazdırınız.

i) “8.63582” stringini double tipine çevirip ekrana yazdırınız.

j) c değişkeninin değerinin bir harf olup olmadığını karar verip, bir koşullu operatör

kullanarak “harftir” yada “harf değildir.” şeklinde ekrana yazdırınız.

k) Klavyeden bir karakter alarak bunu c değişkenine atayınız.

l) s2 ‘nin s1’de ilk bulunduğu konumu ptr değişkenine atayınız.

m) c değişkeninin içeriğinin bir yazı karakteri olup olmadığına karar verdikten sonra

“yazı karakteridir.” ya da “yazı karakteri değildir.” şeklinde ekrana yazdırınız.

n) “1.27 10.3 9.432” stringinden float tipindeki değerleri alıp d, e ve f değişkenlerine

atayınız.

o) s2 dizisindeki stringi s1 dizisine kopyalayınız.

p) s2 dizisindeki her hangi bir karakterin, s1 dizisinde ilk göründüğü konumu ptr

değişkenine atayınız.

q) s1 dizisi içerisindeki string ile s2 dizisi içersindeki stringi karşılaştırarak ekrana

sonucu yazdırınız.

r) c değişkeninin içeriğinin, s1 dizisinde ilk göründüğü konumu ptr değişkenine

atayınız.

s) x, y, z değişkenlerinin içeriğini sprintf kullanarak s1 dizisine atayınız. Her değer 7

karakterlik bir alan kaplamalıdır.

t) s2 dizisindeki 10 karakteri, s1 dizisinde ekleyiniz.

u) s1 dizisinin uzunluğunu bularak ekrana yazdırınız.

v) “-21” stringini int tipine çevirerek ekrana yazdırınız.

w) ptr ’yi s2 içindeki ilk atoma atayın.s2 içindeki atomlar virgüllerle(,) ayrılmıştır.

8.2 sesliHarf dizisine, iki farklı yolla “AEIOU” ilk değerlerini veriniz.

8.3 Aşağıdakiler eğer ekrana bir şey yazdırılıyorlarsa, ne yazdırdıklarını bulunuz ve bu

ifadeler eğer hata içeriyorlarsa bu hataları tanımlayarak düzeltiniz. Aşağıdaki değişken

bildirimlerinin yapıldığını kabul ediniz.

char s1[50] = “veli”, s2[50] = “elli”, s3[50], \*sptr;

a) printf(“%c%s”, toupper(s1[0]), &s1[1]);

355

b) printf (“%s”, strcpy(s3, s2));

c) printf (“%s”, strcat(strcat(strcpy(s3, s1), “ ve “), s2));

d) printf (“%u”, strlen(s1) + strlen (s2));

e) printf (“%u”, strlen(s3));

8.4 Aşağıdaki program parçalarındaki hataları bulup ve nasıl düzeltileceğini açıklayınız.

a) char s[10];

strncpy(s, “merhaba”, 7);

printf (“%s\n”, s);

b) printf(“%s”, ‘a’);

c) char s[12];

strcpy(s, “EveHosgeldin”);

d) if (strcmp(string, string2))

printf(“Stringler eşit\n”);

ÇÖZÜMLER

8.1

a) c = toupper(c);

b) printf(“’%c’%srakam\n”, c, isdigit(c) ? “dır “ : “ değildir”);

c) printf(“%ld\n”, atol(“1234567));

d) printf(“’%c’%skontrol karakteri\n”, c, iscntrl(c) ? “dir” : “ değildir “);

e) gets(s1);

f) puts(s1);

g) ptr = strrchr(s1, c);

h) putchar(c);

i) printf (“%f\n”, atof(“8.63582”));

j) printf (“’%c’”%sharf\n”, c, isalpha(c) ? “, dır” : “değildir”);

k) c = getchar();

l) ptr = strstr(s1, s2);

m) printf (“’%c’ %syazı karakteri\n”,c, isprintf(c) ? “dir” : “değildir”);

n) sscanf(“1.27 10.3 9.432”, “%f%f%f”, &d, &e, &f);

o) strcpy(s1, s2);

p) ptr = strpbrk(s1, s2);

q) printf(“strcmp(s1, s2) = %d\n”, strcmp(s1, s2));

r) ptr = strchr(s1, c);

s) sprintf(s1, “%7d%7d%7d”, x, y, z);

t) strncat(s1, s2, 10);

u) printf (“strlen(s1) = %u\n”, strlen(s1));

v) printf(“%d\n”, atoi(“-21”));

w) ptr = strtok(s2, “,”);

8.2

char sesliHarf[] = “AEIOU”;

char sesliHarf[] = “{‘A’, ‘E’, ‘I’, ‘O’, ‘U’, ‘\0’};

8.3

a) Veli

356

b) elli

c) veli ve elli

d) 8

e) 12

8.4

a) Hata: strncpy fonksiyonu, üçüncü argümanının “merhaba” stringinin uzunluğuna

eşit olduğu için NULL karakteri yazmaz.

Düzeltme: strncpy fonksiyonunun üçüncü argümanının 8 yapılması ya da s[7] ‘ye 0

atanması.

b) Hata: Bir karakter sabitini string gibi yazmaya çalışılması.

Düzeltme: çıktı karakteri için %c kullanmak ya da ‘a’ nın “a” ile değiştirilmesi

c) Hata: s karakter dizisi NULL karakteri de içerecek kadar büyük değildir.

Düzeltme: Daha fazlı elemanı olan bir dizinin bildirilmesi

d) Hata: strcmp fonksiyonu eğer stringler birbirine eşitse, 0 döndürür. if yapısındaki

koşul yanlıştır ve printf çalıştırılmaz.

Düzeltme: koşulda strcmp’nin sonucunun 0 ile karşılaştırılması

ALIŞTIRMALAR

8.5 Klavyeden bir karakter alıp bu karakteri, karakter kütüphanesindeki bütün

fonksiyonları kullanarak test eden bir program yazınız. Program fonksiyondan döndürülen

değerleri ekrana yazdırılmalıdır.

8.6 “Enter” e basılıncaya kadar s[100] karakter dizisine metin girişi yaptıran bir program

yazınız. Metni ekrana sadece küçük harfler ve sadece büyük harfler kullanarak yazdırınız.

8.7 Tamsayıları temsil eden 4 stringi klavyeden alan bir program yazınız. Programınız bu

stringleri tam sayıya çevirdikten sonra toplayarak, toplamı ekrana yazdırsın.

8.8 Ondalıklı sayıları temsil eden 4 stringi klavyeden alan bir program yazınız.

Programınız bu stringleri tam sayıya çevirdikten sonra toplayarak, toplamı ekrana yazdırsın.

8.9 strcmp fonksiyonunu kullanarak, kullanıcının girdiği iki stringi karşılaştıran bir

program yazınız. Programınız, ilk stringin ikincisine eşit ya da ikincisinden küçük veya büyük

olduğuna karar vermelidir.

8.10 strncmp fonksiyonunu kullanarak, kullanıcının girdiği iki stringi karşılaştıran bir

program yazınız. Kaç karakterin karşılaştırılacağı da kullanıcı tarafından girilsin.

Programınız, ilk stringin ikincisine eşit ya da ikincisinden küçük veya büyük olduğuna karar

vermelidir.

8.11 Rasgele sayı üreterek cümle oluşturan bir program yazınız. Programınız, nesne, isim,

fiil, edat isminde dört char tipinde gösterici dizisi kullansın. Programınız isim, nesne, edat,

fiil sırasıyla bu dizilerden rasgele bir kelime seçerek bir cümle oluşturmalıdır. Bir kelime

seçildiğinde dizideki diğer kelimelerin sonuna eklenmelidir ve kelimeler birbirlerinden

boşluklarla ayrılmalıdırlar. Programınız bu şekilde 20 cümle oluşturmalı.

357

8.12 (5 mısralı şiir). Alıştırma 8.11’ de geliştirdiğiniz teknikleri kullanarak, birinci ve iki

satırı beşinci satırla, üçüncü ve dördüncü satırların ise kendi aralarında kafiyeli olduğu şiirler

üreten bir program yazınız. Bir program ile güzel şiirler yazmak zordur ama sonucuna değer.

8.13 İngilizce bir paragrafı pig Latin diline çeviren bir program yazınız. pig Latin dili

genellikle eğlence için kullanılır. Bu forma çeviri için çeşitli yollar bulunmaktadır. Basitçe

aşağıdaki algoritmayı uygulayabilirsiniz.

Bu çeviride strtok fonksiyonunu kullanmalısınız. İngilizce bir kelimeyi bu forma çevirmek

için İngilizce kelimenin ilk harfini kelimenin sonuna taşıyın ve en sona “ay” harflerini

ekleyin. Örneğin “jump” kelimesi “umpjay” kelimesine, “the” kelimesi “hetay” kelimesine

“computer” kelimesi ise “omputercay” kelimesine dönüşmelidir. Bu paragrafın içerdiği

kelimelerin boşluklarla birbirinden ayrıldığını ve hiç bir noktalama işaretinin kullanılmadığını

kabul ediniz. latinKelimeler fonksiyonunu kullanarak bütün kelimeleri ekrana

yazdırınız.(İpucu:strtok için yapılan çağrıda bir atom bulunduğunda,bu atomu gösteren

göstericiyi latinkelimeler fonksiyonuna geçiriniz ve pig Latin kelimeyi yazdırınız)

8.14 Kullanıcıdan (555) 555-5555 formunda bir telefon numarasını string şeklinde alan bir

program yazınız. Program, strtok fonksiyonunu kullanarak alan kodunu bir atom, ilk üç

rakamı bir atom ve son dört rakamı ayrı bir atom olarak açmalıdır. Telefon numarasının yedi

rakamıyla bir string oluşturunuz. Programınız, alan kodunu int tipine, ve geri kalan

numaraları da long tipine çevirerek ekrana yazmalıdır.

8.15 Kullanıcının “enter” e basılıncaya kadar yazdığı metni alan ve bunu strtok

fonksiyonu ile atomlara ayırdıktan sonra atomları ters çevirerek ekrana yazdıran bir program

yazınız.

8.16 Kullanıcının “enter” e basılıncaya kadar yazdığı metni ve bir arama stringini

klavyeden alan bir program yazınız. strstr fonksiyonu kullanarak, aranan stringin ilk

göründüğü konumu metinde buldurun ve bu konumu char \* tipindeki araPtr değişkenine

atayın. Eğer aranan string bulunursa, stringin bulunduğu satırın geri kalanını arama stringi en

başa yazılmış şekilde yazdırın. Daha sonra strstr fonksiyonunu kullanarak arama stringinin

metinde ikinci kez göründüğü konumu buldurun. Eğer arama stringi ikinci kez

bulunursa,stringin bulunduğu satırın geri kalanını stringin ikinci görünümü başta olmak üzere

ekrana yazdırın.. İpucu: strstr fonksiyonunu ikinci çağrısı ilk argüman olarak araPtr + 1 ‘i

içermeli.

8.17 Alıştırma 8.16’daki programı baz alarak, kullanıcının “enter” e basıncaya dek girdiği

metnin içinde yine kullanıcı tarafından girilen arama stringini arayan ve strstr

fonksiyonunun kullanımıyla bu metnin arama stringini kaç kez içerdiğini hesaplayan bir

program yazınız.

8.18 Kullanıcının “enter” e basıncaya dek girdiği metin içerisinde yine kullanıcı tarafından

girilen bir arama karakterinin kaç kez bulunduğunu strchr fonksiyonunu yardımıyla bulan bir

program yazınız.

358

8.19 Alıştırma 8.18’ de yazdığınız programı baz alarak, girilen metinde alfabenin her

harfinin

kaç kez bulunduğunu strchr fonksiyonunun yardımıyla bulan bir program yazınız. Büyük ve

küçük harfleri beraber sayın. Sonuçları bir dizide saklayarak ekrana düzgün bir çizelge

şeklinde yazdırınız.

8.20 Kullanıcının “enter”e basıncaya dek girdiği metinde kaç kelime olduğunu strtok

fonksiyonu yardımıyla bulan bir program yazınız. Kelimelerin boşluk ya da yeni satır

karakteri ile birbirlerinde ayrıldığını kabul ediniz.

8.21 Bölüm 8.6’da anlatılan string karşılaştırma yöntemleri ve 6. Ünitede anlatılan dizi

sıralama tekniklerini kullanarak bir string listesini sıralayan bir program yazınız.

Programınıza veri olarak bulunduğunuz şehirdeki 10-15 köy ismini kullanın.

8.22 Ek D’deki tablo, karakterlerin ASCII karşılıklarını içermektedir. Bu tablo üzerinde

çalışın ve aşağıdakilerin hangilerinin doğru, hangilerinin yanlış olduğuna karar verin.

a) “A” harfi “B” harfinden önce gelir.

b) “9” rakamı “0” rakamından önce gelir.

c) toplama, çıkarma, çarpma ve bölme işlemlerinde kullanılan semboller rakamlardan

önce gelir.

d) Rakamlar, harflerden önce gelir.

e) Eğer bir sıralama programı stringleri artan bir sırada sıralıyorsa sağ parantezi, sol

parantezden daha önce bir sırada yerleştirir.

8.23 String serilerini okuyan ve “b” harfi ile başlayanları yazdıran bir program yazınız.

8.24 String serilerini okuyan ve “ED” harfleriyle bitenlerini ekrana yazdıran bir program

yazınız.

8.25 Kullanıcıya bir ASCII kod girişi yaptıran ve ilgili karakteri ekrana yazdıran bir

program yazınız. Bu program 000-255 arasındaki 3 basamaklı bütün ASCII kodlarını alıp

karşılığındaki karakterleri ekrana yazdıracak şekilde değiştiriniz.

8.26 Ek D’deki ASCII karakter tablosundan yardım alarak Şekil 8.1’deki karakter

kütüphanesi fonksiyonlarının size ait sürümlerini yazınız.

8.27 Şekil 8.5’ teki stringleri sayılara çeviren fonksiyonların kendinize ait sürümlerini

yazınız.

8.28 Şekil 8.17’ deki string kopyalama ve ekleme fonksiyonlarının kendinize ait

sürümlerini yazınız. İlk sürüm dizi belirteç yöntemiyle, ikinci ise göstericileri ve gösterici

aritmetiğini içermeli.

8.29 getchar, gets, putchar ve puts fonksiyonlarının kendinize ait sürümlerini yazınız.

8.30 Şekil 8.20’ deki string karşılaştırma fonksiyonlarının iki ayrı sürümünü yazınız. İlk

sürüm dizi belirteç yöntemiyle, ikinci ise göstericileri ve gösterici aritmetiğini içermeli.

8.31 Şekil 8.22 deki string arama fonksiyonları için kendi sürümlerinizi yazınız.

359

8.32 Şekil 8.30’daki hafıza yönetimi ile ilgili fonksiyonların size ait sürümlerini yazınız.

8.33 Şekil 8.36’daki strlen fonksiyonu için kendi sürümünüzü yazınız. İlk sürüm dizi

belirteç yöntemiyle, ikinci ise göstericileri ve gösterici aritmetiğini içermeli.

ÖZEL BÖLÜM: İLERİ DÜZEYDE STRING İŞLEME ALIŞTIRMALARI

Bundan sonraki alıştırmalar okuyucunun string kullanma becerisini test edecektir. Bu bölüm

ileri düzeyde alıştırmalar içermektedir. Okuyucu, bu problemleri zor ama eğlenceli bulacaktır.

Problemler zorluk seviyeleri bakımından çeşitlilik göstermektedir. Bazılarını çözmede bir

saate yada iki ayrı program yazmaya ihtiyaç duyulur. Diğerleri ise iki yada üç hafta sürecek

laboratuar çalışmalarında gerek duyar. Bir kısmı ise oldukça zor projeler olarak ele alınabilir.

8.34 (Metin analizi) Bilgisayarların string uygulama alıştırmalarında yüksek kapasitelere

sahip olmaları, bazı büyük yazarların yazılarına değişik yaklaşımlarda bulunulmasına yol

açmıştır. En çok William Shakespeare üzerine dikkat çekilmiştir. Bazı dil bilimcilerin

Christopher Marlowe’un yazılarının Shakespeare’ın yazılarına benzerliği konularında ciddi

kanıtları vardır. Araştırmacılar bu iki yazarın yazılarındaki benzerlikleri bulmak için

bilgisayar kullanmışlardır. Bu alıştırma, metinlerin bilgisayarla analizi ile ilgili üç metot

içermektedir.

a) Bir kaç satırlık bir metni okuyan ve bu metinde her harfin kaç kez içerildiğini

hesaplayan bir program yazınız. Örneğin

Olmak, ya da olmamak: işte bütün mesele bu:

paragrafı 5 adet “a”, 2 adet “k” içermektedir.

b) Bir kaç satırlık bir metni okuyan ve bu metin içersinde iki harften, üç harfen, dört

harften vb. oluşan kaçar kelime olduğunu hesaplayan bir program yazınız. Örneğin

Bu soylunun acı çekme niyetinde olması ne fark eder

cümlesi aşağıdakileri içerir.

Kelime Uzunluğu Kaç kez içerildiği

1 0

2 2

3 1

4 2

5 1

6 1

7 0

8 1

9 1

360

c) Bir kaç satırlık bir metni okuyan ve bu metinde her kelimenin kaç kez bulunduğunu

hesaplayan bir program yazınız. Programın ilk sürümü kelimeleri metinde

bulunduğu sırada ekrana yazdırırken daha kullanışlı olan ikinci sürümü ise alfabetik

sırada sıralama yapacak şekilde yazılmalıdır. Örneğin aşağıdaki paragrafta “Bu”

kelimesi iki kez bulnmuştur.

Olmak, ya da olmamak: işte bütün mesele bu:

Bu soylunun acı çekme niyetinde olması ne fark eder

8.35 (Kelime işleme) Bu metindeki string kullanma, gittikçe gelişen kelime işleme için çok

iyi bir örnektir. Kelime işleme sistemlerindeki önemli bir fonksiyon olan ayarla fonksiyonu,

kelimelerin sayfanın sağ veya solundan girintileri ayarlar. Bu sistemle metin, bir daktilo

yazısına nazaran daha düzenli görünmesini sağlar. Bu fonksiyon, metnin sağdan ve soldan

girintilerinin eşit olmasını kelimeler arasına bir veya iki boşluk karakteri koyarak düzenler.

Bir kaç satırdan oluşan bir metinde bu ayarlamayı yapan bir program yazınız. Metnin bir

A4 kağıdına basılacağını ve kağıdın her iki tarafından da ikişer santim boşluk(girinti)

olacağını kabul ediniz.

8.36 (Tarihlerin çeşitli biçimlerde yazılması) Tarihler iş yazışmalarında farklı yollarla

yazılabilir. Aşağıda iki farklı form yazılmıştır.

07/21/55 ve Haziran 21, 1955

Tarihi yukarıdaki biçimlerin birincisi gibi alıp, ikincisine çeviren bir program yazınız.

8.37 (Çek koruma) Bilgisayarlar, maaşlar ya da bazı hesapların ödenmesinde çek yazmak için

sıklıkla kullanılır. 1 milyon dolarlık haftalık ödeme çeklerinin yazıldığı(hata yapılarak)

hakkında değişik hikayeler söylenmektedir. Bilgisayar tarafından insan hatasından yada

bilgisayarın hatasından dolayı yanlış miktarlarda çekler yazılabilir. Sistem tasarımcıları,

sistemlerinin hata yapmasını engellemek için ellerinden geleni yaparlar. Başka bir ciddi

problem ise birisinin kasıtlı olarak çek üzerinde oynama yaparak o çeki bozdurmak

istemesidir. Bir hesapta bu şekilde değişiklikler yapılamaması için bilgisayarlı çek yazma

sistemleri, çek koruma isminde bir teknik kullanırlar. Bilgisayar tarafından basılacak olan

çek, bilgisayarın miktarı yazabilmesi için belli bir sayıda boşluk içerir. Bilgisayarın haftalık

bir ödemeyi yapmak için basacağı bir ödeme çekinin sekiz boşluk karakteri içerdiğini kabul

ediniz. Eğer para miktarı fazla ise 8 boşlukta kullanılmalıdır.. Örneğin :

1 , 2 3 0 . 6 0 (çek miktarı)

- - - - - - - -

1 2 3 4 5 6 7 8 (konum numaraları)

Diğer tarafta eğer para miktarı 1000$ dan daha küçük ise bir kaç boşluk bırakılmalıdır.

Örneğin:

9 9 . 8 7

- - - - - - - -

1 2 3 4 5 6 7 8

bu miktar 3 boşluk içermektedir. Eğer bu çek, boşluk ile yazılacak olursa bir başkası

tarafından kolaylıkla değiştirilebilir. Bu yüzden bir çok sistem boşluk yerine aşağıdaki gibi

361

çeke asteriks(\*) ekler.

\* \* \* 9 9 . 8 7

- - - - - - - -

1 2 3 4 5 6 7 8

Dolar olarak çeke yazılacak olan miktarı kullanıcıdan alan ve eğer gerekirse asterisk karakteri

kullanarak bu miktarı çek koruma formunda yazan bir program yazınız. Para miktarını

yazmada dokuz karakter kullanacağınızı kabul ediniz.

8.38 (Çek miktarının yazı ile yazılması) Yukarıdaki çek koruma tekniğini devam ettirecek

olursak diğer bir yöntem ise çek miktarının hem sayıyla hem de okunuşunun yazı ile

yazılmasıdır. Eğer birisi çek miktarını değiştirmeye kalkışacak olursa yazı ile yazılan kısmını

değiştirmekte oldukça zorlanacaktır.

Bir çok bilgisayarlı çek yazma sistemleri çek miktarını yazı ile yazmaz. Belki bunun ana

sebebi bir çok yüksek seviyeli dilin ileri düzeyde string işleme özelliklerini içermemesidir. Bir

diğer sebepte çek miktarlarının yazı ile yazılma mantığının karışık olmasıdır.

Çek miktarının sayısal değerini alıp yazı ile yazan bir program yazınız. Örneğin 112.43

aşağıdaki gibi yazılmalıdır.

YÜZONİKİ ve 43/100

8.39 (Morse Alfabesi) Belki de en meşhur kod şeması, 1832 yılında Samuel Morse tarafından

telgraf sisteminde kullanılmak için geliştirilen Morse alfabesidir. Morse alfabesi, her harf, her

rakam ve bazı özel karakterler için(virgül, noktalı virgül vb.) noktalardan ve çizgilerden

oluşan kodlar içerir. Ses bazlı sistemlerde nokta kısa sese, çizgi ise uzun sese karşılık gelir.

Diğer gösterimler ise ışığa ve sisteme dayalı sistemlerdir.

Kelimelerin birbirinden ayrılması boşlukla ya da nokta veya çizgiye yer verilmemesiyle olur.

Ses bazlı sistemlerde ise kelimelerin birbirlerinden ayrılması hiç ses iletilmemesiyle yapılır.

Uluslararası Morse alfabesi şekil 8.39 da görülmektedir.

İngilizce bir paragrafı okuyan ve bu paragrafı Morse alfabesine çeviren bir program yazınız.

Ayrıca Morse alfabesinde yazılmış bir paragrafı İngilizce’ye çeviren bir programda yazınız.

Morse Alfabesi ile yazılmış her harf arasında bir boşluk ve her kelime arasında üç boşluk

kullanın.

8.40 (Metrik Çevirme Programı) Kullanıcıya metrik sistemlerin birbirlerine çevrilmesinde

yardımcı olacak bir program yazınız. Programınız kullanıcının birimleri string olarak

belirtmesine izin vermelidir.(örneğin, metrik sistem için santimetre, litre, gram vb ve İngiliz

sistemi için inç, pound vb.) Programınız aşağıdaki gibi sorulara cevap vermelidir.

“2 metre kaç inç yapar?”

“10 kuart kaç litre yapar?”

Programınız yanlış soruları da anlamalıdır. Örneğin

362

“5 kilogram kaç feet yapar?”

sorusu anlamsızdır, çünkü “feet” bir uzunluk, “kilogram” ise ağırlık birimidir.

Karakter Kod Karakter Kod

A .- T -

B -... U ..-

C -.-. V ...-

D -.. W .--

E . X -..-

F ..-. Y -.--

G --. Z --..

H ....

I .. Rakamlar

J .--- 1 .----

K -.- 2 ..---

L .-.. 3 ...--

M -- 4 ....-

N -. 5 .....

O --- 6 -....

P .--. 7 --...

Q --.- 8 ---..

R .-. 9 ----.

S ... 0 -----

Şekil 8.39 Uluslararası Morse alfabesi

8.41 (İhbarname) Bir çok iş yeri, vadesi geçmiş borçları toplamak için oldukça fazla zaman

harcamaktadırlar. İhbarname, borç toplamak amacıyla borcu olanlara ısrarlı bir şekilde çağrı

yapılmasıdır.

Bilgisayarlar ihbarname yazmada sıklıkla kullanılır çünkü borç konusunda sorunlar gittikçe

artmaktadır. Bir teori derki borç eskidikçe, toplanması zorlaşır.

Beş ihbarnamenin metnini içeren bir program yazınız. Programınız girdi olarak aşağıdakileri

almalıdır.

1. Borçlunun ismi

2. Borçlunun adresi

3. Borçlunun hesabı

d) Borç miktarı

e) Borcun vadesi

363

ZOR BİR STRING İŞLEME PROJESİ

8.42 (Çapraz bulmaca) Bir çok insan çapraz bulmaca çözmekle uğraşmıştır ama çok azı

çapraz bulamaca yaratmıştır. Bir çapraz bulmaca yapmak oldukça zordur ve uğraş gerektirir.

Programcının çalışan bir çapraz bulmaca programı yazabilmesi için bir çok konuyu çok iyi

çözmüş olması gerekmektedir. Örneğin bulmacadaki ızgaranın her bir karesi ne ile ifade

edilmelidir? string serileri mi yoksa iki boyutlu diziler mi kullanılmalıdır? Programcı,

programın anlayabileceği ve kullanabileceği bir kelime kaynağına(sözlük) ihtiyaç duyacaktır.

Program string uygulamaları için bu kelimeleri hangi biçimde saklamalıdır? Gerçekten hırslı

bir okuyucu bulmacanın “ipuçları” kısmını “çapraz” ve “aşağı” ifadeler için özet ipuçları

içerecek şekilde yaratmak isteyecektedir. Sadece boş bir bulmaca oluşturarak yazdırmak basit

bir problem değildir.

364

BİÇİMLENDİRİLMİŞ GİRİŞ/ÇIKIŞ

AMAÇLAR

 Giriş ve çıkış akışlarını (akış) anlamak

 Bütün yazdırma biçimlendirme yeteneklerini kullanabilmek

 Bütün giriş biçimlendirme yeteneklerini kullanabilmek

BAŞLIKLAR

9.1 GİRİŞ

9.2 AKIŞLAR ( STREAM )

9.3 printf İLE ÇIKIŞI BİÇİMLENDİRMEK

9.4 TAMSAYILARI YAZDIRMAK

9.5 ONDALIKLI SAYILARI YAZDIRMAK

9.6 STRING VE KARAKTERLERİ YAZDIRMAK

9.7 DİĞER DÖNÜŞÜM BELİRTEÇLERİ

9.8 ALAN GENİŞLİĞİ VE DUYARLIK İLE YAZDIRMAK

9.9 printf BİÇİM-KONTROL DİZESİNDE BAYRAKLARI KULLANMAK

9.10 ÇIKIŞ DİZİLERİNİ VE HAZIR BİLGİLERİ( LITERAL) YAZDIRMAK

9.11 scanf İLE GİRİŞİ BİÇİMLENDİRMEK

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Performans İpuçları\*

Taşınırlık İpuçları\*Yazılım Mühendisliği Gözlemleri\*Çözümlü Alıştırmalar\* Çözümler\*

Alıştırmalar

9.1 GİRİŞ

Bir problemin çözümünün önemli kısımlarından biri de sonuçların gösterimidir. Bu ünitede,

scanf ve printf’in biçimlendirme yeteneklerini detaylı bir şekilde inceleyeceğiz. Bu

fonksiyonlar sırasıyla, verileri standart giriş akışından alırlar ve veriyi standart çıkış

akışından çıkartırlar. Standart giriş ve standart çıkışı kullanan diğer dört fonksiyon, 8.ünitede

anlattığımız gets, puts, getchar ve putchar fonksiyonlarıdır. Bu fonksiyonları çağıran

programlara <stdio.h> eklenmelidir.

printf ve scanf ’in bir çok özelliği, daha önceki ünitelerde anlatılmıştı. Bu ünite, o özellikleri

özetlemekte ve yeni bir çok özellik tanıtmaktadır. 11. ünite, standart giriş/çıkış (stdio)

kütüphanesindeki diğer bir çok fonksiyonu açıklamaktadır.

9.2 AKIŞLAR (STREAM)

Tüm giriş ve çıkış, akışlar ( satırları oluşturmak için organize edilmiş karakter dizileri) ile

yapılır. Her satır, 0 ya da daha fazla karakter içerir ve yeni satır karakteriyle sonlanır. C

standardı, C uygulamalarının sonlandırıcı yeni satır karakteri dahil olmak üzere en az 254

karakterlik satırları desteklemesi gerektiğini belirtmiştir.

365

Program çalışmaya başladığında, programa otomatik olarak üç akış bağlanır. Normalde,

standart giriş akışı klavyeye ve standart çıkış akışı ekrana bağlanır. İşletim sistemleri sıklıkla,

bu akışların başka cihazlara yönlendirilmesine izin verir. Üçüncü akış, standart hata, ekrana

bağlanır. Hata mesajları, standart hata akışından çıkartılır. Akışlar, 11.ünitede daha detaylı

anlatılmıştır.

9.3 printf İLE ÇIKIŞI BİÇİMLENDİRMEK

Kesin çıktıların biçimlendirmesi, printf ile yapılır. Her printf çağrısı, çıkış biçimini belirten

bir biçim kontrol dizesi içerir. Biçim kontrol dizesi, dönüşüm belirteçleri, bayraklar, alan

genişlikleri, duyarlık ve bilgi karakterlerinden oluşur. Bunlar, yüzde işareti (%) ile birlikte

dönüşüm tarifini oluştururlar. printf fonksiyonu, her biri bu ünitede anlatılmış, aşağıdaki

biçimlendirme yeteneklerini gerçekleştirebilir:

1.Ondalıklı sayıları istenen basamağa kadar yuvarlamak.

2.Bir sütun sayıyı, ondalıklı kısımları aynı konumdan başlayacak şekilde hizalamak.

3.Çıktıları sağa dayamak ve sola dayamak

4.Bir satır çıktının istenen konumlarına bilgi karakterleri yerleştirmek.

5.Ondalıklı sayıları üssel biçimde ifade etmek.

6.İşaretsiz tamsayıları sekizlik ve onaltılık biçimde temsil etmek. Sekizlik ve onaltılık

değerler hakkında daha fazla bilgi için Ekler E kısmına bakınız.

7.Her tipteki veriyi istenen alan genişliği ve duyarlıkla yazdırmak.

printf fonksiyonu aşağıdaki biçime sahiptir:

printf ( biçim kontrol dizesi,diğer argümanlar);

Biçim kontrol dizesi, çıktının biçimini tanımlar ve diğer argümanlar (bunlar tercihe bağlıdır),

biçim kontrol dizesindeki her dönüşüm tarifinin karşılığıdır. Her dönüşüm tarifi bir yüzde

işareti ile başlar ve bir dönüşüm belirteci ile sonlanır. Bir biçim kontrol dizesinde birden fazla

dönüşüm tarifi bulunabilir.

Genel Programlama Hataları 9.1

Biçim kontrol dizesini tırnak içine almayı unutmak.

İyi Programlama Alıştırmaları 9.1

Çıktıları gösterim için düzgün bir şekilde yazın. Bu, programı daha okunabilir yapar ve

kullanıcı hatalarını azaltır.

9.4 TAMSAYILARI YAZDIRMAK

Bir tamsayı, 776 ya da –52 gibi, ondalık kısım içermeyen bir sayıdır. Tamsayı değerleri bir

çok biçimden biri ile yazdırılırlar. Şekil 9.1, tamsayı dönüşüm belirteçlerini açıklamaktadır.

Dönüşüm Belirteci Açıklama

d İşaretli bir tamsayıyı(onluk sistemde) gösterir.

i İşaretli bir tamsayıyı(onluk sistemde) gösterir.

(Not: i ve d yalnızca scanf ile kullanıldıklarında farklıdır)

o İşaretsiz bir tamsayıyı(sekizlik sistemde) gösterir.

366

u İşaretsiz bir tamsayıyı(onluk sistemde) gösterir.

x ya da X İşaretsiz bir tamsayıyı(onaltılık sistemde) gösterir.X, 0-9

rakamlarının ve A-F harflerinin gösterilmesini ve x, 0-9

rakamları ve a-f harflerinin gösterilmesini sağlar.

h ya da l (l harfi) Herhangi bir tamsayı belirtecinden önce kullanıldığında

sırasıyla, short ya da long bir tamsayının gösterileceğini

belirtir.

Şekil 9.1 Tamsayı dönüşüm belirteçleri

Şekil 9.2, bir tamsayıyı tüm dönüşüm belirteçlerini kullanarak yazdırmaktadır. Yalnızca eksi

(- ) işaretlerinin yazdırıldığına, artı ( + ) işaretlerinin yazdırılmadığına dikkat ediniz. İleride

artı işaretlerinin yazdırılmasını nasıl sağlayacağımızı anlatacağız. Ayrıca, – 455 değerinin %u

ile okunduğunu ve 65081 işaretsiz değerine çevrildiğine dikkat ediniz.

Genel Programlama Hataları 9.2

Negatif bir değeri, unsigned bir değer bekleyen bir dönüşüm belirteciyle yazdırmaya çalışmak.

1 /\* Şekil 9.2: fig09\_02.c \*/

2 /\* Tamsayı dönüşüm belirteçlerini kullanmak.\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 printf( "%d\n", 455 );

8 printf( "%i\n", 455 ); /\* i ve d printf ifadesinde aynıdırlar. \*/

9 printf( "%d\n", +455 );

10 printf( "%d\n", -455 );

11 printf( "%hd\n", 32000 );

12 printf( "%ld\n", 2000000000 );

13 printf( "%o\n", 455 );

14 printf( "%u\n", 455 );

15 printf( "%u\n", -455 );

16 printf( "%x\n", 455 );

17 printf( "%X\n", 455 );

18

19 return 0;

20 }

455

455

455

-455

32000

2000000000

707

455

65081

1c7

1C7

367

Şekil 9.2 Tamsayı dönüşüm belirteçlerini kullanmak.

9.5 ONDALIKLI SAYILARI YAZDIRMAK

Bir ondalıklı değer, 33.5 ya da 657.983 gibi ondalıklı bir kısım içerir. Ondalıklı sayılar bir

çok biçimden biri ile gösterilirler. Şekil 9.3, ondalıklı sayıların dönüşüm belirteçlerini

açıklamaktadır. e ve E dönüşüm belirteçleri, ondalıklı sayıları üssel yazılım biçiminde

gösterir. Üssel yazılım, matematikte kullanılan bilimsel gösterimin bilgisayarlarda kullanılan

şeklidir. Örneğin, 150.4582 değeri bilimsel gösterimde

1.504582 x 102

şeklinde temsil edilir ve üssel yazılım ile bilgisayarda

1.504582E+02

şeklinde temsil edilir. Bu gösterim 1.504582’in 10’un 2. kuvveti ile ( E+02 ) çarpılacağını

belirtir. E, İngilizce’deki “exponent” yani üs kelimesinin kısaltmasıdır.

e , E ve f dönüşüm belirteçleri ile yazdırılan değerler, noktadan sonra aksi belirtilmedikçe 6

basamak duyarlıkta yazdırılırlar. Diğer duyarlıklar özel olarak belirtilebilir. f dönüşüm

belirteci, her zaman noktanın solunda en az bir basamak yazdırır. e ve E dönüşüm belirteçleri,

üssel kısımdan sonra e ya da E harfini yazdırır ve noktanın solunda kesinlikle bir basamak

yazdırır.

g (G ) dönüşüm belirteci, birbirini izleyen sıfırları yazdırmadan e ( E ) ya da f biçiminde

yazdırır. Örneğin 1.234000 değeri 1.234 olarak yazdırılır. Değerler, eğer üssel yazılım

biçimine çevrildiklerinde değerin üssü – 4’den küçükse ya da belirlenen duyarlığa (g ve G

için aksi belirtilmedikçe 6 basamak) eşit ya da belirlenen duyarlıktan büyükse, e ( E ) ile

yazdırılırlar. Diğer durumlarda ise f dönüşüm belirteciyle yazdırılır. g ya da G ile ondalıklı

kısımda birbirini takip eden sıfırlar yazdırılmaz. Ondalık noktasının yazdırılması için en az bir

ondalıklı basamağa ihtiyaç vardır.%g dönüşüm belirteciyle

0.0000875 8.75e-05

8750000.0 8.75e+06

8.75 8.75

87.50 87.5

875 875

biçiminde yazdırılır. 0.0000875, e kullanmaktadır çünkü üssel yazılım biçimine çevrildiğinde

üs – 4’ten küçüktür. 8750000.0 değeri e kullanır çünkü üs, aksi belirtilmedikçe kullanılan

duyarlığa eşittir.

Dönüşüm Belirteci Açıklama

e ya da E ondalıklı bir değeri üssel yazılım biçiminde gösterir.

f ondalıklı sayıları gösterir.

g ya da G ondalıklı değerleri f ya da e (ya da E) üssel biçiminde gösterir.

L Herhangi bir dönüşüm belirtecinden önce kullanıldığında, bir

long double ondalıklı değerin yazdırılacağını belirtir.

Şekil 9.3 Ondalıklı sayı dönüşüm belirteçleri

368

g ve G dönüşüm belirteçleri için duyarlık, noktanın solundaki sayılarda dahil olmak üzere

önemli kabul edilen en fazla basamak sayısını belirtir. 1234567.0 değeri %g dönüşüm

belirteci kullanılarak 1.23457e+06 biçiminde yazdırılır. ( ondalıklı sayılar için kullanılan tüm

belirteçlerin aksi belirtilmedikçe 6 duyarlığına sahip olduğunu hatırlayınız) Sonuçta, 6 önemli

basamak olduğuna dikkat ediniz. g ile G arasındaki fark, değer üssel yazılım biçiminde

yazdırılırken e ile E arasında oluşan farka benzer. Küçük yazılan g küçük e , büyük yazılan G

büyük E yazılmasına yol açar.

İyi Programlama Alıştırmaları 9.2

Veri yazdırırken, kullanıcının, biçimlendirmeye bağlı olarak verinin duyarlığının

değişebileceğinden (duyarlık belirtme sebebiyle oluşan yuvarlamalar gibi) haberdar

olduğundan emin olun.

Şekil 9.4, ondalıklı sayılar için kullanılan dönüşüm belirteçlerinin hepsini kullanmaktadır.

%E ve %g dönüşüm belirteçlerinin değerin yuvarlanmasına yol açtığına dikkat ediniz.

1 /\* Şekil 9.4: fig09\_04.c \*/

2 /\* Ondalıklı sayı dönüşüm

3 belirteçlerini kullanmak \*/

4

5 #include <stdio.h>

6

7 int main( )

8 {

9 printf( "%e\n", 1234567.89 );

10 printf( "%e\n", +1234567.89 );

11 printf( "%e\n", -1234567.89 );

12 printf( "%E\n", 1234567.89 );

13 printf( "%f\n", 1234567.89 );

14 printf( "%g\n", 1234567.89 );

15 printf( "%G\n", 1234567.89 );

16

17 return 0;

18 }

1.234568e+06

1.234568e+06

-1.234568e+06

1.234568E+06

1.234567.890000

1.23457e+06

1.23457E+06

Şekil 9.4 Ondalıklı sayı dönüşüm belirteçlerini kullanmak

369

9.6 STRING VE KARAKTERLERİ YAZDIRMAK

c ve s dönüşüm belirteçleri sırasıyla, karakterleri ve stringleri yazdırmak için kullanılır. c

dönüşüm belirteci, bir char argümana ihtiyac duyar. s dönüşüm belirteci, argüman olarak

char gösteren bir göstericiye ihtiyaç duyar. s dönüşüm belirteci, sonlandırıcı null karakterle

(‘\0’) karşılaşıncaya dek karakterleri yazdırır. Şekil 9.5’deki program, karakter ve stringleri c

ve s dönüşüm belirteçleri sayesinde yazdırmaktadır.

1 /\* Şekil 9.5: fig09\_05c \*/

2 /\* Karakter ve string dönüşüm belirteçlerini kullanmak \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char karakter = 'A';

8 char string[ ] = "Bu bir stringtir";

9 const char \*stringPtr = "Bu da bir stringtir";

10

11 printf( "%c\n", karakter );

12 printf( "%s\n", " Bu bir stringtir " );

13 printf( "%s\n", string );

14 printf( "%s\n", stringPtr );

15

16 return 0;

17 }

A

Bu bir stringtir

Bu bir stringtir

Bu da bir stringtir

Şekil 9.5 Karakter ve string dönüşüm belirteçlerini kullanmak.

Genel Programlama Hataları 9.3

Bir stringi yazdırmak için %c kullanmak. %c dönüşüm belirteci bir char argüman bekler.Bir string,

char gösteren bir göstericidir(yani char \*)

Genel Programlama Hataları 9.4

char bir argümanı yazdırmak için %s kullanmak. %s dönüşüm belirteci, argüman olarak char

gösteren tipte bir gösterici bekler.

Genel Programlama Hataları 9.5

Karakter stringlerini tek tırnak içine almak bir yazım hatasıdır.Karakter stringleri çift tırnak

içine alınmalıdır.

370

Genel Programlama Hataları 9.6

Bir karakter sabitini çift tırnak içine almak. Bu, gerçekte ikincisi sonlandırıcı null olan iki

karakterlik bir string yaratır. Bir karakter sabiti, tek tırnak içine yazılmış tek bir karakterdir.

9.7 DİĞER DÖNÜŞÜM BELİRTEÇLERİ

Anlatacağımız üç dönüşüm belirteci kalmıştı. Bunlar p, n ve % dönüşüm belirteçleridir

(Şekil 9.6),

Taşınırlık İpuçları 9.1

p dönüşüm belirteci, bir adresi uygulamaya bağımlı bir şekilde (bazı

sistemlerde,onluk sistem yerine onaltılık sistem kullanılır) yazdırır.

Dönüşüm belirteci Açıklama

p Bir gösterici değerini uygulama bağımlı olarak yazdırır.

n O andaki printf ifadesinde yazdırılan karakter sayısını tutar.

İlgili argüman olarak tamsayı gösteren bir gösterici alır.Hiçbir

şey yazdırılmaz.

% Yüzde karakterini yazdırır.

Şekil 9.6 Diğer dönüşüm belirteçleri

n dönüşüm belirteci, o andaki printf ifadesinde yazdırılan karakter sayısını tutar. İlgili

argüman olarak, değerin tutulduğu tamsayı değişkenini gösteren bir gösterici alır. %n

dönüşüm belirteci ile hiçbir şey yazdırılmaz. % dönüşüm belirteci, yüzde işaretini yazdırmak

için kullanılır.

Şekil 9.7’deki %p , ptr’ in değerini ve x’in adresini yazdırır. Bu değerler eştir çünkü ptr ,

x’in adresine atanmıştır. Daha sonra %n , üçüncü printf ifadesi ile yazdırılan karakter

sayısını tamsayı değişkeni olan y içine depolar ve y’in değeri yazdırılır. Son printf ifadesi, bir

karakter stringi içindeki % karakterini yazdırabilmek için %% kullanılır. Her printf

ifadesinin bir değer döndürdüğüne dikkat ediniz. Bu değer, yazdırılan karakter sayısı ya da bir

hata oluştursa negatif bir değerdir.

Genel Programlama Hataları 9.7

Yüzde karakterini, biçim kontrol dizesi içinde %% yerine % kullanarak yazdırmaya

çalışmak.Biçim kontrol dizesi içinde % görüldüğünde, hemen ardından mutlaka bir dönüşüm

belirteci kullanılmalıdır.

1 /\* Şekil 9.7 : fig09\_07.c \*/

2 /\* p,n ve % dönüşüm belirteçlerini kullanmak.\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int \*ptr;

8 int x = 12345, y;

9

10 ptr = &x;

371

11 printf( "ptr nin değeri %p\n", ptr );

12 printf( "x in adresi %p\n\n", &x );

13

14 printf( "Bu satırda yazdırılan toplam karakter sayısı:%n", &y );

15 printf( " %d\n\n", y );

16

17 y = printf( "Bu satırda 27 karakter var\n" );

18 printf( "%d karakter yazıldı\n\n", y );

19

20 printf( " %% işaretinin biçim kontrol dizesinde yazdırılması \n" );

21

22 return 0;

23 }

ptr nin değeri 001F2BB4

x in adresi 001F2BB4

Bu satırda yazdırılan toplam karakter sayısı:45

Bu satırda 27 karakter var

27 karakter yazıldı

% işaretinin biçim kontrol dizesinde yazdırılması

Şekil 9.7 p,n ve % dönüşüm belirteçlerini kullanmak.

9.8 ALAN GENİŞLİĞİ VE DUYARLIK İLE YAZDIRMAK

Verinin yazdırılacağı alanın kesin boyutları alan genişliği ile belirlenir. Eğer alan genişliği

yazdırılacak veriden büyükse, veri o alan içinde otomatik olarak sağa yaslanacaktır. Alan

genişliğini belirten bir tamsayı, dönüşüm tarifi içinde yüzde işareti ( % ) ile dönüşüm belirteci

arasına yerleştirilir. Şekil 9.8, her biri 5 sayıdan oluşan 2 grubu, alan genişliğinden daha az

basamağa sahip olan sayıları sağa yaslayarak, yazdırmaktadır. Alan genişliğinin, alandan daha

geniş değerleri yazdırmak için arttığına ve negatif bir değer için kullanılan eksi işaretinin alan

genişliği içinde bir karakter pozisyonunu kullandığına dikkat ediniz. Alan genişliği tüm

dönüşüm belirteçleriyle kullanılabilir.

Genel Programlama Hataları 9.8

Yazdırılacak sayı için yeterince geniş bir alan sağlamamak.Bu, yazdırılan diğer değerleri

bastırabilir ve şaşırtıcı sonuçlar üretebilir.Verinizi bilin!

1 /\* Şekil 9.8: fig09\_08.c \*/

2 /\* Tamsayıları bir alan içinde sağa yaslamak.\*/

3 #include <stdio.h>

4

5 int main( )

6 {

372

7 printf( "%4d\n", 1 );

8 printf( "%4d\n", 12 );

9 printf( "%4d\n", 123 );

10 printf( "%4d\n", 1234 );

11 printf( "%4d\n\n", 12345 );

12

13 printf( "%4d\n", -1 );

14 printf( "%4d\n", -12 );

15 printf( "%4d\n", -123 );

16 printf( "%4d\n", -1234 );

17 printf( "%4d\n", -12345 );

18

19 return 0;

20 }

1

12

123

1234

12345

-1

-12

-123

-1234

-12345

Şekil 9.8 Tamsayıları bir alan içinde sağa yaslamak.

printf fonksiyonu ayrıca yazdırılacak verinin duyarlığının belirlenmesi yeteneğini sunar.

Duyarlık, farklı veri tipleri için farklı anlamlara gelir. Tamsayı dönüşüm belirteçleri ile

kullanıldığında duyarlık, yazdırılacak minimum basamak sayısını belirtir. Eğer yazdırılan

değer belirlenen duyarlıktan daha az basamağa sahipse, toplam basamak sayısı duyarlığa eşit

olana kadar yazdırılan değerin önüne sıfır eklenir. Tamsayılar için duyarlık, aksi

belirtilmedikçe birdir. e, E ve f ondalıklı sayı dönüşüm belirteçleri ile kullanıldığında

duyarlık, ondalık kısımda yazdırılacak basamak sayısıdır. g ve G dönüşüm belirteçleri ile

kullanıldığında duyarlık, yazdırılacak önemli basamakların maksimum sayısıdır. s dönüşüm

belirteciyle kullanıldığında duyarlık, stringten yazdırılacak en fazla karakter sayısıdır.

Duyarlık kullanmak için, yüzde işareti ile dönüşüm belirteci arasına nokta ( . ) ve duyarlığı

belirten bir tamsayı değeri yerleştirilir. Şekil 9.9, biçim kontrol dizelerinde duyarlığın

kullanımını göstermektedir. Ondalıklı bir değerin, orijinal sayıdaki ondalık kısımdan daha

küçük bir duyarlıkla yazdırıldığında yuvarlandığına dikkat ediniz.

1 /\* Şekil:9.9 fig09\_09.c \*/

2 /\* Çeşitli tiplerdeki verileri yazdırmak için duyarlık kullanmak. \*/

3 #include <stdio.h>

4

373

5 int main( )

6 {

7 int i = 873;

8 double f = 123.94536;

9 char s[ ] = "Nice Yıllaraaa";

10

11 printf( "Tam sayılarda duyarlık kullanımı\n" );

12 printf( "\t%.4d\n\t%.9d\n\n", i, i );

13 printf( "Ondalıklı sayılarda duyarlık kullanımı\n" );

14 printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );

15 printf( "Stringlerde duyarlık kullanımı\n" );

16 printf( "\t%.11s\n", s );

17

18 return 0;

19 }

Tam sayılarda duyarlık kullanımı

0873

000000873

Ondalıklı sayılarda duyarlık kullanımı

123.945

1.239e+02

124

Stringlerde duyarlık kullanımı

Nice Yıllar

Şekil 9.9 Çeşitli tiplerdeki verileri yazdırmak için duyarlık kullanmak.

Alan genişliği ve duyarlık, yüzde işareti ve dönüşüm belirteci arasına, alan genişliğinden

sonra nokta ve noktadan sonra da duyarlık belirtilerek aşağıdaki ifadede olduğu gibi

birleştirilebilir:

printf ( “ %9.3f ” , 123.456789 ) ;

Bu ifade 123.457 yazdıracaktır. Noktadan sonra, ondalık kısımda 3 basamak yazdırılmıştır

(duyarlık 3 olduğundan). Sayı sağa dayalı bir biçimde, 9 basamak genişliğinde bir alan içine

yazdırılmıştır ( alan genişliği 9 olduğundan).

Alan genişliği ve duyarlık, biçim kontrol dizesinden sonra gelen argüman listesi içinde

tamsayı deyimleri olarak belirtilebilir. Bu özelliği kullanmak için, alan genişliği ya da

duyarlığı yerine (ya da ikisi yerine de) yıldız karakteri ( \* ) yerleştirilir. Argüman listesinde,

eşleşen int argümanı hesaplanır ve yıldız işareti yerine kullanılır. Bir alan genişliğinin değeri

pozitif ya da negatif (çıktının alan içinde sola dayalı olmasını sağlar) olabilir.

printf ( “ %\*.\*f ” , 7 , 2 , 98,736 ) ;

374

ifadesi alan genişliği için 7, duyarlık için 2 kullanır ve sağa dayalı bir biçimde 98.74 yazdırır.

9.9 printf BİÇİM-KONTROL DİZESİNDE BAYRAKLARI

KULLANMAK

printf fonksiyonu, çıktı biçimlendirme yeteneklerine ek olarak bayrakları kullanabilir.

Kullanıcının biçim kontrol dizesi içinde kullanabileceği 5 bayrak bulunmaktadır.(Şekil 9.10)

Bayrak Açıklama

-(eksi işareti) Belirlenen alanda çıktıyı sola yaslar.

+(artı işareti) Pozitif değerlerden önce artı ve negatif değerlerden önce eksi

işaretini yazdırır.

boşluk + bayrağı ile yazdırılmamış pozitif bir değerden önce boşluk

yazdırır.

# Sekizlik dönüşüm belirteci(o) ile kullanıldığında yazdırılan

değerin önüne 0 ekler.

x ya da X onaltılık dönüşüm belirteci ile kullanıldığında

yazdırılan değerin önüne 0x ya da 0X ekini ekler.

e, E , f , g ya da G ile yazdırılmış, ondalık kısım içermeyen

ondalıklı sayıların nokta içermesini sağlar. (normalde nokta eğer

kendinden sonra bir basamak varsa yazdırılır) g ve G dönüşüm

belirteçleri için ard arda gelen sıfırlar elenmez.

0(sıfır) Bir alanı, önce sıfırlar gelecek biçimde 0 ile doldurur.

Şekil 9.10 Biçim kontrol dizesi bayrakları

Biçim kontrol dizesi içinde bir bayrak kullanmak için, bayrağı yüzde işaretinin hemen sağına

koymak gerekir. Birden fazla bayrak tek bir dönüşüm belirteci içinde birleştirilebilir.

Şekil 9.11, bir stringin, bir tamsayının, bir karakterin ve ondalıklı bir sayının sağa dayalı ve

sola sayalı bir biçimde yazdırılmasını göstermektedir.

Şekil 9.12, bir pozitif ve bir negatif sayıyı önce + bayrağı ile sonrada + bayrağını

kullanmadan yazdırmaktadır. Eksi işaretinin her iki durumda da yazdırıldığına ancak artı

işaretinin yalnızca + bayrağı kullanıldığında yazıldığına dikkat ediniz.

1 /\* Şekil 9.11: fig09\_11.c \*/

2 /\* Stringlerin bir alanda sola dayalı yazdırılması \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 printf( "%10s%10d%10c%10f\n\n", "selam", 7, 'a', 1.23 );

8 printf( "%-10s%-10d%-10c%-10f\n", "selam", 7, 'a', 1.23 );

375

9 return 0;

10 }

selam 7 a 1.230000

selam 7 a 1.230000

Şekil 9.11 Stringlerin bir alanda sola dayalı yazdırılması

1 /\* Şekil 9.12: fig09\_12.c \*/

2 /\* Sayıların + bayrağı kullanılarak ve +bayrağı kullanılmayarak yazılması \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 printf( "%d\n%d\n", 786, -786 );

8 printf( "%+d\n%+d\n", 786, -786 );

9 return 0;

10 }

786

-786

+786

-786

Şekil 9.12 Pozitif ve negatif sayıların + bayrağı ile ve + bayrağı kullanılmadan yazdırılması.

Şekil 9.13, boşluk bayrağını kullanarak, pozitif bir sayının önüne boşluk yerleştirmektedir.

Bu, aynı sayıda basamağa sahip pozitif ve negatif sayıları hizalamak için oldukça

kullanışlıdır.

Şekil 9.14 , sekizlik bir değerin önüne 0 eklemek, onaltılık değerlerin önüne 0x ve 0X

eklemek ve g ile yazdırılan bir değerde ondalık noktanın yazdırılmasını sağlamak için #

bayrağını kullanmaktadır.

1 /\* Şekil 9.13: fig09\_13.c \*/

2 /\* + veya – ile başlamayan değerlerde

3 boşluk kullanılması \*/

4 #include <stdio.h>

5

6 int main( )

7 {

376

8 printf( "% d\n% d\n", 547, -547 );

9 return 0;

10 }

547

- 547

Şekil 9.13 Boşluk bayrağını kullanmak

1 /\* Şekil 9.14: fig09\_14.c \*/

2 /\* # bayrağının o, x, X dönüşüm belirteçleri ve

3 ondalıklı sayı belirteçleri ile kullanılması \*/

4 #include <stdio.h>

5

6 int main( )

7 {

8 int c = 1427;

9 double p = 1427.0;

10

11 printf( "%#o\n", c );

12 printf( "%#x\n", c );

13 printf( "%#X\n", c );

14 printf( "\n%g\n", p );

15 printf( "%#g\n", p );

16

17 return 0;

18 }

02623

0x593

0x593

1427

1427.00

Şekil 9.14 # bayrağını kullanmak.

Şekil 9.15, + bayrağı ile 0 (sıfır) bayrağını, 452’yi 9 genişliğinde bir alanda + işareti ve alanı

dolduracak kadar 0 ile yazdırmak için birlikte kullanmaktadır. Daha sonra ise, 452’yi sadece 0

(sıfır) bayrağını kullanarak 9 genişliğindeki bir alanda yazdırmaktadır.

1 /\* Şekil 9.15 : fig09\_15.c \*/

2 /\* 0(sıfır) bayrağını kullanmak\*/

3 #include <stdio.h>

4

377

5 int main( )

6 {

7 printf( "%+09d\n", 452 );

8 printf( "%09d\n", 452 );

9

10 return 0;

11 }

+00000452

000000452

Şekil 9.15 0 (sıfır) bayrağını kullanmak

9.10 ÇIKIŞ DİZİLERİNİ VE HAZIR BİLGİLERİ YAZDIRMAK

Bir printf ifadesi içinde yazdırılacak çoğu bilgi karakteri, biçim kontrol dizesi içinde

bulunabilir. Buna rağmen biçim kontrol dizesini sınırlandıran tırnak işareti gibi (“) sorunlu

bazı karakterler vardır. Yeni satır ve tab gibi çeşitli kontrol karakterleri, çıkış sıraları ile temsil

edilmelidir. Bir çıkış sırası ters çizgi işaretinden ( \ ) sonra, bir çıkış karakteri ile oluşturulur.

Şekil 9.16, tüm çıkış sıralarını ve yaptıkları işleri açıklamaktadır.

Genel Programlama Hataları 9.9

printf ifadesi içinde tek tırnak ,çift tırnak ,soru işareti ya da ters çizgi karakterlerini, o karakteri ters

çizgi ile birlikte uygun bir çıkış sırası oluşturacak şekilde kullanmadan yazdırmaya çalışmak.

Çıkış Sırası Açıklama

\ ’ Tek tırnak ( ‘ ) karakterini yazdırır.

\ ” Çift tırnak ( “ ) karakterini yazdırır.

\ ? Soru işareti ( ? ) karakterini yazdırır.

\ \ Ters çizgi ( \ ) karakterini yazdırır.

\ a Duyulabilen (zil) ya da görülebilen bir alarm çalıştırır.

\ b İmleci o andaki konumundan bir geriye taşır.

\ f İmleci bir sonraki sayfanın başına taşır.

\ n İmleci bir sonraki yeni satırın başına taşır.

\ r İmleci o andaki satırın başına taşır.

\ t İmleci bir sonraki yatay tab konumuna taşır.

\ v İmleci bir sonraki düşey tab konumuna taşır.

Şekil 9.16 Çıkış sıraları

9.11 scanf İLE GİRİŞİ BİÇİMLENDİRMEK

Kesin giriş biçimlendirme, scanf sayesinde yapılır. Her scanf ifadesi, girilecek verinin

biçimini tanımlayan bir biçim kontrol dizesi içerir. Biçim kontrol dizesi dönüşüm belirteçleri

ve bilgi karakterleri içerir. scanf fonksiyonu aşağıdaki giriş biçimlendirme yeteneklerine

sahiptir:

1.Her tipte veriyi almak.

2.Bir giriş akışındaki belli karakterleri almak.

3.Giriş akışındaki belli karakterleri atlamak.

378

scanf fonksiyonu aşağıdaki biçimde yazılır:

scanf ( biçim kontrol dizesi , diğer argümanlar ) ;

Biçim kontrol dizesi, girilen verinin biçimini tarif eder ve diğer argümanlar da girilen verinin

depolanacağı değişkenleri gösteren göstericilerdir.

İyi Programlama Alıştırmaları 9.3

Veri girerken, kullanıcıyı tek bir veri parçası ya da az sayıda veri parçasını girmesi için teşvik

edin.Kullanıcının bir anda birden çok veri parçası girebileceği durumlardan kaçının.

Şekil 9.17, her tipte veriyi alabilmek için kullanılan dönüşüm belirteçlerini özetlemektedir.Bu

kısmın geri kalanı, çeşitli scanf dönüşüm belirteçlerini kullanarak okuma yapan bir çok örnek

sunmaktadır.

Dönüşüm Belirteci Açıklama

Tamsayılar

d Tercihe bağlı olarak onluk sistemde işaretli bir tamsayı

okur.İlgili argüman tamsayıyı gösteren bir göstericidir.

i Tercihe bağlı olarak onluk,sekizlik ya da onaltılık sistemde

işaretli bir tamsayı okur.İlgili argümanı, tamsayıyı gösteren bir

göstericidir.

o Sekizlik sistemde bir tamsayı okur.İlgili argüman, işaretsiz bir

tamsayı gösteren bir göstericidir.

u Onluk sistemde işaretsiz bir tamsayıyı okur.İlgili argümanı,

işaretsiz bir tamsayıyı gösteren bir göstericidir.

x ya da X Onaltılık sistemde bir tamsayı okur.İlgili argüman, işaretsiz bir

tamsayıyı gösteren bir göstericidir.

h ya da l Herhangi bir tamsayı dönüşüm belirtecinden önce

yerleştirildiğinde, short ya da long bir tamsayının girileceğini

belirtir.

Ondalıklı tamsayılar

e,E,f,g ya da G Ondalıklı bir tamsayıyı okur. İlgili argüman,

ondalıklı değişkeni gösteren bir göstericidir.

l ya da L Herhangi bir dönüşüm belirtecinden önce yerleştirildiğinde

double ya da long double bir değerin girileceğini belirtir.

379

Karakterler ve stringler

c Bir karakter okur. İlgili argüman, char gösteren bir

göstericidir. Sonlandırıcı null (‘\0’) eklenmez.

s Bir string okur. İlgili argüman, stringi ve sonlandırıcı null

karakteri depolamaya yetecek kadar geniş, char tipinde bir

diziyi gösteren bir göstericidir.

Tarama kümesi

[arama karakterleri] Bir dizide depolanmış karakterleri string içinde arar.

Nadir kullanılanlar

p printf ifadesi ile %p kullanıldığında oluşturulan

biçimle aynı biçimdeki bir adresi okur.

n scanf ile o ana kadar alınan karakter sayısını depolar.İlgili

argüman, tamsayıyı gösteren bir göstericidir.

% Girilen verideki yüzde işaretini(%) atlar.

Şekil 9.17 scanf için dönüşüm belirteçleri

Şekil 9.18, çeşitli tamsayı dönüşüm belirteçlerini kullanarak tamsayılar okumakta ve

tamsayıları onluk sistemde sayılar olarak yazdırmaktadır. %i’nin onluk, sekizlik ve onaltılık

tamsayıları alabildiğine dikkat ediniz.

1 /\* Şekil 9.18: fig09\_18.c \*/

2 /\* Tamsayı girişi\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int a, b, c, d, e, f, g;

8

9 printf( "Yedi tamsayı girin: " );

10 scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );

11 printf( "Girilen sayılar, onluk sistemde tamsayılar olarak gösterilecek:\n" );

12 printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );

13

14 return 0;

15 }

Yedi tamsayı girin: -70 –70 070 0x70 70 70 70

Girilen sayılar, onluk sistemde tamsayılar olarak gösterilecek:

-70 –70 56 112 56 70 112

Şekil 9.18 Tamsayı dönüşüm belirteçleri ile veri okumak.

380

Ondalıklı sayılar girilirken, e , E , f , g ya da G ondalıklı sayı dönüşüm belirteçlerinden

herhangi biri kullanılabilir. Şekil 9.19, üç ondalıklı sayı okumaktadır ve bu sayılardan her

birini okumak için, üç dönüşüm belirtecinden birini kullanmaktadır. Daha sonra üç sayıyı da f

dönüşüm belirteciyle yazdırmaktadır. Programın çıktısı, ondalıklı sayıların kesin olmadığı

gerçeğini bir kez daha göstermektedir. Bu gerçek, yazdırılan ikinci değerde açıkça

gösterilmektedir.

1 /\* Şekil 9.19: fig09\_19.c \*/

2 /\* Ondalıklı sayıları girme\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 double a, b, c;

8

9 printf( "Üç ondalıklı sayı girin: \n" );

10 scanf( "%le%lf%lg", &a, &b, &c );

11 printf( "Farklı ondalıklı\n" );

12 printf( "sayı gösterimleri:\n" );

13 printf( "%f\n%f\n%f\n", a, b, c );

14

15 return 0;

16 }

Üç ondalıklı sayı girin:

1.27987 1.27987e+03 3.38476e-06

Farklı ondalıklı

sayı gösterimleri:

1.279870

1279.869995

0.000003

Şekil 9.19 Ondalıklı sayı dönüşüm belirteçleri ile okuma yapmak.

Karakterler ve stringler sırasıyla, c ve s dönüşüm belirteçleri ile alınırlar. Şekil 9.20,

kullanıcıdan bir string girmesini istemektedir. Program, ilk karakteri %c ile almakta ve

karakter değişkeni olan x içine depolamaktadır. Daha sonra stringin geri kalanını %s ile

almakta ve karakter dizisi olan y içinde depolamaktadır.

Bir karakter dizisi, tarama kümesi kullanılarak girilebilir. Bir tarama kümesi, kare parantez [ ]

içine yazılmış karakter kümesidir ve biçim kontrol dizesinde yüzde işaretinden sonra yazılır.

Tarama kümesi, giriş akışındaki karakterler arasında tarama kümesi içinde belirtilenlerle

eşleşenleri tarar. Herhangi bir anda karakter eşlemesi olduğunda, eşlenen karakter, tarama

kümesinin ilgili argümanı olan karakter dizisini gösteren gösterici içine depolanır. Tarama

kümesi, karakter almayı tarama kümesi içinde yer almayan ilk karakterle karşılaşıldığında

durdurur. Eğer giriş akışındaki ilk karakter tarama kümesindeki karakterlerle eşleşmezse,

dizide yalnızca null karakter depolanır. Şekil 9.21, [aeiou] tarama kümesini kullanarak, giriş

381

akışında sesli harfleri taramaktadır. Girişin ilk yedi harfinin okunduğuna dikkat ediniz.

Sekizinci harf (h), tarama kümesi içinde olmadığından arama durmuştur.

1 /\* Şekil 9.20: fig09\_20.c \*/

2 /\* Karakter ve stringleri girme \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char x, y[ 9 ];

8

9 printf( "Bir string girin: " );

10 scanf( "%c%s", &x, y );

11

12 printf( "Karakter \"%c\" ", x );

13 printf( " ve string \"%s\"\n", y );

14 printf( "\ngirdiniz.”);

15

16 return 0;

17 }

Bir string girin: Pazar

Karakter “P” ve string “azar”

girdiniz.

Şekil 9.20 Karakter ve stringleri girmek

1 /\* Şekil 9.21: fig09\_21.c \*/

2 /\* Tarama kümesi kullanma\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char z[ 9 ];

8

9 printf( "Bir string girin: " );

10 scanf( "%[aeiou]", z );

11 printf( "\"%s\" girdiniz.\n", z );

12

13 return 0;

14 }

Bir string girin: ooeeooahah

“ooeeooa” girdiniz.

Şekil 9.21 Tarama kümesi kullanmak

382

Tarama kümesi aynı zamanda, küme içinde bulunmayan karakterleri taramak için de

kullanılabilir. Ters bir tarama kümesi yaratmak için, kare parantezler içinde arama

karakterlerinden önce şapka karakterini ( ^ ) yerleştirmek gerekir. Bu, karakter kümesi içinde

yer almayan karakterlerin depolanmasını sağlar. Ters bir tarama kümesi içinde bulunan bir

karakterle karşılaşıldığında giriş sonlanır. Şekil 9.22, [ ^ aeiou] ters tarama kümesini

kullanarak sessiz harfleri aramaktadır.

1 /\* Şekil 9.22: fig09\_22.c \*/

2 /\* Ters bir tarama kümesi kullanımı \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 char z[ 9 ] = { '\0' };

8

9 printf( "Bir string girin: " );

10 scanf( "%[^aeiou]", z );

11 printf( "Girilen string: \"%s\"\n", z );

12

13 return 0;

14 }

Bir string gitin: String

Girilen string : “Str”

Şekil 9.22 Ters bir tarama kümesi kullanmak

Alan genişliği bir scanf dönüşüm belirteci içinde kullanılarak, giriş akışından belli sayıda

karakterin okunması sağlanabilir. Şekil 9.23 , ard arda girilen rakamları iki basamaklı bir sayı

ve giriş akışından geriye kalan rakamları içeren bir tamsayı olarak almaktadır.

1 /\* Şekil 9.23: fig09\_23.c \*/

2 /\* Alan genişliği ile veri girmek \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int x, y;

8

9 printf( "6 basamaklı bir tamsayı girin: " );

10 scanf( "%2d%d", &x, &y );

11 printf( "Girdiğiniz tamsayı: %d ve %d\n", x, y );

12

13 return 0;

14 }

6 basamaklı bir tamsayı girin: 123456

Girdiğiniz tamsayı: 12 ve 3456

Şekil 9.23 Alan genişliği ile veri girmek

383

Genellikle, giriş akışındaki bazı karakterleri atlamak gerekir.Örneğin,bir tarih

11-10-1999

şeklinde girilebilir.

Tarihteki her sayı depolanmalıdır fakat sayıları ayıran tire işaretleri ( - ) kullanılmamalıdır.

Gereksiz karakterleri elemek için, elenecek karakterleri scanf’in biçim kontrol dizesi içine

yazmak gerekir. Örneğin, girişteki tire işaretlerini ( - ) elemek için

scanf (“ %d-%d-%d ”, &ay , &gun , &yil );

ifadesi kullanılır. Bu scanf, az önceki girişte tire işaretlerini elese de, tarihi

10/11/1999

şeklinde de girmek mümkündür.

Bu durumda az önceki scanf, gereksiz karakterleri eleyemez. Bu sebepten, scanf atama

bastırma karakteri olan \* karakterini sunar. Atama bastırma karakteri, scanf ’in girişten her

tipte veriyi okuması ve bir değişkene atamadan ihmal etmesini sağlar. Şekil 9.24, atama

bastırma karakterini giriş akışındaki bir karakterin okunacağını ve ihmal edileceğini belirtmek

için %c dönüşüm belirteci içinde kullanmaktadır. Yalnızca ay, gün ve yıl depolanacaktır.

Değişkenlerin değerleri, değerlerin doğru bir şekilde girildiğini göstermek amacıyla

yazdırılmıştır. Argüman listesindeki değişkenlerin hiçbirinin atama bastırma karakterini

kullanan dönüşüm belirteçleriyle eşleşmediğine dikkat ediniz. Çünkü bu dönüşüm belirteçleri

için hiçbir atama yapılmayacaktır.

1 /\* Şekil 9.24: fig09\_24.c \*/

2 /\* Giriş akışından karakter okumak ve karakterleri ihmal etmek \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int ay1, gun1, yil1, ay2, gun2, yil2;

8

9 printf( "aa-gg-yyyy biçiminde bir tarih girin: " );

10 scanf( "%d%\*c%d%\*c%d", &ay1, &gun1, &yil1 );

11 printf( "ay = %d gün = %d yıl = %d\n\n",

12 ay1, gun1, yil1 );

13 printf( "aa/gg/yyyy biçiminde bir tarih girin: " );

14 scanf( "%d%\*c%d%\*c%d", &ay2, &gun2, &yil2 );

15 printf( "ay = %d gün = %d yıl = %d\n",

16 ay2, gun2, yil2 );

17

18 return 0;

19 }

384

aa-gg-yyyy biçiminde bir tarih girin: 11-18-2000

ay = 11 gün = 18 yıl = 2000

aa/gg/yyyy biçiminde bir tarih girin: 11/18/2000

ay = 11 gün = 18 yıl = 2000

Şekil 9.24 Giriş akışından karakter okumak ve karakterleri ihmal etmek

ÖZET

 Tüm giriş ve çıkış, akışlar (satırları oluşturmak için organize edilmiş karakter dizileri)

ile yapılır. Her satır, 0 ya da daha fazla karakter içerir ve yeni satır karakteriyle

sonlanır

 Normalde, standart giriş akışı klavyeye ve standart çıkış akışı ekrana bağlanır.

 İşletim sistemleri sıklıkla, standart giriş ve standart çıkış akışlarının başka cihazlara

yönlendirilmesine izin verir.

 printf biçim kontrol dizesi, çıktı değerlerinin hangi biçimde görüneceğini tanımlar.

Biçim kontrol dizesi, dönüşüm belirteçleri, bayraklar, alan genişlikleri , duyarlık ve

bilgi karakterlerinden oluşur.

 Tamsayılar şu dönüşüm belirteçlerinden biriyle yazdırılır: d ya da i, işaretli bir

tamsayıyı, o işaretsiz bir tamsayıyı sekizlik sistemde, u işaretsiz bir tamsayıyı onluk

sistemde , x ya da X işaretsiz bir tamsayıyı onaltılık sistemde gösterir. h ya da l

herhangi bir tamsayı belirtecinden önce kullanıldığında sırasıyla, short ya da long bir

tamsayının gösterileceğini belirtir.

 Ondalıklı sayılar şu dönüşüm belirteçlerinden biri ile yazdırılır: e ya da E üssel

yazılım biçimi için, f her zamanki ondalıklı gösterim için, g ya da G , f ya da e (ya

da E) biçiminden biri ile göstermek için kullanılır. g ya da G dönüşüm belirteci

belirtildiğinde, değerin üssü – 4’den küçükse ya da belirlenen duyarlığa eşit ya da

belirlenen duyarlıktan büyükse e (ya da E) dönüşüm belirteci kullanılır.

 g ve G dönüşüm belirteçleri için duyarlık, noktanın solundaki sayılarda dahil olmak

üzere önemli kabul edilen en fazla basamak sayısını belirtir.

 c dönüşüm belirteci, karakterleri yazdırmak için kullanılır.

 s dönüşüm belirteci, stringleri yazdırmak için kullanılır.

 p dönüşüm belirteci, bir adresi uygulamaya bağımlı bir şekilde (bazı sistemlerde,

onluk sistem yerine onaltılık sistem kullanılır) yazdırır.

 n dönüşüm belirteci, o andaki printf ifadesinde yazdırılan karakter sayısını tutar. İlgili

argüman olarak, değerin tutulduğu tamsayı değişkenini gösteren bir gösterici alır.

 %% dönüşüm belirteci , % bilgisini yazdırmak için kullanılır.

 Eğer alan genişliği yazdırılacak veriden büyükse,veri o alan içinde otomatik olarak

sağa yaslanır.

 Alan genişliği tüm dönüşüm belirteçleriyle kullanılabilir.

 Tamsayı dönüşüm belirteçleri ile kullanıldığında duyarlık, yazdırılacak minimum

basamak sayısını belirtir. Eğer yazdırılan değer belirlenen duyarlıktan daha az

basamağa sahipse, toplam basamak sayısı kesinliğe eşit olana kadar yazdırılan değerin

önüne sıfır eklenir.

 e, E ve f ondalıklı sayı dönüşüm belirteçleri ile kullanıldığında duyarlık, ondalık

kısımda yazdırılacak basamak sayısıdır. g ve G dönüşüm belirteçleri ile

kullanıldığında duyarlık, yazdırılacak önemli basamakların maksimum sayısıdır.

385

 s dönüşüm belirteciyle kullanıldığında duyarlık, stringten yazdırılacak en fazla

karakter sayısıdır.

 Alan genişliği ve duyarlık, yüzde işareti ve dönüşüm belirteci arasına, alan

genişliğinden sonra nokta ve noktadan sonra da duyarlık belirtilerek birleştirilebilir

 Alan genişliği ve kesinliği, biçim kontrol dizesinden sonra gelen argüman listesi

içinde tamsayı deyimleri olarak belirtilebilir. Bu özelliği kullanmak için, alan genişliği

ya da duyarlığı yerine (ya da ikisi yerine de) yıldız karakteri( \* ) yerleştirilir. Argüman

listesinde, eşleşen argüman hesaplanır ve yıldız karakteri yerine kullanılır. Bir alan

genişliğinin değeri pozitif ya da negatif olabilir ancak duyarlık pozitif olmalıdır.

 - bayrağı, belirlenen alanda çıktıyı sola yaslar.

 + bayrağı, pozitif değerlerden önce artı ve negatif değerlerden önce eksi işaretini

yazdırır. Boşluk bayrağı, + bayrağı ile yazdırılmamış pozitif bir değerden önce boşluk

yazdırır.

 # bayrağı, sekizlik sistemde yazdırılmış değerlerin önüne 0 ekini, onaltılık sistemde

yazdırılmış değerlerin önüne 0x ya da 0X ekini ekler. e, E , f , g ya da G ile

yazdırılmış, ondalık kısım içermeyen ondalıklı sayıların nokta içermesini sağlar.

(normalde nokta eğer kendinden sonra bir basamak varsa yazdırılır)

 0 (sıfır)bayrağı bir alanı, önce sıfırlar gelecek biçimde 0 ile doldurur.

 Kesin giriş biçimlendirme, scanf kütüphane fonksiyonu sayesinde yapılır.

 Tamsayılar d, i, o, u, x ve X dönüşüm belirteçleriyle girilir. d ve i dönüşüm

belirteçleri tercihe bağlı işaretli bir tamsayıyı girmek için, o , u , x ya da X ise işaretsiz

tamsayıları girmek için kullanılır. h ya da l herhangi bir tamsayı dönüşüm

belirtecinden önce yerleştirildiğinde, short ya da long bir tamsayının girileceğini

belirtir.

 Ondalıklı sayılar e, E, f, g ya da G dönüşüm belirteçleri ile girilirler. l ya da L

herhangi bir dönüşüm belirtecinden önce yerleştirildiğinde double ya da long double

bir değerin girileceğini belirtir.

 Karakterler c dönüşüm belirteci ile girilirler.

 Stringler s dönüşüm belirteciyle girilirler.

 Tarama kümesi, girilen karakterler arasında tarama kümesi içinde belirtilenlerle

eşleşenleri tarar. Herhangi bir anda karakter eşlemesi olduğunda,eşlenen karakter,bir

dizi içinde depolanır.Tarama kümesi, karakter almayı tarama kümesi içinde yer

almayan ilk karakterle karşılaşıldığında durdurur.

 Ters bir tarama kümesi yaratmak için, kare parantezler içinde arama karakterlerinden

önce şapka karakterini ( ^ ) yerleştirmek gerekir. Bu, tarama kümesi içinde bulunan

bir karakterle karşılaşılıncaya kadar, karakter kümesi içinde yer almayan karakterlerin

depolanmasını sağlar.

 Adres değerleri p dönüşüm belirteciyle girilir.

 n dönüşüm belirteci, scanf ile o ana kadar alınan karakter sayısını depolar.İlgili

argüman int gösteren bir göstericidir.

 %% dönüşüm belirteci, girişte tek bir % karakteriyle eşleşir.

 Atama bastırma karakteri , girişten veri okur ve veriyi ihmal eder.

 Alan genişliği bir scanf dönüşüm belirteci içinde kullanılarak, giriş akışından belli

sayıda karakterin okunması sağlanabilir.

ÇEVRİLEN TERİMLER

alignment.................................................... hizalama

assignment suppression character......................... atama bastırma karakteri (\*)

386

caret......................................................................... şapka ( ^ )

conversion specification........................................ dönüşüm tarifi

conversion specifier.............................................. dönüşüm belirteci

escape sequence.................................................... çıkış sırası

field width........................................................... alan genişliği

flag...................................................................... bayrak

format control string........................................ biçim kontrol dizesi

literal characters.............................................. bilgi karakterleri

precision........................................................... duyarlık

rounding................................................... yuvarlama

scan set.................................................... tarama kümesi

stream.................................................... akış

GENEL PROGRAMLAMA HATALARI

9.1 Biçim kontrol dizesini tırnak içine almayı unutmak.

9.2 Negatif bir değeri, unsigned bir değer bekleyen bir dönüşüm belirteciyle yazdırmaya

çalışmak.

9.3 Bir stringi yazdırmak için %c kullanmak. %c dönüşüm belirteci bir char argüman

bekler.Bir string, char gösteren bir göstericidir.(yani char \*)

9.4 char bir argümanı yazdırmak için %s kullanmak. %s dönüşüm belirteci, argüman olarak

char gösteren tipte bir gösterici bekler.

9.5 Karakter stringlerini tek tırnak içine almak bir yazım hatasıdır.Karakter stringleri çift

tırnak içine alınmalıdır.

9.6 Bir karakter sabitini çift tırnak içine almak.Bu gerçekte, ikincisi sonlandırıcı null olan iki

karakterlik bir string yaratır.Bir karakter sabiti, tek tırnak içine yazılmış tek bir

karakterdir.

9.7 Yüzde karakterini, biçim kontrol dizesi içinde %% yerine % kullanarak yazdırmaya

çalışmak.Biçim kontrol dizesi içinde % görüldüğünde, hemen ardından mutlaka bir

dönüşüm belirteci kullanılmalıdır.

9.8 Yazdırılacak sayı için yeterince geniş bir alan sağlamamak.Bu,yazdırılan diğer değerleri

bastırabilir ve şaşırtıcı sonuçlar üretebilir.Verinizi bilin!

9.9 printf ifadesi içinde tek tırnak , çift tırnak, soru işareti ya da ters çizgi karakterlerini, o

karakteri ters çizgi ile birlikte uygun bir çıkış sırası oluşturacak şekilde kullanmadan

yazdırmaya çalışmak.

İYİ PROGRAMLAMA ALIŞTIRMALARI

9.1 Çıktıları gösterim için düzgün bir şekilde yazın.Bu, programı daha okunabilir yapar ve

kullanıcı hatalarını azaltır.

9.2 Veri yazdırırken, kullanıcının, biçimlendirmeye bağlı olarak verinin duyarlığının

değişebileceğinden (duyarlık belirtme sebebiyle oluşan yuvarlamalar gibi) haberdar

olduğundan emin olun.

9.3 Veri girerken, kullanıcıyı tek bir veri parçası ya da az sayıda veri parçasını girmesi için

teşvik edin.Kullanıcının bir anda birden çok veri parçası girebileceği durumlardan

kaçının.

TAŞINIRLIK İPUÇLARI

387

9.1 p dönüşüm belirteci, bir adresi uygulamaya bağımlı bir şekilde(bazı sistemlerde,onluk

sistem yerine onaltılık sistem kullanılır) yazdırır.

ÇÖZÜMLÜ ALIŞTIRMALAR

9.1 Aşağıdaki boşlukları doldurunuz.

a) Bütün giriş ve çıkışlar \_\_\_\_\_\_\_ biçiminde ifade edilir.

b) \_\_\_\_\_\_\_ akışı klavyeye bağlıdır.

c) \_\_\_\_\_\_\_ akışı bilgisayar ekranına bağlıdır.

d) Çıktı duyarlılıkları \_\_\_\_\_\_\_ fonksiyonu ile gerçekleştirilir.

e) Biçim kontrol dizesi \_\_\_\_\_\_\_, \_\_\_\_\_\_\_, \_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ içerir.

f) \_\_\_\_\_\_\_ ya da \_\_\_\_\_\_\_ dönüşüm belirteçleri, çıktı olarak onluk sistemde işaretli bir

tamsayı yazdırmakta kullanılır.

g) \_\_\_\_\_\_\_,\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ dönüşüm belirteçleri, işaretsiz tamsayıları,

sekizlik,onluk ve onaltılık sistemde gösterirler.

h) \_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ belirteçleri tamsayı dönüşüm belirteçlerinden önce kullanılarak

, bu tamsayıların long yada short tipinde gösterileceğini belirtirler.

i) \_\_\_\_\_\_\_ dönüşüm belirteci, ondalıklı bir değeri, üssel yazılım biçiminde gösterir.

j) \_\_\_\_\_\_\_belirteci ondalıklı sayı dönüşüm belirtecinden önce kullanılarak bu sayıların

long yada double tipinde gösterileceğini belirtir.

k) e, E ve f dönüşüm belirteçleri, eğer bir duyarlılık derecesi belirtilmemişse ondalık

noktanın sağından itibaren \_\_\_\_\_\_\_ basamak duyarlılığında gösterirler.

l) \_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ dönüşüm belirteçleri sırasıyla stringleri ve karakterleri yazmada

kullanılırlar.

m) Bütün stringler \_\_\_\_\_\_\_ ile biter.

n) Bir printf dönüşüm belirteci içinde alan genişliği ve duyarlılığı, biçim kontrol

dizesinden sonra gelen argüman listesi içinde tamsayı deyimleri olarak

belirtilebilir. Bu özelliği kullanmak için, alan genişliği ya da duyarlığı yerine (ya da

ikisi yerine de \_\_\_\_\_\_\_\_\_\_ yerleştirilir.

o) \_\_\_\_\_\_\_ bayrağı, çıktının alan içerisinde sola yaslanmasını sağlar.

p) \_\_\_\_\_\_\_ bayrağı, değerlerin artı yada eksi işaretleriyle gösterilmelerini sağlar.

q) Belirli bir duyarlılıkta değer alınması \_\_\_\_\_\_\_ fonksiyonu ile yapılır.

r) \_\_\_\_\_\_\_, bir string içerisinde belli karakterler aramada ve karakterleri dizilerde

saklamakta kullanılır.

s) \_\_\_\_\_\_\_ dönüşüm belirteci, sekizlik, onluk ve onaltılık sistemde işaretli bir tamsayıyı

almak için kullanılır.

t) \_\_\_\_\_\_\_dönüşüm belirteci, double değer alınmasında kullanılır.

u) \_\_\_\_\_\_\_, giriş akışından veri okumada ve herhangi bir değişkene atanmadan ihmal

edilmesinde kullanılır.

v) \_\_\_\_\_\_\_ bir scanf dönüşüm belirtecinde, giriş akışından belli bir sayıda karakter yada

rakam okunacağını göstermek için kullanılabilir.

9.2 Aşağıdaki ifadelerdeki hataları bulunuz ve nasıl düzeltileceğini açıklayınız.

a) Aşağıdaki ifade ‘c’ karakterini yazdırır.

printf(“%s\n”, ‘c’);

b) Aşağıdaki ifade 9.375% yazdırır.

printf(“%.3f%”, 9.375);

c) Aşağıdaki ifade “Pazar” stringinin ilk karakterini yazdırır

printf(“%c\n”, “Pazar”);

388

d) printf (“” Tırnak içinde bir string “”);

e) printf (“%d%d, 12, 20);

f) printf (“%c”, “x”);

g) printf (“%s\n”, ‘Huseyin&Metin’);

9.3 Aşağıdakiler için birer ifade yazınız.

a) 10 alan genişliği içinde sağa yaslanmış 1234 yazdırın.

b) 123.456789 sayısını, üssel gösterimde, işaretli olarak (+ ya da -) 3 basamak

duyarlılıkta yazdırın.

c) sayi değişkenine klavyeden bir double değer alın.

d) 0 ile başlayacak şekilde 100 sayısını sekizlik sistemde yazdırın.

e) string karakter dizisine bir string yazdırın.

f) n dizisine, rakam olmayan bir karakter denk gelene kadar, karakter yazın.

g) 87.4573 double değerini, x ve y değişkenlerini, alan genişliğini ve duyarlılığını

gösterecek şekilde kullanarak yazdırın.

h) 3.5% biçiminde bir değer alın. Bu yüzde değerini float tipinde olan yuzde

değişkeninde saklayın ve % işaretini ortadan kaldırın. Atama bastırma karakteri

kullanmayın.

i) 3.333333 sayısını, long double tipinde ve işaretli(+ ya da -) olarak 20 karakterlik bir

alana 3 duyarlılıkta yazdırınız.

ÇÖZÜMLER

9.1 a)Akışlar b) Standart giriş c) Standart çıkış d) printf e) Dönüşüm belirteçleri, bayraklar,

alan genişlikleri, duyarlılıklar ve bilgi karakterler. f) d, i g) o, u, x (ya da X) h) h, l i) e

(ya da E) j) L k) 6 l) s, c m) NULL(‘\0’) n) yıldız karakteri(\*) o) –(eksi) p) +(artı) q)

scanf r) Tarama kümesi s) i t) le, lE, lf, lg ya da lG u) Atama bastırma karakteri (\*) v)

Alan genişliği

9.2

a) Hata: s dönüşüm belirteci char tipinde bir argüman almıştır.

Düzeltme: ‘c’ karakterini yazdırmak için %c dönüşüm belirtecini kullanmak yada

‘c’ yi “c” olarak değiştirmek.

b) Hata: % bilgi karakteri %% dönüşüm belirteci kullanılmadan yazılmaya

çalışılmıştır.

Düzeltme: % bilgi karakterini yazmak için %% dönüşüm belirtecini kullanın.

c) Hata: c dönüşüm belirteci char tipinde bir argüman almıştır.

Düzeltme: “Pazar” kelimesinin ilk harfini yazdırmak için %1s dönüşüm

belirtecinin kullanılması.

d) Hata: “ bilgi karakteri \” kullanılmadan yazdırılmaya çalışılmıştır.

Düzeltme: Ana tırnakların içindeki tırnakların \” ile değiştirilmesi.

e) Hata: Biçim kontrol dizesi, tırnak(“) ile kapatılmamıştır.

Düzeltme: %d%d’yi tırnak(“) ile kapatmak.

f) Hata: x karakteri tırnak(“) ile kapatılmıştır.

Düzeltme: x karakterini (‘) tırnağı ile kapatmak.

g) Hata: Yazdırılacak string, (‘) tırnağı ile kapatılmıştır.

Düzeltme: (‘) tırnağı yerine (“) kullanılması

9.3

a) printf (“10d\n”, 1234);

b) printf (“%+.3e\n”, 123.456789);

389

c) scanf(“%lf”, &sayi);

d) printf (“%#\n”, 100);

e) scanf(“%s”, string);

f) scanf(“%[^0123456789]”, n);

g) printf(“%\*.\*f\n”, x, y, 87.4573);

h) scanf(“%f%%”, &yuzde);

i) printf (“%+20.3Lf\n”, 3.333333);

ALIŞTIRMALAR

9.4 Aşağıdakiler için bir printf yada scanf ifadesi yazınız.

a) unsigned int tipindeki 40000 sayısını 15 basamak genişliğinde bir alan içinde sola

dayalı olarak 8 basamak biçiminde yazdırınız.

b) heks değişkenine onaltılık sayı sisteminden bir sayı alınız.

c) 200 sayısını işaretli ve işaretsiz olarak yazdırınız.

d) Başında 0x olmak üzere 100 sayısını onaltılık sistemde yazdırınız.

e) p harfiyle karşılaşıncaya kadar s dizisine karakter yazınız.

f) 1.234 sayısını, sayının önünde sıfırlar olacak şekilde 9 basamak genişliğinde bir alan

içine yazdırınız.

g) ss:dd:sn biçiminde zamanı alınız. saati saat, dakikayı dakika ve saniyeyi saniye

değişkenlerinde saklayınız. Giriş akışında (:) ile karşılaşınca bunları atlayınız. Atama

bastıma karakterini kullanınız.

h) Standart girişten “karakterler” stringini alın ve s dizisinde saklayın. Tırnak(“”)

işaretleri diziye yazılmasın.

i) ss:dd:sn biçiminde zamanı alınız. saati saat, dakikayı dakika ve saniyeyi saniye

değişkenlerinde saklayınız. Giriş akışında (:) ile karşılaşınca bunları atlayınız . Atama

bastırma karakterini kullanmayınız.

9.5 Aşağıdaki ifadeler sonucunda ekrana ne yazdırılacağını gösteriniz. Eğer bir ifade yanlış

ise nedenini belirtiniz.

a) printf (“%-10d\n”, 10000);

b) printf (“%c\n”, “Bu bir stringtir.”);

c) printf (“%\*.\*lf\n”, 8, 3, 1024.98765);

d) printf (“%#o\n%#X\n%#e\n”, 17, 17, 1008.83689);

e) printf (“% ld\n%+ld\n”, 1000000, 1000000);

f) printf (“%10.2E\n”, 444.93738);

g) printf (“%10.2g\n”, 444.93738);

h) printf (“%d\n”, 10.987);

9.6 Aşağıdaki program parçacıklarındaki hataları bulunuz ve nasıl düzeltileceğini açıklayınız.

a) printf (“%s\n”, ‘Iyi ki dogdun’);

b) printf (“%c\n”, ‘Merhaba’);

c) printf (“%c\n”, “Bu bir stringtir”);

d) Aşağıdaki ifade “Huseyin & Metin” yazar.

printf (“”%s””, “Huseyin & Metin”);

e) char gun[] = “Pazar”;

printf (“%s\n”, gun[3]);

f) printf (‘İsminizi Girin: ‘);

g) printf (%f, 123.456);

h) Aşağıdaki ifade ‘O’ ve ‘K’ karakterlerini ekrana yazdırır.

390

printf (“%s%s\n”, ‘O’, ‘K’);

i) char s[10];

scanf(“%c”, s[7]);

9.7 10 elemanlık sayi dizisine, 1 ile 1000 arasında 10 rasgele sayı atayan bir program yazınız.

Her değer için, o değeri ve toplam yazılan karakter sayısını ekrana yazdırın. Her değer için

kullanılan karakter sayısını %d dönüşüm belirtecini kullanarak yazdırın. Çıktı aşağıdaki

şekilde olmalıdır.

Değer Toplam karakter sayısı

342 3

1000 7

963 10

6 11

vs.

9.8 scanf ifadesinde kullanılan %d ve %i dönüşüm belirteçlerinin farkını gösteren bir program

yazınız.

scanf(“%i%d”, &x, &y);

printf(“%d %d\n”, x, y);

ifadelerini kullanarak değerleri alın ve yazdırın. Programı aşağıdaki veri kümesini kullanarak

test ediniz.

10 10

-10 -10

010 010

0x10 0x10

9.9 %p dönüşüm belirtecini ve diğer bütün tamsayı dönüşüm belirteçlerini kullanarak

gösterici değerlerini yazdıran bir program yazınız. Hangileri değişik değerler yazdırdı?

Hangilerinde hatalar meydana geldi? %p sisteminizdeki adresi hangi biçimde gösterdi?

9.10 12345 tamsayısını ve 1.2345 ondalıklı sayısını, farklı boyutlardaki alanlarda yazdıran bir

program yazınız. Bu değerler, basamak sayılarından daha küçük alanlarda yazdırıldıklarında

ne oldu?

9.11 100.463627 sayısını, en yakın onluk rakama , en yakın yüzlük rakama, en yakın binlik

rakama ve en yakın on binlik rakama yuvarlayarak yazdırınız.

9.12 Klavyeden bir sayı alarak bu stringin uzunluğunu bulan ve bu stringi uzunluğunu iki katı

alan içinde yazdıran bir program yazınız.

9.13 0 ile 212 arasında değerler alan Fahrenayt tamsayı değerini, float tipindeki derece

değerine 3 basamak duyarlılıkta çeviren bir program yazınız.

derece = 5.0 / 9.0 \* (fahrenayt – 32);

391

formülünü kullanınız. Çıktı, her biri 10 karakterlik, sağa yaslı iki sütun halinde olmalıdır ve

derecelerden önce, değerlerin pozitif mi negatif mi olduğunu gösteren artı ya da eksi işareti

gelmelidir.

9.14 Şekil 9.16’daki bütün çıkış sıralarını test eden bir program yazınız. İmleci hareket ettiren

çıkış sıraları için, çıkış sırasını yazdırmadan önce ve yazdırdıktan sonra bir karakter

yazdırınız ki imlecin nereye hareket ettiği belli olsun.

9.15 ? karakterinin printf biçim kontrol dizesi içinde /? çıkış sırası kullanılmadan string bilgi

karakteri biçiminde yazdırmanın mümkün olup olmadığını test eden bir program yazınız.

9.16 Bütün scanf tamsayı dönüşüm belirteçlerini kullanarak, 437 sayısını klavyeden aldıran

bir program yazınız. Tüm giriş değerlerini, tüm tamsayı dönüşüm belirteçlerini kullanarak

ekrana yazdırınız.

9.17 1.2345 sayısını e, f ve g dönüşüm belirteçlerini kullanarak klavyeden alan bir program

yazınız. Her değişkenin değerini ekrana yazdırarak bu dönüşüm belirteçlerinin her birinin bu

sayıyı almada kullanılabileceğini ispatlayınız.

9.18 Bazı programlama dillerinde stringler (‘’) ya da (“”) tırnakları ile alınırlar. hnk, “hnk” ve

‘hnk’ stringlerini alan bir C programı yazınız. Tırnak işaretleri C tarafından ihmal mi edildi

yoksa stringin birer parçalarıymış gibi mi işlem gördü?

9.19 printf ifadesi içindeki biçim kontrol dizesinde, %c dönüşüm belirtecini kullanarak; sabit

karakter çıkış sırası ‘\?’ kullanılmadan, ? karakterinin, ‘?’ karakter sabiti biçiminde yazdırılıp

yazdırılmayacağına karar veren bir program yazınız.

9.20 g dönüşüm belirtecini kullanarak 9876.12345 değerini ekrana yazdıran bir program

yazınız. Bu değeri 1’den 9’a kadar değişen duyarlılıkta yazdırınız.

392

YAPILAR, BİRLİKLER, BİT İŞLEME VE SAYMA SABİTLERİ

( ENUMARATIONS )

AMAÇLAR

 Yapılar,birlikler ve sayma sabitleri oluşturabilmek ve kullanmak.

 Yapıları fonksiyonlara değere göre çağırma ve referansa göre çağırma ile geçirebilmek

 Bit operatörleriyle veri işleyebilmek

 Verileri daha verimli bir şekilde depolayabilmek için bit alanları yaratmak

BAŞLIKLAR

10.1 GİRİŞ

10.2 YAPI TANIMLAMALARI

10.3 YAPILARA İLK DEĞER ATAMAK

10.4 YAPI ELEMANLARINA ULAŞMAK

10.5 YAPILARI FONKSİYONLARLA KULLANMAK

10.6 typedef

10.7 ÖRNEK:YÜKSEK PERFORMANSLI KART KARMA VE DAĞITMA

10.8 BİRLİKLER

10.9 BİT OPERATÖRLERİ

10.10 BİT ALANLARI

10.11 SAYMA SABİTLERİ

Özet\*Genel Programlama Hataları\*İyi Programlama Alıştırmaları\*Taşınırlık

İpuçları\*Performans İpuçları\*Yazılım Mühendisliği Gözlemleri\*Cevaplı

Alıştırmalar\*Cevaplar\*Alıştırmalar

10.1 GİRİŞ

Yapılar, birbirleriyle ilişkili değişkenlerin bir isim altında toplanmasıdır.

Diziler aynı veri tipinde elemanlar içerebilirken yapılar değişik veri

tiplerinde elemanlar içerebilir. Yapılar, dosya ( bakınız 11.ünite ) içinde

tutulacak kayıtları oluşturmakta kullanılırlar. Göstericiler ve yapılar, bağlı

listeler, yığınlar, sıralar ve ağaçlar gibi daha karmaşık veri yapılarının

(bakınız 12.ünite) oluşturulmasını kolaylaştırırlar.

10.2 YAPI TANIMLAMALARI

Yapılar, diğer tipte nesneler kullanılarak oluşturulan, türetilmiş veri tipleridir. Aşağıdaki veri

tanımını inceleyiniz:

struct kart{

char \*taraf;

char \*takim;

393

};

struct anahtar kelimesi yapı tanımını başlatır. kart tanıtıcısı yapı etiketidir (structure tag).

Yapı etiketleri, yapı tanımına isim verir ve struct anahtar kelimesiyle kullanılarak yapı

tipinde değişkenler bildirir. Bu örnekte, yapı tipi struct kart’tır. Yapı tanımında parantezler

içinde bildirilen değişkenler yapı elemanlarıdır. Aynı yapının elemanları, kendilerine özel

isimlere sahip olmalıdır ancak farklı iki yapı aynı isimde elemanlar içerebilir. Bu, herhangi bir

karışıklığa yol açmaz ( ileride bunun sebebini göreceğiz). Her yapı tanımı noktalı virgül ile

sonlanmalıdır.

Genel Programlama Hataları 10.1

Yapı tanımını sonlandıran noktalı virgülü unutmak.

struct kart tanımı, char \* tipinde iki eleman ( taraf ve takim ) içermektedir. Yapı

elemanları, temel veri tipleri değişkenleri ( int, float vb.) ya da dizi ve diğer yapılar gibi

toplulukların değişkenleri olabilir. 6. ünitede gördüğümüz gibi, bir dizinin her elemanı aynı

tipte olmalıdır. Yapı elemanları ise değişik veri tiplerinden olabilir. Örneğin, aşağıdaki yapı

struct isci{

char adi[20];

char soyadi[20];

int yas;

char cinsiyet;

double saatlikUcreti;

};

ad ve soyadı için karakter dizisi elemanları, işçinin yaşı için bir int eleman, işçinin cinsiyeti

için ‘E’ ya da ‘K’ içeren bir char eleman ve işçinin saatlik ücreti için bir double eleman

içermektedir.

Bir yapı kendi örneğini içeremez. Örneğin, struct isci tipinde bir değişken, struct isci tanımı

içinde bildirilemez. struct isci’yi gösteren bir gösterici ise yapı tanımlaması içinde

bildirilebilir. Örneğin,

struct isci2{

char adi[20];

char soyadi[20];

int yas;

char cinsiyet;

double saatlikUcreti;

struct isci2 kisi; //hata

struct isci2 \*ePtr; //gösterici

};

tanımlamasında, struct isci2 kendi kendisinin bir örneğini ( kisi ) içermektedir ve bu bir

hatadır. ePtr ise (struct isci2 tipini gösteren bir göstericidir) tanımlama içinde kullanılabilir.

Aynı yapı tipini gösteren bir gösterici elemanına sahip yapılara, kendine dönüşlü ( self

referential ) yapılar denir. Kendine dönüşlü yapılar, 12.ünitede çeşitli bağlı veri yapıları

oluşturmada kullanılacaktır.

394

Az önceki yapı tanımlaması hafızada yer ayırmaz, bunun yerine değişkenler bildirmek için

kullanılacak yeni bir veri tipi oluşturur. Yapı değişkenleri diğer tiplerdeki değişkenler gibi

bildirilirler.

struct kart a, deste[52], \*cPtr;

bildirimi, struct kart tipinde bir a değişkeni, struct kart tipinde 52 elemana sahip bir deste

dizisi ve struct kart ’ı gösteren bir gösterici değişkeni bildirir.

Verilen bir yapı tipindeki değişkenler, değişken isimleri yapı tanımının sonundaki parantez ile

yapı tanımlamasını sonlandıran noktalı virgül arasına, virgüllerle ayrılmış bir biçimde

yazılarak bildirilebilir. Örneğin, az önceki bildirim struct kart tanımlamasıyla aşağıdaki

biçimde birleştirilebilir:

struct kart{

char \*taraf;

char \*takim;

} a, deste[52], \*cPtr;

Yapı etiketi isteğe bağlıdır. Yapı etiket ismi içermeyen bir yapı tanımlamasında, yapı

tipindeki değişkenler yalnızca yapı tanımlaması içinde bildirilebilirler. Ayrı bir bildirim ile

bildirilemezler.

İyi Programlama Alıştırmaları 10.1

Yeni bir yapı tipi oluştururken her zaman yapı etiket ismi kullanın. Yapı etiket ismi, programda daha

sonradan o yapı tipinde yeni değişkenler bildirmek için gereklidir.

İyi Programlama Alıştırmaları 10.2

Anlamlı bir yapı etiket ismi kullanmak programın daha anlaşılır olmasını sağlar.

Yapılarla kullanılabilen geçerli işlemler : yapı değişkenlerini aynı tipte yapı değişkenlerine

atamak, bir yapı değişkeninin adresini (&) almak, yapı değişkenlerine erişmek (bakınız kısım

10.4) ve yapı değişkeninin boyutunu belirlemek için sizeof operatörünü kullanmaktır.

Genel Programlama Hataları 10.2

Bir tipte yapıyı başka bir tipteki yapıya atamak.

Yapılar, == ve != operatörleri kullanılarak karşılaştırılamazlar çünkü yapı elemanları hafızada

ardışık byte’lar içinde bulunmak zorunda değildir. Bazen yapılar içinde boşluklar bulunabilir

çünkü bilgisayarlar belli veri tiplerini özel sınırlar içinde depolarlar. Bu sınırlar, verileri

bilgisayarda tutmak için kullanılan standart hafıza birimleri olarak düşünülebilir. Bu standart

birim 1 (halfword), 2 (word) ya da 4 byte (double word) uzunluğunda olabilir. Aşağıdaki yapı

tanımlamasını inceleyelim;

struct ornek{

char c;

int i;

} numune1,numune2;

395

Bu yapı tanımlaması ile struct ornek tipinde iki değişken; numune1 ve numune2

bildirilmiştir. 2-byte sınırlar kullanan bir bilgisayar, struct ornek yapısının her elemanını bir

sınıra hizalayabilir. Yani her elemanı bir sınırın başlangıcına yerleştirir. ( bu, her makinede

değişebilir)

Şekil 10.1’de, struct ornek tipinde bir değişkenin nasıl hizalandığını gösteren bir örnek

bulacaksınız. Bu örnek, yapının elemanlarının ‘a’ karakterine ve 97 tamsayısına atandığı

düşünülerek verilmiştir. Bu örnekte yapının elemanlarının atandığı değerlerin bit karşılıkları

verilmiştir. Eğer elemanlar sınırların başlangıçlarına hizalanırsa, struct ornek tipindeki

değişkenlerin depolanması esnasında 1-byte’lık bir boşluk oluşacaktır. Bu 1 byte içindeki

değer tanımlanmamıştır. Gerçekte numune1 ve numune2’nin elemanlarının değerleri eşit

bile olsa, karşılaştırıldıklarında (1-byte çok büyük olasılıkla eş değerler içermeyeceğinden)

eşit olmayacaklardır.

Genel Programlama Hataları 10.3

Yapıları karşılaştırmak bir yazım hatasıdır.

Taşınırlık İpuçları 10.1

Belli bir tipte veri parçalarının boyutu, makine bağımlı olduğu için ve depolama hizalama hususları

da makine bağımlı olduğundan, yapıların gösterilmesi de makine bağımlıdır.

byte 0 1 2 3

Şekil 10.1 struct ornek tipinde bir değişkenin sınırlar içinde muhtemel hizalanışının temsili

10.3 YAPILARA İLK DEĞER ATAMAK

Yapılara, dizilerde olduğu gibi atama listeleri ile atama yapılır. Yapıya

değer atamak için, yapı değişkeninin adından sonra eşittir işareti ve küme

parantezleri içinde virgüllerle ayrılmış atama değerleri kullanılır. Örneğin,

struct kart a={“İki”,”Kupa”};

bildirimi daha önceden tanımlanmış struct kart tipinde bir a değişkeni yaratır ve bu

değişkenin taraf elemanına “İki” ve takim elemanına “Kupa” değerini atar. Eğer atama

listesinde yapı elemanlarından daha az sayıda atama değeri varsa, kalan elemanlar otomatik

olarak 0’a (ya da eleman gösterici ise NULL’a) atanır. Yapı değişkenleri fonksiyon tanımı

dışında bildirilirse ve dışarıda yapılan bu bildirimde özel olarak değerlere atanmazsa, ilk

değer olarak 0’a ya da NULL’a atanırlar. Yapı değişkenleri, aynı tipte yapı değişkenlerine

atandıkları atama ifadelerinde ya da yapı elemanlarına değerlerin atandığı atama ifadelerinde

değerlere atanabilirler.

10.4 YAPI ELEMANLARINA ULAŞMAK

Yapı elemanlarına ulaşmak için iki operatör kullanılır: Yapı elemanı operatörü (.) (aynı

zamanda nokta operatörü olarak da bilinir) ve yapı gösterici operatörü ( ->) (aynı zamanda ok

0110001 00000000 01100001

396

operatörü olarak da adlandırılır) Yapı elemanı operatörü, yapı elemanına yapı değişkeninin

ismini kullanarak erişir. Örneğin, az önceki bildirimden sonra a yapısının takim elemanını

yazdırmak için

printf(“%s”, a.takim);

ifadesi kullanılır.

Yapı gösterici operatörü ( eksi işareti(-) ve büyüktür işareti( >) arasında boşluk bırakmadan

yazılır ) yapı elemanına, yapıyı gösteren bir gösterici ile ulaşır. struct kart yapısını göstermek

için bir aPtr göstericisinin bildirildiğini ve a yapısının adresinin aPtr’ye atandığını

düşünelim. a yapısının takim elemanını yazdırmak için

printf(“%s”, aPtr -> takim);

ifadesi kullanılır.

aPtr-> takim deyimi, (\*aPtr).takim ile eşdeğerdir. Burada parantezler gereklidir çünkü yapı

elemanı operatörü ( . ), gösterici operatöründen ( \* ) daha yüksek önceliğe sahiptir. Yapı

gösterici operatörü ve yapı elemanı operatörü, parantez ve dizilerde kullanılan köşeli parantez

operatörüyle ( [ ] ) birlikte en yüksek önceliğe sahiptir ve soldan sağa doğru işler.

İyi Programlama Alıştırmaları 10.3

Farklı tipteki yapıların elemanları için aynı isimleri kullanmaktan kaçının.Buna izin verilmiştir ancak

karışıklık yaratabilir.

İyi Programlama Alıştırmaları 10.4

-> ve . operatörlerinden önce ve sonra boşluk bırakmayınız. Bu sayede, bu operatörlerin kullanıldığı

deyimlerin aslında tek bir değişken ismi olduğu vurgulanmış olur.

Genel Programlama Hataları 10.4

Yapı gösterici operatörünü yazarken – ve > arasına boşluk koymak. ( ya da ?: operatörü haricinde

birden çok karakter kullanılarak yazılan operatörler arasına boşluk koymak)

Genel Programlama Hataları 10.5

Yapı elemanının ismini tek başına kullanarak yapı elemanına ulaşmaya çalışmak

Genel Programlama Hataları 10.6

Bir yapı elemanını, gösterici ve yapı elemanı operatörü kullanarak belirtirken parantez kullanmamak.

(örneğin \*aPtr.takim bir yazım hatasıdır.)

Şekil 10.2’deki program yapı elemanı operatörü ve yapı gösterici operatörünün kullanımını

göstermektedir. Yapı elemanı operatörü kullanılarak, a yapısının elemanlarına sırasıyla “As”

ve “Maca” değerleri atanmıştır (satır 16 ve 17). aPtr göstericisi, a yapısının adresine

atanmıştır (satır 18). printf fonksiyonu a yapısının elemanlarını

1) değişken ismi ( a ) ve yapı elemanı operatörünü birlikte kullanarak

2) yapı gösterici operatörüyle aPtr göstericisini birlikte kullanarak,

3) Gösterdiği nesneye erişilmiş aPtr göstericisini yapı elemanı operatörüyle birlikte

kullanarak yazdırmaktadır (19-22.satır).

397

1 /\* Şekil 10.2: fig10\_02.c

2 yapı elemanı operatörünü ve

3 yapı gösterici elemanını kullanmak.\*/

4 #include <stdio.h>

5

6 struct kart {

7 char \*taraf;

8 char \*takim;

9 };

10

11 int main( )

12 {

13 struct kart a;

14 struct kart \*aPtr;

15

16 a.taraf = "As";

17 a.takim = "Maça";

18 aPtr = &a;

19 printf( "%s %s%s\n%s %s%s\n%s %s%s\n",

20 a.takim, a.taraf, ”ı”,

21 aPtr->takim, aPtr->taraf, “ı”,

22 ( \*aPtr ).takim, ( \*aPtr ).taraf, “ı” );

23 return 0;

24 }

Maça Ası

Maça Ası

Maça Ası

Şekil 10.2 Yapı elemanı operatörünü ve yapı gösterici elemanını kullanmak.

10.5 YAPILARI FONKSİYONLARLA KULLANMAK

Yapılar fonksiyonlara, yapı elemanlarının bağımsız bir şekilde geçirilmesiyle, tüm yapının

geçirilmesiyle ya da yapıyı gösteren bir göstericinin geçirilmesiyle geçirilirler. Yapılar ya da

yapı elemanları fonksiyonlara geçirilirken, değere göre çağırma ile geçirilirler. Bu sebepten,

çağırıcının yapı elemanları çağrılan fonksiyonla değiştirilemez.

Bir yapıyı referansa göre çağırmak için yapı değişkeninin adresi geçirilir. Yapı dizileri, diğer

tüm diziler gibi, otomatik olarak referansa göre geçirilir.

6.ünitede, bir dizinin yapılar kullanılarak değere göre çağırma ile geçirilebileceğini

söylemiştik. Bir diziyi değere göre çağırma ile geçirebilmek için dizinin eleman olarak

kullanıldığı bir yapı oluşturmak gerekir. Yapılar değere göre geçirildiğinden, dizi de değere

göre geçirilmiş olur.

Genel Programlama Hataları 10.7

398

Yapıların, diziler gibi otomatik olarak referansa göre çağırma ile geçirildiklerini düşünmek ve

çağırıcının yapısını çağırılan fonksiyon içinde değiştirmeye çalışmak.

Performans İpuçları 10.1

Yapıları referansa göre çağırma ile geçirmek, yapıları değere göre çağırma ile geçirmekten

(tüm yapının kopyasının oluşturulması gerekir) daha verimlidir.

10.6 typedef

typedef anahtar kelimesi, daha önceden tanımlanmış veri tipleri için eşanlamlı sözcükler (ya

da takma isimler) yaratan bir mekanizma sağlar. Yapı tipi isimleri genellikle typedef ile

tanımlanarak daha kısa tip isimleri oluşturulur.Örneğin,

typedef struct kart Kart;

ifadesi struct kard tipi ile eş anlamda kullanılan, Kart isminde yeni bir tip yaratır. C

programcıları typedef anahtar kelimesini, yapı tipi tanımlarken kullanırlar böylece yapı

etiketi kullanmaya gerek kalmaz. Örneğin,

typedef struct{

char \*taraf;

char \*takim;

} Kart;

tanımı, Kart yapı tipini ayrı bir typedef ifadesi kullanmaya gerek kalmadan yaratır.

İyi Programlama Alıştırmaları 10.5

typedef isimlerinin ilk harflerini büyük harf ile yazarak, bu isimlerin başka tip isimleri için eş anlamlı

isimler olduğunu vurgulamak.

Artık Kart, struct kart tipinde değişkenler bildirmek için kullanılabilir.

Kart deste[52];

bildirimi, 52 Kart yapısından (yani struct kart tipinde değişkenlerden) oluşan bir dizi

bildirir. typedef ile yeni bir isim yaratmak yeni bir tip yaratmaz; typedef daha önceden var

olan bir tip ismi için, eş anlamlı olarak kullanılabilecek yeni tip isimleri yaratır. Anlamlı bir

isim, programın daha anlaşılır olmasını sağlar. Örneğin, az önceki bildirimi okuduğumuzda

deste’nin 52 Kart’ın dizisi olduğunu biliriz.

typedef sıklıkla, temel veri tipleri için eş anlamlı isimler yaratmada kullanılır. Örneğin, 4-byte

tamsayılara ihtiyaç duyan bir program, bir sistemde int, başka bir sistemde ise long tipini

kullanabilir. Taşınılabilir programlar için genellikle typedef kullanılarak, 4-byte tamsayılar

için Tamsayi gibi eşanlamlı bir isim yaratılır. Tamsayi ismi, programın tüm sistemlerde

çalışabilmesi için programda yalnızca bir kez değiştirilir.

Taşınırlık İpuçları 10.2

typedef kullanmak programı daha taşınır bir hale getirir.

399

10.7 ÖRNEK: YÜKSEK PERFORMANSLI KART KARMA VE

DAĞITMA

Şekil 10.3’teki program, 7.ünitede tartışılan kart karma ve dağıtma uygulamasına

dayanmaktadır. Program, kartların destesini, yapılardan oluşan bir dizi ile temsil etmektedir.

Program, yüksek performanslı kart karma ve dağıtma algoritmaları kullanmaktadır. Programın

çıktısı Şekil 10.4’te gösterilmiştir.

1 /\* Şekil. 10.3: fig10\_03.c

2 Yapıların kullanılmasıyla kart karılması ve dağıtılması \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5 #include <time.h>

6

7 struct kart {

8 const char \*taraf;

9 const char \*takim;

10 };

11

12 typedef struct kart Kart;

13

14 void desteDoldur( Kart \* const, const char \*[ ],

15 const char \*[ ] );

16 void desteyiKar ( Kart \* const );

17 void dagit ( const Kart \* const );

18

19 int main( )

20 {

21 Kart deste[ 52 ];

22 const char \*taraf [ ] = { "As", "İki", "Üç",

23 "Dört", "Beş",

24 "Altı", "Yedi", "Sekiz",

25 "Dokuz", "On",

26 "Vale", "Kız", "Papaz"};

27 const char \*takim [ ] = { "Kupa", "Karo",

28 "Sinek", "Maça"};

29

30 srand( time( NULL ) );

31

32 desteDoldur( deste, taraf, takim );

33 desteyiKar( deste );

34 dagit( deste );

35 return 0;

36 }

37

38 void desteDoldur( Kart \* const wDeste, const char \* wTaraf[ ],

400

39 const char \* wTakim[ ] )

40 {

41 int i;

42

43 for ( i = 0; i <= 51; i++ ) {

44 wDeste[ i ].taraf = wTaraf[ i % 13 ];

45 wDeste[ i ].takim = wTakim[ i / 13 ];

46 }

47 }

48

49 void desteyiKar( Kart \* const wDeste )

50 {

51 int i, j;

52 Kart gecici;

53

54 for ( i = 0; i <= 51; i++ ) {

55 j = rand( ) % 52;

56 gecici = wDeste[ i ];

57 wDeste[ i ] = wDeste[ j ];

58 wDeste[ j ] = gecici;

59 }

60 }

61

62 void dagit( const Kart \* const wDeste )

63 {

64 int i;

65

66 for ( i = 0; i <= 51; i++ )

67 printf( "%5s %-8s%c", wDeste[ i ].takim,

68 wDeste[ i ].taraf,

69 ( i + 1 ) % 2 ? '\t' : '\n' );

70 }

Şekil 10.3 Yüksek Performanslı kart karma ve dağıtma uygulaması

Karo Sekiz Kupa As

Sinek Sekiz Sinek Beş

Kupa Yedi Karo İki

Sinek As Karo On

Maça İki Karo Altı

Maça Yedi Sinek İki

Sinek Vale Maça On

Kupa Papaz Karo Vale

Kupa Üç Karo Üç

Sinek Üç Sinek Dokuz

Kupa On Kupa İki

Sinek On Karo Yedi

Sinek Altı Maça Kız

Kupa Altı Maça Üç

Karo Dokuz Karo As

Maça Vale Sinek Beş

401

Karo Papaz Sinek Yedi

Maça Dokuz Kupa Dört

Maça Altı Maça Sekiz

Karo Kız Karo Beş

Maça As Kupa Dokuz

Sinek Papaz Kupa Beş

Maça Papaz Karo Dört

Kupa Kız Kupa Sekiz

Maça Dört Kupa Vale

Sinek Dört Sinek Kız

Şekil 10.4 Yüksek performanslı kart karma ve dağıtma uygulamasının çıktıları.

Programda, desteDoldur fonksiyonu (satır 38’de tanımlanmıştır), her takımın As’ından

Papaz’ına kadar tüm kartların değerlerini yazarak bu değerleri Kart dizisine atamaktadır.

Kart dizisi, desteyiKar fonksiyonuna (49.satırda tanımlanmıştır) geçirilerek (satır 33) yüksek

performanslı karma algoritması gerçekleştirilmiştir. desteyiKar fonksiyonu, 52 Kart yapısını

argüman olarak almaktadır. Fonksiyon 52 kartı (dizi belirteçleri 0’dan 51’e kadar olan

kartları) for yapısıyla dolaşmaktadır (satır 54). Her kart için 0 ile 51 arasında bir sayı rasgele

seçilmektedir. Daha sonra o andaki Kart yapısıyla, rasgele seçilen Kart yapısı dizi içinde

değiştirilmektedir (satır 56-satır 58). Tüm dizi bir kez geçirilerek toplam 52 değiştirme

yapılmıştır ve Kart yapılarının dizisi karılmıştır. Bu algoritma, 7.ünitede gösterilen karma

algoritması gibi belirsiz ertelemeden etkilenmez. Kart yapıları dizi içinde yer

değiştirdiğinden, dagit fonksiyonu (62.satırda tanımlanmıştır) yüksek performanslı dağıtma

algoritmasını gerçekleştirebilmek için karılmış kartların dizisini yalnızca bir kez geçirecektir.

Genel Programlama Hataları 10.8

Yapılardan oluşan dizilerde, bağımsız yapıları belirtmek için kullanılan dizi belirteçlerini

unutmak.

10.8 BİRLİKLER

Birlikler (yapılar gibi), türetilmiş veri tipleridir. Birlik elemanları aynı depolama alanını

kullanırlar. Bir programdaki farklı durumlar için bazı değişkenler kullanılmazken bazıları

kullanılır. Bu sebepten, bir birlik kullanılmayan değişkenler için hafızayı boş yere işgal etmek

yerine ayrılan alanı kullanır. Bir birliğin elemanları her tipte olabilir. Bir birliği

depolayabilmek için kullanılan byte sayısı en az birliğin en büyük elemanını tutabilecek kadar

olmalıdır. Çoğu durumda birlikler, iki ya da daha fazla veri tipi içerir. Bir anda yalnızca bir

eleman, bu sebepten de yalnızca bir veri tipi kullanılabilir. Birlik içindeki verinin uygun tiple

kullanılmasını sağlamak programcının sorumluluğundadır.

Genel Programlama Hataları 10.9

Birlik içinde depolanmış verinin tipini farklı bir tip ile kullanmak bir mantık hatasıdır.

Taşınırlık İpuçları 10.3

Eğer birlikte depolanan veri yanlış bir tiple kullanılırsa, sonuçlar uygulamaya bağlı olarak farklılık

gösterebilir.

Bir birlik union anahtar kelimesiyle, yapılarla aynı biçimde bildirilir.

402

union sayi{

int x;

double y;

};

birlik bildirimi sayi’nin birlik tipinde olduğunu ve int x ile double y elemanlarına sahip

olduğunu belirtir. Birlik tanımlamaları bir programda main’den önce yer alır. Böylece

tanımlama, programdaki tüm fonksiyonlarda değişken bildirmek için kullanılabilir.

Yazılım Mühendisliği Gözlemleri 10.1

struct bildiriminde olduğu gibi bir union bildirimi de yeni bir tip yaratır. union ya da struct

bildirimini fonksiyonların dışında yapmak global değişkenler yaratmaz.

Birliklerde yapılabilen işlemler şunlardır: bir birliği aynı tipte başka bir birliğe atamak,

birliğin adresini almak (&) ve birlik elemanlarına yapı elemanı operatörü ve yapı gösterici

operatörü kullanarak erişmek. Birlikler, yapılarda anlattığımız sebeplerden dolayı, == ve !=

operatörüyle karşılaştırılamazlar.

Bildirimde bir birliğe, yalnızca ilk birlik elemanının tipiyle aynı olan değerler atanabilir.

Örneğin az önceki birlik bildiriminden sonra yapılacak

union sayi deger ={10};

ataması geçerli olacaktır çünkü atanan değer int tipindedir. Fakat aşağıdaki bildirim

geçersizdir:

union sayi deger ={1.43};

Genel Programlama Hataları 10.10

Birlikleri karşılaştırmak yazım hatasıdır.

Genel Programlama Hataları 10.11

Bir birlik bildiriminde, birliğin ilk elemanının tipinden farklı tipte bir değer ile atama

yapmak.

Taşınırlık İpuçları 10.4

Bir birliği depolamak için gerekli olan alan, uygulamadan uygulamaya farklılık gösterebilir.

Taşınırlık İpuçları 10.5

Bazı birlikler başka bilgisayar sistemlerine kolaylıkla taşınamaz. Bir birliğin taşınılabilirliği

genellikle sistemde birliğin eleman tiplerinin depolanmasında kullanılan hizalama

yöntemlerine dayanır.

Performans İpuçları 10.2

Birlikler depolama alanında kazanç sağlar.

Şekil 10.5’deki program, birlik içinde int ve double olarak depolanan değerleri yazdırmak

için union sayi tipindeki deger değişkenini (satır 12) kullanmaktadır. Programın çıktısı

403

uygulama bağımlıdır. Programın çıktısı, double bir değerin makinenin içindeki temsilinin int

temsilinden oldukça farklı olabileceğini göstermektedir.

1 /\* Şekil 10.5: fig10\_05.c

2 Birlik Örneği \*/

3 #include <stdio.h>

4

5 union sayi {

6 int x;

7 double y;

8 };

9

10 int main( )

11 {

12 union sayi deger;

13

14 deger.x = 100;

15 printf( "%s\n%s\n%s%d\n%s%f\n\n",

16 "Tamsayı üyesine bir sayı koyun",

17 "ve bütün üyeleri yazdırın.",

18 "int: ", deger.x,

19 "double:\n", deger.y );

20

21 deger.y = 100.0;

22 printf( "%s\n%s\n%s%d\n%s%f\n",

23 "Ondalıklı sayı üyesine bir sayı koyun",

24 "ve bütün üyeleri yazdırın.",

25 "int: ", deger.x,

26 "double:\n", deger.y );

27 return 0;

28 }

Tamsayı üyesine bir sayı koyun

ve bütün üyeleri yazdırın.

int: 100

double:

-92559592117433136000000000000000000000000000000000000000000000.00000

Ondalıklı sayı üyesine bir sayı koyun

ve bütün üyeleri yazdırın.

int: 0

double:

100.000000

Şekil 10.5 Bir birliğin değerini içerdiği veri tipleri cinsinden yazdırmak

10.9 BİT OPERATÖRLERİ

404

Bilgisayarın içinde tüm veriler bitlerle temsil edilir. Her bit yalnızca 0 ya da 1 değerini

alabilir. Çoğu sistemde 8 bit bir byte’ı oluşturur. Byte, char tipi için standart depolama

birimidir. Diğer veri tipleri daha büyük sayıda byte’lar içinde saklanır. Bit operatörleri

operandlarının (char, int ve long, hem signed hem de unsigned ) bitlerini yönetmek için

kullanılır. Bit operatörleriyle genellikle işaretsiz tamsayılar ( unsigned int ) kullanılır.

Taşınırlık İpuçları 10.6

Verilerin bitleriyle yapılan işlemler makinelere bağımlıdır.

Bu kısımdaki bit operatörü açıklamaları, tamsayı operandların ikilik sistemdeki gösterimlerini

göstermektedir. İkilik sistem hakkında daha detaylı açıklama için Ekler E’yi inceleyiniz.Bu

kısımdaki programlar Microsoft Visual C++ ile denenmiştir. Bit işlemleri makine-bağımlı

olduğundan bu programlar sizin sisteminizde çalışmayabilir.

Bit operatörleri şunlardır: AND(&), OR( | ), EXCLUSIVE OR( ^ ), sola kaydırma(<<), sağa

kaydırma(>>) ve tümleyen(~) . AND, OR, EXCLUSIVE OR operatörleri operandlarını bit bit

karşılaştırır. AND operatörü (bit düzeyinde VE operatörü de denir), iki operandında da ilgili

bitte 1 varsa sonuçtaki biti 1 yapar. OR operatörü (bit düzeyinde VEYA operatörü de denir),

operandlarındaki bitlerden birinde ya da ikisinde birden 1 varsa sonuçtaki biti 1

yapar.EXCLUSIVE OR (bit düzeyinde ÖZEL VEYA operatörü de denir) operatörü,

operandlarındaki bitlerden yalnızca biri 1 ise sonuçtaki biti 1 yapar. Sola kaydırma operatörü,

soldaki operandındaki bitleri sağdaki operandında belirtilen sayı kadar sola kaydırır. Sağa

kaydırma operatörü, soldaki operandındaki bitleri sağdaki operandında belirtilen sayı kadar

sağa kaydırır. Tümleyen operatörü, operandındaki 0 olan tüm bitleri 1 ve operandındaki 1

olan tüm bitleri 0 yapar. Bit operatörlerinin detaylı açıklamaları ilerdeki örneklerde

bulunabilir. Bit operatörleri Şekil 10.6’da özetlenmiştir.

Operatör Tanımlama

& AND İki operandın ikisinin de ilgili bitlerinde 1 varsa sonuçtaki bitler

1 yapılır.

| OR İki operandın ilgili bitlerinden en az biri 1 ise sonuçtaki bitler 1

yapılır.

^ EXCLUSIVE OR İki operandın ilgili bitlerinden yalnızca biri 1 ise sonuçtaki bitler

1 yapılır.

<< Sola kaydırma İlk operandındaki bitleri ikinci operandında belirtilen sayı kadar

sola kaydırır.Sağdan itibaren 0 ile doldurur.

>> Sağa kaydırma İlk operandındaki bitleri ikinci operandında belirtilen sayı kadar

sağa kaydırır.Soldan itibaren yapılacak doldurma makine

bağımlıdır.

~ Tümleyen Tüm 0 bitleri 1,tüm 1 bitleri 0 yapılır.

Şekil 10.6 Bit operatörleri

Bit operatörlerini kullanırken, değerleri ikilik sistemde yazdırmak bu operatörlerin etkilerini

gösterebilmek için oldukça kullanışlıdır. Şekil 10.7’deki program, unsigned bir tamsayıyı her

biri 8 bitten oluşan gruplarla ikilik gösterimde yazdırmaktadır.

405

1 /\* Şekil 10.7: fig10\_07.c

2 unsigned bir tamsayıyı bitlerle yazdırmak. \*/

3 #include <stdio.h>

4

5 void bitleriGoster( unsigned );

6

7 int main( )

8 {

9 unsigned x;

10

11 printf( "İşaretsiz bir tamsayı giriniz: " );

12 scanf( "%u", &x );

13 bitleriGoster( x );

14 return 0;

15 }

16

17 void bitleriGoster( unsigned deger )

18 {

19 unsigned c, maske = 1 << 31;

20

21 printf( "%7u = ", deger );

22

23 for ( c = 1; c <= 32; c++ ) {

24 putchar( deger & maske ? '1' : '0' );

25 deger <<= 1;

26

27 if ( c % 8 == 0 )

28 putchar( ' ' );

29 }

30

31 putchar( '\n' );

32 }

İşaretsiz bir tamsayı giriniz: 65000

65000 = 11111101 11101000

Şekil 10.7 unsigned bir tamsayıyı bitlerle yazdırmak.

bitleriGoster fonksiyonu (17.satırda tanımlanmıştır), AND operatörünü kullanarak deger

değişkeni ile maske değişkenini birleştirmektedir. Sıklıkla, AND operatörü maske adı verilen

bir operandla kullanılır. Bu operand, bazı bitleri 1 olan bir tamsayı değeridir. Maskeler bir

değerdeki bazı bitleri seçerken bazı bitleri de saklamak için kullanılır. bitleriGoster

fonksiyonunda maske değişkeni

1<<31 (10000000 00000000 00000000)

değerine atanmıştır.

406

Sola kaydırma operatörü, 1 değerini maske içindeki en düşük değerlikli (en sağdaki) bitten en

yüksek değerlikli (en soldaki) bite doğru kaydırır ve sağdan itibaren bitleri 0 ile doldurur.

putchar(deger&maske? ‘1’ : ’0’ );

ifadesi, deger değişkeninin o anda en solda bulunan biti için 1 ya da 0 yazdırılmasına karar

verir. deger ve maske değişkenleri & kullanılarak birleştirildiğinde, deger değişkeni içindeki

en yüksek değerlikli bit hariç diğer tüm bitler maskelenir (gizlenir) çünkü AND operatörünün

herhangi bir operandı 0 ise sonuç 0’dır. Eğer en soldaki bit 1 ise, deger&maske, 1 sonucunu

verir ve 1 yazdırılır. Eğer en soldaki bit 0 ise 0 yazdırılır. deger değişkeni daha sonra

deger<<1 deyimi ile bir bit sola kaydırılır (bu deger =deger<<1 yazmak ile aynıdır). Bu

işlemler, unsigned bir değişken olan deger içindeki tüm bitler için teker teker yapılır. Şekil

10.8, AND operatörüyle iki bitin birleştirilmesinden oluşacak sonuçları göstermektedir.

Genel Programlama Hataları 10.12

AND operatörü(&) yerine mantıksal VE operatörünü(&&) kullanmak ya da tam tersi.

Bit1 Bit2 Bit1&Bit2

0 0 0

1 0 0

0 1 0

1 1 1

Şekil 10.8 İki bit AND operatörüyle birleştirildiğinde oluşan sonuçlar.

Şekil 10.9, AND, OR, EXCLUSIVE OR ve tümleyen operatörlerinin kullanımını

göstermektedir. Program, bitleriGoster fonksiyonunu unsigned tamsayı değerlerini

yazdırmak için kullanmaktadır. Programın çıktısı Şekil 10.10’da gösterilmiştir.

1 /\* Şekil 10.9: fig10\_09.c

2 AND ,OR ,EXCLUSIVE OR ve

3 tümleyen operatörlerinin kullanımı \*/

4 #include <stdio.h>

5

6 void bitleriGoster( unsigned );

7

8 int main( )

9 {

10 unsigned sayi1, sayi2, maske, bitBirle;

11

12 sayi1 = 65535;

13 maske = 1;

14 printf( "Aşağıdakileri birleştirmenin sonucu \n" );

15 bitleriGoster( sayi1 );

16 bitleriGoster( maske );

17 printf( "AND operatorü & kullanıldığında\n" );

18 bitleriGoster( sayi1 & maske );

19

407

20 sayi1 = 15;

21 bitBirle = 241;

22 printf( "\nAşağıdakileri birleştirmenin sonucu \n" );

23 bitleriGoster( sayi1 );

24 bitleriGoster( bitBirle );

25 printf( "OR operatorü | kullanıldığında\n" );

26 bitleriGoster( sayi1 | bitBirle );

27

28 sayi1 = 139;

29 sayi2 = 199;

30 printf( "\nAşağıdakileri birleştirmenin sonucu\n" );

31 bitleriGoster( sayi1 );

32 bitleriGoster( sayi2 );

33 printf( "exclusive OR operatorü ^ kullanıldığında\n" );

34 bitleriGoster( sayi1 ^ sayi2 );

35

36 sayi1 = 21845;

37 printf( "\nAşağıdakinin Bire tümleyeni \n" );

38 bitleriGoster( sayi1 );

39 printf( "şöyle gösterilir\n" );

40 bitleriGoster( ~sayi1 );

41

42 return 0;

43 }

44

45 void bitleriGoster( unsigned deger )

46 {

47 unsigned c, maskeGoster = 1 << 31;

48

49 printf( "%7u = ", deger );

50

51 for ( c = 1; c <= 32; c++ ) {

52 putchar( deger & maskeGoster ? '1' : '0' );

53 deger <<= 1;

54

55 if ( c % 8 == 0 )

56 putchar( ' ' );

57 }

58

59 putchar( '\n' );

60 }

Şekil 10.9 AND ,OR ,EXCLUSIVE OR ve tümleyen operatörlerinin kullanımı

Aşağıdakileri birleştirmenin sonucu

65535 = 00000000 00000000 11111111 11111111

1 = 00000000 00000000 00000000 00000000

AND operatorü & kullanıldığında

1 = 00000000 00000000 00000000 00000001

408

Aşağıdakileri birleştirmenin sonucu

15 = 00000000 00000000 00000000 00001111

241 = 00000000 00000000 00000000 11110001

OR operatorü | kullanıldığında

255 = 00000000 00000000 00000000 11111111

Aşağıdakileri birleştirmenin sonucu

139 = 00000000 00000000 00000000 10001011

199 = 00000000 00000000 00000000 11000111

exclusive OR operatorü ^ kullanıldığında

76 = 00000000 00000000 00000000 01001100

Aşağıdakinin bire tümleyeni

21845 = 00000000 00000000 01010101 01010101

şöyle gösterilir

43690 = 11111111 11111111 10101010 10101010

Şekil 10.10 Şekil 10.9’daki programın çıktısı

Şekil 10.9’da, tamsayı değişkeni maske 13.satırda 1 değerine (00000000 00000000 00000000

00000001) ve sayi1 değişkeni 65535 değerine (00000000 00000000 11111111 11111111)

atanmıştır. maske ve sayi1 AND operatörü (&) kullanılarak sayi1&maske deyimi ile

birleştirildiğinde sonuç 00000000 00000000 00000000 00000001 olur. sayi1 değişkeni

içindeki en düşük değerlikli bit hariç tüm bitler, sayi1 değişkeni maske değişkeni ile AND

operatörü sayesinde birleştirildiğinden maskelenir (gizlenir).

OR operatörü bir operandındaki belli bitleri 1 yapmak için kullanılır. Şekil 10.9’daki sayi1

değişkeni 20.satırda 15 değerine (00000000 00000000 00000000 00001111) ve bitBirle

değişkeni 21.satırda 241 değerine (00000000 00000000 00000000 11110001) atanmıştır.

sayi1 ve bitBirle değişkenleri OR operatörü kullanılarak sayi1 | bitBirle deyimi içinde

birleştirildiğinde sonuç 255 (00000000 00000000 00000000 11111111) olur. Şekil 10.11, OR

operatörü ile iki bitin birleştirilmesinden oluşan sonuçları göstermektedir.

Genel Programlama Hataları 10.13

OR operatörü (|) yerine mantıksal VEYA operatörünü (||) kullanmak ya da tam tersi

EXCLUSIVE OR operatörü (^) operandlarındaki ilgili bitlerden yalnızca biri 1 ise sonuçtaki

biti 1 yapar. Şekil 10.9’da, sayi1 ve sayi2 değişkenleri sırasıyla 139 (00000000 00000000

00000000 10001011) ve 199 (00000000 00000000 00000000 11000111) değerlerine

atanmıştır. Bu değişkenler EXCLUSIVE OR operatörü kullanılarak sayi1^sayi2 deyimi ile

birleştirildiğinde, sonuç 000000000 00000000 00000000 01001100 olmaktadır. Şekil 10.12,

iki bitin EXCLUSIVE OR operatörü ile birleştirilmesinden oluşan sonuçları özetlemektedir.

Tümleyen operatörü (~) ,operandında 1 olan tüm bitleri sonuçta 0, operandında 0 olan tüm

bitleri sonuçta 1 yapar. Bu yüzden de “değerin bire tümleyenini almak” da denir. Şekil

10.9’da, sayi1 değişkeni 36.satırda 21845 değerine (00000000 00000000 01010101

01010101) değerine atanmıştır. ~sayi1 deyimi hesaplandığında sonuç 00000000 00000000

10101010 10101010 olmaktadır.

409

Bit1 Bit2 Bit1|Bit2

0 0 0

1 0 1

0 1 1

1 1 1

Şekil 10.11 İki biti OR operatörü | ile birleştirmek.

Bit1 Bit2 Bit1^Bit2

0 0 0

1 0 1

0 1 1

1 1 0

Şekil 10.12 İki biti EXCLUSIVE OR operatörü ^ ile birleştirmek

Şekil 10.13’deki program, sola kaydırma(<<) ve sağa kaydırma(>>) operatörlerini

göstermektedir. bitleriGoster fonksiyonu, unsigned tamsayı değerlerini yazdırmak için

kullanılmıştır. Sola kaydırma operatörü (<<), soldaki operandındaki bitleri sağdaki

operandında belirtilen sayı kadar sola kaydırır. Sağdan boşaltılan bitler 0 ile doldurulurken,

sola kaydırılan 1’ler kaybolur. Şekil 10.13’deki programda, sayi1 değişkeni 9.satırda 960

değerine (000000000 000000000 00000011 11000000) atanmıştır. sayi1 değişkeninin

sayi1<<8 deyimi ile 8 bit sola kaydırılması 49152 (00000000 00000000 11000000

00000000) sonucunu vermektedir (satır 15).

1 /\* Şekil 10.13: fig10\_13.c

2 Kaydırma operatörlerinin kullanılması \*/

3 #include <stdio.h>

4

5 void bitleriGoster( unsigned );

6

7 int main( )

8 {

9 unsigned sayi1 = 960;

10

11 printf( "\n Sola kaydırma operatörünün << kullanılmasıyla \n" );

12 bitleriGoster( sayi1 );

13 printf( "sayısını 8 bit pozisyonu " );

14 printf( " sola kaydırmanın sonucu \n" );

15 bitleriGoster( sayi1 << 8 );

16

17 printf( "\n Sağa kaydırma operatörünün >> kullanılmasıyla \n" );

18 bitleriGoster( sayi1 );

19 printf( " sayısını 8 bit pozisyonu " );

20 printf( " sağa kaydırmanın sonucu: \n" );

21 bitleriGoster( sayi1 >> 8 );

22 return 0;

23 }

410

24

25 void bitleriGoster( unsigned deger )

26 {

27 unsigned c, maskeyiGoster = 1 << 31;

28

29 printf( "%7u = ", deger );

30

31 for ( c = 1; c <= 32; c++ ) {

32 putchar( deger & maskeyiGoster ? '1' : '0' );

33 deger <<= 1;

34

35 if ( c % 8 == 0 )

36 putchar( ' ' );

37 }

38

39 putchar( '\n' );

40 }

Sola kaydırma operatörünün << kullanılmasıyla

960 = 00000011 11000000

sayısını 8 bit pozisyonu

sola kaydırmanın sonucu:

245760 =00000000 00000011 11000000 00000000

Sağa kaydırma operatörünün >> kullanılmasıyla

960 = 00000011 11000000

sayısını 8 bit pozisyonu

sağa kaydırmanın sonucu:

3 = 00000000 00000000 00000000 00000011

Şekil 10.13 Kaydırma operatörlerini kullanmak

Sağa kaydırma operatörü (>>), soldaki operandındaki bitleri sağdaki operandında belirtilen

sayı kadar sağa kaydırır. unsigned bir tamsayı üzerinde sağa kaydırma yapmak soldan

boşaltılan bitlerin 0 ile doldurulmasını sağlar, sağdan kaydırılan 1’ler kaybolur. Şekil

10.13’deki programda, sayi1’i sayi1>>8 deyimi ile sağa kaydırmak 3 (00000000 00000011)

değerini verir (satır 21).

Genel Programlama Hataları 10.14

Bir değeri kaydırırken eğer sağdaki operand negatif ise ya da sağdaki operand soldaki

operandın bit sayısından büyükse, kaydırma tanımsızdır.

Taşınırlık İpuçları 10.7

Sağa kaydırma makine bağımlıdır. İşaretli bir tamsayıyı sağa kaydırma bazı makinelerde

boşaltılan bitleri 0 bazılarında ise 1 ile doldurur.

411

Her bit operatörünün (tümleyen operatörü hariç) bir atama operatörü bulunur. Bu bit atama

operatörleri şekil 10.14’de gösterilmiştir ve 3.ünitede tanıtılan aritmetik operatörlerine benzer

bir biçimde kullanılırlar.

Şekil 10.15 şu ana kadar gördüğümüz operatörlerin öncelik sıralarını ve işleyişlerini

göstermektedir.Operatörlerin önceliği yukarıdan aşağıya gidildikçe azalmaktadır.

Bit Atama Operatörleri

& = AND atama operatörü

| = OR atama operatörü

^ = EXCLUSIVE OR atama operatörü

<< = Sola kaydırma atama operatörü

>> = Sağa Kaydırma atama operatörü

Şekil 10.14 Bit atama operatörleri

Operatör İşleyiş sırası Tip

( ) [ ] . -> soldan sağa en yüksek

++ -- + - (tip) ! & \* ~ sizeof sağdan sola tekli

\* / % soldan sağa multiplicative

+ - soldan sağa additive

<< >> soldan sağa kaydırma

< <= > >= soldan sağa karşılaştırma

= = != soldan sağa eşitlik

& soldan sağa AND

^ soldan sağa EXCLUSIVE OR

| soldan sağa OR

&& soldan sağa mantıksal ve

|| soldan sağa mantıksal veya

?: sağdan sola koşullu

= += -= \*= /= &= |= ^= <<= >>= sağdan sola atama

Şekil 10.15 Operatör öncelikleri ve işleyişleri

10.10 BİT ALANLARI

C, bir birliğin ya da yapının unsigned ya da int elemanlarının kaç bit içinde ( bit alanı olarak

bilinir) depolanacağını belirlememize imkan tanır. Bit alanları, verileri gerekli en az sayıda bit

içinde tutarak daha iyi bir hafıza kullanımı sağlar. Bit alanları int ya da unsigned olarak

bildirilir.

Performans İpuçları 10.3

Bit alanları depolamada kazanç sağlar.

412

Aşağıdaki yapı tanımlamasını inceleyelim:

struct bitKart{

unsigned taraf :4;

unsigned takim:2;

unsigned renk :1;

};

Bu tanımlama 52 kartlık bir desteyi temsil etmek için, 3 unsigned bit alanı ( taraf,takim ve

renk) içerir. Bir bit alanı, unsigned ya da int bir eleman isminden sonra iki nokta üst üste( : )

ve alanın genişliğini belirten bir tamsayı sabiti ile (elemanın depolanacağı bit sayısını belirten

bir sabit ile) bildirilir. Genişliği belirten sabit, 0 ile sisteminizde int depolamak için kullanılan

toplam bit sayısı arasında bir tamsayı olmak zorundadır. Örneklerimiz 4-byte (32 bit) tamsayı

kullanan bir bilgisayarda denenmiştir.

Az önceki yapı tanımlaması taraf elemanının 4 bit içinde, takim elemanının 2 bit ve renk

elemanının 1 bit içinde depolandığını belirtmektedir. Bit sayısı her yapı elamanı için istenen

aralığa bağlıdır. taraf elemanı 0 (As) ile 12 (Papaz) arasında değerler depolamaktadır. 4 bit, 0

ile 15 arasındaki değerleri tutabilir. takim elemanı 0 ile 3 arasında değerler (0= karo, 1=kupa,

2=sinek , 3=maça) depolamaktadır. 2 bit, 0 ile 3 arasındaki değerleri depolayabilir. renk

elemanı 0 (kırmızı) ve 1(siyah) değerlerini tutmaktadır. 1 bit, 0 ya da 1 değerlerini

depolayabilir.

Şekil 10.16 (çıktısı şekil 10.17’de gösterilmiştir) 52 struct bitKart yapısından oluşan deste

dizisini yaratmaktadır(satır 19). desteDoldur fonksiyonu (27.satırda tanımlanmıştır) 52 kartı

deste dizisi içine yerleştirmekte ve dagıt fonksiyonu (41.satırda tanımlanmıştır) 52 kartı

yazdırmaktadır. Yapıların bit alanı elemanlarına, diğer yapı elemanlarına ulaşıldığı biçimde

ulaşıldığına dikkat ediniz. renk elemanı, renkli gösterim yapabilen sistemlerde kart rengini

belirtmek için kullanılmıştır.

1 /\* Şekil 10.16: fig10\_16.c

2 Bit alanları kullanma örneği \*/

3

4 #include <stdio.h>

5

6 struct bitKart {

7 unsigned taraf : 4;

8 unsigned takim : 2;

9 unsigned renk : 1;

10 };

11

12 typedef struct bitKart Kart;

13

14 void desteDoldur( Kart \* const );

15 void dagit( const Kart \* const );

16

17 int main( )

18 {

413

19 Kart deste[ 52 ];

20

21 desteDoldur( deste );

22 dagit( deste );

23

24 return 0;

25 }

26

27 void desteDoldur( Kart \* const wDeste )

28 {

29 int i;

30

31 for ( i = 0; i <= 51; i++ ) {

32 wDeste[ i ].taraf = i % 13;

33 wDeste[ i ].takim = i / 13;

34 wDeste[ i ].renk = i / 26;

35 }

36 }

37

38 /\* dagir fonksiyonu kartları iki sütun biçiminde yazar

39 Sütun 1 k1 belirteçli 0-25 kartlarını içerir

40 Sütun 2 k2 belirteçli 26-51 kartlarını içerir \*/

41 void dagit( const Kart \* const wDeste )

42 {

43 int k1, k2;

44

45 for ( k1 = 0, k2 = k1 + 26; k1 <= 25; k1++, k2++ ) {

46 printf( "Kart:%3d Takim:%2d Renk:%2d ",

47 wDeste[ k1 ].taraf, wDeste[ k1 ].takim,

48 wDeste[ k1 ].renk );

49 printf( "Kart:%3d Takim:%2d Renk:%2d\n",

50 wDeste[ k2 ].taraf,

51 wDeste[ k2 ].takim, wDeste[ k2 ].renk );

52 }

53 }

Şekil 10.16 Bir deste kartı depolamak için bit alanları kullanmak.

Kart: 0 Takım: 0 Renk: 0 Kart: 0 Takım: 2 Renk: 1

Kart: 1 Takım: 0 Renk: 0 Kart: 1 Takım: 2 Renk: 1

Kart: 2 Takım: 0 Renk: 0 Kart: 2 Takım: 2 Renk: 1

Kart: 3 Takım: 0 Renk: 0 Kart: 3 Takım: 2 Renk: 1

Kart: 4 Takım: 0 Renk: 0 Kart: 4 Takım: 2 Renk: 1

Kart: 5 Takım: 0 Renk: 0 Kart: 5 Takım: 2 Renk: 1

Kart: 6 Takım: 0 Renk: 0 Kart: 6 Takım: 2 Renk: 1

Kart: 7 Takım: 0 Renk: 0 Kart: 7 Takım: 2 Renk: 1

414

Kart: 8 Takım: 0 Renk: 0 Kart: 8 Takım: 2 Renk: 1

Kart: 9 Takım: 0 Renk: 0 Kart: 9 Takım: 2 Renk: 1

Kart: 10 Takım: 0 Renk: 0 Kart: 10 Takım: 2 Renk: 1

Kart: 11 Takım: 0 Renk: 0 Kart: 11 Takım: 2 Renk: 1

Kart: 12 Takım: 0 Renk: 0 Kart: 12 Takım: 2 Renk: 1

Kart: 0 Takım: 1 Renk: 0 Kart: 0 Takım: 3 Renk: 1

Kart: 1 Takım: 1 Renk: 0 Kart: 1 Takım: 3 Renk: 1

Kart: 2 Takım: 1 Renk: 0 Kart: 2 Takım: 3 Renk: 1

Kart: 3 Takım: 1 Renk: 0 Kart: 3 Takım: 3 Renk: 1

Kart: 4 Takım: 1 Renk: 0 Kart: 4 Takım: 3 Renk: 1

Kart: 5 Takım: 1 Renk: 0 Kart: 5 Takım: 3 Renk: 1

Kart: 6 Takım: 1 Renk: 0 Kart: 6 Takım: 3 Renk: 1

Kart: 7 Takım: 1 Renk: 0 Kart: 7 Takım: 3 Renk: 1

Kart: 8 Takım: 1 Renk: 0 Kart: 8 Takım: 3 Renk: 1

Kart: 9 Takım: 1 Renk: 0 Kart: 9 Takım: 3 Renk: 1

Kart: 10 Takım: 1 Renk: 0 Kart: 10 Takım: 3 Renk: 1

Kart: 11 Takım: 1 Renk: 0 Kart: 11 Takım: 3 Renk: 1

Kart: 12 Takım: 1 Renk: 0 Kart: 12 Takım: 3 Renk: 1

Şekil 10.17 Şekil 10.16’daki programın çıktısı.

Yapı içinde, boşluk bırakmak için kullanılan isimsiz bit alanları belirlemek mümkündür.

Örneğin,

struct ornek{

unsigned a :13;

unsigned :19;

unsigned b : 4;

};

yapı tanımı, 19 bitlik isimsiz bir alanı boşluk bırakmak için (padding) kullanmaktadır. Bu 19

bit içinde hiçbir şey depolanamaz. b elemanı (4-byte tamsayı kullanan bilgisayarımızda)

başka bir depolama alanında depolanır.

0 genişliğinde isimsiz bir bit alanı, bir sonraki bit alanını yeni bir depolama birimi sınırına

hizalamakta kullanılır.Örneğin,

struct ornek{

unsigned a:13;

unsigned : 0;

unsigned b: 4;

};

yapı tanımı, isimsiz 0 bitlik alanı, a’nın depolandığı depolama biriminde kalan bitleri (ne

kadar varsa) atlamak için ve b’yi yeni bir depolama birimine hizalamak için kullanır.

Taşınırlık İpuçları 10.8

Bit alanı işlemleri makine bağımlıdır. Örneğin bazı bilgisayarlar bit alanlarının sınırları

geçmesine izin verirken diğerleri vermeyebilir.

415

Genel Programlama Hataları 10.15

Bir bit alanının içindeki bitlere sanki bir dizi elemanına erişir gibi erişmeye çalışmak.Bit

alanları bitlerden oluşan bir dizi değildir.

Genel Programlama Hataları 10.16

Bir bit alanının adresini almaya çalışmak(& operatörüyle bit alanlarının adresleri alınamaz

çünkü bit alanlarının adresi yoktur.)

Performans İpuçları 10.4

Bit alanları depolama alanından kazanç sağlasa da derleyicinin makine kodlarını daha yavaş

üretmesine sebep olurlar.Bu, adreslenebilir bir depolama alanında erişilebilecek alanların

belirlenmesi için, fazladan makine dili işlemlerinin yapılması nedeniyle oluşur.Bu zamanmekan değişimlerinin bir çok örneğinden biridir.

10.11 SAYMA SABİTLERİ

C, kullanıcı tarafından tanımlanabilen son veri tipi olan sayma tipini sunar. Bir sayma, enum

anahtar kelimesiyle tanıtılır ve tanıtıcılar ile temsil edilen tamsayı sabitlerinin kümesidir. Bu

sayma sabitleri, değerleri otomatik olarak belirlenen sembolik sabitlerdir. enum içindeki

değerler aksi belirtilmedikçe 0 ile başlar ve 1 arttırılır. Örneğin,

enum aylar{OCA, SUB, MAR, NIS, MAY, HAZ, TEM, AGU, EYL, EKI, KAS, ARA};

yeni bir tip olan enum aylar tipini yaratır. Bu sayma, tanıtıcıları 0 ile 11 arasında tamsayılar

yapar. Ayları 1’den 12’ye kadar saydırmak için

enum aylar{OCA = 1, SUB, MAR, NIS, MAY, HAZ, TEM, AGU,EYL,

EKI, KAS, ARA};

kullanılır. Burada ilk değer özel olarak 1 yapıldığından, kalan değerler 1 arttırılarak 1-12

değerleri oluşturulur. Saymada kullanılan tanıtıcılar özel olmalıdır. Her sayma sabitinin

değeri, atama değerine bir değer atanarak istenen özel bir değer yapılabilir. Saymada, birden

çok eleman aynı sabit değere sahip olabilir. Şekil 10.18’daki programda, sayma değişkeni ay,

for yapısı içinde kullanılarak, ayIsmi dizisinden yılın aylarını yazdırmak için kullanılmıştır.

ayIsmi[0] ’ı boş bir string “ ” yaptığımıza dikkat ediniz. Bazı programcılar, ayIsmi[0] ’ı

\*\*\*HATA\*\*\* gibi bir değere atayarak bir mantık hatası oluştuğunu göstermeyi tercih

edebilirler.

Genel Programlama Hataları 10.17

Tanımlandıktan sonra bir sayma sabitine değer atamak yazım hatasıdır.

İyi Programlama Alıştırmaları 10.6

Sayma sabitleri için yalnızca büyük harfler kullanın. Bu, sabitlerin programda daha belirgin

hale gelmesini sağlatır ve programcıya sayma sabitlerinin değişken olmadığını hatırlatır.

1 /\* Şekil 10.18: fig10\_18.c

2 Sayma tipi kullanma \*/

416

3 #include <stdio.h>

4

5 enum aylar { OCA = 1, SUB, MAR, NIS, MAY, HAZ,

6 TEM, AGU, EYL, EKI, KAS, ARA };

7

8 int main( )

9 {

10 enum aylar ay;

11 const char \*ayIsmi[] = { "", "Ocak", "Subat", "Mart",

12 "Nisan", "Mayıs", "Haziran", "Temmuz",

13 "Ağustos", "Eylül", "Ekim",

14 "Kasım", "Aralık" };

15

16 for ( ay = OCA; ay <=ARA; ay++ )

17 printf( "%2d%11s\n", ay, ayIsmi[ ay ] );

18

19 return 0;

20 }

1 Ocak

2 Subat

3 Mart

4 Nisan

5 Mayıs

6 Haziran

7 Temmuz

8 Ağustos

9 Eylül

10 Ekim

11 Kasım

12 Aralık

Şekil 10.18 Sayma kullanmak

ÖZET

 Yapılar, birbirleriyle ilişkili değişkenlerin bir isim altında toplanmasıdır

 Yapılar değişik veri tiplerinde değişkenler içerebilir

 struct anahtar kelimesi yapı tanımını başlatır. Yapı tanımında parantezler içinde

bildirilen değişkenler yapı elemanlarıdır.

 Aynı yapının elemanları, kendilerine özel isimlere sahip olmalıdır.

 Bir yapı tanımlaması, değişkenler bildirmek için kullanılacak yeni bir veri tipi

oluşturur.

 Yapı değişkenleri bildirmenin iki yöntemi vardır. İlk yöntem, diğer veri tiplerindeki

değişkenlerin bildiriminde yapıldığı gibi değişkenleri struct etiket\_ismi tipini

417

kullanarak bildirmek. İkinci yöntem, değişkenleri yapı tanımının en son parantezi ile

yapı tanımını sonlandıran noktalı virgül arasına yerleştirmektir.

 Bir yapıda etiket ismi kullanmak tercihe bağlıdır. Eğer yapı etiket ismi kullanılmadan

tanımlanırsa, türetilmiş veri tipindeki değişkenler yapı tanımı içinde bildirilmelidir ve

yeni yapı tipinde başka değişkenler bildirilemez.

 Yapıya değer atamak için, yapı değişkeninin adından sonra eşittir işareti ve küme

parantezleri içinde virgüllerle ayrılmış atama değerleri kullanılır.Eğer atama listesinde

yapı elemanlarından daha az sayıda atama değeri varsa, kalan elemanlar otomatik

olarak 0’a (ya da eleman gösterici ise NULL’a) atanır.

 Yapıların tümü aynı tipteki yapı değişkenlerine atanabilirler.

 Bir yapı değişkeni, aynı tipteki bir yapı değişkenine atandığı atama ifadelerinde ilk

değerlere atanabilir.

 Yapı elemanı operatörü, yapı elemanına yapı değişkeninin ismini kullanarak erişir.

 Yapı gösterici operatörü (eksi işareti(-) ve büyüktür işareti( >) arasında boşluk

bırakmadan yazılır) yapı elemanına, yapıyı gösteren bir gösterici ile ulaşır

 Yapılar ya da yapı elemanları fonksiyonlara geçirilirken, değere göre çağırma ile

geçirilirler.

 Bir yapıyı referansa göre çağırmak için yapı değişkeninin adresi geçirilir

 Yapı dizileri, diğer tüm diziler gibi, otomatik olarak referansa göre geçirilir.

 typedef ile yeni bir isim yaratmak yeni bir tip yaratmaz; typedef daha önceden var

olan bir tip ismi için, eş anlamlı olarak kullanılabilecek yeni tip isimleri yaratır

 Birlikler (yapılar gibi), türetilmiş veri tipleridir. Birlik elemanları aynı depolama

alanını kullanırlar.Birlik elemanları herhangi bir tipte olabilirler.

 Bir birliği depolayabilmek için kullanılan byte sayısı en az birliğin en büyük elemanını

tutabilecek kadar olmalıdır.Çoğu durumda birlikler iki ya da daha fazla veri tipi

içerirler. Bir anda yalnızca bir eleman,bu sebepten de yalnızca bir veri tipi

kullanılabilir

 Bir birlik union anahtar kelimesiyle, yapılarla aynı biçimde bildirilir.

 Bir birliğe, yalnızca ilk birlik elemanının tipiyle aynı olan değerler atanabilir.

 AND operatörü (bit düzeyinde VE operatörü de denir), iki operand kullanır ve iki

operandında da ilgili bitte 1 varsa sonuçtaki biti 1 yapar.

 Maskeler bir değerdeki bazı bitleri seçerken bazı bitleri de saklamak için kullanılır

 OR operatörü (bit düzeyinde VEYA operatörü de denir), iki operand kullanır ve

operandlarındaki bitlerden birinde ya da ikisinde birden 1 varsa sonuçtaki biti 1 yapar.

 Her bit operatörünün (tümleyen operatörü hariç) bir atama operatörü bulunur

 EXCLUSIVE OR (bit düzeyinde özel VEYA operatörü de denir) operatörü,

operandlarındaki bitlerden yalnızca biri 1 ise sonuçtaki biti 1 yapar.

 Sola kaydırma operatörü, soldaki operandındaki bitleri sağdaki operandında belirtilen

sayı kadar sola kaydırır. Sağdan boşaltılan bitler 0 ile doldurulur.

 Sağa kaydırma operatörü ,soldaki operandındaki bitleri sağdaki operandında belirtilen

sayı kadar sağa kaydırır. İşaretsiz bir tamsayı üzerinde sağa kaydırma yapmak soldan

boşaltılan bitlerin 0 ile doldurulmasını sağlatır.İşaretli tamsayılarda soldan boşaltılan

bitler 0 ya da 1 ile doldurulabilir.Bu, her makinede farklılık gösterebilir.

 Tümleyen operatörü, operandındaki 0 olan tüm bitleri 1 ve operandındaki 1 olan tüm

bitleri 0 yapar.

 Bit alanları, verileri gerekli en az sayıda bit içinde tutarak daha iyi bir hafıza kullanımı

sağlar.

 Bit alanları int ya da unsigned olarak bildirilir.

418

 Bir bit alanı, unsigned ya da int bir eleman isminden sonra iki nokta üst üste( : ) ve

alanın genişliğini belirten bir tamsayı sabiti ile(elemanın depolanacağı bit sayısını

belirten bir sabit ile) bildirilir.

 Genişliği belirten sabit, 0 ile sisteminizde int depolamak için kullanılan toplam bit

sayısı arasında bir tamsayı olmak zorundadır.

 Yapı içinde, boşluk bırakmak için kullanılan isimsiz bit alanları belirlemek

mümkündür.

 0 genişliğinde isimsiz bir bit alanı, bir sonraki bit alanını yeni bir depolama birimi

sınırına hizalamakta kullanılır

 Bir sayma, enum anahtar kelimesiyle tanıtılır ve tanıtıcılarla temsil edilen tamsayı

kümesidir. enum içindeki değerler aksi belirtilmedikçe 0 ile başlar ve 1 arttırılır.

ÇEVRİLEN TERİMLER

bit field..................... bit alanı

bitwise operators...... bit operatörleri

complementing........ tümleyenini almak

derived type............. türetilmiş tip

enumaration............ sayma sabiti

mask.............. ......... maske

structure tag............ yapı etiketi

tag name ................. etiket ismi

GENEL PROGRAMLAMA HATALARI

10.1 Yapı tanımını sonlandıran noktalı virgülü unutmak.

10.2 Bir tipte yapıyı başka bir tipteki yapıya atamak

10.3 Yapıları karşılaştırmak bir yazım hatasıdır

10.4 Yapı gösterici operatörünü yazarken – ve > arasına boşluk koymak.(ya da ?: operatörü haricinde

birden çok karakter kullanılarak yazılan operatörler arasına boşluk koymak)

10.5 Yapı elemanının ismini tek başına kullanarak yapı elemanına ulaşmaya çalışmak

10.6 Bir yapı elemanını, gösterici ve yapı elemanı operatörü kullanarak belirtirken parantez

kullanmamak.(örneğin \*aPtr.takim bir yazım hatasıdır.)

10.7 Yapıların, diziler gibi otomatik olarak referansa göre çağırma ile geçirildiklerini

düşünmek ve çağırıcının yapısını çağırılan fonksiyon içinde değiştirmeye çalışmak.

10.8 Yapılardan oluşan dizilerde, bağımsız yapıları belirtmek için kullanılan dizi belirteçlerini

unutmak

10.9 Birlik içinde depolanmış verinin tipini farklı bir tip ile kullanmak bir mantık hatasıdır.

10.10 Birlikleri karşılaştırmak yazım hatasıdır.

10.11 Bir birlik bildiriminde, birliğin ilk elemanının tipinden farklı tipte bir değer ile atama

yapmak

10.12 AND operatörü(&) yerine mantıksal VE operatörünü(&&) kullanmak ya da tam tersi.

10.13 OR operatörü (|) yerine mantıksal VEYA operatörünü (||) kullanmak ya da tam tersi

10.14 Bir değeri kaydırırken eğer sağdaki operand negatif ise ya da sağdaki operand soldaki

operandın bit sayısından büyükse, kaydırma tanımsızdır.

10.15 Bir bit alanının içindeki bitlere sanki bir dizi elemanına erişir gibi erişmeye çalışmak.

Bit alanları bitlerden oluşan bir dizi değildir.

10.16 Bir bit alanının adresini almaya çalışmak(& operatörüyle bit alanlarının adresleri

alınamaz çünkü bit alanlarının adresi yoktur.)

10.17 Tanımlandıktan sonra bir sayma sabitine değer atamak yazım hatasıdır.

419

İYİ PROGRAMLAMA ALIŞTIRMALARI

10.1 Yeni bir yapı tipi oluştururken her zaman yapı etiket ismi kullanın.Yapı etiket ismi, programda

daha sonradan o yapı tipinde yeni değişkenler bildirmek için gereklidir.

10.2 Anlamlı bir yapı etiket ismi kullanmak programın daha anlaşılır olmasını sağlar.

10.3 Farklı tipteki yapıların elemanları için aynı isimleri kullanmaktan kaçının.Buna izin verilmiştir

ancak karışıklık yaratabilir.

10.4 -> ve . operatörlerinden önce ve sonra boşluk bırakmayınız.Bu sayede, bu operatörlerin

kullanıldığı deyimlerin aslında tek bir değişken ismi olduğu vurgulanmış olur.

10.5 typedef isimlerinin ilk harflerini büyük harf ile yazarak, bu isimlerin başka tip isimleri

için eş anlamlı isimler olduğunu vurgulamak.

10.6 Sayma sabitleri için yalnızca büyük harfler kullanın.Bu, sabitlerin programda daha

belirgin hale gelmesini sağlatır ve programcıya sayma sabitlerinin değişken olmadığını

hatırlatır.

TAŞINIRLIK İPUÇLARI

10.1 Belli bir tipte veri parçalarının boyutu, makine bağımlı olduğu için ve depolama

hizalama hususları da makine bağımlı olduğundan, yapıların gösterilmesi de makine

bağımlıdır.

10.2 typedef kullanmak programı daha taşınır bir hale getirir.

10.3 Eğer birlikte depolanan veri yanlış bir tiple kullanılırsa, sonuçlar uygulamaya bağlı

olarak farklılık gösterebilir.

10.4 Bir birliği depolamak için gerekli olan alan, uygulamadan uygulamaya farklılık

gösterebilir.

10.5 Bazı birlikler başka bilgisayar sistemlerine kolaylıkla taşınamaz.Bir birliğin

taşınılabilirliği, genellikle sistemde birliğin eleman tiplerinin depolanmasında kullanılan

hizalama yöntemlerine dayanır.

10.6 Verilerin bitleriyle yapılan işlemler makinelere bağımlıdır.

10.7 Sağa kaydırma makine bağımlıdır.İşaretli bir tamsayıyı sağa kaydırma bazı makinelerde

boşaltılan bitleri 0 bazılarında ise 1 ile doldurur.

10.8 Bit alanı işlemleri makine bağımlıdır.Örneğin bazı bilgisayarlar bit alanlarının sınırları

geçmesine izin verirken diğerleri vermeyebilir.

PERFORMANS İPUÇLARI

10.1 Yapıları referansa göre çağırma ile geçirmek, yapıları değere göre çağırma ile

geçirmekten (tüm yapının kopyasının oluşturulması gerekir) daha verimlidir.

10.2 Birlikler depolama alanında kazanç sağlar.

10.3 Bit alanları depolamada kazanç sağlar.

10.4 Bit alanları depolama alanından kazanç sağlasa da derleyicinin makine kodlarını daha

yavaş üretmesine sebep olurlar.Bu, adreslenebilir bir depolama alanında erişilebilecek

alanların belirlenmesi için, fazladan makine dili işlemlerinin yapılması nedeniyle

oluşur.Bu mekan-zaman değişimlerinin bir çok örneğinden biridir

YAZILIM MÜHENDİSLİĞİ GÖZLEMLERİ

10.1 struct bildiriminde olduğu gibi bir union bildirimi de yeni bir tip yaratır.union ya

420

da struct bildirimini fonksiyonların dışında yapmak global değişkenler yaratmaz.

ÇÖZÜMLÜ ALIŞTIRMALAR

10.1 Aşağıdaki boşlukları doldurunuz.

a) \_\_\_\_\_\_, birbiriyle ilgili değişkenlerin tek isim altında toplanmasıdır.

b) \_\_\_\_\_\_ aynı depolamayı kullanan değişkenlerin tek isim altında toplanmasıdır.

c) \_\_\_\_\_\_ ifadesi kullanıldığında her iki operandındaki ilgili bitleri 1 ise sonuç 1 olur.

Aksi takdirde, bitler 0 yapılır.

d) Yapı tanımlamalarında bildirilen değişkenlere yapının \_\_\_\_\_\_ denir.

e) \_\_\_\_\_\_ ifadesi kullanıldığında, her iki operandında ilgili bitlerinin en az biri 1 ise

sonuç 1 olur. Aksi takdirde, bitler 0 yapılır.

f) \_\_\_\_\_\_ anahtar kelimesi, yapı bildiriminde kullanılır.

g) \_\_\_\_\_\_ anahtar kelimesi, daha önceden tanımlanmış bir yapının eşitini oluşturmada

kullanılır.

h) \_\_\_\_\_\_ ifadesi kullanıldığında her iki operandında ilgili bir biti 1 ise sonuç 1 olur.

Aksi takdirde bitler 0 yapılır.

i) AND bit operatörü &, bit stringinden istenen bitleri, diğer bitleri sıfırlama yoluyla

seçmede kullanılır. Buna \_\_\_\_\_\_ denir.

j) \_\_\_\_\_\_ anahtar kelimesi, birlik tanıtmak için kullanılır.

k) Yapının ismine, genellikle yapı \_\_\_\_\_\_ denir.

l) Bir yapı üyesine, \_\_\_\_\_\_ operatörü yada \_\_\_\_\_\_ operatörü ile erişilir.

m) \_\_\_\_\_\_ ve \_\_\_\_\_\_ operatörleri, sırasıyla bitleri sola ya da sağa kaydırmada

kullanılır.

n) \_\_\_\_\_\_ , sabitlerle ifade edilen tamsayılar kümesidir.

10.2 Aşağıdakilerden hangilerinin doğru yada hangilerinin yanlış olduğuna karar veriniz.

Yanlış olanların neden yanlış olduğunu açıklayınız.

a) Yapılar sadece bir veri tipi içerebilirler.

b) Birlikler, eşit olup olmadıklarının anlaşılabilmesi için karşılaştırılabilirler.

c) Bir yapıda yapı etiketi kullanımı tercihe bağlıdır.

d) Farklı yapıların üyeleri farklı isimlere sahip olmak zorundadır.

e) typedef anahtar kelimesi, yeni veri tiplerinin tanımlanmasında kullanılır.

f) Yapılar, fonksiyonlara her zaman referansa göre çağırma ile geçirilirler.

g) Yapılar, karşılaştırılamazlar.

10.3 Aşağıdaki ifadeleri gerçekleştirecek ifade ya da ifadeleri yazınız.

a) parca isminde bir yapı tanımlayınız. Yapı, parcaNumarasi isimli int tipinde bir

üyeyi ve değerlerinin uzunluğu 25 karaktere kadar olabilen parcaAdi isimli char

tipinde bir diziyi içersin.

b) struct parca tipi için eş anlamlı olarak kullanılabilecek Parca‘yı tanımlayınız.

421

c) Parca yapısını kullanarak, a’yı struct parca tipinin değişkeni, b[10] ‘u struct

parca tipinin dizisi ve ptr değişkenini de struct parca tipine gösterici olarak

bildiriniz.

d) a değişkeninin üyeleri için, klavyeden parça numarasını ve parça ismini aldırınız.

e) a değişkeninin üyelerini b dizisinin üçüncü elemanına atayınız.

f) b dizisinin adresini ptr göstericisine atayınız.

g) b dizisinin üçüncü elemanını, ptr değişkenini ve yapı gösterici operatörlerini

kullanarak ekrana yazdırınız.

10.4 Aşağıdakilerde hataları bulunuz.

a) char tipinde, taraf, takim isminde iki gösterici içeren struct kart ifadesinin daha

önceden tanımlandığını kabul ediniz. c değişkeninin struct kart tipinde bir

değişken olacak biçimde bildirildiğini ve cPtr değişkeninin struct kart tipindeki

bir yapıyı gösterecek tipte bir değişken olarak bildirildiğini varsayınız.. cPtr

değişkenine c değişkenin adresi atanmıştır.

printf(“%s\n”, \*cPtr->taraf);

b) char tipinde, taraf, takim isminde iki gösterici içeren struct kart ifadesinin daha

önceden tanımlandığını kabul ediniz. Aynı zamanda kupa[13] dizisinin de struct

kart yapısında bildirildiğini kabul ediniz. Aşağıdaki ifade, dizinin 10. elemanının

taraf üyesini ekrana yazdırır.

printf(“%s\n”, kupa.taraf);

c) union degerler {

char w;

float x;

double y;

} v = {1.27};

d) struct adam {

char soyad[15];

char isim[15];

int yas;

}

e) struct adam yapısının d şıkkında olduğu gibi, ancak uygun düzeltmeler yapılarak

tanımlandığını kabul ediniz.

adam d;

f) p değişkeninin struct adam tipinde tanımlandığını ve c değişkeninin struct kart

tipinde bildirildiğini kabul ediniz.

p = c;

422

ÇÖZÜMLER

10.1 a) yapı, b) birlik c) AND (&) bit operatörü d) üyeler e) OR bit operatörü f) struct

g) typedef h) OR (^) bit operatörü i) maske j) union k) yapı etiketi l) yapı üyesi, yapı

göstericisi m) sola kaydırma operatörü (<<), sağa kaydırma operatörü (>>) n) sayma

sabitleri

10.2

a) Yanlış. Bir yapı farklı veri tipleri içerebilir.

b) Yanlış. Birlikler karşılaştırılamazlar. Sebebi ise, yapılarda da karşılaşılan

hizalama problemleridir.

c) Doğru

d) Yanlış. Farklı yapıların üyeleri aynı isime sahip olabilirler ama aynı yapının

üyeleri aynı isime sahip olamazlar.

e) Yanlış. typedef anahtar kelimesi, daha önce tanımlanmış veri tiplerine yeni(eş)

isimler tanımlamada kullanılır.

f) Yanlış. Yapılar, fonksiyonlara her zaman değere göre çağırılarak geçerler.

g) Doğru. Hizalama problemleri yüzünden.

10.3

a) struct parca {

int parcaNumarasi;

char parcaAdi[25];

}

b) typedef struct parca Parca;

c) Parca a, b[10], \*ptr;

d) scanf(“%d%s”, &a.parcaNumarasi, &a.parcaAdi);

e) b[3] = a;

f) ptr = b;

g) printf (“%d %s\n”, (ptr + 3) -> parcaAdi, (ptr + 3) -> parcaAdi);

10.4

a) Hata: \*cptr değişkeninde kullanılan parantezler, ifadenin yanlış bir sırada

çalışmasına yol açar.

b) Hata: Dizi belirteci ihmal edilmiştir. ifade kupa[10].taraf şeklinde olmalıdır.

c) Hata: Bir birliğe, sadece ilk üyesiyle aynı tipte olan bir değerle ilk atama

yapılabilir.

d) Hata: Yapı ifadesinin sonuna noktalı virgül konulmalıdır.

e) Hata: Değişken bildiriminde struct anahtar kelimesi kullanılmamıştır.

f) Hata: Farklı yapı tipleri birbirlerine atanamaz.

ALIŞTIRMALAR

10.5 Aşağıdaki yapıları ve birlikleri tanımlayınız.

a) envanter yapısı, parcaAdi[30] karakter dizisi, parcaNumarasi tamsayı

değişkeni, ucret ondalıklı sayı değişkeni, stok tamsayı değişkeni ve sırala

tamsayı değişkenini içermektedir.

b) data birliği, char c, short s, long l, float f, ve double d değişkenlerini

içermektedir.

423

c) adres ismindeki yapı, sokakAdresi[25], sehir[20], ulke[3] ve postaKodu[6]

karakter dizilerini içermektedir.

d) ogrenci yapısı, ad[15], soyad[15] dizilerini ve c şıkkındaki adres yapısı

tipindeki evAdresi değişkenini içermektedir.

e) test yapısı, 16 bitlik, 1 bit genişliğinde bir alanı içermektedir. bit alanları a

harfinden p harfine kadardır.

10.6 Aşağıdaki, yapı tanımlamaları ve değişken bildirimleri verildiğine göre, aşağıdaki yapı

üyelerine erişimi sağlayan ifadeleri yazınız.

struct musteri {

char soyad[15];

char ad[15];

int musteriNumarasi;

struct {

char telNumarasi[11];

char adres[50];

char sehir[15];

char bolge[3];

char postaKodu[6];

} kisisel;

} musteriKaydi,, \*musteriPtr;

musteriPtr = &musteriKaydi;

a) musteriKaydi yapısının soyad üyesi

b) musteriPtr ile gösterilen yapının soyad üyesi

c) musteriKaydi yapısının ad üyesi

d) musteriPtr ile gösterilen yapının ad üyesi

e) musteriKaydi yapısının musteriNumarasi üyesi

f) musteriPtr ile gösterilen musteriNumarasi üyesi

g) musteriKaydi yapısının kisisel üyesinin telNumarasi üyesi

h) musteriPtr ile gösterilen kisisel üyesinin telNumarasi üyesi

i) musteriKaydi yapısının kisisel üyesinin adres üyesi

j) musteriPtr ile gösterilen yapının kisisel üyesinin adres üyesi

k) musteriKaydi yapısının kisisel üyesinin sehir üyesi

l) musteriPtr ile gösterilen yapının kisisel üyesinin sehir üyesi

m) musteriKaydi yapısının kisisel üyesinin bolge üyesi

n) musteriPtr ile gösterilen yapının kisisel üyesinin bolge üyesi

o) musteriKaydi yapısının kisisel üyesinin postaKodu üyesi

p) musteriPtr ile gösterilen yapının kisisel üyesinin postaKodu üyesi

10.7 Şekil 10.16’daki programı, kartları yüksek performanslı kart karma algoritmasını

kullanacak biçimde değiştiriniz (Şekil 10.3’ de gösterildiği gibi). Karma işleminden sonra

424

desteyi iki sütun halinde Şekil 10.4 de olduğu gibi yazdırınız. Önce kartların rengini

yazdırınız.

10.8 char c, short s, int i ve long l üyelerini içerecen tamsayi birliğini yaratınız. Klavyeden

char, short, int ve long tipinde değerler alan ve bu değerleri union tamsayi birliğinin

değişkenlerinde saklayan bir program yazınız. Her birlik elemanı, bir char, bir short, bir int

ve bir long şeklinde ekrana yazdırılmalıdır. Bu değerler her zaman doğru olarak yazdırıldı

mı?

10.9 float f, double d ve long double l üyelerini içeren ondalikliSayi isminde bir birlik

oluşturunuz. Klavyeden float, double ve long double tipinde değerler alarak bunları union

ondalikliSayi birliğinde saklayan bir program yazınız. Her birlik elemanı, bir float, bir

double ve bir long double şeklinde ekrana yazdırılmalıdır. Bu değerler her zaman doğru

olarak yazdırıldı mı?

10.10 Bir tamsayı değişkenini sağa 4 bit kaydıran bir program yazınız. kaydırma

operasyonundan önce ve sonra bu bitleri ekrana yazdırınız. Sisteminiz, boş kalan bitler için 0

veya 1 koyuyor mu?

10.11 Eğer bilgisayarınız, 4-byte tamsayılar kullanıyorsa Şekil 10.7’ deki programı 4 byte

sayılarla çalışacak şekilde değiştiriniz.

10.12 unsigned tipindeki bir tamsayıyı sola 1 bit kaydırmak demek 2 ile çarpmak demektir.

kuvvet2 adında bir fonksiyon yazınız. Fonksiyonunuz, sayi ve kuvvet adında iki tamsayı

argümanı alsın ve aşağıdaki ifadeyi hesaplasın:

sayi \* 2kuvvet

Sonucu hesaplamada kaydırma operatörünü kullanın. Programınız, sonucu tamsayılar ve bitler

biçiminde ekrana yazdırsın.

10.13 Sola kaydırma operatörü, iki karakter değerinin, bir unsigned tamsayı değişkeni içine

yerleştirilmesinde kullanılabilir. Klavyeden, iki karakter alan ve bunları karakterPaketle

fonksiyonuna gönderen bir program yazınız. Bu iki karakteri, unsigned integer tipine

çevirmek için, ilk karakteri unsigned integer değişkenine atayınız ve bu değişkeni 8 bit sola

kaydırınız. Daha sonra unsigned tipindeki bu değişkeni, ikinci karakter ile OR operatörünü

kullanarak birleştiriniz. Programınız, karakterlerin unsigned tipine doğru paketlendiğini

göstermek için çıktıyı, karakterleri paketlenmeden önce ve paketlendikten sonra bitler

biçiminde ekrana yazdırmalıdır.

10.14 Sağa kaydırma operatörü, AND operatörü ve bir maske kullanarak KarakterPaketAc

isminde bir fonksiyon yazınız. Fonksiyonunuz, Alıştırma 10.13’deki unsigned integer

tipindeki değeri alarak iki karakter haline getirmelidir. İki karakterin oluşturulması için

unsigned integer tipindeki değişkeni, 65280(00000000 00000000 11111111 00000000) ile

maskeleyiniz ve sonucu sağa 8 bit kaydırınız. Sonucu char tipinde bir değişkene atayınız.

Daha sonra unsigned integer tipindeki değişkeni 255 (00000000 00000000 00000000

11111111) ile maskeleyiniz ve sonucu char tipindeki diğer bir değişkene atayınız. Paketin

açılmasının doğru olarak yapıldığını göstermek için programınız unsigned integer tipindeki

değeri işlemlerden önce ve sonra bitler biçiminde ekrana yazdırılmalıdır.

425

10.15 Eğer bilgisayarınız 4-byte tamsayılar kullanıyorsa, Şekil 10.13’ deki programı 4

karakteri paket yapacak şekilde değiştiriniz.

10.16 Eğer bilgisayarınız, 4-byte tamsayılar kullanıyorsa Alıştırma 10.14’deki

KarakterPaketAc fonksiyonunu 4 karakter oluşturacak şekilde değiştiriniz. Karakterleri

oluşturmak için kullanacağınız maskeleri, 255 değerini 8 bit sola 0, 1, 2 ve 3 kez kaydırarak

(oluşturacağınız byte’a göre) oluşturunuz.

10.17 unsigned integer tipindeki bir değerin bitlerini tersten yazan bir program yazınız.

Programınız kullanıcından bir değer almalı ve bu değeri bitleriCevir fonksiyonuna

göndererek tersten yazdırmalıdır. Programınızın bitleri doğru olarak tersten yazdırdığını

görebilmek için bu değeri işlemler yapılmadan önce ve yapıldıktan sonra ekrana yazdırınız.

10.18 Şekil 10.7’deki bitleriGoster fonksiyonunu 2-byte tamsayılar ve 4-byte tamsayılar

kullanan sistemlerle uyumlu olacak şekilde değiştiriniz. İpucu:sizeof operatörünü kullanarak

makinede kullanılan tamsayı boyutunu bulabilirsiniz.

10.19 Aşağıdaki program kat isminde bir fonksiyon kullanarak, klavyeden girilen tamsayının

herhangi bir X tamsayısının tam katı olup olmadığını karar vermektedir. kat fonksiyonunu

inceleyin ve X tamsayısının değerini bulunuz.

1 /\* ex10\_19.c \*/

2 #include <stdio.h>

3

4 int kat ( int );

5

6 int main( )

7 {

8 int y;

9

10 printf( " 1 ile 32000 arasında bir tamsayı giriniz: " );

11 scanf( "%d", &y );

12

13 if ( kat( y ) )

14 printf( "%d X’in katı\n", y );

15 else

16 printf( "%d X’in katı değil\n", y );

17

18 return 0;

19 }

20

21 int kat( int sayi )

22 {

23 int i, maske = 1, carp = 1;

24

25 for ( i = 1; i <= 10; i++, maske <<= 1 )

26 if ( ( sayi & maske ) != 0 ) {

27 carp = 0;

28 break;

29 }

426

30

31 return carp;

32 }

10.20 Aşağıdaki program ne yapar?

1 /\* ex10\_20.c \*/

2 #include <stdio.h>

3

4 int gizem( unsigned );

5

6 int main( )

7 {

8 unsigned x;

9

10 printf( "Bir tamsayı giriniz: " );

11 scanf( "%u", &x );

12 printf( "Sonuç %d\n", gizem( x ) );

13 return 0;

14 }

15

16 int gizem( unsigned bitler )

17 {

18 unsigned i, maske = 1 << 31, toplam = 0;

19

20 for ( i = 1; i <= 32; i++, bitler <<= 1 )

21 if ( ( bitler & maske ) == maske )

22 ++toplam;

23

24 return !( toplam % 2 ) ? 1 : 0;

25 }

427

DOSYA İŞLEME

AMAÇLAR

 Dosyalar yaratabilmek, dosyalara yazma ve dosyadan okuma yapabilmek, dosyaları

güncelleyebilmek.

 Sıralı erişimle dosya işlemeyi tanımak

 Rasgele erişimle dosya işlemeyi tanımak

BAŞLIKLAR

11.1 GİRİŞ

11.2 VERİ HİYERARŞİSİ

11.3 DOSYA VE AKIŞLAR (STREAM)

11.4 SIRALI ERİŞİMLİ DOSYA YARATMAK

11.5 SIRALI ERİŞİMLİ DOSYADAN VERİ OKUMAK

11.6 RASGELE ERİŞİMLİ DOSYALAR

11.7 RASGELE ERİŞİMLİ DOSYA YARATMAK

11.8 RASGELE ERİŞİMLİ DOSYAYA RASGELE VERİ YAZMAK

11.9 RASGELE ERİŞİMLİ DOSYADAN RASGELE VERİ OKUMAK

11.10 ÖRNEK: EVRAK İŞLEME SİSTEMİ

11.1 GİRİŞ

Değişkenler ve diziler içinde depolanan veriler geçicidir ; bu türde veriler program

sonlandığında kaybolurlar. Dosyalar, büyük miktarda veriyi kalıcı olarak tutmak için

kullanılır. Bilgisayarlar dosyaları ikincil depolama cihazlarında, özellikle de disk depolama

cihazlarında tutarlar. Bu ünitede, veri dosyalarının nasıl yaratıldığını, güncellendiğini ve C

programları ile nasıl işlendiğini açıklayacağız. Sıralı erişimli dosyalar ve rasgele erişimli

dosyaların üzerinde duracağız.

11.2 VERİ HİYERARŞİSİ

Sonuçta, bilgisayar tarafından işlenen tüm veriler, sıfır ve birlerin kombinasyonlarına

indirgenirler. Bunun sebebi, iki kararlı durum içeren elektronik cihazları üretmenin basit ve

ekonomik olmasıdır. İki kararlı durumdan biri 0’ı, diğeri ise 1’i temsil eder. Bilgisayarlar

tarafından gerçekleştirilen etkileyici fonksiyonların yalnızca 1 ve 0’ların temel işlemlerini

içermesi dikkate değer bir noktadır.

Bir bilgisayardaki en küçük veri parçası 0 ya da 1 değerini alabilir. Böyle veri parçalarına bit

( ikili basamak anlamına gelen binary digit teriminin kısaltmasıdır, basamak iki değerden

birini alabilir) denir. Bilgisayar devreleri, bir bitin değerini anlamak, bite değer yerleştirmek

ve bit değerlerini tersine çevirmek (0 ise 1’e, 1 ise 0’a) gibi basit bit işlemlerini gerçekleştirir.

Programcılar için bitler biçimindeki düşük seviyeli verilerle çalışmak oldukça zahmetlidir.

Bunun yerine, programcılar rakamlar (yani 0, 1, 2, 3, 4, 5, 6, 7, 8, 9), harfler (yani A-Z, a-z)

ve özel semboller ( örneğin $, @, %, &, \*, ”, :, ? ve diğerleri ) formundaki verilerle çalışmayı

428

tercih ederler. Rakamlar, harfler ve özel semboller karakterler olarak bilinir. Bir bilgisayarda

programlar yazmak ve veri parçalarını temsil etmek için kullanılan tüm karakterlerin

kümesine, bilgisayarın karakter seti denir. Bilgisayarlar yalnızca 1 ve 0’ları işleyebildiğinden,

bilgisayarın karakter setindeki her karakter, 1 ve 0’ların değişik biçimde dizilişleriyle (bu

dizilişe byte denir) temsil edilir. Bugün, çok yaygın olarak, bir byte 8 bitten oluşur.

Programcılar programlarını ve veri parçalarını karakterlerle yaratabilirler ; bilgisayarlarda bu

karakterleri, bitlerin dizilişleri biçiminde yönetir ve işlerler.

Karakterlerin bitlerden oluşması gibi alanlar da karakterlerden oluşur. Bir alan ( field ),

karakterlerin anlam içerecek şekilde dizilişidir. Örneğin, yalnızca büyük ve küçük harfler

kullanılarak oluşturulan bir alan, bir kişinin ismini temsil etmek için kullanılabilir.

Bilgisayarlar tarafından işlenen veri parçaları, bitlerden karakterlere, karakterlerden alanlara

ve bu şekilde ilerleyerek daha büyük ve karmaşık bir hale geldikçe bir veri hiyerarşisi

oluşturur.

Bir kayıt (örneğin C’ de struct), bir çok alanın bir araya gelmesiyle oluşur. Örneğin, bir

bordro sisteminde bir işçi için tutulan kayıt şu alanları içerebilir:

1. Sosyal Güvenlik Numarası

2. İsim

3. Adres

4. Saatlik Ücret

5. Muaf olduğu haklar

6. Yıllık kazancı

7. Vergi miktarı

Bu sebepten, bir kayıt, ilgili alanların topluluğudur. Az önceki örnekte alanların her biri aynı

işçiye aitti. Tabii ki, bir işyerinde birden fazla işçi çalışıyor ve bu işçilerin her biri için bordro

kaydı tutuluyor olabilir. Bir dosya ( file ), ilgili kayıtların topluluğudur. Bir şirketin bordro

dosyası normalde her işçi için bir kayıt içerir. Bu sebepten, küçük bir şirket için bordro

dosyası 22 kayıt içerirken, büyük bir şirket için 100000 kayıt içerebilir. Bir şirket için her biri

milyonlarca karakter içeren, yüzlerce ya da binlerce dosyaya sahip olmak alışılmadık bir

durum değildir. Lazer optik disklerin popülerliğinin atması ve multimedya teknolojisi ile

trilyon byte’lık dosyalar çok yakında bilgisayar marketlerinde yerini alacaktır. Şekil 11.1, veri

hiyerarşisini temsil etmektedir.

Dosyadan bir kaydı geri elde etmek için en az bir kayıt, kayıt anahtarı (record key) olarak

seçilmelidir. Bir kayıt anahtarı, bir kaydın bir kişiye ya da varlığa ait olduğunu belirler.

Örneğin, bu kısımda tanımlanan bordro kaydında, Sosyal Güvenlik Numarası kayıt anahtarı

olarak seçilebilir.

Bir dosyadaki kayıtları organize etmenin bir çok yolu vardır. Bu yollardan en popüler olanı

sıralı dosyalardır (sequential file ). Sıralı dosyalarda kayıtlar, kayıt anahtarının sırasına göre

depolanırlar. Bordro dosyasında kayıtlar genellikle Sosyal Güvenlik Numarasına göre

sıralanır. Dosyadaki ilk işçi kaydı, en düşük Sosyal Güvenlik Numarasına sahiptir ve daha

sonra gelen kayıtlar daha büyük Sosyal Güvenlik Numarası içerir.

Bir çok işte verileri saklamak için farklı dosyalar kullanılır. Örneğin, şirketler bordro

dosyalarına, müşterilerden alınacak paraların tutulduğu dosyalara, diğer şirketlere ödenecek

429

borç dosyalarına, sarfiyat listelerinin tutulduğu dosyalara ve başka tiplerde bir çok dosyaya

sahip olabilirler. İlgili dosyaların topluluğuna genellikle veri tabanı (database) denir. Veri

tabanı oluşturmak ve veri tabanını yönetmek için kullanılan programlara veri tabanı yönetim

sistemleri(Data Base Management System) denir.

01001010

1

Şekil 11.1 Veri hiyerarşisi

11.3 DOSYA VE AKIŞLAR ( STREAM )

C, her dosyayı basit olarak bitlerin ard arda geldiği bir akış olarak görür (Şekil 11.2). Her

dosya ya dosya sonu belirteci (end-of-file) ya da sistemde yönetici veri yapısı tarafından

belirlenmiş özel bir byte sayısı ile sonlanır. Bir dosya açıldığında, dosya ile ilgili bir akış

ilişkilendirilir. Program çalışmaya başladığında, üç dosya ve bu dosyalarla ilişkili akışlar;

standart giriş (standart input), standart çıkış (standart output) ve standart hata (standart error)

otomatik olarak açılır. Akışlar,dosyalar ile program arasında haberleşme kanalları oluşturur.

Örneğin, tandart giriş akışı programın klavyeden veri okumasını ve standart çıkış akışı

programın ekrana veri yazdırmasını sağlar. Bir dosyayı açmak, dosyayı işlemek için gerekli

Metin Hüseyin

Ahmet Veli

Judy Blue

Tom Green

Mert Ali

Judy Blue

Judy

Bit

Byte(ASCII karakter J )

Alan

Kayıt

Dosya

430

FILE yapısını (<stdio.h> içinde tanımlanmıştır) gösteren bir gösterici döndürür. Bu yapı,

açık dosya tablosu (open file table) adı verilen işletim sistemi dizisi için dizin gösteren bir

dosya belirteci (file descriptor) içerir. Her dizi elemanı, işletim sisteminin bir dosyayı

yönetebilmesi için kullandığı dosya kontrol bloğunu (File Control Block) içerir. Standart

giriş, standart çıkış ve standart hata stdin,stdout ve stderr göstericileri ile yönetilirler.

0 1 2 3 4 5 6 7 8 9 ... n-1

Şekil 11.2 C’nin n byte’lık bir dosyayı ele alışı.

Standart kütüphane, dosyalardan okuma yapmak ve dosyalara veri yazmak için bir çok

fonksiyon sunmaktadır. fgetc fonksiyonu, getchar gibi, dosyadan bir karakter okur. fgetc,

karakterin okunacağı dosyayı gösteren bir FILE göstericisi alır. fgetc ( stdin ) çağrısı,

stdin’den yani standart girişten bir karakter okur. Bu çağrı, getchar( ) çağrısı ile eşdeğerdir.

fputc fonksiyonu, argüman olarak yazdırılacak bir karakter ve karakterin yazdırılacağı

dosyayı gösteren bir gösterici alır. fputc ( ‘a’, stdout ) çağrısı, ‘a’ karakterini stdout’a yani

standart çıkışa yazdırır. Bu çağrı putchar ( ‘a’ ) ile eşdeğerdir.

Standart girişten veri okuyan ve standart çıkışa veri yazdıran diğer bir çok fonksiyonun benzer

isimlerde dosyaları işleme fonksiyonları bulunur. Örneğin, fgets ve fputs fonksiyonları

dosyadan bir satır okumak ya da dosyaya bir satır yazdırmak için kullanılabilir. Bu

fonksiyonların benzerleri 8. ünitede anlattığımız gets ve puts fonksiyonlarıdır. Bundan

sonraki kısımlarda, scanf ve printf fonksiyonlarının dosya işleyen eşdeğerleri olan fscanf ve

fprintf fonksiyonlarını tanıtacağız. Daha sonra ise fread ve fwrite fonksiyonlarını

tartışacağız.

11.4 SIRALI ERİŞİMLİ DOSYA YARATMAK

C, dosyalar için özel bir yapı kullanmaz. Bu sebepten, bir dosyadaki kayıt gibi gösterimler C

dilinin bir parçası değildir. Bu yüzden, programcı her uygulama için gerekli dosya yapısını

kendisi oluşturmalıdır. Aşağıdaki örnekte, programcının bir dosyada kayıt yapısını nasıl

kullandığını göreceğiz.

Şekil 11.3, bir şirketin müşterilerine verdiği hizmetlerin karşılığı olarak alacağı miktarların

kayıtlarını tutmak amacıyla kullanabileceği, basit bir sıralı erişimli dosya oluşturmaktadır.

Her müşteri için program, müşterinin hesap numarasını, müşterinin ismini ve müşterinin

borcunu (daha önceden aldığı hizmetler karşılığında şirkete ödeyeceği miktarı) almaktadır.

Her müşteri için alınan bu veriler, o müşteri için bir kayıt oluşturmakta kullanılmaktadır.

Hesap numarası, bu uygulama için kayıt anahtarı olarak kullanılmıştır. Bu dosya, hesap

numarasına göre yaratılacak ve yönetilecektir. Bu program, kullanıcının kayıtları hesap

numarası sırasına göre girdiğini kabul etmektedir. Daha ayrıntılı bir sistemde, kullanıcının

kayıtları istediği sırada girebilmesine izin veren bir sıralama yeteneği kullanılabilir. Böylelikle

kayıtlar önce sıraya konur daha sonra da dosyaya yazılır.

Dosya sonu belirteci

431

1 /\* Şekil 11.3: fig11\_03.c

2 Sıralı bir dosya yaratmak \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int hesap;

8 char isim[ 30 ];

9 double bakiye;

10 FILE \*cfPtr; /\* cfPtr = musteri.dat dosya göstericisi \*/

11

12 if ( ( cfPtr = fopen( "musteri.dat", "w" ) ) == NULL )

13 printf( "Dosya açılamadı\n" );

14 else {

15 printf( "Hesap Numarasını, ismi ve bakiyeyi giriniz.\n" );

16 printf( "EOF girerek veri girişini sonlandırın.\n" );

17 printf( "? " );

18 scanf( "%d%s%lf", &hesap, isim, &bakiye);

19

20 while ( !feof( stdin ) ) {

21 fprintf ( cfPtr, "%d %s %.2f\n",

22 hesap, isim, bakiye );

23 printf( "? " );

24 scanf( "%d%s%lf", &hesap, isim, &bakiye);

25 }

26

27 fclose( cfPtr );

28 }

29

30 return 0;

31 }

Hesap Numarasını, ismi ve bakiyeyi giriniz.

EOF girerek veri girişini sonlandırın.

? 100 Jones 24.98

? 200 Doe 345.67

? 300 White 0.00

? 400 Stone –42.16

? 500 Rich 224.62

?

Şekil 11.3 Sıralı bir dosya yaratmak.

432

Şimdi programı inceleyelim.

FILE \*cfPtr;

ifadesi, cfPtr’nin FILE yapısını gösteren bir gösterici olduğunu belirtmektedir. C programı,

her dosyayı ayrı bir FILE yapısıyla yönetir. Programcı dosyaları kullanabilmek için FILE

yapısının özelliklerini bilmelidir. İleride FILE yapısının işletim sisteminin kontrol bloğunu

(FCB) nasıl dolaylı bir biçimde idare ettiğini özel olarak göreceğiz.

Taşınırlık İpuçları 11.1

FILE yapısı işletim sistemine bağımlıdır (her sistemin dosyaları ele alışlarına göre yapı

elemanları değişiklik gösterebilir ).

Her açık dosya , o dosyayı belirtmek için kullanılan FILE tipinde bildirilmiş bir göstericiye

sahip olmak zorundadır.

if( ( cfPtr = fopen ( “ musteri . dat ” , ” w ” ) ) = = NULL )

satırı, program tarafından kullanılacak dosyaya bir isim vermektedir ( “musteri.dat”) ve

dosya ile bir haberleşme yolu kurmaktadır. cfPtr dosya göstericisi, fopen ile açılan dosyadaki

FILE yapısını gösteren bir gösterici olarak belirlenmiştir. fopen fonksiyonu iki argüman alır :

bir dosya ismi ve dosya açma modu. ”w” dosya açma modu, dosyanın yazma işlemi yapmak

için açılacağını belirtir. Eğer daha önceden var olmayan bir dosya yazma yapmak için açılırsa,

fopen o dosyayı yaratır. Eğer varolan bir dosya yazma yapmak için açılırsa, dosyanın

içindekiler hiçbir uyarı yapılmadan silinir. Programda, if yapısı cfPtr dosya göstericisinin

NULL olup olmadığına karar vermek için kullanılmıştır (Eğer NULL ise dosya açılmamıştır).

Eğer gösterici NULL ise, bir hata mesajı yazdırılır ve program sonlandırılır. Gösterici NULL

değilse veriler işlenir ve dosyaya yazılır.

Genel Programlama Hataları 11.1

Kullanıcı dosyanın içeriğini korumak isterken, var olan bir dosyayı yazma yapmak(“w”) için

açmak. Bu durumda hiçbir uyarı yapılmadan dosyanın içeriği kaybolur.

Genel Programlama Hataları 11.2

Program içinde kullanmadan önce dosyayı açmayı unutmak.

Program, kullanıcıya her kayıt için çeşitli alanları doldurması ya da veri girişinin sonlandığını

belirten dosya sonu belirtecini girmesini söyleyen bir mesaj yazdırır. Şekil 11.4, çeşitli

bilgisayar sistemleri için dosya sonu belirtecini girerken kullanılan tuş birleşimlerini

listelemektedir.

while ( ! feof ( stdin ) )

satırı, feof fonksiyonunu kullanarak stdin’in belirttiği dosyanın, dosya sonu belirtecinin elde

edilip edilmediğine karar verir. Dosya sonu belirteci, programa işlenecek daha fazla veri

kalmadığını söyler. Şekil 11.3’teki programda, kullanıcı dosya sonu belirteci için kullanılan

tuş birleşimini girdiğinde, standart giriş için dosya sonu belirteci elde edilir. feof

fonksiyonunun argümanı, dosya sonu belirteci için test edilen dosyayı gösteren göstericidir

(bu durumda stdin).

433

Fonksiyon, dosya sonu belirteci elde edilince sıfırdan farklı bir değer ( doğru) döndürür, aksi

durumda ise sıfır döndürür. Bu programda, feof çağrısını içeren while yapısı dosya sonu

belirteci elde edilmediği sürece çalışmaya devam eder.

fprintf ( cfPtr , %d %s %.2f \n ” , hesap ,isim , bakiye);

ifadesi veriyi musteri.dat dosyasına yazar. Veri, dosyayı okumak için tasarlanmış başka bir

program ile geri elde edilebilir (bakınız Kısım 11.5). fprintf fonksiyonu ile printf

fonksiyonu, fprintf fonksiyonunun argüman olarak verinin yazılacağı dosyayı gösteren bir

gösterici alması haricinde eşdeğerdir.

Bilgisayar Sistemi Tuş Birleşimi

UNIX sistemleri <return> <ctrl>d

IBM PC ve türevleri <ctrl>z

Macintosh <ctrl>d

Şekil 11.4 Çeşitli bilgisayar sistemleri için dosya sonu belirteci tuş birleşimleri.

Genel Programlama Hataları 11.3

Bir dosyayı belirtmek için yanlış dosya göstericisini kullanmak.

İyi Programlama Alıştırmaları 11.1

Bir programdan dosya işleme fonksiyonlarına yapılan çağrıların doğru dosya göstericisini

içerdiğinden emin olun.

Kullanıcı dosya sonu belirtecini girdikten sonra, program musteri.dat dosyasını fclose ile

kapatır ve program sonlanır. fclose fonksiyonu da bir dosya göstericisi (dosya ismi yerine)

alır. fclose fonksiyonu özel olarak çağrılmamışsa, işletim sistemi dosyayı program

sonlandığında kapatır. Bu işletim sistemlerinin “housekeeping” özelliklerinden biridir.

İyi Programlama Alıştırmaları 11.2

Programın dosyayı yeniden kullanmayacağını öğrendikten sonra, mümkün olduğunca çabuk

dosyayı kapatmak.

Performans İpuçları 11.1

Bir dosyayı kapatmak diğer kullanıcıların ya da programların beklediği

kaynakları serbest bırakır.

Şekil 11.3’teki programın örnek çıktısında, kullanıcı 5 hesap için gerekli bilgileri girdikten

sonra dosya sonu belirteci sinyalini yollayarak, veri girişinin sonlandığını belirtmektedir.

Örnek çıktı, kayıtların dosyada nasıl gözüktüğünü göstermemektedir. Kayıtların başarılı bir

şekilde yaratıldığını onaylamak için bir sonraki kısımda dosyayı okuyup içeriği yazdıran bir

program anlatacağız.

434

Şekil 11.5, FILE göstericileri, FILE yapıları ve hafızadaki FCB’ler arasındaki ilişkiyi

göstermektedir. ”musteri.dat” dosyası açıldığında, dosya için bir FCB hafızaya kopyalanır.

Şekil, fopen tarafından döndürülen gösterici ile işletim sisteminin dosyayı yönetmek için

kullandığı FCB arasındaki bağlantıyı göstermektedir.

Programlar hiçbir dosyayı işlemeyebilir, tek bir dosya ya da birden çok dosya işleyebilir.

Programda kullanılan her dosyanın özel bir ismi olmalıdır ( bu isim başka hiçbir dosya

tarafından kullanılamaz) ve fopen tarafından döndürülecek farklı bir göstericiye sahip

olmalıdır. Dosya açıldıktan sonra dosya işleyen diğer fonksiyonlar dosyayı uygun bir gösterici

ile belirtmelidir. Dosyalar herhangi bir modda açılabilir (Şekil 11.6). Bir dosyayı yaratmak

için ya da dosyaya yazmadan önce dosyanın tüm içeriğini silmek için kullanılacak dosyayı,

yazma modunda ( “w” ) açmak gerekir. Varolan bir dosyadan okuma yapmak için, dosyayı

okuma modunda ( “r” ) açmak gerekir. Varolan bir dosyaya kayıtlar eklemek için dosyayı

ekleme modunda açmak ( “a” ) gerekir. Dosyaya hem yazma yapmak hem de dosyadan

okuma yapmak için dosyayı güncelleme yapan üç moddan birinde açmak gerekir. ( “r +”,

”w +”, ”a+” ). ”r +” modu, dosyayı okuma ve yazma yapmak için açar. ”w +” modu, yazma

ve okuma için bir dosya yaratır. Eğer dosya daha önceden yaratılmışsa, “w+” modu dosyayı

açar ve önceki tüm içerikler kaybolur. ”a+” modu, dosyayı okuma ve yazma yapmak için

açar. Tüm yazma işlemleri dosyanın sonuna yapılır. Eğer dosya daha önceden yoksa, yaratılır.

Eğer dosyayı herhangi bir modda açarken bir hata oluşursa, fopen NULL döndürür. Bazı

genel hatalar şunlardır:

Genel Programlama Hataları 11.4

Var olmayan bir dosyayı okuma yapmak için açmak.

Genel Programlama Hataları 11.5

Dosyaya uygun erişim hakkı verilmeden dosyayı okuma ya da yazma yapmak

için açmak(Bu işletim sistemine bağlıdır)

Genel Programlama Hataları 11.6

Yeterli disk alanı olmadan dosyayı yazma yapmak için açmak.

Genel Programlama Hataları 11.7

Bir dosyayı yanlış bir dosya modu ile açmak yıkıcı hatalara yol açabilir.Örneğin,güncelleme

modu (“r +”) ile açılması gereken bir dosyayı yazma modunda(“w +”) açmak, dosyanın

bütün içeriğinin silinmesine sebep olur.

İyi Programlama Alıştırmaları 11.3

Eğer dosyanın içeriği değiştirilmeyecekse dosyayı yalnızca okuma modunda

açmak .Bu, dosya

içeriğinin istemsiz olarak değiştirilmesini engeller.Bu, en az yetki prensibinin

başka bir örneğidir.

435

Kullanıcının buna erişme hakkı vardır Yalnızca işletim sisteminin buna

erişme hakkı vardır.

1

cfPtr = fopen ( “ musteri.dat” , ”w” );

fopen FILE yapısını(<stdio.h>

içinde tanımlanmıştır) gösteren bir

gösterici döndürür.

2

FILE yapısı “musteri.dat”

için bir belirteç içerir.Bu belirteç

Açık Dosya Tablosu için bir dizin

belirten küçük bir tamsayıdır.

3

Program,fprintf(cfPtr,”%d %s %.2f”,

hesap,isim,bakiye);

gibi bir giriş/çıkış çağrısı yaptığında,

FILE yapısı içindeki belirteci (7) bulur

ve bu belirteci Açık Dosya Tablosundaki

FCB’ yi bulmak için kullanır.

Şekil 11.5 FILE göstericileri, FILE yapıları ve FCB’ler arasındaki ilişki

yeniPtr

7

“musteri.dat” için FCB

Açık Dosya Tablosu

Program bir işletim

sistemi servisini

çağırır.Bu servis FCB

içindeki veriyi

kullanarak diskteki

dosyaya tüm giriş/çıkış

işlemlerini kontrol

eder.Not:Kullanıcı

FCB’ ye doğrudan

erişemez.

Bu giriş

dosya açıldığında

FCB ’den diske

kopyalanır.

4

436

Mod Tanım

r Bir dosyayı okumak için aç

w Yazma yapmak için bir dosya yarat.

a Ekle;bir dosyayı sonuna ekleme yapmak için aç ya da

yarat

r+ Bir dosyayı güncellemek (okuma ve yazma yapmak)için

aç

w+ Güncelleme yapmak için bir dosya yarat.Dosya daha

önceden varsa önceki içeriği silinir.

a+ Ekle:bir dosyayı güncellemek için aç ya da yarat;yazma

işlemi dosyanın sonuna yapılır.

Şekil 11.6 Dosya açma modları

11.5 SIRALI ERİŞİMLİ DOSYADAN VERİ OKUMAK

Veriler dosyalarda tutulur. Böylece, verinin işlenmesi gerektiğinde yeniden elde edilirler. Bir

önceki kısım, sıralı erişimli bir dosyanın nasıl yaratılacağını göstermişti. Bu kısımda, sıralı

erişimli bir dosyadan nasıl okuma yapacağımızı göreceğiz.

Şekil 11.7’deki program, Şekil 11.3’te yaratılan “musteri.dat” dosyasındaki kayıtları okuyup

kayıtların içeriklerini yazdıracaktır.

FILE \*cfPtr;

ifadesi cfPtr’nin bir FILE göstericisi olduğunu belirtmektedir.

if ( ( cfPtr = fopen ( “ musteri . dat ” , ” r ” ) ) = = NULL )

satırı “musteri.dat” dosyasını okuma yapmak için ( “r” ) açmaya çalışmaktadır ve dosyanın

başarılı bir şekilde açılıp açılmadığına ( eğer fopen NULL döndürmezse dosya açılmış

demektir) karar vermektedir.

1 /\* Şekil 11.7: fig11\_07.c

2 Sıralı erişimli bir dosyadan okuma ve dosyaya yazma yapmak.\*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int hesap;

8 char ad[ 30 ];

9 double bakiye;

10 FILE \*cfPtr; /\* cfPtr = musteri.dat dosya göstericisi \*/

11

12 if ( ( cfPtr = fopen( "musteri.dat", "r" ) ) == NULL )

13 printf( "Dosya açılamadı\n" );

437

14 else {

15 printf( "%-10s%-13s%s\n", "Hesap", "İsim", "Bakiye" );

16 fscanf( cfPtr, "%d%s%lf", &hesap, ad, &bakiye);

17

18 while ( !feof( cfPtr ) ) {

19 printf( "%-10d%-13s%7.2f\n", hesap, ad, bakiye);

20 fscanf( cfPtr, "%d%s%lf", &hesap, ad, &bakiye);

21 }

22

23 fclose( cfPtr );

24 }

25

26 return 0;

27 }

Hesap İsim Bakiye

100 Jones 24.98

200 Doe 345.67

300 White 0.00

400 Stone –42.16

500 Rich 224.62

Şekil 11.7 Sıralı erişimli bir dosyayı okuyup, yazdırmak.

fscanf ( cfPtr , ” %d %s %f ”, &hesap , isim , &bakiye);

ifadesi dosyadan bir kayıt okur. fscanf fonksiyonu, fscanf’in okuma yapılacak dosyayı

gösteren bir göstericiyi argüman olarak kullanması dışında scanf fonksiyonu ile denktir. Az

önceki ifade ilk kez çalıştırıldıktan sonra, hesap 100 , isim “Jones” ve bakiye 24.98 değerine

sahip olacaktır. Daha sonraki fscanf fonksiyon çağrıları, dosyadan başka bir kayıt okur ve

hesap, isim ve bakiye değişkenleri yeni değerler alır. Dosyanın sonuna erişildiğinde dosya

kapatılır ve program sona erer.

Sıralı erişimli bir dosyadan verileri geri elde etmek için, program okuma yapmaya dosyanın

başından başlar ve istenen veri bulunana kadar sırayla tüm veriler okunur. Bir dosyadaki

veriyi (dosyanın başından başlayarak), program çalışırken ard arda bir çok kez işlemek

gerekebilir.

rewind ( cfPtr ) ;

gibi bir ifade, programın dosya pozisyon göstericisini (dosyada okuma ya da yazma yapılacak

bir sonraki byte numarasını belirtir) cfPtr ile gösterilen dosyanın başına ( 0.byte’a ) geri

döndürmesini sağlatır. Dosya pozisyon göstericisi aslında bir gösterici değildir. Daha doğrusu

bu, dosyada okuma ya da yazma yapılacak bir sonraki byte’ın konumu belirten bir tamsayı

değeridir. Buna çoğu zaman dosya offseti denir. Dosya pozisyon göstericisi, FILE yapısının

her dosyayla ilişkili bir elemanıdır.

Şimdi de bir kredi yöneticisinin şirketin kredi durumunu özetleyen listeleri elde etmesini

sağlayan programımızı ( Şekil 11.8 ) inceleyelim. Bu program, şirkete borcu olmayan

438

müşterilerin listesini, şirketin kredi verdiği müşterilerin listesini ve şirkete aldığı hizmetler

karşılığında para veren müşterilerin listesini özetleyecektir. Şirketin verdiği krediler negatif

bir değer, şirketin müşterilerine sağladığı hizmetler karşısında aldığı paralar pozitif bir

değerdir.

1 /\* Şekil 11.8: fig11\_08.c

2 Kredi araştırma programı \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 int secim, hesap;

8 double bakiye;

9 char isim[ 30 ];

10 FILE \*cfPtr;

11

12 if ( ( cfPtr = fopen( "musteri.dat", "r" ) ) == NULL )

13 printf( "Dosya açılamadı\n" );

14 else {

15 printf( "Seçiminiz : \n"

16 " 1 - Sıfır bakiyesi olan hesapları listele \n"

17 " 2 - Kredili hesapları listele \n"

18 " 3 - Borcu olan hesapları listele\n"

19 " 4 - Çıkış \n? " );

20 scanf( "%d", &secim);

21

22 while ( secim!= 4 ) {

23 fscanf( cfPtr, "%d%s%lf", &hesap, isim, &bakiye);

24

25

26 switch ( secim ) {

27 case 1:

28 printf( "\nSıfır bakiyesi olan hesaplar:\n" );

29

30 while ( !feof( cfPtr ) ) {

31

32 if ( bakiye == 0 )

33 printf( "%-10d%-13s%7.2f\n",

34 hesap, isim, bakiye);

35

36 fscanf( cfPtr, "%d%s%lf",

37 &hesap, isim, &bakiye);

38

39 }

40

41 break;

42 case 2:

43 printf( "\nKredili hesapların listesi:\n" );

44

439

45 while ( !feof( cfPtr ) ) {

46

47 if ( bakiye < 0 )

48 printf( "%-10d%-13s%7.2f\n",

49 hesap, isim, bakiye);

50

51 fscanf( cfPtr, "%d%s%lf",

52 &hesap, isim, &bakiye);

53

54 }

55

56 break;

57 case 3:

58 printf( "\nBorcu olan hesaplar:\n" );

59

60 while ( !feof( cfPtr ) ) {

61

62 if ( bakiye > 0 )

63 printf( "%-10d%-13s%7.2f\n",

64 hesap, isim, bakiye);

65

66 fscanf( cfPtr, "%d%s%lf",

67 &hesap, isim, &bakiye);

68 }

69

70 break;

71 }

72

73 rewind( cfPtr );

74 printf( "\n? " );

75 scanf( "%d", &secim);

76 }

77

78 printf( "Çıkış.\n" );

79 fclose( cfPtr );

80 }

81

82 return 0;

83 }

Şekil 11.8 Kredi araştırma programı

Program bir menü yazdırmakta ve kredi yöneticisinin kredi bilgilerini elde edebilmesi için üç

seçenekten birini seçmesine izin vermektedir. 1. seçenek şirkete borcu olmayan müşterilerin

listesini, 2.seçenek şirketin kredi verdiği müşterilerin listesini ve 3. seçenek şirkete aldığı

hizmetler karşılığında para veren müşterilerin listesini özetleyecektir. 4. seçenek programı

sonlandırmaktadır.Programın örnek bir çıktısı Şekil 11.9’da gösterilmiştir.

440

Seçiminiz :

1 - Sıfır bakiyesi olan hesapları listele

2 - Kredili hesapları listele

3 - Borcu olan hesapları listele

4 - Çıkış

? 1

Sıfır bakiyesi olan hesaplar:

300 White 0.00

? 2

Kredili hesapların listesi:

400 Stone -42.16

? 3

Borcu olan hesaplar:

100 Jones 24.98

200 Doe 345.67

500 Rich 224.62

? 4

Çıkış.

Şekil 11.9 Kredi araştırma programının(Şekil 11.8) çıktısı.

Bu tarzda sıralı erişimli dosyalarda, dosyadaki diğer verilere zarar verme ihtimali olmadan

değiştirme yapılamadığına dikkat ediniz. Örneğin, eğer “White” ismi “Worthington” olarak

değiştirilecekse, yeni isim doğrudan eski ismin üzerine yazılamaz. White için kayıt dosyaya

300 White 0.00

şeklinde yazılmıştır.

Eğer kayıt, aynı konumdan başlanarak farklı bir isimle dosyaya yazdırılırsa kayıt

300 Worthington 0.00

olacaktır. Yeni kayıt eski kayıttan daha büyüktür. ”Worthington” içindeki ikinci “o”

karakterinden sonraki karakterler, dosyadaki bir sonraki kaydın başlangıcından itibaren

yazılacaktır. Buradaki problem, fprintf ve fscanf kullanarak yapılan biçimlendirilmiş

giriş/çıkış modelinde alanların ( bu sebepten de kayıtların ) büyüklüğünün değişebilmesidir.

Örneğin, 7, 14, -117, 2074 ve 27383 değerlerinin tümü int tipi değerlerdir ve bilgisayarda

441

aynı sayıda byte içinde tutulurlar fakat ekranda ya da fprintf kullanılarak diske

yazdırıldıklarında ise farklı boyutlu alanlara yazdırılır.

Bu sebepten, sıralı erişimde fprintf ve fscanf kayıtları yerinde güncellemek için kullanılmaz.

Bunun yerine, tüm dosya yeniden yazılır. Bu tarzda bir sıralı erişimli dosyada, önceki ismi

değiştirmek için 300 White 0.00 kaydından önceki kayıtlar yeni bir dosyaya kopyalanır, yeni

kayıt yazdırılır ve 300 White 0.00 kaydından sonraki kayıtlar yeni dosyaya kopyalanır. Bu bir

kaydı güncellemek için tüm kayıtların işlenmesini gerektirir.

11.6 RASGELE ERİŞİMLİ DOSYALAR

Daha önceden belirttiğimiz gibi, biçimlendirilmiş çıktı fonksiyonu fprintf ile oluşturulan

kayıtlar aynı uzunlukta olmak zorunda değildir. Ancak, rasgele erişimli bir dosyadaki tüm

kayıtlar sabit uzunluğa sahiptirler ve diğer kayıtların aranmasına gerek kalmadan doğrudan

(ve bu sebepten hızlıca) erişilebilirler. Bu, uçuş rezervasyon sistemleri, banka sistemleri, satış

noktası sistemleri ve diğer evrak işleme sistemleri gibi veriye hızlı erişim gerektiren sistemler

için rasgele erişimli dosyaları uygun yapar. Rasgele erişimli dosyaları uygulamak için başka

yollarda bulunmaktadır ancak biz burada anlatımımızı sabit uzunlukta kayıtlar yaklaşımını

kullanma ile kısıtlı tutacağız.

Rasgele erişimli bir dosyada her kayıt sabit uzunluğa sahip olduğundan, kaydın dosyanın

başlangıcına göre konumu anahtar kaydın bir fonksiyonu olarak bulunabilir. İleride bunun

büyük dosyalarda bile kayıtlara hızlı erişimi nasıl sağladığını göreceğiz.

Şekil 11.10, rasgele erişimli bir dosyayı göstermek için kullanılan yollardan birini tasvir

etmektedir. Böyle bir dosya bir çok vagonu olan bir yük treni gibidir.Vagonlardan bazıları boş

bazıları ise kargo yüklü olabilir. Trendeki her vagon aynı uzunluğa sahiptir.

Rasgele erişimli dosyalara, dosyadaki diğer verilere zarar vermeden yeni veriler eklenebilir.

Daha önceden depolanmış veriler, tüm dosyanın yeniden yazılmasına gerek kalmadan

güncellenebilir ya da silinebilir. İlerideki kısımlarda, rasgele erişimli bir dosyanın nasıl

yaratıldığını, böyle dosyalara nasıl veri girildiğini, dosyadaki verilerin hem rasgele hem de

sıralı bir biçimde nasıl okunduğunu ve daha fazla kullanılmayan verilerin dosyadan nasıl

silindiğini açıklayacağız.

0 100 200 300 400 500

100 100 100 100 100 100

byte byte byte byte byte byte

442

Şekil 11.10 C’ in rasgele erişimli bir dosyayı ele alışı.

11.7 RASGELE ERİŞİMLİ DOSYA YARATMAK

fwrite fonksiyonu, hafızada belirlenmiş bir konumdan aldığı belli sayıdaki byte’ı dosyaya

aktarır. Bu veriler dosyaya, dosya pozisyon göstericisi ile gösterilen konumdan itibaren

yazılır. fread fonksiyonu, dosya içinde dosya pozisyon göstericisi ile belirlenen konumdan

aldığı belli sayıdaki byte’ı , belirlenen adresten başlayarak hafızaya aktarır. Şimdi, bir tamsayı

yazarken ( 4-byte tamsayılar için ) tamsayının değerine göre 1 basamak ya da 11 basamak

yazdırabilen ( 10 basamak artı işaret, her biri 1 byte’ lık alanda depolanır)

fprintf ( fptr , ” % d ” , sayi ) ;

yerine, her zaman sayi değişkenindeki 4 byte’ı (ya da 2-byte tamsayı kullanan sistemler için 2

byte’ı) fPtr ile gösterilen dosyaya yazan

fwrite ( &sayi , sizeof (int) , 1 , fPtr ) ;

ifadesini kullanabiliriz (1 argümanını birazdan açıklayacağız). Daha sonra, fread bu

byte’ların 4’ünü sayi değişkeni içine okumak için kullanılabilir. fread ve fwrite fonksiyonları

değişkenin değerine göre okuma ya da yazma yapmak yerine, verileri her zaman sabit boyutta

olacak biçimde okuyup ve yazar. Ancak bu fonksiyonların kullandıkları veriler, printf ve

scanf’in insanlar tarafından kolaylıkla anlaşılabilecek biçimi yerine, bilgisayarda “çiğ veri”

(byte’lar halinde veri) biçiminde işlenir .

fwrite ve fread fonksiyonları, veri dizilerini diske okuyup yazabilir ya da diskten veri dizileri

içine okuma ve yazma yapabilir. fread ve fwrite fonksiyonlarındaki üçüncü argüman, diskten

diziye okunacak ya da diziden diske yazılacak eleman sayısını belirtir. Az önceki fwrite

fonksiyon çağrısı, diske tek bir tamsayı yazdırdığından üçüncü argüman 1’dir (sanki bir

dizinin bir elemanı yazdırılıyormuş gibi de düşünebiliriz).

Dosya işleyen programlar çok nadir olarak dosyadaki tek bir alana yazma yapar. Normalde,

örneklerde göstereceğimiz gibi her seferinde bir struct yazarlar.

Şimdi aşağıdaki problemi inceleyelim:

Sabit uzunlukta 100 adet kaydı tutabilecek bir hesap takip sistemi yaratacağız. Her kayıt,kayıt

anahtarı olarak kullanılacak bir hesap numarası içermelidir. Ayrıca kayıtlarda, isim, soy isim

ve bakiye belirtilmelidir. Program bir hesabı güncelleyebilecek, yeni bir hesap kaydı

oluşturabilecek, eski bir kaydı silebilecek ve bütün hesap kayıtlarını yazdırmak için,

biçimlendirilmiş bir metin dosyası içinde listeleyecektir.

Bundan sonraki birkaç kısım, bu sistemi yaratmak için gerekli olan teknikleri anlatmaktadır.

Şekil 11.11’deki program, rasgele erişimli bir dosyanın nasıl açılacağını, struct ile bir kayıt

biçiminin oluşturuluşunu ve veriler diske yazıldıktan sonra dosyanın kapatılışını

göstermektedir. Bu program, “kredi.dat” dosyasındaki 100 kaydın hepsini de fwrite

fonksiyonu kullanarak boş struct’lara atamaktadır. Her boş struct, hesap numarası için 0, soy

isim için NULL ( tırnak işaretlerinin arası boş bırakılarak temsil edilmiştir), isim için NULL

443

ve bakiye için 0.00 içermektedir. Bu dosya, bu şekilde ilk değerlere atanarak dosya için diskte

bir alan oluşturulmuş ve bir kaydın veri içerip içermediğine karar vermek mümkün hale

gelmiştir.

1 /\* Şekil 11.11: fig11\_11.c

2 Rasgele erişimli bir dosyayı sıralı biçimde oluşturmak. \*/

3 #include <stdio.h>

4

5 struct musteriVerisi{

6 int hesapNo;

7 char soyisim [ 15 ];

8 char isim[ 10 ];

9 double bakiye;

10 };

11

12 int main( )

13 {

14 int i;

15 struct musteriVerisi bosVeri = { 0, "", "", 0.0 };

16 FILE \*cfPtr;

17

18 if ( ( cfPtr = fopen( "kredi.dat", "w" ) ) == NULL )

19 printf( "Dosya açılamadı\n" );

20 else {

21

22 for ( i = 1; i <= 100; i++ )

23 fwrite( &bosVeri,

24 sizeof ( struct musteriVerisi), 1, cfPtr );

25

26 fclose ( cfPtr );

27 }

28

29 return 0;

30 }

Şekil 11.11 Rasgele erişimli bir dosyayı sıralı biçimde oluşturmak.

fwrite fonksiyonu dosyaya bir veri bloğu ( belli sayıda byte ) yazar. Programımızda,

fwrite ( &bosVeri , sizeof ( struct musteriVerisi ) , 1 , cfPtr ) ;

ifadesi, cfPtr ile gösterilen dosyaya sizeof ( struct musteriVerisi ) boyutundaki bosVeri

yapısının yazdırılmasını sağlamaktadır. sizeof operatörü, parantezler içindeki nesnenin

(burada bu nesne struct musteriVerisi‘dir) boyutunu byte olarak döndürür. sizeof operatörü,

işaretsiz bir tamsayı döndüren derleme zamanlı tekli bir operatördür. sizeof operatörü,

herhangi bir veri ya da deyimin boyutunu byte olarak belirlemek için kullanılır. Örneğin,

sizeof ( int ) bir bilgisayarda tamsayıların, 2 byte içinde mi yoksa 4 byte içinde mi

tutulduğunu belirlemek için kullanılır.

444

Performans İpuçları 11.2

fwrite fonksiyonu, aslında bir nesne dizisindeki bir çok elemanı yazdırmak için kullanılır.

Birden fazla dizi elemanını yazdırmak için, programcı diziyi gösteren bir göstericiyi fwrite

çağrısında ilk argüman olarak kullanır ve yazdırılacak eleman sayısını da üçüncü argüman ile

belirtir.Az önceki ifadede, fwrite dizi elemanı olmayan tek bir nesneyi yazdırmak için

kullanılmıştı.Tek bir nesneyi yazdırmak, bir dizinin tek bir elemanını yazdırmakla eşdeğerdir.

Bu sebepten, fwrite çağrısı içinde 1 kullanılmıştır.

11.8 RASGELE ERİŞİMLİ DOSYAYA RASGELE VERİ YAZMAK

Şekil 11.12’deki program, ”kredi.dat” dosyasına veri yazar. Bu program, veriyi dosyada belli

konumlara yerleştirebilmek için fseek ve fwrite kombinasyonlarını kullanır. fseek

fonksiyonu, dosya pozisyon göstericisini dosyada belli bir konuma götürür ve daha sonra

fwrite veriyi yazar. Programın örnek bir çıktısı Şekil 11.3’te gösterilmiştir.

1 /\* Şekil 11.12: fig11\_12.c

2 Rasgele erişimli bir dosyaya rasgele veri yazmak \*/

3

4 #include <stdio.h>

5

6 struct musteriVerisi {

7 int hesapNo;

8 char soyisim[ 15 ];

9 char isim[ 10 ];

10 double bakiye;

11 };

12

13 int main( )

14 {

15 FILE \*cfPtr;

16 struct musteriVerisi musteri= { 0, "", "", 0.0 };

17

18 if ( ( cfPtr = fopen( "kredi.dat", "r+" ) ) == NULL )

19 printf( "Dosya açılamadı.\n" );

20 else {

21 printf( "Hesap numarasını giriniz: "

22 " ( 1 den 100’ e kadar, çıkış için 0) \n? " );

23 scanf( "%d", &musteri.hesapNo );

24

25 while (musteri.hesapNo!= 0 ) {

26 printf( "soyisim, isim, bakiye giriniz:\n? " );

27 fscanf( stdin, "%s%s%lf", musteri.soyisim,

28 musteri.isim, &musteri.bakiye);

29 fseek( cfPtr, ( musteri.hesapNo- 1 ) \*

30 sizeof( struct musteriVerisi), SEEK\_SET );

Çoğu programcı, yanlış bir biçimde, sizeof’un bir fonksiyon olduğunu ve sizeof kullanmanın çalışma

zamanında fazladan bir fonksiyon çağrısı gerektireceğini düşünür.Bu şekilde bir durum yoktur çünkü

sizeof derleme zamanlı bir operatördür.

445

31 fwrite( &musteri, sizeof( struct musteriVerisi), 1,

32 cfPtr );

33 printf( "Hesap Numarasını giriniz: \n? " );

34 scanf( "%d", &musteri.hesapNo);

35 }

36

37 fclose( cfPtr );

38 }

39

40 return 0;

41 }

Şekil 11.12 Rasgele erişimli bir dosyaya rasgele veri yazmak

Hesap Numarasını giriniz:

? 37

soyisim, isim bakiye giriniz:

? Karaca Huseyin 0.00

Hesap Numarasını giriniz:

? 29

soyisim, isim bakiye giriniz:

? Zavrak Metin –24.54

Hesap Numarasını giriniz:

? 96

soyisim, isim bakiye giriniz:

? Aksoy Ekrem 34.98

Hesap Numarasını giriniz:

? 88

soyisim, isim bakiye giriniz:

? Alkılıçgil Erdem 258.34

Hesap Numarasını giriniz:

? 33

soyisim, isim bakiye giriniz:

? Acındı Alper 314.33

Hesap Numarasını giriniz:

? 0

Şekil 11.13 Şekil 11.12’deki programın örnek bir çıktısı.

fseek ( cfPtr , ( musteri.hesapNo - 1) \* sizeof ( struct musteriVerisi ) , SEEK\_SET ) ;

ifadesi, cfPtr ile belirlenen dosyadaki dosya pozisyon göstericisini, ( musteri.hesapNo - 1) \*

sizeof ( struct musteriVerisi ) deyimi ile hesaplanan byte konumuna taşır. Bu deyimin değeri

genellikle, offset ya da yer değiştirme olarak bilinir. Byte konumları 0 ile başladığından ve

hesap numarası 1 ile 100 arasında olduğundan, kaydın byte konumu hesaplanırken hesap

numarasından 1 çıkartılır. Bu sebepten, kayıt 1 için dosya pozisyon göstericisi dosyadaki 0.

byte’a taşınır. SEEK\_SET sembolik sabiti, dosya pozisyon göstericisinin yer değiştirmesinin

dosyanın başındaki konumdan itibaren yapılacağını belirtir. Yukarıdaki ifadede belirtildiği

gibi, hesap numarası 1 iken dosya pozisyon göstericisi dosyanın başındadır çünkü byte

446

konumu 0 olarak hesaplanmaktadır. Şekil 11.14, hafızadaki FILE yapısını belirten dosya

pozisyon göstericisini tasvir etmektedir. Dosya pozisyon göstericisi okuma ya da yazma

yapılacak bir sonraki byte’ın, dosyanın başından itibaren 5 byte sonra olduğunu

belirtmektedir.

fseek fonksiyonunun prototipi ANSI standardına göre aşağıdaki biçimdedir:

int fseek(FILE \*stream, long int offset, int whence);

Burada offset, stream ile belirtilen dosyadaki whence konumundan itibaren byte sayısıdır.

whence argümanı, dosyada aramanın başlayacağı konumu belirten SEEK\_SET,

SEEK\_CUR ve SEEK\_END değerlerinden birini alabilir. SEEK\_SET aramanın dosyanın

başından başlayacağını; SEEK\_CUR aramanın dosyadaki o anda bulunulan konumdan

başlayacağını ve SEEK\_END aramanın dosya sonundan başlayacağını belirtir. Bu üç

sembolik sabit stdio.h öncü dosyası içinde tanımlanmıştır.

Şekil 11.14 Dosyanın başlangıcından itibaren 5 byte’lık bir yer değiştirmeyi belirten dosya

pozisyon göstericisi

11.9 RASGELE ERİŞİMLİ DOSYADAN RASGELE VERİ OKUMAK

fread fonksiyonu, bir dosyadan hafızaya belli sayıda byte okur. Örneğin,

fread (&musteri,sizeof ( struct musteriVerisi) ,1, cfPtr);

ifadesi, cfPtr ile belirlenen dosyadan sizeof ( struct musteriVerisi) ile belirlenen sayıda

byte’ı okur ve bu veriyi musteri yapısına depolar. Dosyadan okunacak byte’ların konumu,

cfPtr

5

0 1 2 3 4 5 6 7 8 9 .....

byte numarası

(dosya pozisyon

göstericisi)

447

dosya pozisyon göstericisi tarafından belirlenir. fread fonksiyonu, okunacak elemanların

sayısı ve okunan elemanların depolanacağı diziyi gösteren bir gösterici belirtilerek, diziden

sabit boyuttaki birden çok elemanı okumak için kullanılabilir. Az önceki ifade, bir elemanın

okunacağını belirtiyordu. Daha fazla eleman okumak için, okunacak eleman sayısı fread

fonksiyonunun üçüncü argümanında belirtilmelidir.

Şekil 11.15, “kredi.dat” dosyasındaki her kaydı sırayla okumakta ve her kaydın veri içerip

içermediğine karar verip, veri içeren kayıtları biçimlendirilmiş bir şekilde yazdırmaktadır.

feof fonksiyonu dosyanın sonuna ne zaman ulaşıldığına karar vermektedir ve fread

fonksiyonu diskteki veriyi musteriVerisi yapısı olan musteri içine aktarır.

1 /\* Şekil 11.15: fig11\_15.c

2 Rasgele erişimli dosyadan sıralı okuma yapmak.\*/

3 #include <stdio.h>

4

5 struct musteriVerisi {

6 int hesapNo;

7 char soyisim [ 15 ];

8 char isim[ 10 ];

9 double bakiye;

10 };

11

12 int main( )

13 {

14 FILE \*cfPtr;

15 struct musteriVerisi musteri = { 0, "", "", 0.0 };

16

17

18 if ( ( cfPtr = fopen( "kredi.dat", "r" ) ) == NULL )

19 printf( "Dosya açılamadı.\n" );

20 else {

21 printf( "%-6s%-16s%-11s%10s\n", "HspNo", "Soyisim ",

22 "İsim", "Bakiye" );

23

24 while ( !feof( cfPtr ) ) {

25 fread( &musteri, sizeof( struct musteriVerisi), 1,

26 cfPtr );

27

28 if ( musteri.hesapNo != 0 )

29 printf( "%-6d%-16s%-11s%10.2f\n",

30 musteri.hesapNo, musteri.soyisim,

31 musteri.isim, musteri.bakiye);

32 }

33

34 fclose( cfPtr );

35 }

36

37 return 0;

448

38 }

HspNo Soyisim İsim Bakiye

29 Zavrak Metin -24.54

33 Alper Acındı 314.33

37 Huseyin Karaca 0.00

88 Erdem Alkılıçgil 258.34

96 Ekrem Aksoy 34.98

Şekil 11.15 Rasgele erişimli dosyadan sıralı okuma yapmak.

11.10 ÖRNEK: EVRAK İŞLEME SİSTEMİ

Şimdi, rasgele erişimli dosyaları kullanarak, oldukça güçlü bir evrak işleme sistemi

göstereceğiz. Bu program, bir bankanın hesap bilgilerini yönetmektedir. Program, var olan

hesapları güncellemekte, hesapları silmekte, yeni hesaplar eklemekte ve tüm hesapları

listeleyerek yazdırmak için bir metin dosyasında tutmaktadır. Şekil 11.11’deki programın

çalıştırılarak, kredi.dat dosyasını yarattığını varsayıyoruz.

Programın 5 seçeneği bulunmaktadır. 1. seçenek, metinDosyasi fonksiyonunu çağırarak tüm

hesapları, hesaplar.txt adlı bir dosyaya biçimlendirilmiş bir şekilde kaydetmektedir.

hesaplar.txt ,ileride yazdırılarak hesapların durumu incelenebilir. Fonksiyon, fread ve Şekil

11.15’te kullanılan sıralı erişimli dosya tekniklerini kullanmaktadır. 1. seçenek seçildikten

sonra hesaplar.txt dosyası şunları içermektedir:

HspNo Soyisim İsim Bakiye

29 Zavrak Metin -24.54

33 Alper Acındı 314.33

37 Huseyin Karaca 0.00

88 Erdem Alkılıçgil 258.34

96 Ekrem Aksoy 34.98

2.seçenek bir hesabı güncellemek için, kayitGuncelle fonksiyonunu çağırmaktadır.

Fonksiyon, yalnızca varolan bir kaydı güncellemektedir. Bu sebepten, öncelikle kullanıcı

tarafından belirtilen kaydın boş olup olmadığına kontrol eder. Kayıt fread ile musteri yapısı

içine aktarılır ve hesapNo elemanı 0 ile karşılaştırılır. Eğer 0 ise, kayıt bilgi içermemektedir

ve kaydın boş olduğunu belirten bir mesaj yazdırılır. Daha sonra menü yeniden görüntülenir.

Eğer kayıt bilgi içeriyorsa, kayitGuncelle fonksiyonu müşteriden alınan miktarın ya da

müşteriye verilen miktarın girilmesini ister. Bu bilgi girildikten sonra borç hesaplanır ve

dosyadaki kaydın üzerine yazılır. Seçenek 2 için çıktı şu şekilde gözükecektir.

449

Güncellenecek Hesap Numarasını giriniz.: ( 1 - 100 ): 37

37 Huseyin Karaca 0.00

borç ( + ) ya da ödeme ( - ) giriniz: +87.99

37 Huseyin Karaca 87.99

3.seçenek, yeniKayit fonksiyonunu kullanarak dosyaya yeni bir hesap ekler. Eğer, kullanıcı

daha önceden var olan bir hesap numarası girerse, fonksiyon bu kaydın bilgi içerdiğini

belirten bir hata mesajı yazdırır ve menü yeniden görüntülenir. Bu fonksiyon, Şekil

11.12’deki programın yeni bir kayıt eklerken izlediği sürecin aynısını kullanmaktadır.

Seçenek 3 için çıktı şu şekilde gözükecektir.

Yeni hesap numarasını giriniz ( 1 - 100 ): 22

Soyisim, isim ve bakiye giriniz

? Gorkem Sahin 247.45

4.seçenek, kayitSil fonksiyonunu dosyadaki bir kaydı silmek için çağırır. Silme işlemi,

kullanıcıya silinecek kayıt numarası sorularak ve kayıt yeniden ilk değerlere (0, NULL,

NULL) atanarak yapılır. Eğer hesap bilgi içermiyorsa, kayitSil fonksiyonu kaydın

bulunmadığını belirten bir hata mesajı yazdırır. 5. seçenek programı sonlandırır. Program,

Şekil 11.16’da gösterilmiştir. ”kredi.dat” dosyasının güncelleme yapmak için “r +” modu

(yazma ve okuma) kullanılarak açıldığına dikkat ediniz.

1 /\* Şekil 11.16: fig11\_16.c

2 Bu program rasgele erişimli bir dosyayı açar,

3 dosyaya daha önceden yazılmış veriyi günceller,

4 dosyaya yerleştirilecek yeni veriyi oluştururur,

5 ve dosyada önceden varolan verileri siler \*/

6 #include <stdio.h>

7

8 struct musteriVerisi{

9 int hspNo;

10 char soyisim [ 15 ];

11 char isim[ 10 ];

12 double bakiye;

13 };

14

15 int secimGir( void );

16 void metinDosyasi ( FILE \* );

17 void kayitGuncelle ( FILE \* );

450

18 void yeniKayit ( FILE \* );

19 void kayitSil ( FILE \* );

20

21 int main( )

22 {

23 FILE \*cfPtr;

24 int secim;

25

26 if ( ( cfPtr = fopen( "kredi.dat", "r +" ) ) == NULL )

27 printf( "Dosya açılamadı.\n" );

28 else {

29

30 while ( ( secim = secimGir( ) ) != 5 ) {

31

32 switch ( secim ) {

33 case 1:

34 metinDosyasi ( cfPtr );

35 break;

36 case 2:

37 kayitGuncelle( cfPtr );

38 break;

39 case 3:

40 yeniKayit( cfPtr );

41 break;

42 case 4:

43 kayitSil( cfPtr );

44 break;

45 }

46 }

47

48 fclose( cfPtr );

49 }

50

51 return 0;

52 }

53

54 void metinDosyasi( FILE \*okuPtr )

55 {

56 FILE \*yazPtr;

57 struct musteriVerisi musteri= { 0, "", "", 0.0 };

58

59 if ( (yazPtr = fopen( "hesaplar.txt", "w" ) ) == NULL )

60 printf( "Dosya açılamadı.\n" );

61 else {

62 rewind( okuPtr );

63 fprintf( yazPtr, "%-6s%-16s%-11s%10s\n",

64 "HspNo", "Soyisim", "İsim","Bakiye" );

65

66 while ( !feof( okuPtr ) ) {

67 fread( &musteri, sizeof( struct musteriVerisi), 1, okuPtr );

451

68

69 if ( musteri.hesapNo != 0 )

70 fprintf( yazPtr, "%-6d%-16s%-11s%10.2f\n",

71 musteri.hspNo, musteri.soyisim,

72 musteri.isim, musteri.bakiye );

73 }

74

75 fclose( yazPtr );

76 }

77

78 }

79

80 void kayitGuncelle ( FILE \*fPtr )

81 {

82 int hesap;

83 double guncelle;

84 struct musteriVerisi musteri = { 0, "", "", 0.0 };

85

86 printf( "Güncellenecek Hesap Numarasını giriniz.: ( 1 - 100 ): " );

87 scanf( "%d", &hesap);

88 fseek( fPtr, ( hesap - 1 ) \* sizeof( struct musteriVerisi ),

89 SEEK\_SET );

90 fread( &musteri, sizeof ( struct musteriVerisi ), 1, fPtr );

91

92 if ( musteri.hspNo== 0 )

93 printf( "Hesap#%d hakkında bilgi yok.\n", hesap);

94 else {

95 printf( "%-6d%-16s%-11s%10.2f\n\n",

96 musteri.hspNo, musteri.soyisim,

97 musteri.isim, musteri.bakiye);

98 printf( "borç ( + ) ya da ödeme ( - ) giriniz: " );

99 scanf( "%lf", &guncelle);

100 musteri.bakiye += guncelle;

101 printf( "%-6d%-16s%-11s%10.2f\n",

102 musteri.hspNo, musteri.soyisim,

103 musteri.isim, musteri.bakiye);

104 fseek( fPtr, ( hesap- 1 ) \* sizeof( struct musteriVerisi),

105 SEEK\_SET );

106 fwrite( &musteri, sizeof( struct musteriVerisi), 1, fPtr );

107 }

108 }

109

110 void kayitSil( FILE \*fPtr )

111 {

112 struct musteriVerisi musteri, bosMusteri= { 0, "", "", 0 };

113 int hesapNum;

114

115 printf( "Silinecek hesap numarasını giriniz ( 1 - 100 ): " );

116 scanf( "%d", &hesapNum);

117 fseek( fPtr, ( hesapNum - 1 ) \* sizeof( struct musteriVerisi),

452

118 SEEK\_SET );

119 fread( &musteri, sizeof( struct musteriVerisi), 1, fPtr );

120

121 if ( musteri.hspNo == 0 )

122 printf( "Hesap%d bulunamadı.\n", hesapNum);

123 else {

124 fseek( fPtr, ( hesapNum - 1 ) \* sizeof( struct musteriVerisi),

125 SEEK\_SET );

126 fwrite( &bosMusteri, sizeof( struct musteriVerisi), 1, fPtr );

127 }

128 }

129

130 void yeniKayit ( FILE \*fPtr )

131 {

132 struct musteriVerisi musteri= { 0, "", "", 0.0 };

133 int hesapNum;

134 printf( "Yeni hesap numarasını giriniz ( 1 - 100 ): " );

135 scanf( "%d", &hesapNum);

136 fseek( fPtr, ( hesapNum - 1 ) \* sizeof( struct musteriVerisi),

137 SEEK\_SET );

138 fread( &musteri, sizeof( struct musteriVerisi), 1, fPtr );

140

141 if ( musteri.hspNo!= 0 )

142 printf( "Hesap #%d zaten var.\n", musteri.hspNo);

143 else {

144 printf( "Soyisim, isim ve bakiye giriniz\n? " );

145 scanf( "%s%s%lf", &musteri.soyisim, &musteri.isim,

146 &musteri.bakiye);

147 musteri.hspNo= hesapNum;

148 fseek( fPtr, ( musteri.hspNo- 1 ) \*

149 sizeof( struct musteriVerisi), SEEK\_SET );

150 fwrite( &musteri, sizeof( struct musteriVerisi), 1, fPtr );

151 }

152 }

153

154 int secimGir( void )

155 {

156 int menuSec;

157

158 printf( "\nSeçiminiz.\n"

159 "1 – Yazdırmak için\n"

160 " \"hesaplar.txt\" isminde metin dosyası oluştur\n"

161 "2 – Hesap Güncelle \n"

162 "3 – Yeni hesap oluştur \n"

163 "4 – Hesap sil \n"

164 "5 - Çıkış \n? " );

165 scanf( "%d", &menuGir);

166 return menuGir;

167 }

453

Şekil 11.16 Banka hesap programı.

ÖZET

 Bir bilgisayarda işlenen tüm veriler bir ve sıfırlara indirgenir.

 Bilgisayardaki en küçük veri parçası 1 ya da 0 değerini alabilir.Böyle bir veri

parçasına bit (iki değerden birini alabilen rakam anlamına gelen binary digit teriminin

kısaltmasıdır) denir.

 Rakamlar, harfler ve özel semboller karakterler olarak bilinir. Bir bilgisayarda

programlar yazmak ve veri parçalarını temsil etmek için kullanılan tüm karakterlerin

kümesine, bilgisayarın karakter seti denir. Bilgisayarlar yalnızca 1 ve 0’ları

işleyebildiğinden, bilgisayarın karakter setindeki her karakter 1 ve 0’ların değişik

biçimde dizilişleriyle (bu dizilişe byte denir) temsil edilir.

 Bir alan karakterlerin anlam içerecek şekilde dizilişidir.

 Bir kayıt, bir çok alanın bir araya gelmesiyle oluşur.

 Dosyadan bir kaydı geri elde etmek için en az bir kayıt, kayıt anahtarı olarak

seçilmelidir. Bir kayıt anahtarı, bir kaydın bir kişiye ya da varlığa ait olduğunu

belirler.

 Bir dosyadaki kayıtları organize etmenin en popüler yolu, istenen veriye ulaşılana dek

tüm kayıtların ard arda erişildiği sıralı erişimli dosyalardır.

 İlgili dosyaların topluluğuna genellikle veri tabanı denir. Veri tabanı oluşturmak ve

veri tabanını yönetmek için kullanılan programlara veri tabanı yönetim sistemleri

(Data Base Management System) denir.

 C, her dosyayı basit olarak bitlerin ard arda geldiği bir akış olarak görür .

 Program çalışmaya başladığında,üç dosya ve bu dosyalarla ilişkili akışlar; standart

giriş (standart input), standart çıkış (standart output) ve standart hata (standart error)

otomatik olarak açılır.

 Standart giriş, standart çıkış ve standart hataya atanan dosya göstericileri stdin,stdout

ve stderr göstericileridir.

 fgetc fonksiyonu, belirlenen dosyadan bir karakter okur. fputs belirlenen dosyaya bir

karakter yazdırır.

 fgets fonksiyonu belirlenen dosyadan bir satır okur. fputs fonksiyonu belirlenen

dosyaya bir satır yazdırır.

 FILE, stdio.h öncü dosyasında tanımlanmış bir yapı tipidir. Programcı, dosyaları

kullanabilmek için bu yapının özelliklerini bilmelidir. Bir dosya açıldığında, dosyanın

FILE yapısını gösteren bir gösterici döndürülür.

 fopen fonksiyonu iki argüman alır (bir dosya ismi ve dosya açma modu) ve dosyayı

açar. Eğer varolan bir dosya açılırsa, dosyanın içindekiler hiçbir uyarı yapılmadan

silinir. Eğer daha önceden var olmayan bir dosya yazma yapmak için açılırsa, fopen o

dosyayı yaratır.

 feof fonksiyonu, bir dosya için dosya sonu belirtecinin elde edilip edilmediğine karar

verir.

 fprintf fonksiyonu ile printf fonksiyonu, fprintf fonksiyonunun argüman olarak

verinin yazılacağı dosyayı gösteren bir gösterici alması haricinde eşdeğerdir.

 fclose fonksiyonu argümanı ile gösterilen dosyayı kapatır.

 Bir dosyayı yaratmak için ya da dosyaya yazmadan önce dosyanın tüm içeriğini

silmek için dosyayı yazma modunda (“w”) açmak gerekir. Varolan bir dosyadan

okuma yapmak için, dosyayı okuma modunda ( “r” ) açmak gerekir. Varolan bir

454

dosyaya kayıtlar eklemek için dosyayı ekleme modunda açmak ( “a” ) gerekir.

Dosyaya hem yazma yapmak hem de dosyadan okuma yapmak için dosyayı

güncelleyen üç moddan birinde açmak gerekir ( “r +”,”w +”,”a+” ) .”r +” modu

dosyayı, okuma ve yazma yapmak için açar. ”w +” modu, yazma ve okuma için bir

dosya yaratır. Eğer dosya daha önceden yaratılmışsa “w+” modu dosyayı açar ve

önceki tüm içerikler kaybolur. ”a+” modu, dosyayı okuma ve yazma yapmak için

açar. Bu modda açılan dosyalarda tüm yazma işlemleri dosyanın sonuna yapılır. Eğer

dosya daha önceden yoksa, yaratılır.

 fscanf fonksiyonu, fscanf’in okuma yapılacak dosyayı gösteren bir göstericiyi

(normalde stdin ’den farklı) argüman olarak kullanması dışında scanf fonksiyonu ile

denktir.

 rewind fonksiyonu, programın dosya pozisyon göstericisini belirlenen dosyanın

başına geri döndürmesini sağlatır.

 Rasgele erişimli dosyalar bir kayda doğrudan erişmek için kullanılır.

 Rasgele erişimi sağlamak için veriler sabit uzunluklardaki kayıtlar içinde tutulur.

Rasgele erişimli bir dosyadaki tüm kayıtlar sabit uzunluğa sahip olduğundan

bilgisayar dosyanın başlangıcına göre kaydın konumunu hızlıca (kayıt anahtarının bir

fonksiyonu olarak) bulabilir.

 Rasgele erişimli dosyalara, dosyadaki diğer verilere zarar vermeden yeni veriler

eklenebilir. Daha önceden depolanmış sabit uzunluktaki kayıtlar tüm dosyayı yeniden

yazmaya gerek kalmadan güncellenebilir ya da silinebilir.

 fwrite fonksiyonu, bir veri bloğunu ( belli sayıdaki byte’ı) dosyaya yazar.

 sizeof operatörü, operandının boyutunu byte olarak döndüren derleme zamanlı bir

operatördür.

 fseek fonksiyonu, aramanın başlatıldığı konumuna bağlı olarak, dosya pozisyon

göstericisini dosyada istenen konuma taşır. Arama şu üç konumdan birinden

başlayabilir: SEEK\_SET, SEEK\_CUR ve SEEK\_END. SEEK\_SET aramanın

dosyanın başından başlayacağını; SEEK\_CUR aramanın dosyadaki o anda bulunulan

konumdan başlayacağını ve SEEK\_END aramanın dosya sonundan başlayacağını

belirtir

 fread bir dosyadan veri bloğu ( belli sayıda byte ) okur.

ÇEVRİLEN TERİMLER

binary digit.......................... ikili basamak

character field..................... karakter alanı

data hierarchy..................... veri hiyerarşisi

database............................. veritabanı

database management system veritabanı yönetim sistemi

displacement..... yerdeğiştirme

end-of-file.............. dosya sonu

end-of-file................. dosya sonu belirteci

field......................... alan

file open mode................... dosya açma modu

file pointer.......................... dosya göstericisi

file position pointer............ dosya pozisyon göstericisi

formatted input/output......... biçimlendirilmiş giriş/çıkış

random access.................. rasgele erişim

random access file..................... rasgele erişimli dosya

record.......................... kayıt

455

record key kayıt anahtarı

sequential access file......... sıralı erişimli dosya

GENEL PROGRAMLAMA HATALARI

11.1 Kullanıcı dosyanın içeriğini korumak isterken,var olan bir dosyayı yazma yapmak(“w”)

için açmak. Bu durumda hiçbir uyarı yapılmadan dosyanın içeriği kaybolur.

11.2 Program içinde kullanmadan önce dosyayı açmayı unutmak.

11.3 Bir dosyayı belirtmek için yanlış dosya göstericisini kullanmak.

11.4 Var olmayan bir dosyayı okuma yapmak için açmak.

11.5 Dosyaya uygun erişim hakkı verilmeden dosyayı okuma ya da yazma yapmak için

açmak (Bu işletim sistemine bağlıdır)

11.6 Yeterli disk alanı olmadan dosyayı yazma yapmak için açmak.

11.7 Bir dosyayı yanlış bir dosya modu ile açmak yıkıcı hatalara yol açabilir. Örneğin,

güncelleme modu(“r +”) ile açılması gereken bir dosyayı, yazma modunda(“w +”)

açmak dosyanın bütün içeriğinin silinmesine sebep olur.

İYİ PROGRAMLAMA ALIŞTIRMALARI

11.1 Bir programdan dosya işleme fonksiyonlarına yapılan çağrıların doğru dosya

göstericisini içerdiğinden emin olun.

11.2 Programın dosyayı yeniden kullanmayacağını öğrendikten sonra, mümkün olduğunca

çabuk dosyayı kapatmak

11.3 Eğer dosyanın içeriği değiştirilmeyecekse, dosyayı yalnızca okuma modunda açmak .Bu

dosya içeriğinin istemsiz olarak değiştirilmesini engeller.Bu, en az yetki prensibinin

başka bir örneğidir

PERFORMANS İPUÇLARI

11.1 Bir dosyayı kapatmak, diğer kullanıcıların ya da programların beklediği

kaynakları serbest bırakır.

11.2 Çoğu programcı, yanlış bir biçimde, sizeof’un bir fonksiyon olduğunu ve sizeof

kullanmanın çalışma zamanında fazladan bir fonksiyon çağrısı gerektireceğini

düşünür.Bu şekilde bir durum yoktur çünkü sizeof derleme zamanlı bir

operatördür.

TAŞINABİLİRLİK İPUÇLARI

11.1 FILE yapısı işletim sistemine bağımlıdır.(her sistemin dosyaları ele alışlarına göre yapı

elemanları değişiklik gösterebilir.)

ÇÖZÜMLÜ ALIŞTIRMALAR

11.1 Aşağıdaki boşlukları doldurunuz

a) En nihayetinde, bütün veriler bilgisayar tarafından işlenmek üzere \_\_\_\_\_\_ ve

\_\_\_\_’lere indirgenirler.

b) Bilgisayar tarafından işlenebilecek en küçük veri parçasına \_\_\_\_\_\_ denir.

456

c) \_\_\_\_\_\_, birbiriyle ilgisi olan kayıtlar topluluğudur.

d) Rakamlar, harfler ve semboller \_\_\_\_\_\_ olarak isimlendirilir.

e) \_\_\_\_\_\_ ,birbiriyle ilgisi olan dosyalar topluluğudur.

f) \_\_\_\_\_\_ fonksiyonu, bir dosyayı kapatır.

g) \_\_\_\_\_\_ ifadesi, scanf komutunun stdin’den veri okumasına benzer bir biçimde, bir

dosyadan veri okur.

h) \_\_\_\_\_\_ fonksiyonu, belli bir dosyadan bir karakter okur.

i) \_\_\_\_\_\_ fonksiyonu, belli bir dosyadan giriş tuşuna basılıncaya kadar yazılmış metni

okur.

j) \_\_\_\_\_\_ fonksiyonu, bir dosya açar.

k) \_\_\_\_\_\_ fonksiyonu, rasgele erişim uygulamalarında bir dosyadan veri okumada

kullanılır.

l) \_\_\_\_\_\_ fonksiyonu dosya pozisyon göstericisini dosya içinde istenen yere taşır.

11.2 Aşağıdakilerin hangilerinin doğru hangilerinin yanlış olduğuna karar veriniz (Yanlış

olanların neden yanlış olduğunu açıklayınız.):

a) fscanf fonksiyonu, standart girişten veri okumak için kullanılamaz.

b) Standart giriş, standart çıkış ve standart hata akışları açmak için programcı fopen

kullanmalıdır.

c) Bir dosyayı kapatmak için programda fclose kullanılmalıdır.

d) Sıralı dosyalarda, eğer dosya pozisyon göstericisi dosyanın başlangıcını

göstermiyorsa, başlangıcı göstermesi için dosya kapatılmalı ve tekrar açılmalıdır.

e) fprintf standart çıkışa yazabilir.

f) Sıralı erişimli dosyaların güncellenmesi, başka bir verinin üzerine yazma yapılmadan

gerçeklenebilir.

g) Rasgele erişimli dosyalarda belli bir kaydı bulmak için bütün kayıtların gözden

geçirilmesine gerek yoktur.

h) Rasgele erişimli dosyalarda kayıtların boyutları aynı değildir.

i) fseek fonksiyonu sadece dosyanın başını bulabilir.

11.3 Aşağıdakileri gerçekleştirecek birer ifade yazınız. Bütün bu ifadelerin aynı program

içersinde olacağını kabul ediniz.

a) “eskiAnakayit.dat” dosyasını okuma işlemi için açan ve döndürülen dosya

göstericisini ePtr’ye atayan ifadeyi yazınız.

b) “kayitlar.dat” dosyasını okuma işlemi için açan ve döndürülen dosya gösterisini

kPtr’ ye atayan ifadeyi yazınız.

c) “yeniAnakayit.dat” dosyasını yazma işlemi için açan ve döndürülen dosya

göstericisini yPtr’ye atayan ifadeyi yazınız.

d) “eskiAnakayit.dat” dosyasından bir kayıt okuyan ifadeyi yazınız. Kayıt,

hesapNum tamsayısı, isim stringi ve suankiBakiye ondalıklı sayısından

oluşmalıdır.

e) “kayitlar.dat” dosyasından bir kayıt okuyan ifadeyi yazınız. Kayıt hesapNum

tamsayısını ve dolar ondalıklı sayısını içermeli.

f) “yeniAnakayit.dat” dosyasına bir kayıt yazan ifadeyi yazınız. Kayıt, HesapNum

tamsayısını, isim stringini ve suankiBakiye ondalıklı sayısını içermeli

11.4 Aşağıdaki kod parçalarındaki hataları bulunuz ve nasıl düzeltileceklerini açıklayınız.

a) fPtr(“odemeler.dat”) ile belirtilen dosya açılmamıştır.

b) fprintf(fPtr, “%d%s%s\n”, hesap, sirket, miktar);

457

c) open(“al.dat”, “r+”);

d) Aşağıdaki ifade, “odemeler.dat” dosyasından bir kayıt okumalıdır. odePtr bu

dosyayı göstermekte ve alPtr, “al.dat” dosyasını göstermektedir.

fscanf(“alPtr, “%d%s%d\n”, &hesap, sirket, &miktar);

e) “araclar.dat” dosyası, eski verilerin silinmeden yeni verilerin eklenmesi için

açılmalıdır.

if ((arPtr = fopen(“araclar.dat”, “w”)) != NULL)

f) “kurslar.dat” dosyası içeriği değiştirilmeden sadece ekleme yapılması için

açılmalıdır.

if ((kursPtr = fopen(“kurslar.dat”, “w+”)) != NULL)

ÇÖZÜMLER

11.1 a) 1, 0 b) Bit c) Dosya d) Karakter e) Veri tabanı f) fclose g) fscanf h) getc ya da

fgetc i) fgets j) fopen k) fread l) fseek

11.2

a) Yanlış. fscanf fonksiyonu, fscanf çağrısında standart giriş akışının göstericisi

kullanılarak standart girişten okuma yapmak için kullanılabilir.

b) Yanlış. Bu üç akış, C programı çalıştırdığında otomatik olarak çalıştırılır.

c) Yanlış. Bu dosyalar programın çalışması bittiğinde otomatik olarak kapatılacaktır.

Her dosya ayrı olarak fclose ile kapatılabilir.

d) Yanlış. rewind fonksiyonu dosya pozisyon göstericisinin pozisyonunu dosyanın

başını gösterek şekilde değiştirmek için kullanılabilir.

e) Doğru

f) Yanlış. Çoğu durumda sıralı dosya kayıtları, aynı uzunlukta değildir ancak bir kaydı

güncellerken diğerinin üzerine yazılmasına yol açılabilir.

g) Doğru

h) Yanlış. Rasgele erişimli dosyalarda, kayıtlar aynı uzunluktadırlar.

i) Yanlış. Dosya pozisyon göstericisinin dosya içindeki o andaki konumuna göre,

dosyanın başından itibaren ve dosyanın sonundan itibaren arama yapmak

mümkündür.

11.3

a) ePtr = fopen(“eskiAnakayit.dat”, “r”);

b) kPtr = fopen(“kayitlar.dat”, “r”);

c) yPtr = fopen(“yeniAnakayit.dat”, “w”);

d) fscanf(ePtr, “%d%s%f”, &hesapNum, isim, &suankiBakiye);

e) fscanf(kPtr, “%d%f”, &hesapNum, &dolar);

f) fprintf(yPtr, “%d%s%.2f”, hesapNum, isim, suankiBakiye);

11.4

a) Hata: “odemeler.dat” dosyası, dosya göstericisi kullanılarak açılmamıştır.

Düzeltme: “odemeler.dat” dosyasının yazmaya veya okunmaya açılması için, fopen

fonksiyonunun kullanılması gerekmektedir.

b) Hata: open fonksiyonu bir ANSI C fonksiyonu değildir.

Düzeltme: fopen fonksiyonunu kullanın.

c) Hata: fscan ifadesi, “odemeler.dat” dosyası için yanlış dosya göstericisini

kullanmaktadır.

458

Düzeltme: “odemeler.dat” dosyasının odePtr dosya göstericisi ile gösterilmesi.

d) Hata: Dosya (“w”) ile açıldığı için, dosyanın önceki içeriği silinmiştir.

Düzeltme: Dosyaya yeni kayıtların eklenmesi için (“r+”) ya da (“a”) ile açılması

gerekmektedir. .

e) Hata: “kurslar.dat” dosyası güncelleme için açılmıştır ama “w+” modu kullanıldığı

için eski içeriği silinmiştir.

Düzeltme: Dosyanın“a” modunda açılması.

ALIŞTIRMALAR

11.5 Aşağıdaki boşlukları doldurunuz.

a) Bilgisayarlar, yüksek miktardaki verileri saklamak için \_\_\_\_\_\_ gibi ikincil depolama

aygıtlarını kullanırlar.

b) \_\_\_\_\_\_ bir çok alandan oluşur.

c) Rakamları, harfleri ve boşlukları içeren alana \_\_\_\_\_\_ alanı denir.

d) Bir dosyadan belli bazı kayıtları almayı kolaylaştırmak için her kayıttaki bir alan

\_\_\_\_\_\_ olarak seçilir.

e) Bilgisayar sistemlerinde saklanan bilgilerin büyük kısmı \_\_\_\_\_\_ dosyalarda saklanır.

f) Bir anlamı olan ilgili karakterlerin kümesine\_\_\_\_\_\_\_\_\_denir.

g) Bir C programı çalışmaya başladığında, otomatik olarak açılan üç dosyayı gösteren

dosya göstericileri; \_\_\_\_\_\_,\_\_\_\_\_\_ ve \_\_\_\_\_\_ dir.

h) \_\_\_\_\_\_ fonksiyonu, belli bir dosyaya bir karakter yazar.

i) \_\_\_\_\_\_ fonksiyonu, giriş tuşuna basılıncaya kadar alınan metni, dosyaya yazar.

j) \_\_\_\_\_\_ fonksiyonu, genellikle rasgele erişimli dosyalara veri yazar.

k) \_\_\_\_\_\_ fonksiyonu, dosya pozisyon göstericisinin pozisyonunu dosyanın başına

taşır.

11.6 Aşağıdakilerin hangilerinin doğru, hangilerinin yanlış olduğuna karar veriniz(Yanlış

olanların neden yanlış olduğunu açıklayınız)

a) Bilgisayarların işlettiği fonksiyonlar, sıfır ve birlerin yönetimini içerir.

b) İnsanlar, karakterleri ve alanları yönetmek yerine, bitleri yönetmeyi tercih ederler.

c) İnsanlar, programları ve verileri karakterler olarak kullanırlar, bilgisayarlar ise bu

karakterlerden sıfır ve bir grupları oluştururlar ve bu grupları yönetirler.

d) Birisinin posta kodu, sayısal bir alana örnektir.

e) Birisinin sokak adresi, bilgisayar uygulamalarında alfabetik alanlar olarak kabul

edilirler.

f) Bilgisayar tarafından işlenen veri elemanları, biz alanlardan karakterlere, karakterlerden

bitlere ilerledikçe, veri elemanlarının daha büyük ve daha kompleks hale geleceği bir

veri hiyerarşisi oluştururlar.

g) Bir kayıt anahtarı, bir kaydın hangi alana ait olduğunu belirtir.

h) Bir çok organizasyon, verileri bir dosyada saklayarak bilgisayarın işlem yapmasını

kolaylaştırır.

i) Dosyalar, C programlarında hep isimleriyle kullanılırlar.

j) Bir program bir dosya oluşturduğunda, bu dosya bilgisayar tarafından daha sonra

kullanılmak üzere hafızada tutulur.

11.7 Alıştırma 11.3’ de kullanıcıya her şık için birer ifade yazdırılmıştı. Gerçekte bu ifadeler,

önemli bir dosya işleme programı türünün temelini oluşturur. Bu tür, dosya eşleme

programıdır. Mesleki veri işleme uygulamalarında, genellikle her bir sistemde bir çok dosya

kullanılır. Örneğin, hesap kayıtları tutan bir sitemde, bir ana dosya, müşteriler hakkında

459

detaylı bilgi içerir. Bu bilgiler, müşterinin adı, adresi, telefon numarası, bakiyesi, kredi limiti,

kontrat anlaşmaları ve müşterinin daha önce yaptığı harcamalar ve ödemeler için bir özet

içerir.

Para alış verişi (satışlar, ödemeler vs.) başladığında bunlar bir dosyaya kayıt edilirler. Her iş

süreci sonunda (bazı şirketler için bir ay, bazıları için bir hafta ve bazı durumlarda bir gün) bu

alış verişler (Alıştırma 11.3’ de kayitlar.dat olarak adlandırılmıştır.) ana dosyaya(Alıştırma

11.3 de eskiAnakayit.dat olarak adlandırılmıştır) kayıt edilirler, böylece veriler güncellenmiş

olur. Bu güncelleme her çalıştırıldığında ana dosyanın (“yeniAnakayit.dat”) isminde

kaydedilir. Bu dosya ise, bir sonraki güncellemede tekrar kullanılır.

Dosya eşleme programları mutlaka, tek dosya programlarının çözemediği problemleri

çözmelidirler. Örneğin, her zaman bir eşleme olmayabilir. Bir müşteri, bir iş süreci içerisinde

hiç bir harcama veya ödeme yapmamış olabilir. Böylece alış veriş dosyasında bu müşteri için

bir kayıt yer almaz. Diğer taraftan, başka bir müşteri ise bu şirketle yeni iş yapmaya başlamış

ve şirket bu müşteriyi ana dosyaya kayıt etmek için şans bulamamış olabilir.

Alıştırma 11.3 de yazdığınız ifadeleri bir dosya eşleme programına temel olarak alın ve bütün

programı yazınız. Her dosyada kayıt anahtarı olarak hesap numarasını kullanınız. Her

dosyanın sıralı dosya türünde ve hesap numaralarının artarak sıralandığını kabul ediniz.

Bir eşleşme meydana geldiğinde (aynı hesap numarası hem ana dosyada hem de alış veriş

dosyasında görüldüğünde), alış veriş dosyasındaki para miktarını ana dosyadaki bakiyeye

ekleyiniz ve dosyayı “yeniAnakayit.dat” olarak saklayınız (Harcamaların eksi miktarlarla,

ödemelerin ise artı miktarlarla gösterildiğini kabul edin.). Belirli bir hesaba ana kayıtta

rastlanıyor ama alış veriş kaydında rastlanmıyorsa , sadece ana kaydı “yeniAnakayit.dat”

olarak saklayın. Eğer belirli bir hesaba alış veriş kaydında rastlanırken ana kayıtta

rastlanmıyorsa, “Eşlenmemiş alış veriş kaydı....” (mesajın geri kalanına hesap numarasını

yazdırınız) şekline ekrana bir mesaj yazdırınız.

11.8 Alıştırma 11.7’deki programı yazdıktan sonra, bir test verisi oluşturacak ve Alıştırma

11.7 deki programın çalışmasını kontrol edecek küçük bir program yazınız. Aşağıdaki örnek

hesap verisini kullanın

Ana Dosya:

Hesap numarası İsim Bakiye

100 Metin Zavrak 348.17

300 Hüseyin Karaca 27.19

500 Ekrem Aksoy 0.00

700 Gül Erkal -14.22

Alışveriş Dosyası:

Hesap numarası Dolar miktarı

100 27.14

300 62.11

400 100.56

900 82.17

460

11.9 Alıştırma 11.7’ deki programı, Alıştırma 11.8 deki örnek test verisini kullanacak şekilde

çalıştırın. Kısım 11.7’ deki listeleme programı yardımıyla yeni ana kayıt dosyasını yazdırın ve

sonuçları dikkatlice kontrol edin.

11.10 Aynı kayıt numarasıyla birden fazla alış veriş kaydı olması mümkündür (genellikle

olur). Bu, belli bir müşteri bir iş süreci içerisinde bir kaç kez harcama veya ödeme yaptığında

gerçekleşir. Alıştırma 11.7’de yazdığınız programı aynı kayıt anahtarıyla bir kaç kez

karşılaşılabilme olasılığını destekleyecek şekilde tekrar yazınız. Alıştırma 11.8 ‘deki test

verisini ise aşağıdaki kayıtları da içerecek şekilde değiştiriniz.

Hesap Numarası Dolar miktarı

300 83.89

700 80.78

700 1.53

11.11 Aşağıdakileri gerçekleştirecek ifadeleri yazınız.

struct kisi {

char soyisim[15];

char isim[15];

char yas[2];

};

yapısının tanımlanmış olduğunu ve dosyanın yazma için açılmış olduğunu kabul edin.

a) “isimyas.dat” dosyasını 100 kayıt içerecek şekilde ve bu kayıtların ilk içeriği, soyisim

= “atanmadi” , ilkisim = “ ” ve yas = “0” olacak şekilde ilk değerlerini atayınız.

b) 10 adet soyisim, isim ve yaş girdisi yaptırarak bunları dosyaya yazınız.

c) Herhangi bir kaydı güncelleyiniz (değiştiriniz). Eğer bu kayıt daha önceden

yapılmamışsa “Bu kayıt yok” yazdırınız.

d) Herhangi bir kaydı a şıkkındaki ilk değerleri alacak şekilde siliniz.

11.12 Bir hırdavat dükkanının sahibisiniz ve bir envanter defteri tutmanız gerekmektedir.

Defterde hangi ürünlerin elinizde bulunduğunu, ne kadar bulunduğunu ve her birinin

ücretinin saklamanız gerekmektedir.”hirdavat.dat” dosyasını 100 boş kayıt içerecek

şekilde oluşturan ve ürünleriniz hakkında gerekli verileri girebilmenizi, ürünlerinizi

listeleyebilmenizi, her hangi bir ürünü silebilmenizi veya güncelleyebilmenizi sağlayan

bir program yazınız. Ürün numarası kayıt numarası olabilir. Dosyanızı oluşturmaya

başlamak için aşağıdaki bilgileri kullanın.

Kayıt # Ürün Adı Miktar Ücret

3 Elektrikli testere 7 57.98

17 Çekiç 76 11.99

24 Testere 21 11.00

39 Çim biçme makinesi 3 79.50

56 Güçlü testere 18 99.99

68 Tornavida 106 6.99

77 Balyoz 11 21.50

83 Burgu 34 7.50

461

11.13 Telefon Numaralarından Kelime Oluşturma. Standart telefonlar 0’dan 9’a kadar

rakamları içerir. 2’den 9’a kadar olan rakamlar sayesinde harflerde aşağıdaki şekilde

olduğu gibi ifade edilebilir.

Rakam Harf

2 A B C

3 D E F

4 G H I

5 J K L

6 M N O

7 P R S

8 T U V

9 W X Y

Bir çok insan telefon numaralarını ezberlemekte güçlük çeker. Bu yüzden numaraların

harflerle olan ilgisinden faydalanırlar. Örneğin telefon numarası 487-3946 olan bir kişi

yukarıdaki tablo yardımıyla “HUSEYIN” kelimesine ulaşır.

Şirketler genellikle, müşterilerinin kolaylıkla hatırlayabileceği telefon numaralarını kullanmak

isterler. Eğer bir şirket, müşterilerine bu yolla telefonlarının kolaylıkla hatırlanabilmesini

sağlarlarsa telefonla daha çok aranacakları kesindir.

Her yedi harfli kelime tam olarak bir telefon numarasını göstermektedir. Örneğin, çanta satan

bir dükkan, 226-8224 (CANTACI) telefon numarasıyla, kolayca hatırlanabilir bir telefon

numarasına sahip olabilir. Yada bir kırtasiye, 548-2724 (KITAPCI) telefon numarasını

kullanmak ister.

7 rakamlık bir telefon numarasını alan ve bu rakamların gösterebileceği bütün kelimeleri bir

dosyaya yazan bir program yazınız. Bu şekilde 2187(üçün yedinci kuvveti) kelime vardır.

Programınız 0 veya l ile başlayan telefon numaralarını içermesin.

11.14 Eğer bilgisayarınızda bir sözlüğünüz varsa Alıştırma 11.13 de yaptığınız programı,

bulduğu kelimeleri sözlükten kontrol edecek şekilde değiştirin. Yedi adet rakamın harflerle

ifade edilmesinde bu rakamlar iki kelimenin birleşimini içerebilir.(867-8229 telefon numarası

“TOSTCAY” a karşılık gelir.)

11.15 Şekil 8.15 deki programı, getchar ve puts fonksiyonları yerine fgetc ve fputs

fonksiyonlarını kullanacak şekilde değiştiriniz. Programınız kullanıcıya, standart girişten

okuma ya da standart çıkışa yazma veya dosyadan okuma ya da dosyaya yazma şeklinde

seçeneklerini sunmalıdır. Eğer kullanıcı ikinci seçeneği girerse, dosya isimlerini de girmelidir.

11.16 sizeof operatörünü kullanarak bilgisayarınızda kullandığınız veri tiplerinin

uzunluklarını byte cinsinden bulan bir program yazınız. Sonuçları “veriboyutu.dat” isminde

bir dosyaya yazdırın. Sonuçlar dosyaya aşağıdaki biçimde yazılmalıdır:

462

Veri tipi Boyut

char 1

unsigned char 1

short int 2

unsigned short int 2

int 4

unsigned int 4

long int 4

unsigned long int 4

float 4

double 8

long double 16

Not: Sizin bilgisayarınızdaki ver tiplerinin boyutları yukarıdaki gibi olmayabilir.

11.17 Alıştırma 7.19’da Simpletron Makina Dilini(SMD) kullanan bir bilgisayarın yazılım

simülasyonunu yazmıştınız. Bu simülasyonda bir SMD programı çalıştırmak istediğinizde

mutalaka bu programı simülasyon programına klavyeden girmek zorundaydınız. Eğer bu

işlem sırasında bir hata yaparsanız, simülasyon programını tekrar çalıştırarak programı tekrar

girmek zorundasınız. Hataları en aza indirmek ve zamandan kazanmak için, SMD

programının klavye yerine bir dosyadan okunabilmesi daha iyi olacaktır.

a) Alıştırma 7.19’da yazdığınız simülasyon programını değiştirerek SMD

programının kullanıcı tarafından belirtilen bir dosayadan alınmasını sağlayınız.

b) Simpletron çalıştıktan sonra, yazmaçların ( register ) ve hafızasının içeriğini ekrana

yazar. Bu çıktıyı ekran yerine bir dosyaya yazdırmak daha iyi olacağı için

simülasyon programınızda gerekli değişiklikleri yapınız

463

VERİ YAPILARI

AMAÇLAR

 Veri nesneleri için dinamik olarak hafıza tahsis etmek ve tahsis edilen alanı

boşaltabilmek

 Göstericiler, kendine dönüşlü yapılar ve yineleme kullanarak bağlı veri yapıları

oluşturabilmek.

 Bağlı listeler, sıralar,yığınlar ve ikili ağaçlar oluşturabilmek ve yönetebilmek.

 Bağlı veri yapılarının bir çok önemli uygulamasını anlamak.

BAŞLIKLAR

12.1 GİRİŞ

12.2 KENDİNE DÖNÜŞLÜ YAPILAR

12.3 DİNAMİK HAFIZA TAHSİSİ

12.4 BAĞLI LİSTELER

12.5 YIĞINLAR

12.6 SIRALAR

12.7 AĞAÇLAR

12.1 GİRİŞ

Şimdiye kadar tek belirteçli diziler,iki boyutlu diziler ve structlar gibi sabit boyutta olan veri

yapılarını inceledik.Bu ünite, çalışma zamanında boyutları büyüyebilen ve küçülebilen

dinamik veri yapılarını tanıtmaktadır.Bağlı listeler , bir satırda dizilmiş veri nesnelerinin

birlikleridir.Bir bağlı listedeki her noktaya ekleme ve her noktadan çıkarma yapılabilir.

Yığınlar ,derleyicilerde ve işletim sistemlerinde önemlidir.Yığınlara ekleme ve yığınlardan

çıkarma, yalnızca yığının iki ucundan birinde yapılabilir.Sıralar ,bekleme satırlarını temsil

eder.Sıralara ekleme yalnızca arkadan (kuyruk olarak da bilinir) ve sıralardan çıkarma

yalnızca önden (baş olarak da bilinir) yapılır. İkili ağaçlar , verinin çok hızlı aranması ve

dizilmesini,veri nesnelerinin kopyalarının elenmesini, dosya sistemi dizinlerinin temsil

edilmesini ve deyimlerin makine diline çevrilmesini sağlar.Bu veri yapılarından her birinin

oldukça ilginç uygulamaları vardır.

Her veri yapısı tipini inceleyecek ve bu veri yapılarını oluşturan ve yöneten programlar

sunacağız.Kitabın diğer kısmında( C++ ‘a ve nesneye yönelik programlamaya giriş) veri

soyutlamayı (veri abstraction) çalışacağız. Bu teknik bize, bu veri yapılarını oldukça farklı bir

şekilde kullanarak daha kolay elde edilebilir ve özellikle de daha kolay yeniden kullanılabilir

yazılımlar tasarlamamızı sağlayacaktır.

Bu ünite oldukça zorlayıcı bir kısımdır.Programlar oldukça güçlüdür ve daha önceki

ünitelerde öğrenilen çoğu konuyu içermektedir.Programlar, ağırlıkla gösterici yönetimine

dayanmaktadır.Bu konu, çoğu kişinin C’ de en zor başlıklar arasında yer aldığını düşündüğü

bir konudur.Bu ünite, daha ileri düzeydeki programlama kurslarında kullanabileceğiniz

oldukça pratik programlarla donatılmıştır.Ayrıca bu ünite, veri yapılarının pratik

uygulamalarını vurgulayan oldukça zengin alıştırmalar içermektedir.

464

Umarız “Kendi Derleyicinizi Geliştirmek” adlı kısımdaki proje üzerinde çalışırsınız.C

programlarınızı çalıştırabilmeniz için, programlarınızı makine diline dönüştüren bir derleyici

kullanıyorsunuz.Bu projede , kendi derleyicinizi oluşturacaksınız.Derleyiciniz, BASIC dilinin

ilk versiyonlarındakine benzer, basit ama güçlü komutlarla yazılmış ifadelerden oluşan bir

dosyayı okuyacaktır.Daha sonra, bu ifadeleri Simpletron Makine Dili (SMD) komutlarına

dönüştürecektir.SMD dilini, 7.ünitedeki “Kendi bilgisayarınızı geliştirmek” kısmında

öğrenmiştiniz.Simpletron Gerçekleyici programınız, derleyiciniz tarafından üretilen SMD

programını çalıştıracaktır.Bu proje, bu kursta öğrendiğiniz bir çok konuyu uygulayabilmeniz

için muhteşem bir fırsat vermektedir.Bu özel kısım, sizi yüksek seviyeli dilin tüm tarifleri

üzerinde yürütecek ve her tipte yüksek seviyeli dili makine diline çevirmenizi sağlayacak

algoritmaları tanımlayacaktır.Eğer zor işlerle uğraşmaktan hoşlanıyorsanız, alıştırmalardaki

derleyici ve Simpletron Gerçekleyici kısımları üzerinde uğraşmanızı tavsiye ediyoruz.

12.2 KENDİNE DÖNÜŞLÜ YAPILAR

Kendine dönüşlü bir yapı , yapı tipiyle aynı tipte bir yapıyı gösteren bir gösterici elemanına

sahiptir.Örneğin;

struct dugum{

int veri;

struct dugum \*yeniPtr;

};

tanımı, struct dugum tipini tanımlamaktadır.struct dugum tipindeki yapının iki elemanı

vardır:tamsayı elemanı veri ve bir gösterici olan yeniPtr elemanı. yeniPtr elemanı, struct

dugum (burada tanımlanan veri tipiyle aynı tipte bir yapı , bu sebepten de kendine dönüşlü

terimi kullanılır) tipindeki bir yapıyı göstermektedir. yeniPtr elemanı, bağ(link) olarak

bilinir.yeniPtr , struct dugum tipindeki bir yapıyı yine aynı tipteki başka bir yapıya

bağlamak için kullanılabilir. Kendine dönüşlü yapılar birbirine bağlanarak listeler, sıralar,

yığınlar ve ağaçlar gibi kullanışlı veri tipleri oluşturmakta kullanılabilirler.Şekil 12.1, bir liste

oluşturmak için bağlanmış kendine dönüşlü iki yapıyı tasvir etmektedir.İkinci kendine

dönüşlü yapının bağ elemanında, bağın başka bir yapıyı göstermediğini belirtmek için bir ters

bölü işaretinin( NULL ’u temsil etmektedir) kullanıldığına dikkat ediniz.Ters bölü işareti

yalnızca gösterimlerde kullanılmaktadır,C’ deki ters bölü karakteriyle bir alakası

yoktur.NULL gösterici(null karakterin string sonunu göstermesi gibi) veri yapısının sonunu

gösterir.

Genel Programlama Hataları 12.1

Bir listenin son bağına NULL yerleştirmemek.

12.3 DİNAMİK HAFIZA TAHSİSİ

Dinamik veri yapıları yaratmak ve yönetmek için dinamik hafıza tahsisi yapmak

gerekir.Dinamik hafıza tahsisi programın çalışma zamanında yeni düğümleri tutabilmesi için

daha fazla hafıza alanı elde etme yeteneğidir.Dinamik hafıza tahsisinin limiti bilgisayardaki

kullanılabilir fiziksel hafıza miktarı kadar ya da sanal hafıza sistemlerinde sanal hafıza kadar

büyük olabilir.Sıklıkla, limitler daha küçüktür çünkü kullanılabilir hafıza kullanıcılar arasında

paylaştırılmalıdır.

465

Şekil 12.1 İki kendine dönüşlü yapı birbirine bağlanmıştır.

malloc ve free fonksiyonları ile sizeof operatörü, dinamik hafıza tahsisi için gereklidir.malloc

fonksiyonu, tahsis edilecek byte sayısını argüman olarak alır ve tahsis edilen alanı gösteren

void\* tipte bir gösterici döndürür.void\* bir gösterici her tipteki göstericiye atanabilir.malloc

fonksiyonu normalde sizeof operatörü ile kullanılır.Örneğin,

yeniPtr = malloc ( sizeof ( struct dugum ) )

ifadesi, struct dugum tipindeki yapının boyutunu byte olarak hesaplar,sizeof(struct dugum)

byte’lık yeni bir hafıza alanı tahsis eder ve yeniPtr değişkene tahsis edilen alan için bir

göstericiyi depolar. Eğer uygun hafıza yoksa, malloc NULL bir gösterici döndürür.

free fonksiyonu tahsis edilen alanı serbest bırakır(hafıza sisteme geri döndürülür,böylece

ileride yeniden tahsis edilebilir)Az önceki malloc çağrısı ile dinamik olarak tahsis edilen

hafıza alanını serbest bırakmak için

free ( yeniPtr ) ;

ifadesi kullanılır.

İlerleyen kısımlar listeleri, yığınları, sıraları ve ağaçları anlatacaktır. Bu veri yapılarının her

biri, dinamik hafıza tahsisi ve kendine dönüşlü yapılar sayesinde oluşturulur ve yönetilir.

Taşınırlık İpuçları 12.1

Bir yapının boyutu, elemanlarının boyutları toplamına eşit olmak zorunda değildir.Bunun

sebebi, çeşitli makinelerde hizalamanın farklı yapılmasıdır(10.üniteye bakınız)

Genel Programlama Hataları 12.2

Bir yapının boyutunun, elemanlarının boyutlarının toplamına eşit olduğunu düşünmek.

İyi Programlama Alıştırmaları 12.1

Bir yapının boyutuna karar vermek için sizeof operatörünü kullanmak.

İyi Programlama Alıştırmaları 12.2

malloc kullanırken,geri dönüş değerinin NULL gösterici olup olmadığına kontrol etmek.Eğer

istenen hafıza tahsis edilemezse bir hata mesajı yazdırmak.

Genel Programlama Hataları 12.3

Dinamik olarak tahsis edilen hafızaya ihtiyaç kalmadığında, tahsis edilen

hafızayı sisteme geri döndürmemek.Bu, sistemin olması gerekenden daha

erken bir zamanda hafıza sıkıntısı çekmesine sebep olur.

15 10

466

İyi Programlama Alıştırmaları 12.3

Dinamik olarak tahsis edilen hafızaya ihtiyaç kalmadığında,hafızayı sisteme anında geri

döndürmek için free kullanın.

Genel Programlama Hataları 12.4

malloc ile tahsis edilmemiş bir hafıza alanını serbest bırakmak(boşaltmak)

Genel Programlama Hataları 12.5

Serbest bırakılmış bir hafıza alanından bahsetmek ve kullanmaya çalışmak.

12.4 BAĞLI LİSTELER

Bir bağlı liste,düğüm adı verilen ve gösterici bağları sayesinde birleştirilmiş(bu sebepten bağlı

terimi kullanılmaktadır) kendine dönüşlü yapıların doğrusal bir birlikteliğidir.Bağlı bir listeye,

listenin ilk düğümünü gösteren gösterici sayesinde erişilir.Sonradan gelen düğümlere, her

düğüm içindeki gösterici elemanları sayesinde erişilir.Geleneksel olarak,son düğümdeki bağ

göstericisi, listenin sonunu belirtmek amacıyla NULL yapılır.Bağlı listelerde veri dinamik

olarak tutulur.Her düğüm ihtiyaç duyulduğunda oluşturulur.Bir düğüm, başka struct’larda

dahil olmak üzere her tipte veri içerebilir.Yığınlar ve sıralar da doğrusal veri yapılarıdır ve

göreceğimiz gibi bağlı listelerin kısıtlanmış versiyonlarıdır.Ağaçlar doğrusal olmayan veri

yapılarıdır.

Veri listeleri dizilerde tutulabilir ancak bağlı listeler bir çok avantaj sağlar.Bağlı bir liste, veri

yapısında temsil edilen veri elemanlarının sayısı bir kerede tahmin edilemediği durumlarda

uygundur.Bağlı listeler dinamiktir, bu sebepten gerektikçe listenin uzunluğu artabilir ya da

kısalabilir.Dizinin boyutu ise hafıza tahsis edildikten sonra değiştirilemez.Diziler dolabilir.

Bağlı listeler ise sistem dinamik hafıza tahsisi istemlerini karşılayamadığında dolar.

Performans İpuçları 12.1

Bir dizi, beklenen veri parçalarının sayısından daha fazla eleman içerecek biçimde

tanımlanabilir ancak bu hafızayı boşuna harcayabilir.Bağlı listeler bu durumlarda daha iyi

bir hafıza kullanımı sağlar.

Bağlı listeler, her yeni eleman uygun noktadan eklenerek sıralı bir şekle getirilebilir.

Performans İpuçları 12.2

Sıralanmış bir dizide ekleme ve silme zaman alıcı olabilir.Silinen ya da eklenen elemandan

sonraki tüm elemanlar uygun bir şekilde kaydırılmalıdır.

Performans İpuçları 12.3

Bir dizinin elemanları hafızada art arda depolanır.Bu, her dizi elemanına

ani erişime izin verir çünkü elemanın adresi, dizinin başlangıcına göre

uzaklığı hesaplanarak bulunabilir.Bağlı listeler elemanlarına bu şekilde bir

ani erişimi gerçekleştiremez.

Bağlı listelerin düğümleri normalde hafızada art arda tutulmaz.Mantık olarak , listedeki her

düğüm sanki art arda geliyormuş gibi gözükür.Şekil 12.2, birkaç düğümden oluşan bir bağlı

liste göstermektedir.

467

Şekil 12.2 Bir bağlı listenin grafik tasviri.

Performans İpuçları 12.4

Çalışma zamanında büyüyüp küçülebilen dinamik hafıza tahsisini(dizilerin yerine) kullanmak

hafızayı verimli kullanmamızı sağlar.Ancak, göstericilerin alan kapladığını ve dinamik hafıza

tahsisi yapmak için yapılacak fonksiyon çağrılarının bir yük getirebileceğini aklınızda tutun.

Şekil 12.3, (çıktısı Şekil 12.4’te gösterilmiştir.) karakterlerin listesini yönetmektedir.Program

iki seçenek sunmaktadır: 1)Listeye alfabetik sırada bir karakter eklemek(ekle fonksiyonu) ve

2)Listeden bir karakter silmek(sil fonksiyonu)

Bu program oldukça büyük ve karmaşık bir programdır.Programın detaylı bir açıklaması

yapılacaktır.Alıştırma 12.20’de, okuyucuya bir listeyi tersten yazdıran bir yinelemeli

fonksiyonu gerçeklemesi ve Alıştırma 12.21’de, bağlı bir listede belli bir veri parçasını arayan

yinelemeli bir fonksiyon yazması sorulmuştur.

1 /\* Şekil 12.3: fig12\_03.c

2 Bir liste oluşturmak ve yönetmek \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 struct listeDugumu { /\* kendine dönüşlü yapılar \*/

7 char veri;

8 struct listeDugumu \*sonrakiPtr;

9 };

10

11 typedef struct listeDugumu ListeDugumu;

12 typedef ListeDugumu \*ListeDugumuPtr;

13

14 void ekle( ListeDugumuPtr \*, char );

15 char sil( ListeDugumuPtr \*, char );

16 int bosMu( ListeDugumuPtr );

17 void listeyiYazdir( ListeDugumuPtr );

18 void menu( void );

19

20 int main( )

21 {

22 ListeDugumuPtr baslangicPtr = NULL;

23 int secim;

17 29 93

..........

468

24 char secimNo;

25

26 menu( ); /\* menüyü göster \*/

27 printf( "? " );

28 scanf( "%d", &secim );

29

30 while ( secim != 3 ) {

31

32 switch ( secim ) {

33 case 1:

34 printf( "Bir karakter giriniz: " );

35 scanf( "

\n%c", &secimNo );

36 ekle( &baslangicPtr, secimNo );

37 listeyiYazdir( baslangicPtr );

38 break;

39 case 2:

40 if ( !bosMu( baslangicPtr ) ) {

41 printf( "Silinecek karakteri giriniz: " );

42 scanf( "

\n%c", &secimNo );

43

44 if ( sil( &baslangicPtr, secimNo ) ) {

45 printf( "%c silindi.

\n", secimNo );

46 listeyiYazdir( baslangicPtr );

47

}

48 else

49 printf( "%c bulunamadı.

\

n

\n", secimNo );

50

}

51 else

52 printf( "Liste boştur.

\

n

\n" );

53

54 break;

55 default:

56 printf( "Geçersiz seçim.

\

n

\n" );

57 menu( );

58 break;

59

}

60

61 printf( "? " );

62 scanf( "%d", &secim );

63

}

64

65 printf( "Program sonlandı.

\n" );

66 return 0;

67

}

68

69 /\* menüyü yazdır \*/

70 void menu( void

)

71 {

72 printf( "Seçiminizi girin:

\n"

73 " 1 listeye eleman eklemek için .

\n"

469

74 " 2 listeden eleman silmek için.\n"

75 " 3 çıkış.\n" );

76 }

77

78 /\* Listeye sıralı biçimde yeni eleman ekle \*/

79 void ekle( ListeDugumuPtr \*sPtr, char deger )

80 {

81 ListeDugumuPtr yeniPtr, oncekiPtr, suandakiPtr;

82

83 yeniPtr = malloc( sizeof( ListeDugumu ) );

84

85 if ( yeniPtr != NULL ) { /\* boş alan var mı \*/

86 yeniPtr->veri = deger;

87 yeniPtr->sonrakiPtr = NULL;

88

89 oncekiPtr = NULL;

90 suandakiPtr = \*sPtr;

91

92 while ( suandakiPtr != NULL && deger > suandakiPtr->veri ) {

93 oncekiPtr = suandakiPtr; /\*bir sonraki düğüme ... \*/

94 suandakiPtr = suandakiPtr->sonrakiPtr; /\* ....git \*/

95 }

96

97 if ( oncekiPtr == NULL ) {

98 yeniPtr->sonrakiPtr = \*sPtr;

99 \*sPtr = yeniPtr;

100 }

101 else {

102 oncekiPtr->sonrakiPtr = yeniPtr;

103 yeniPtr->sonrakiPtr = suandakiPtr;

104 }

105 }

106 else

107 printf( "%c eklenemedi.Yetersiz hafıza.\n", deger );

108 }

109

110 /\* Bir liste elemanını silmek \*/

111 char sil( ListeDugumuPtr \*sPtr, char deger )

112 {

113 ListeDugumuPtr oncekiPtr, suandakiPtr, geciciPtr;

114

115 if ( deger == ( \*sPtr )->veri ) {

116 geciciPtr = \*sPtr;

117 \*sPtr = ( \*sPtr )->sonrakiPtr;

118 free( geciciPtr );

119 return deger;

120 }

121 else {

122 oncekiPtr = \*sPtr;

123 suandakiPtr = ( \*sPtr )->sonrakiPtr;

470

124

125 while ( suandakiPtr != NULL && suandakiPtr->veri != deger ) {

126 oncekiPtr = suandakiPtr; /\* bir sonraki düğüme... \*/

127 suandakiPtr = suandakiPtr->sonrakiPtr; /\* ... git \*/

128 }

129

130 if ( suandakiPtr != NULL ) {

131 geciciPtr = suandakiPtr;

132 oncekiPtr->sonrakiPtr = suandakiPtr->sonrakiPtr;

133 free( geciciPtr );

134 return deger;

135 }

136 }

137

138 return '\0';

139 }

140

141 /\* Eğer liste boşsa 1 döndür,değilse 0 döndür \*/

142 int bosMu( ListeDugumuPtr sPtr )

143 {

144 return sPtr == NULL;

145 }

146

147 /\* Listeyi yazdır \*/

148 void listeyiYazdir( ListeDugumuPtr suandakiPtr )

149 {

150 if ( suandakiPtr == NULL )

151 printf( "Liste boştur.\n\n" );

152 else {

153 printf( "Liste:\n" );

154

155 while ( suandakiPtr != NULL ) {

156 printf( "%c --> ", suandakiPtr->veri );

157 suandakiPtr = suandakiPtr->sonrakiPtr;

158 }

159

160 printf( "NULL\n\n" );

161 }

162 }

Şekil 12.3 Bir listeye düğümler eklemek ve listedeki düğümleri silmek

Seçiminizi girin

1 listeye eleman eklemek için

2 listeden eleman silmek için.

3 çıkış.

? 1

Bir karakter giriniz : B

471

Liste:

B-->NULL

? 1

Bir karakter giriniz : A

Liste:

A-->B-->NULL

? 1

Bir karakter giriniz: C

Liste:

A-->B-->C-->NULL

? 2

Silinecek karakteri giriniz: D

D bulunamadı

? 2

Silinecek karakteri giriniz: B

B silindi

Liste :

A-->C-->NULL

? 2

Silinecek karakteri giriniz: C

C silindi

Liste :

A-->NULL

? 2

Silinecek karakteri giriniz: A

A silindi

Liste boştur

? 4

Geçersiz seçim

Seçiminizi girin

1 listeye eleman eklemek için

2 listeden eleman silmek için.

3 çıkış.

? 3

Program sonlandı

Şekil 12.4 Şekil 12.3’teki programın çıktısı.

472

Bağlı listelerin iki temel fonksiyonu, ekle(satır 79) ve sil(satır 111) fonksiyonlarıdır. bosMu

fonksiyonu (satır 142), listeyi herhangi bir biçimde değiştirmemektedir.Aksine, listenin boş

olup olmadığına(listenin ilk düğümünü gösteren göstericinin NULL olup olmadığına) karar

vermektedir.Eğer liste boşsa 1,değilse 0 döndürülmektedir.listeyiYazdir fonksiyonu, listeyi

yazdırmaktadır.

Karakterler listeye alfabetik bir sırada eklenmektedir.ekle fonksiyonu, listenin ve eklenecek

karakterin adresini alır.Listenin adresi, listenin başlangıcına bir değer ekleneceğinde

gereklidir.Listenin adresini bilmek (yani listenin ilk düğümünü gösteren göstericiyi bilmek),

listenin referansa göre çağırma ile değiştirilebilmesine imkan sağlar.Listenin kendisi de bir

gösterici olduğundan(ilk elemanına),listenin adresini geçirmek, göstericiyi gösteren bir

gösterici oluşmasına sebep olur.Bu, oldukça karmaşık bir gösterimdir ve oldukça dikkatli

programlama yapmayı gerektirir.Listeye karakter eklemek için izlenen adımlar

şunlardır(bakınız Şekil 12.5):

1. malloc çağırarak bir düğüm oluşturulur.

Tahsis edilen hafızanın adresi yeniPtr’ ye atanır.

Eklenecek karakter yeniPtr->veri ‘ye atanır.

yeniPtr->sonrakiPtr ’ye NULL atanır.

2. oncekiPtr ’ye ilk değer olarak NULL atanır

suandakiPtr ’ye ilk değer olarak \*sPtr (listenin başlangıcını gösteren gösterici)

atanır.

oncekiPtr ve suandakiPtr ,ekleme noktasından önceki ve ekleme noktasından

sonraki düğümlerin konumlarını tutacaktır.

3. suandakiPtr NULL olmadıkça ve eklenecek değer suandakiPtr->veri ‘den büyük

oldukça,

suandakiPtr oncekiPtr’ ye atanır ve

suandakiPtr listedeki sonraki düğüme ilerletilir.Bu, değer için ekleme noktasının

konumunu belirler.

4. Eğer suandakiPtr NULL ise yeni düğüm, listedeki ilk düğüm olarak eklenir.

\*sPtr yeniPtr->sonrakiPtr ’ye atanır(yeni düğüm bağlantısı önceki ilk

düğümü gösterir)

yeniPtr \*sPtr ’ye atanır( \*sPtr ,yeni düğümü gösterir)

Aksi takdirde, eğer oncekiPtr NULL değilse,yeni düğüm içeriye yerleştirilir.

yeniPtr oncekiPtr-> sonrakiPtr ’ye atanır(önceki düğüm yeni düğümü

gösterir)

suandakiPtr yeniPtr->sonrakiPtr ’ye atanır(yeni düğüm bağı o andaki düğümü

gösterir)

473

Şekil 12.5 Bir düğümü listeye sıralı bir şekilde eklemek

İyi Programlama Alıştırmaları 12.4

Yeni bir düğümün bağ elemanına NULL atamak.Göstericiler kullanılmadan önce ilk

değerlere atanmalıdır.

Şekil 12.5 , ‘C’ karakterini içeren bir düğümün sıralı bir listeye eklenişini göstermektedir.

Şeklin a) kısmı, listeyi ve düğümü ekleme yapılmadan önce göstermektedir.Şeklin b) kısmı,

yeni düğüm eklendikten sonra oluşan sonucu göstermektedir.Yeniden atama yapılan

göstericiler kesikli çizgilerle gösterilmiştir.

sil fonksiyonu, listenin başlangıcını gösteren göstericinin adresini ve silinecek karakteri

alır.Listeden karakter silmede izlenecek adımlar şunlardır:

1. Eğer silinecek karakter listedeki ilk düğüm içinde tutulan karakter ile eşleşirse,

\*sPtr geciciPtr ’ye atanır.(geciciPtr ,ihtiyaç duyulmayan hafızayı

serbest bırakmak için kullanılacaktır)

(\*sPtr)->sonrakiPtr \*sPtr ‘ye atanır. ( \*sPtr artık listedeki ikinci düğümü

A B D

C

E

\*sPtr oncekiPtr suandakiPtr

yeniPtr

A B D

C

E

\*sPtr oncekiPtr suandakiPtr

yeniPtr

b)

a)

474

göstermektedir.)

geciciPtr ile gösterilen hafıza free ile serbest bırakılır ve silinen karakter geri

döndürülür.

2. Aksi takdirde,

oncekiPtr ’ye ilk değer olarak \*sPtr ve

suandakiPtr’ ye ilk değer olarak (\*sPtr)->sonrakiPtr atanır.

3. suandakiPtr NULL olmadıkça ve silinecek değer suandakiPtr->veri ’ye eşit

olmadıkça,

suandakiPtr oncekiPtr ’ye atanır ve

suandakiPtr->sonrakiPtr suandakiPtr ’ye atanır.Bu,silinecek karakter eğer listede

yer alıyorsa karakterin konumunu bulur.

4. Eğer suandakiPtr NULL değilse,

suandakiPtr geciciPtr ’ye atanır.

suandakiPtr->sonrakiPtr oncekiPtr->sonrakiPtr ’ye atanır.

geciciPtr ile gösterilen düğüm serbest bırakılır ve listeden silinen karakter döndürülür.

Eğer suandakiPtr NULL ise, silinecek karakterin liste içinde bulunamadığını

belirtmek için null karakter(‘\0’) döndürülür.

Şekil 12.6, bağlı bir listeden bir düğümün silinişini göstermektedir.Şeklin a) kısmı, listeyi az

önceki ekleme işleminden sonraki biçimiyle göstermektedir.Şeklin b) kısmı, oncekiPtr bağ

elemanının yeniden atanışını ve suandakiPtr’ in geciciPtr ‘ye atanmasını göstermektedir.

geciciPtr , ‘C’ yi depolamak için kullanılan hafızayı serbest bırakmak için kullanılmaktadır.

a)

b)

A B C D E

\*sPtr oncekiPtr suandakiPtr

A B D E

\*sPtr oncekiPtr suandakiPtr

C

geciciPtr

475

Şekil 12.6 Bir listeden düğüm silmek

listeyiYazdir fonksiyonu(satır 148), listenin başlangıcını gösteren göstericiyi argüman olarak

almakta ve göstericiyi, suandakiPtr olarak kullanmaktadır.Fonksiyon ilk önce, listenin boş

olup olmadığına karar vermektedir.Eğer boşsa, listeyiYazdir fonksiyonu “Liste boştur”

yazdırmakta ve sonlanmaktadır.Aksi takdirde ise,listedeki veriyi yazdırmaktadır.suandakiPtr

boş olmadığı sürece, suandakiPtr->veri fonksiyon tarafından yazdırılmaktadır ve

suandakiPtr->sonrakiPtr ’ye atanmaktadır.Eğer listenin son düğümündeki bağ NULL

değilse,yazdırma algoritması listenin sonrasında yer alanları da yazdırmaya çalışacak ve bir

hata oluşacaktır.Yazdırma algoritması bağlı listeler,yığınlar ve sıralar için aynıdır.

12.5 YIĞINLAR

Bir yığın,bağlı listelerin kısıtlanmış bir çeşididir.Yeni düğümler yığına yalnızca en üstten

eklenir ve düğümler yığının yalnızca en üstünden çıkartılabilir.Bu sebepten,yığınlar son giren

ilk çıkar(LIFO Last in-First out) veri yapıları olarak adlandırılır.Bir yığın,yığının en üstteki

elemanını gösteren bir gösterici sayesinde kullanılır.Yığının en sonundaki bağ elemanı NULL

yapılarak, bu elemanın yığının sonu olduğu belirtilir.

Şekil 12.7, bir çok düğümden oluşan bir yığını göstermektedir.Yığınlar ve bağlı listeler

arasındaki fark, ekleme ve çıkarmaların listede her noktadan yapılabilmesi ancak yığınlarda

yalnızca en üstten yapılabilmesidir.

Şekil 12.7 Bir yığının grafik gösterimi

Genel Programlama Hataları 12.6

Bir yığının en son düğümündeki bağı NULL yapmamak.

Bir yığınla ilgili işlemlerde kullanılan temel fonksiyonlar, push ve pop fonksiyonlarıdır.push

fonksiyonu, yeni bir düğüm yaratır ve yığının üstüne yerleştirir.pop fonksiyonu, yığının

üstündeki düğümü çıkartır,çıkartılan bu düğüm için tahsis edilmiş olan hafızayı serbest bırakır

ve çıkartılmış değeri döndürür.

Şekil 12.8 (çıktısı Şekil 12.9’da gösterilmiştir),tamsayılardan oluşan bir yığını

işlemektedir.Program üç seçenek sunmaktadır:1) yığına bir değer ekle(push fonksiyonu)

2)yığından bir değer çıkart(pop fonksiyonu)

3)programı sonlandır.

8 2 ..... 3

yiginPtr

476

1 /\* Şekil 12.8: fig12\_08.c

2 dinamik yığın programı \*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 struct yiginDugumu { /\* kendine dönüşlü yapı \*/

7 int veri;

8 struct yiginDugumu \*sonrakiPtr;

9 };

10

11 typedef struct yiginDugumu YiginDugumu;

12 typedef YiginDugumu \*YiginDugumuPtr;

13

14 void push( YiginDugumuPtr \*, int );

15 int pop( YiginDugumuPtr \* );

16 int bosMu( YiginDugumuPtr );

17 void yiginYazdir( YiginDugumuPtr );

18 void menu( void );

19

20 int main( )

21 {

22 YiginDugumuPtr yiginPtr = NULL; /\* yığının en başını gösterir. \*/

23 int secim, deger;

24

25 menu( );

26 printf( "? " );

27 scanf( "%d", &secim );

28

29 while ( secim != 3 ) {

30

31 switch ( secim ) {

32 case 1: /\* değeri yığına push eder(atar) \*/

33 printf( "Bir tamsayı giriniz: " );

34 scanf( "%d", &deger );

35 push( &yiginPtr, deger );

36 yiginYazdir( yiginPtr );

37 break;

38 case 2: /\* yıgından değeri pop et (çek) \*/

39 if ( !bosMu( yiginPtr ) )

40 printf( "Çekilen değer: %d.\n",

41 pop( &yiginPtr ) );

42

43 yiginYazdir( yiginPtr );

44 break;

45 default:

46 printf( "Geçersiz secim.\n\n" );

47 menu( );

48 break;

49 }

477

50

51 printf( "? " );

52 scanf( "%d", &secim );

53

}

54

55 printf( "Program sonlandı.

\n" );

56 return 0;

57

}

58

59 /\* Menüyü yazdır \*/

60 void menu( void

)

61 {

62 printf( "Seçiminizi giriniz:

\n"

63 "1 Yığına bir değer push etmek için

\n"

64 "2 Yığından değeri pop etmek için

\n"

65 "3 Programı sonlandır

\n" );

66

}

67

68 /\* Yığının başına bir düğüm ekle \*/

69 void push( YiginDugumuPtr \*ustPtr, int info )

70 {

71 YiginDugumuPtr yeniPtr;

72

73 yeniPtr = malloc( sizeof( YiginDugumu ) );

74 if ( yeniPtr != NULL ) {

75 yeniPtr

->veri = info;

76 yeniPtr

->sonrakiPtr = \*ustPtr;

77 \*ustPtr = yeniPtr;

78

}

79 else

80 printf( "%d eklenemedi.Yetersiz hafıza.

\n",

81 info );

82

}

83

84 /\* Yığının üstünden bir düğüm çıkart \*/

85 int pop( YiginDugumuPtr \*ustPtr )

86 {

87 YiginDugumuPtr geciciPtr;

88 int popDeger;

89

90 geciciPtr = \*ustPtr;

91 popDeger = ( \*ustPtr )

->veri;

92 \*ustPtr = ( \*ustPtr )

->sonrakiPtr;

93 free( geciciPtr );

94 return popDeger;

95

}

96

97 /\* Yığını yazdır \*/

98 void yiginYazdir( YiginDugumuPtr suandakiPtr )

99 {

478

100 if ( suandakiPtr == NULL )

101 printf( "Yığın boştur.\n\n" );

102 else {

103 printf( "Yığın:\n" );

104

105 while ( suandakiPtr != NULL ) {

106 printf( "%d --> ", suandakiPtr->veri );

107 suandakiPtr = suandakiPtr->sonrakiPtr;

108 }

109

110 printf( "NULL\n\n" );

111 }

112 }

113

114 /\* Yığın boş mu? \*/

115 int bosMu( YiginDugumuPtr ustPtr )

116 {

117 return ustPtr == NULL;

118 }

Şekil 12.8 Basit bir yığın programı

Seçiminizi giriniz:

1 Yığına bir değer push etmek için

2 Yığından değeri pop etmek için

3 Programı sonlandır

? 1

Bir tamsayı giriniz:5

Yığın:

5-->NULL

? 1

Bir tamsayı giriniz: 6

Yığın:

6-->5-->NULL

? 1

Bir tamsayı giriniz: 4

Yığın:

4-->6-->5-->NULL

? 2

Çekilen değer: 4

Yığın:

6-->5-->NULL

? 2

Çekilen değer: 6

Yığın:

5-->NULL

479

? 2

Çekilen değer:5

Yığın boştur.

? 2

Yığın boştur

? 4

Geçersiz seçim

Seçiminizi giriniz:

1 Yığına bir değer push etmek için

2 Yığından değeri pop etmek için

3 Programı sonlandır

? 3

Program sonlandı

Şekil 12.9 Şekil 12.8’deki programın örnek bir çıktısı

push fonksiyonu, yığının en üstüne yeni bir düğüm yerleştirir.Fonksiyon üç adımdan

oluşmaktadır:

1. malloc çağırarak yeni bir düğüm yarat , tahsis edilen hafıza konumunu yeniPtr ’ye

ata,yığına yerleştirilecek değeri yeniPtr->veri ’ye ata ve yeniPtr-sonrakiPtr ’ye NULL

ata.

2. \*ustPtr ’yi(yığının en üstünü gösteren gösterici) yeniPtr->sonrakiPtr ’ye ata .Böylece

yeniPtr’in bağ elemanı daha önceki en üst düğümü gösterir.

3. yeniPtr ’yi \*ustPtr ’ye ata.Böylelikle \*ustPtr artık yeni yığının en üstünü gösterecektir.

\*ustPtr ’yi içeren işlemler, yiginPtr ’in değerini main içinde değiştirmektedir.Şekil 12.10,

push fonksiyonunu tasvir etmektedir.Şeklin a) kısmı yığını ve yeni düğümü, push işleminden

önce göstermektedir.Şeklin b) kısmındaki kesikli çizgiler, 12 değerini içeren yeni düğümü

yığının en üstüne yerleştiren push işleminin 2 ve 3.adımlarını göstermektedir.

Şekil 12.10 push işlemi

12

7 11

12

7 11

\*ustPtr

\*ustPtr

yeniPtr

yeniPtr

a)

b)

480

pop fonksiyonu (satır 85) , yığından bir düğümü çıkartır. main ’in , pop fonksiyonunu

çağırmadan önce, yığının boş olup olmadığına karar verdiğine dikkat ediniz.pop işlemi 5

adım içermektedir:

1.\*ustPtr ’yi geciciPtr ’ye ata.(geciciPtr ,gereksiz hafızayı serbest bırakmak için

kullanılacaktır)

2.(\*ustPtr)->veri ‘yi popdeger ’e ata(en üst düğümdeki değeri sakla)

3.(\*ustPtr)->sonrakiPtr ’yi \*ustPtr ’ye ata(yeni düğümün adresini \*ustPtr ’ye ata)

4.geciciPtr ile gösterilen hafızayı serbest bırak

5.popdeger ’i çağırıcıya geri döndür(Şekil 12.8’deki programda main)

Şekil 12.11, pop fonksiyonunu tasvir etmektedir.a) kısmı yığını, az önceki push işleminden

sonraki biçimiyle göstermektedir.b) kısmı geciciPtr ’in yığının ilk düğümünü göstermesini ve

ustPtr ’in yığının ikinci düğümünü göstermesini tasvir etmektedir.free fonksiyonu, geciciPtr

ile gösterilen hafızayı serbest bırakmak için kullanılmıştır.

Şekil 12.11 pop işlemi

Yığınların bir çok ilginç uygulaması vardır.Örneğin,bir fonksiyon çağrısı yapıldığında,

çağrılan fonksiyon çağırıcısına nasıl geri döneceğini bilmek zorundadır,bu yüzden geri dönüş

adresi yığına yazılır.Eğer bir çok fonksiyon çağrısı art arda yapılırsa,geri dönüş değerleri

yığına, son giren ilk çıkar sırasına göre yazılır.Böylece, her fonksiyon kendi çağırıcısına geri

dönebilir.Yığınlar, yinelemeli fonksiyon çağrılarını da yinelemeli olmayan fonksiyon

çağrılarında olduğu gibi destekler.

Yığınlar, bir fonksiyonun her çağrısında yaratılan otomatik değişkenler için bir alan içerir.

Fonksiyon çağırıcısına geri döndüğünde,o fonksiyonun otomatik değişkenleri için ayrılan alan

yığından çıkartılır ve bu değişkenler artık program tarafından bilinemez.Yığınlar, derleyiciler

tarafından deyimlerin hesaplanması ve makine dili kodlarının oluşturulması esnasında

kullanılır.Alıştırmalar yığınların bir çok uygulamasını araştırmaktadır.

12.6 SIRALAR

Oldukça yaygın bir diğer veri yapısı da sıralardır.Bir sıra , marketlerde kasada oluşan sıralara

benzer.İlk önce, sıradaki ilk kişinin işleri yapılır ve diğer müşteriler sıraya yalnızca sıranın

sonundan girebilirler.Sıra düğümleri, yalnızca sıranın başından çıkartılır ve yalnızca sıranın

kuyruğundan eklenirler.Bu sebepten,bir sıra ilk giren ilk çıkar (FIFO First in First out) veri

7 11

7 11

12

12

\*ustPtr

\*ustPtr

a)

b)

geciciPtr

481

yapısı olarak adlandırılır.Ekleme ve çıkarma işlemleri, sirayaGir ve siradanCik olarak

bilinir.

Sıraların bilgisayar sistemlerinde bir çok uygulaması vardır.Çoğu bilgisayar, yalnızca tek bir

işlemciye sahiptir,bu yüzden bir anda yalnızca tek kullanıcıya hizmet verir.Diğer

kullanıcıların istekleri sıraya konur.Her istek, sadece en öndeki isteğin gerektirdiği işlemler

sonlanınca, sırada ileriye doğru ilerler.En öndeki istek, bir sonra servis sağlanacak istektir.

Sıralar ayrıca yazdırma işlemlerinde kullanılırlar.Çok kullanıcılı bir ortamda yalnızca tek bir

yazıcı bulunuyor olabilir.Bir çok kullanıcı yazdırılacak çıktılar üretebilir.Yazıcı meşgulken

hala başka çıktılar üretilebilir.Bunlar yazıcı uygun hale gelene kadar diskte bir sıra içinde

tutulurlar.

Bilgi paketleri de bilgisayar ağlarında sırada beklerler.Bir ağ düğümüne paket ulaştığında,bu

paket gideceği yere kadar,bir sonraki düğüme yönlendirilmelidir.Yönlendirici düğüm, bir

anda yalnızca tek bir paketi yönlendirebilir,bu yüzden daha sonradan gelen paketler

yönlendirilene kadar sıraya konur. Şekil 12.12 , birkaç düğümden oluşan bir sırayı tasvir

etmektedir.Sıranın başındaki ve sonundaki göstericilere dikkat ediniz.

Genel Programlama Hataları 12.7

Bir sıranın son düğümündeki bağı NULL yapmamak.

Şekil 12.12 Bir sıranın grafik gösterimi.

Şekil 12.13 (çıktısı Şekil 12.14’te gösterilmiştir), sıra işlemleri yapmaktadır.Program birkaç

seçenek sunmaktadır:sıraya yeni bir düğüm eklemek(sirayaGir fonksiyonu),sıradan bir

düğümü çıkartmak(siradanCik fonksiyonu) ve programın sonlanması.

1 /\* Fig. 12.13: fig12\_13.c

2 Bir sıranın işletimi ve yönetimi \*/

3

4 #include <stdio.h>

5 #include <stdlib.h>

6

7 struct siraDugumu { /\* kendine dönüşlü yapı \*/

8 char veri;

9 struct siraDugumu \*sonrakiPtr;

10 };

11

12 typedef struct siraDugumu SiraDugumu;

\*ustPtr

....

basPtr kuyrukPtr

482

13 typedef SiraDugumu \*SiraDugumuPtr;

14

15 /\* fonksiyon prototipleri \*/

16 void sirayiYazdir( SiraDugumuPtr );

17 int bosMu( SiraDugumuPtr );

18 char siradanCik( SiraDugumuPtr \*, SiraDugumuPtr \* );

19 void sirayaGir( SiraDugumuPtr \*, SiraDugumuPtr \*, char );

20 void menu( void );

21

22 int main()

23 {

24 SiraDugumuPtr basPtr = NULL, kuyrukPtr = NULL;

25 int secim;

26 char secimNo;

27

28 menu();

29 printf( "? " );

30 scanf( "%d", &secim );

31

32 while ( secim != 3 ) {

33

34 switch( secim ) {

35

36 case 1:

37 printf( "Bir karakter giriniz: " );

38 scanf( "\n%c", &secimNo );

39 sirayaGir( &basPtr, &kuyrukPtr, secimNo );

40 sirayiYazdir( basPtr );

41 break;

42 case 2:

43 if ( !bosMu( basPtr ) ) {

44 secimNo = siradanCik( &basPtr, &kuyrukPtr );

45 printf( "%c sıradan çıkartılmıştır.\n", secimNo );

46 }

47

48 sirayiYazdir( basPtr );

49 break;

50

51 default:

52 printf( "Geçersiz seçim.\n\n" );

53 menu();

54 break;

55 }

56

57 printf( "? " );

58 scanf( "%d", &secim );

59 }

60

61 printf( "Program sonlandı.\n" );

62 return 0;

483

63 }

64

65 void menu( void )

66 {

67 printf ( "Seçiminizi giriniz:\n"

68 " 1 Sıraya eleman eklemek için\n"

69 " 2 Sıradan eleman çıkarmak için\n"

70 " 3 Programdan Çık\n" );

71 }

72

73 void sirayaGir( SiraDugumuPtr \*basPtr, SiraDugumuPtr \*kuyrukPtr,

74 char deger )

75 {

76 SiraDugumuPtr yeniPtr;

77

78 yeniPtr = malloc( sizeof( SiraDugumu ) );

79

80 if ( yeniPtr != NULL ) {

81 yeniPtr->veri = deger;

82 yeniPtr->sonrakiPtr = NULL;

83

84 if ( bosMu( \*basPtr ) )

85 \*basPtr = yeniPtr;

86 else

87 ( \*kuyrukPtr )->sonrakiPtr = yeniPtr;

88

89 \*kuyrukPtr = yeniPtr;

90 }

91 else

92 printf( "%c eklenemedi. Yetersiz hafıza.\n",

93 deger );

94 }

95

96 char siradanCik( SiraDugumuPtr \*basPtr, SiraDugumuPtr \*kuyrukPtr )

97 {

98 char deger;

99 SiraDugumuPtr geciciPtr;

100

101 deger = ( \*basPtr )->veri;

102 geciciPtr = \*basPtr;

103 \*basPtr = ( \*basPtr )->sonrakiPtr;

104

105 if ( \*basPtr == NULL )

106 \*kuyrukPtr = NULL;

107

108 free( geciciPtr );

109 return deger;

110 }

111

112 int bosMu( SiraDugumuPtr basPtr )

484

113 {

114 return basPtr == NULL;

115 }

116

117 void sirayiYazdir( SiraDugumuPtr currentPtr )

118 {

119 if ( currentPtr == NULL )

120 printf( "Sıra boştur.\n\n" );

121 else {

122 printf( "Sıra:\n" );

123

124 while ( currentPtr != NULL ) {

125 printf( "%c --> ", currentPtr->veri );

126 currentPtr = currentPtr->sonrakiPtr;

127 }

128

129 printf( "NULL\n\n" );

130 }

131 }

Şekil 12.13 Bir sırayı işlemek

Seçiminizi giriniz:

1 Sıraya eleman eklemek için

2 Sıradan eleman çıkarmak için

3 Programdan Çık

? 1

Bir karakter giriniz: A

Sıra:

A--> NULL

? 1

Bir karakter giriniz: B

Sıra:

A-->B-->NULL

? 1

Bir karakter giriniz: C

Sıra:

A-->B-->C-->NULL

? 2

A sıradan çıkartılmıştır

Sıra:

B-->C-->NULL

? 2

B sıradan çıkartılmıştır

Sıra:

C-->NULL

485

? 2

C sıradan çıkartılmıştır

Sıra boştur

? 2

Sıra boştur

? 4

Geçersiz seçim

Seçiminizi giriniz:

1 Sıraya eleman eklemek için

2 Sıradan eleman çıkarmak için

3 Programdan Çık

? 3

Program sonlandı

Şekil 12.14 Şekil 12.13’teki programın örnek çıktısı

sirayaGir fonksiyonu(satır 73), main ’den üç argüman alır : sıranın başını gösteren

göstericinin adresi, sıranın kuyruğunu gösteren göstericinin adresi ve sıraya eklenecek

değer.Fonksiyon üç adımdan oluşmaktadır.

1.Yeni bir düğüm yarat:malloc çağır,tahsis edilen hafıza konumunu yeniPtr ‘ye ata,sıraya

eklenecek değeri yeniPtr->veri ‘ye ata ve yeniPtr->sonrakiPtr ’ye NULL ata.

2. Eğer sıra boşsa, yeniPtr ‘yi \*basPtr ‘ye ata ,aksi takdirde yeniPtr göstericisini

( \*kuyrukPtr) -> sonrakiPtr ’ye ata.

3. yeniPtr ’yi \*kuyrukPtr ’ye ata.

Şekil 12.15, sirayaGir işlemini göstermektedir. a) kısmı sırayı ve yeni düğümü, işlemden

önceki durumlarıyla göstermektedir. b) kısmındaki kesikli çizgiler boş olmayan bir sıranın

sonuna yeni bir düğüm ekleyen sirayaGir fonksiyonunun 2 ve 3. adımlarını göstermektedir.

Şekil 12.15 sirayaGir işleminin grafik gösterimi

R A D N

\*basPtr \*kuyrukPtr yeniPtr

a)

b)

R A D N

\*basPtr \*kuyrukPtr yeniPtr

486

siradanCik fonksiyonu(satır 96), sıranın başını gösteren göstericinin adresini ve sıranın

kuyruğunu gösteren göstericinin adresini argüman olarak alıp sıradaki ilk düğümü sıradan

çıkartır.Fonksiyon 6 adımdan oluşmaktadır.

1.(\* basPtr)->veri ‘yi deger ‘e ata.(veriyi sakla)

2.\*basPtr’yi geciciPtr ’ye ata.(geciciPtr gereksiz hafızayı serbest bırakmak için

kullanılacaktır)

3.(\*basPtr)->sonrakiPtr ‘yi \* basPtr ’ye ata.(\*basPtr artık sıradaki yeni düğümü

göstermektedir.)

4.Eğer \*basPtr NULL ise,\*kuyrukPtr ’ye NULL ata.

5.geciciPtr ile gösterilen hafızayı serbest bırak

6.deger ’i çağırıcıya döndür.(siradanCik fonksiyonu Şekil 12.12’teki programda main

içinden çağrılmıştır)

Şekil 12.16’da, siradanCik fonksiyonunu tasvir edilmiştir.Şeklin a) kısmı, sırayı az önceki

sirayaGir işleminden sonraki haliyle göstermektedir.b) kısmı ise, geciciPtr çıkarılan düğümü

gösterirken ve basPtr yeni sıranın ilk düğümünü gösterirken çizilmiştir.free fonksiyonu

geciciPtr ile gösterilen hafızanın serbest bırakılması için kullanılmıştır.

Şekil 12.16 siradanCik işleminin grafik gösterimi

12.7 AĞAÇLAR

Bağlı listeler,yığınlar ve sıralar doğrusal veri yapılarıdır.Bir ağaç , doğrusal olmayan, iki

boyutlu ve özel amaçlı bir veri yapısıdır.Ağaç düğümleri iki ya da daha fazla bağ içerebilir.Bu

kısım ikili ağaçları(Şekil 12.17) açıklamaktadır.İkili ağaçlar, tüm düğümleri iki bağ

içeren(hiçbiri , biri ya da ikisi birden NULL olabilir) ağaçlardır.Kök düğüm(root node)

ağaçtaki ilk düğümdür.Kökteki her bağ bir çocuğu(child) belirtir.İlk sol çocuk, sol ağaççıktaki

(subtree) ilk düğümdür ve sağ çocuk sağ ağaççıktaki ilk düğümdür.Bir düğümün çocuklarına

kardeşler(siblings) denir.Çocukları olmayan düğüme, yaprak düğüm(leaf node) denir.

Bilgisayar uzmanları ağaçları, kök düğümden aşağıya doğru(doğadaki ağaçların tam tersi)

çizerler.

R A D N

\*basPtr \*kuyrukPtr

R A D N

\*basPtr \*kuyrukPtr

\*geciciPtr

b)

a)

487

Bu kısımda,ikili arama ağacı adı verilen özel bir ikili ağacı inceleyeceğiz.İkili arama

ağacı(düğüm değerleri diğeriyle aynı değere sahip olmayan), herhangi bir sol ağaççıktaki

değerlerin, ebeveyn(parent) düğümündeki değerlerden daha küçük olması ve herhangi bir sağ

ağaççıktaki değerlerin, ebeveyn düğümündeki değerlerden büyük olması karakteristiğine

sahiptir.Şekil 12.18, 12 değere sahip bir ikili arama ağacını göstermektedir.Değerleri temsil

eden ikili arama ağacının şeklinin, değerlerin ağaca yerleştirilme sırasına göre

değişebileceğine dikkat ediniz.

Genel Programlama Hataları 12.8

Bir ağacın yaprak düğümlerindeki bağları NULL yapmamak.

Şekil 12.19 (çıktısı şekil 12.20’de gösterilmiştir) , ikili bir arama ağacı yaratmakta ve ağacın

içinde 3 yoldan ilerlemektedir.Program, rasgele 10 sayı üretmekte ve sayıları ağacın içine,

birbirinin aynısı olan değerler hariç, yerleştirmektedir.

Şekil 12.17 İkili ağacın grafik gösterimi

Şekil 12.18 İkili arama ağacı

47

25 77

11 43 65 93

7 17 31 44 68

488

1 /\* Şekil 12.19: fig12\_19.c 2 İkili bir ağaç yarat ve bu ağaçta öncesol, öncedüğüm 3 ve önceçocuk biçiminde ilerlemek \*/ 4 #include <stdio.h> 5 #include <stdlib.h> 6 #include <time.h> 78 struct agacDugumu { 9 struct agacDugumu \*solPtr;

10 int veri;

11 struct agacDugumu \*sagPtr;

12 };

13

14 typedef struct agacDugumu AgacDugumu;

15 typedef AgacDugumu \*AgacDugumuPtr;

16

17 void dugumEkle( AgacDugumuPtr \*, int );

18 void onceSol( AgacDugumuPtr );

19 void onceDugum( AgacDugumuPtr );

20 void onceCocuk( AgacDugumuPtr );

21

22 int main()

23 {

24 int i, secimNo;

25 AgacDugumuPtr kokPtr = NULL;

26

27 srand( time( NULL ) );

28

29 /\* Ağaca 1 ile 15 arasında rasgele değerler yerleştir \*/

30 printf( "Ağaca yerleştirilen değerler:

\n" );

31

32 for ( i = 1; i <= 10; i++ ) {

33 secimNo = rand() % 15;

34 printf( "%3d", secimNo );

35 dugumEkle( &kokPtr, secimNo );

36

}

37

38 /\* ağaçta onceDugum biçiminde ilerle\*/

39 printf( "

\

n

\nonceDugum ilerleme :

\n" );

40 onceDugum( kokPtr );

41

42 /\* ağaçta onceSol biçiminde ilerle\*/

43 printf( "

\

n

\nonceSol ilerleme :

\n" );

44 onceSol( kokPtr );

45

46 /\* ağaçta onceCocuk biçiminde ilerle \*/

47 printf( "

\

n

\nonceCocuk ilerleme:

\n" );

48 onceCocuk( kokPtr );

489

49

50 return 0;

51

}

52

53 void dugumEkle( AgacDugumuPtr \*agacPtr, int deger )

54 {

55 if ( \*agacPtr == NULL ) { /\* \*agacPtr NULL’dur \*/

56 \*agacPtr = malloc( sizeof( AgacDugumu ) );

57

58 if ( \*agacPtr != NULL ) {

59 ( \*agacPtr )

->veri = deger;

60 ( \*agacPtr )

->solPtr = NULL;

61 ( \*agacPtr )

->sagPtr = NULL;

62

}

63 else

64 printf( "%d eklenemedi. Yetersiz hafıza.

\n",

65 deger );

66

}

67 else

68 if ( deger < ( \*agacPtr )

->veri )

69 dugumEkle( &( ( \*agacPtr )

->solPtr ), deger );

70 else if ( deger > ( \*agacPtr )

->veri )

71 dugumEkle( &( ( \*agacPtr )

->sagPtr ), deger );

72 else

73 printf( "kopya" );

74

}

75

76 void onceSol( AgacDugumuPtr agacPtr )

77 {

78 if ( agacPtr != NULL ) {

79 onceSol( agacPtr

->solPtr );

80 printf( "%3d", agacPtr

->veri );

81 onceSol( agacPtr

->sagPtr );

82

}

83

}

84

85 void onceDugum( AgacDugumuPtr agacPtr )

86 {

87 if ( agacPtr != NULL ) {

88 printf( "%3d", agacPtr

->veri );

89 onceDugum( agacPtr

->solPtr );

90 onceDugum( agacPtr

->sagPtr );

91

}

92

}

93

94 void onceCocuk( AgacDugumuPtr agacPtr )

95

{

96 if ( agacPtr != NULL ) {

97 onceCocuk( agacPtr

->solPtr );

98 onceCocuk( agacPtr

->sagPtr );

490

99 printf( "%3d", agacPtr->veri );

100 }

101 }

Şekil 12.19 İkili bir ağaç yaratma ve içinde ilerleme

Ağaca yerleştirilen değerler:

7 8 0 6 14 1 0kopya 13 0kopya 7kopya

onceSol ilerleme:

7 0 6 1 8 14 13

onceDugum ilerleme:

0 1 6 7 8 13 14

oncecocuk ilerleme:

1 6 0 13 14 8 7

Şekil 12.20 Şekil 12.19’daki programın örnek çıktısı

Şekil 12.19’da, ikili arama ağacı yaratmak ve ağaç içinde ilerlemek için kullanılan

fonksiyonlar yinelemelidir.dugumekle fonksiyonu (satır 53),ağacın adresini ve ağaçta

saklanacak tamsayı değerini argüman olarak alır.Bir düğüm, ikili arama ağacına yalnızca bir

yaprak düğüm olarak eklenebilir.İkili arama ağacına düğüm eklemek için izlenen adımlar

şunlardır :

1.Eğer \*agacPtr NULL ise yeni bir düğüm yarat.malloc çağır,tahsis edilen hafızayı

\*agacPtr ’ye ata,( \*agacPtr)->veri ’ye saklanacak tamsayıyı ata, ( \*agacPtr)->solPtr ve

( \*agacPtr)->sagPtr ’ye NULL ata ve kontrolü çağırıcıya döndür.(main ya da daha önceki

dugumekle fonksiyonuna)

2.Eğer \*agacPtr ‘in değeri NULL değilse ve eklenecek değer ( \*agacPtr)->veri ‘den

küçükse, dugumekle fonksiyonu ( \*agacPtr)->solPtr ‘in adresi ile çağrılır.Aksi takdirde,

dugumekle fonksiyonu ( \*agacPtr)->sagPtr ‘in adresi ile çağrılır.Yineleme, NULL gösterici

bulunana kadar devam eder daha sonra ise yeni düğüm eklemek için 1.adım çalıştırılır.

oncesol(satır 76),oncedugum(satır 85) ve oncecocuk(satır 94) fonksiyonlarının hepsi de bir

ağaç alır(yani ağacın kök düğümünü gösteren bir gösterici alır) ve ağaç içinde ilerler.

oncesol ilerlemenin adımları:

1.sol ağaççıkta oncesol biçimde ilerle

2.Düğümdeki değeri işle

3.Sağ ağaççıkta oncesol biçimde ilerle.

Bir düğümdeki değer, sol ağaççığındaki değerler işlenene kadar işleme tabii tutulmaz.Şekil

12.21’deki ağaçta oncesol ilerleme şu şekildedir:

491

6 13 17 27 33 42 48

27

13 42

6 17 33 48

Şekil 12.21 İkili arama ağacı

İkili arama ağacında oncesol ilerlemenin, düğüm değerlerini artan bir sırada yazdırdığına

dikkat ediniz.İkili arama ağacı yaratma süreci, aslında veriyi sıralamaktadır ve bu yüzden bu

sürece ikili ağaç sıralama denir.

oncedugum ilerlemenin adımları:

1.Düğümdeki değeri işle

2.Sol ağaççıkta oncedugum biçiminde ilerle

3.Sağ ağaççıkta oncedugum biçiminde ilerle

Her düğümdeki değer, düğüme gelindiğinde işlenir.Verilen bir düğümdeki değer işlendikten

sonra,sol ağaççıktaki değerler işlenir ve daha sonra sağ ağaççıktaki değerler işlenir.Şekil

12.21’deki ağaçta oncedugum ilerleme şu şekildedir:

27 13 6 17 42 33 48

oncecocuk ilerlemenin adımları:

1.Sol ağaççıkta oncecocuk biçiminde ilerle.

2.Sağ ağaççıkta oncecocuk biçiminde ilerle.

3.Düğümdeki değeri işle

Düğümlerdeki değer , çocuklarının değerleri yazdırılana kadar işlenmez.Şekil 12.21’deki

ağaçta oncecocuk ilerleme şu şekildedir:

6 17 13 33 48 42 27

İkili arama ağacı, kopyaların elenmesini sağlar.Ağaç yaratılırken,bir değerin kopyasının

eklenmeye çalışması fark edilir çünkü kopya değer orijinal değerle aynı yolu ilerleyecektir.Bu

sebepten,kopya değer en sonunda kendisiyle aynı değere sahip düğümle karşılaştırılacaktır.

Kopya değer bu noktada kolaylıkla elenebilir.

İkili bir ağaçta, anahtar bir değerle eşleşen değeri aramak da oldukça hızlıdır.Eğer ağaç sıkıca

paketlenmişse,her seviye bir önceki seviyenin iki katı kadar eleman içerecektir.n elemanlı bir

ikili arama ağacı, en fazla log2n seviye içerecektir ve bu sebepten bir eşlemeyi bulmak ya da

eşleme olmadığına karar vermek en fazla log2n karşılaştırma gerektirecektir.Bu

sebepten,örneğin,1000 elemanlı bir ikili arama ağacında(sıkıca paketlenmiş),10’dan fazla

492

karşılaştırma yapmaya gerek yoktur çünkü 2

10>1000. Sıkıca paketlenmiş ve 1000000 elemanlı

ikili arama ağacında 20’den fazla karşılaştırma yapmaya gerek yoktur çünkü 220>1000000

Alıştırmalarda,ikili ağaçtan değer silme,ikili ağacı iki boyutlu ağaç biçiminde yazdırma ve

ikili ağaçta seviye sıralı ilerleme gibi bir çok algoritma gösterilmiştir.İkili ağaçta seviye sıralı

ilerleme, kök düğüm seviyesinden başlayarak ağaçtaki düğümleri satır satır ziyaret

etmektedir. Ağacın her seviyesinde,düğümler soldan sağa ziyaret edilmektedir.Diğer ikili

ağaç alıştırmaları ikili bir ağacın kopya değerler içerebilmesi,ikili ağaca string değerlerinin

yerleştirilmesi ve ikili ağaçta kaç adet seviye bulunduğunun belirlenmesi gibi sorular

içermektedir.

ÖZET

 Kendine dönüşlü yapılar , yapı tipiyle aynı tipte bir yapıyı gösteren ve link adı bir

elemana sahiptir.

 Kendine dönüşlü yapılar birbirine bağlanarak listeler, sıralar, yığınlar ve ağaçlar gibi

kullanışlı veri tipleri oluşturmakta kullanılabilirler

 Dinamik hafıza tahsisi, programın çalışma zamanında yeni bir veri nesnesini

tutabilmek hafızadan byte blokları ayırır.

 malloc fonksiyonu, tahsis edilecek byte sayısını argüman olarak alır ve tahsis edilen

alanı gösteren void\* tipte bir gösterici döndürür.malloc fonksiyonu normalde sizeof

operatörü ile kullanılır.sizeof operatörü hafıza tahsis edilen yapının boyutunu byte

cinsinden belirler.

 free fonksiyonu tahsis edilen hafıza alanını serbest bırakır.

 Bir bağlı liste, birleştirilmiş bir grup kendine dönüşlü yapının bir birlikteliğidir.

 Bağlı bir liste, dinamik bir veri yapısıdır.Listenin uzunluğu gerektikçe artabilir ya da

azalabilir.

 Bağlı listeler, hafıza yeterli oldukça büyümeye devam edebilir.

 Bağlı listeler, göstericilerin yeniden atanması sayesinde basit veri ekleme ve

çıkarmalar yapabilir.

 Yığınlar ve sıralar,bağlı listelerin özelleştirilmiş biçimleridir.

 Yeni düğümler yığına yalnızca en üstten eklenir ve düğümler yığının yalnızca en

üstünden çıkartılabilir.Bu sebepten,yığınlar son giren ilk çıkar(LIFO Last in-First out)

veri yapıları olarak adlandırılır.

 Yığının son düğümündeki bağ elemanı,yığının sonu olduğunu belirtmek için,NULL

yapılır.

 Bir yığınla ilgili işlemlerde kullanılan temel fonksiyonlar, push ve pop

fonksiyonlarıdır.push fonksiyonu, yeni bir düğüm yaratır ve yığının üstüne

yerleştirir.pop fonksiyonu, yığının üstündeki düğümü çıkartır,çıkartılan bu düğüm için

tahsis edilmiş olan hafızayı serbest bırakır ve çıkartılmış değeri döndürür.

 Sıra veri yapılarında düğümler, yalnızca sıranın başından çıkartılır ve yalnızca sıranın

kuyruğundan eklenirler.Bu sebepten,bir sıra ilk giren ilk çıkar (FIFO First in First out)

veri yapısı olarak adlandırılır.Ekleme ve çıkarma işlemleri, sirayaGir ve siradanCik

olarak bilinir.

 Bir ağaç ,bağlı listeler,sıralar ve yığınlara göre daha karmaşık veri yapılarıdır.Ağaçlar

iki boyutlu ve her düğüm için iki ya da daha fazla bağ içeren veri yapılarıdır.

 İkili ağaçlar bir düğüm için iki bağ içerirler.

493

 Kök düğüm, ağaçtaki ilk düğümdür.

 Kökteki her bağ bir çocuğu(child) belirtir.İlk sol çocuk, sol ağaççıktaki (subtree) ilk

düğümdür ve sağ çocuk sağ ağaççıktaki ilk düğümdür.Bir düğümün çocuklarına

kardeşler(siblings) denir.Çocukları olmayan düğüme, yaprak düğüm(leaf node) denir.

 İkili arama ağacı(düğüm değerleri diğeriyle aynı değere sahip olmayan), herhangi bir

sol ağaççıktaki değerlerin, ebeveyn(parent) düğümündeki değerlerden daha küçük

olması ve herhangi bir sağ ağaççıktaki değerlerin, ebeveyn düğümündeki değerlerden

büyük olması karakteristiğine sahiptir.

 İkili ağaçta oncesol ilerleme,sol ağaççıkta oncesol biçimde ilerler,düğümdeki değeri

işler ve sağ ağaççıkta oncesol biçimde ilerler.Bir düğümdeki değer, sol ağaççığındaki

değerler işlenene kadar işleme tabii tutulmaz.

 oncedugum ilerleme,düğümdeki değeri işler,sol ağaççıkta oncedugum biçiminde

ilerler ve sağ ağaççıkta oncedugum biçiminde ilerler.Her düğümdeki değer, düğüme

gelindiğinde işlenir.

 oncecocuk ilerleme,sol ağaççıkta oncecocuk biçiminde ilerler,sağ ağaççıkta

oncecocuk biçiminde ilerler ve düğümdeki değeri işler.Düğümlerdeki değer ,

çocuklarının değerleri yazdırılana kadar işlenmez.

ÇEVRİLEN TERİMLER

binary search tree...................... ikili arama ağacı

binary tree................................. ikili ağaç

child node................................. çocuk düğüm

dynamic data structures........... dinamik veri yapıları

dynamic memory allocation.... dinamik hafıza tahsisi

inserting a node....................... düğüm ekleme

leaf node................................. yaprak düğüm

linear data structures............ doğrusal veri yapıları

linked list............................... bağlı liste

node ..................................... düğüm

parent node.......................... ebeveyn düğüm

queue................................... sıra

siblings.............................. kardeşler

root node............................ kök düğüm

stack................................. yığın

subtree............................ ağaççık

tree.................................. ağaç

GENEL PROGRAMLAMA HATALARI

12.1 Bir listenin son bağına NULL yerleştirmemek.

12.2 Bir yapının boyutunun, elemanlarının boyutlarının toplamına eşit olduğunu düşünmek.

12.3 Dinamik olarak tahsis edilen hafızaya ihtiyaç kalmadığında, tahsis edilen hafızayı

sisteme geri döndürmemek.Bu, sistemin olması gerekenden daha erken bir zamanda hafıza

sıkıntısı çekmesine sebep olur.

12.4 malloc ile tahsis edilmemiş bir hafıza alanını serbest bırakmak(boşaltmak)

494

12.5 Serbest bırakılmış bir hafıza alanından bahsetmek ve kullanmaya

çalışmak.

12.6 Bir yığının en son düğümündeki bağı NULL yapmamak.

12.7 Bir sıranın son düğümündeki bağı NULL yapmamak

12.8 Bir ağacın yaprak düğümlerindeki bağları NULL yapmamak.

İYİ PROGRAMLAMA ALIŞTIRMALARI

12.1 Bir yapının boyutuna karar vermek için sizeof operatörünü kullanmak.

12.2 malloc kullanırken,geri dönüş değerinin NULL gösterici olup olmadığına kontrol

etmek.Eğer istenen hafıza tahsis edilemezse bir hata mesajı yazdırmak.

12.3 Dinamik olarak tahsis edilen hafızaya ihtiyaç kalmadığında, tahsis edilen hafızayı

sisteme geri döndürmemek.Bu, sistemin olması gerekenden daha erken bir zamanda hafıza

sıkıntısı çekmesine sebep olur.

12.4 Yeni bir düğümün bağ elemanına NULL atamak.Göstericiler kullanılmadan önce ilk

değerlere atanmalıdır.

PERFORMANS İPUÇLARI

12.1 Bir dizi, beklenen veri parçalarının sayısından daha fazla eleman içerecek biçimde

tanımlanabilir ancak bu hafızayı boşuna harcayabilir.Bağlı listeler bu durumlarda daha iyi bir

hafıza kullanımı sağlar.

12.2 Sıralanmış bir dizide ekleme ve silme zaman alıcı olabilir.Silinen ya da eklenen

elemandan sonraki tüm elemanlar uygun bir şekilde kaydırılmalıdır.

12.3 Bir dizinin elemanları hafızada art arda depolanır.Bu, her dizi

elemanına ani erişime izin verir çünkü elemanın adresi, dizinin başlangıcına

göre uzaklığı hesaplanarak bulunabilir.Bağlı listeler elemanlarına bu şekilde

bir ani erişimi gerçekleştiremez.

12.4 Çalışma zamanında büyüyüp küçülebilen dinamik hafıza tahsisini(dizilerin yerine)

kullanmak hafızayı verimli kullanmamızı sağlar.Ancak, göstericilerin alan kapladığını ve

dinamik hafıza tahsisi yapmak için yapılacak fonksiyon çağrılarının bir yük getirebileceğini

aklınızda tutun.

TAŞINIRLIK İPUÇLARI

12.1 Bir yapının boyutu, elemanlarının boyutları toplamına eşit olmak zorunda değildir.Bunun

sebebi, çeşitli makinelerde hizalamanın farklı yapılmasıdır(10.üniteye bakınız)

ÇÖZÜMLÜ ALIŞTIRMALAR

12.1 Aşağıdaki boşlukları doldurunuz.

a) Kendine \_\_\_\_\_\_ yapılar, dinamik veri yapıları oluşturmada kullanılır.

b) \_\_\_\_\_\_ fonksiyonu dinamik hafıza tahsisinde kullanılır.

c) \_\_\_\_\_\_, yeni düğümlerin yalnızca en üstten eklenebildiği ve yalnızca en üstünden

çıkartılabildiği bağlı listelerdir.

d) Listeyi değiştirmeden,yalnızca listeyi inceleyen fonksiyonlara\_\_\_\_\_\_\_\_\_denir

e) \_\_\_\_\_\_ veri yapısında eklenen ilk düğümler, silinen ilk düğümlerdir.

f) Bir bağlı listede, bir sonraki düğümü gösteren göstericiye \_\_\_\_\_\_ denir.

495

g) \_\_\_\_\_\_ fonksiyonu, dinamik hafıza tahsisini serbest bırakmak için kullanılır.

h) \_\_\_\_\_\_ , düğümlerin sadece en üstten eklenebildiği ve en sondan silinebildiği bir

çeşit bağlı listedir.

i) \_\_\_\_\_\_ lineer olmayan, iki boyutlu, iki ya da daha fazla bağ içeren veri yapılarıdır.

j) Yığın, \_\_\_\_\_\_ veri yapısı olarak bilinir. Çünkü en sona eklenen düğüm ilk silinen

düğümdür.

k) \_\_\_\_\_\_ ağaçlar, tüm düğümleri iki bağ içeren ağaçlardır.

l) Bir ağacın ilk düğümüne \_\_\_\_\_\_ denir.

m) Bir ağaçtaki her düğüm, o düğümün \_\_\_\_\_\_, \_\_\_\_\_\_ ya da \_\_\_\_\_\_ gösterir.

n) Çocukları olmayan düğüme \_\_\_\_\_\_ denir.

o) İkili ağaçta ilerleme algoritmaları\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_\_ olarak

adlandırılır.

12.2 Yığın ile bağlı listelerin farkları nelerdir?

12.3 Yığın ile sıra arasındaki farklar nelerdir?

12.4 Aşağıdakileri gerçekleştiren ifade ya da ifadeleri yazınız. Bütün yönetimlerin main

fonksiyonunda olduğunu (gösterici değişkenlerinin adreslerine gerek yoktur) ve aşağıdaki

tanımlamaları kabul ediniz.

struct notDugum {

char soyisim[20];

float not;

struct notDugum \*sonrakiPtr;

};

typedef struct notDugum NOTDUGUM;

typedef NOTDUGUM \*NOTDUGUMPTR;

a) Listenin başını gösteren, baslangicPtr isminde bir gösterici tanımlayınız. Liste boştur.

b) NOTDUGUM tipinde ve NOTDUGUMPTR’nin yeniPtr göstericisiyle gösterilen

yeni bir düğüm oluşturunuz. “Huseyin” stringini soyisim üyesine, 91.5 değerini ise

not üyesine(strcpy kullanın) atayınız.

c) baslangicPtr tarafından gösterilen listenin iki düğüm içerdiğini kabul ediniz. Biri

“Huseyin”, diğeri ise “Metin”. Düğümler alfabetik sıradadırlar. soyisim ve not

verilerini içeren aşağıdaki düğümlerin eklenmesini sağlayınız.

“Ahmet” 85.0

“Tarik” 73.5

“Remzi” 66.5

d) Bütün düğümlerdeki verileri ekrana yazdıran bir while döngüsü yazınız. Listede

hareket etmek için suandakiPtr göstericisini kullanınız.

e) Listedeki bütün düğümleri silen ve her düğümle alakalı hafızayı serbest bırakan bir

while döngüsü yazınız. suandakiPtr ve geciciPtr göstericileri ile listede hareket

ediniz ve hafızayı serbest bırakınız.

12.5 Şekil 12.22’deki ikili ağaçta oncecocuk, oncesol, oncedugum şeklindeki ilerlemeleri

elinizle çizerek kanıtlayınız.

496

CEVAPLAR

12.1 a) dönüşlü b) malloc c) yığın d)???? e) FIFO f) bağ g) free h) sıra i) ağaç j) LIFO k)

ikili l) kök m) çocuk ya da ağaççık n) yaprak o) oncecocuk,oncesol,oncedugum

12.2 Bağlı bir listede herhangi bir yere düğüm eklemek veya herhangi bir yerden düğüm

çıkarmak mümkündür. Yığında ise düğümler sadece en üste eklenebilir ve yine en üstten

silinebilirler.

49

28 83

18 40 71 97

11 19 32 44 69 72 92 99

Şekil 12.22 14 düğümlü ikili arama ağacı

12.3 Bir sıranın, başını ve kuyruğunu gösteren göstericileri vardır. Bu sayede, düğümler

kuyruğa eklenebilirler ve baştan silinebilirler. Bir yığının ise ekleme ve silme işlemleri

yapabileceği şekilde yığının sadece en üstünü gösteren bir göstericisi vardır.

12.4

a) NOTDUGUMPTR baslangicPTR = NULL;

b) NOTDUGUMPTR yeniPtr;

yeniPtr = malloc(sizeof(NOTDUGUM));

strcpy(yeniPtr->soyisim, “Huseyin”);

yeniPtr->not = 91.5;

yeniPtr->sonrakiPtr = NULL;

c) “Ahmet” in eklenmesi için:

oncekiPtr NULL’dur suankiPtr listenin ilk elemanını göstermektedir.

yeniPtr->sonrakiPtr = suankiPtr;

baslangicPtr = yeniPtr;

“Tarık” ın eklenmesi için:

oncekiPtr listedeki son elemanı göstermektedir.( “Metin”i içeren düğümü)

suankiPtr NULL’dur.

yeniPtr->sonrakiPtr = suankiPtr;

oncekiPtr->sonrakiPtr = yeniPtr;

“Remzi” nin eklenmesi için:

oncekiPtr “Huseyin”i içeren düğümü göstermektedir.

suankiPtr “Remzi”yi içeren düğümü göstermektedir.

yeniPtr->sonrakiPtr = suankiPtr;

oncekiPtr->sonrakiPtr = yeniPtr;

497

d) suankiPtr = baslangicPtr;

while( suankiPtr != NULL) {

printf(“Soyisim = %s\nNot = %6.2f\n”,

suankiPtr->soyisim, suankiPtr->not);

suankiPtr = suankiPtr->sonrakiPtr;

}

e) suankiPtr = baslangicPtr;

while (suankiPtr != NULL) {

geciciPtr = suankiPtr;

suankiPtr = suankiPtr->suankiPtr;

free(geciciPtr);

}

baslangicPtr = NULL;

12.5 oncesol ilerleme:

11 18 19 28 32 40 44 49 69 71 72 83 92 97 99

oncedugum ilerleme:

49 28 18 11 19 40 32 44 83 71 69 72 97 92 99

oncecocuk ilerleme:

11 19 18 32 44 40 28 69 72 71 92 99 97 83 49

ALIŞTIRMALAR

12.6 İki bağlı listeyi birbirine bağlayan bir program yazınız. Programınız, bagla isminde,

listeler için göstericileri argüman olarak alan bir fonksiyon içersin ve ikinci listeyi birinciye

bağlasın.

12.7 İki sıralanmış tamsayı listesini birleştirerek tek sıralı tamsayı listesi yapan bir program

yazınız.birlestir fonksiyonu, birleştirilecek listelerin en üst düğümünün göstericilerini

argüman olarak almalı ve birleştirilen listenin ilk düğümünün göstericisini döndürmeli.

12.8 Bir bağlı listeye 0 ile 100 arasından 25 rasgele sayıyı sıralı olarak yerleştiren bir program

yazınız. Programınız elemanların toplamını ve float tipinde ortalamalarını bulmalıdır.

12.9 10 karakterlik bir bağlı liste oluşturan bir program yazınız. Programınız daha sonra bu

listenin bir kopyasını tersten oluştursun.

12.10 Kullanıcının enter tuşuna basıncaya kadar girdiği metni yığın kullanarak tersten

yazdıran bir program yazınız.

498

12.11 Bir stringin baştan ve sondan başlanarak okunduğunda aynı olup olmadığını yığın

kullanarak anlayan bir program yazınız. Programınız boşluk karakterini ve noktalama

işaretlerini ihmal etmelidir.

12.12 Yığınlar, derleyicilere ifadelerin çalıştırılmasında ve makine dili kodunun

oluşturulmasında yardım ederler. Bu ve bundan sonraki alıştırmada, derleyicilerin sadece

sabitler, operatörler ve parantezlerden oluşan aritmetik işlemleri nasıl işlediğini göreceğiz.

İnsanlar genellikle ifadeleri, 3 + 4 ve 7 / 9 şeklinde yazarlar. Burada operatörler(+ ya da /)

operandların arasına yazılmıştır. Buna ortaEk(infix) gösterimi denir. Bilgisayarlar ise

operatörün en sağa yazıldığı sonEk gösterimini tercih ederler. Az önceki ifadeler sonek

olarak 3 4 + ve 7 9 / şeklinde gösterilir.

Karışık bir ortaek gösterimini derleyici öncelikle sonek gösterimine çevirir ve bu şekilde

işlem yapar. Bu algoritmalar için sadece ifadenin soldan sağa doğru işlenmesi gerekmektedir.

Bu algoritmalar, işlemlerini yapmak için yığın kullanırlar ve bu yığınlar farklı amaçlarla

kullanılırlar.

Bu alıştırmada ortaek gösterimini sonek gösterimine çeviren bir C algoritması yazacaksınız.

( 6 + 2 ) \* 5 – 8 / 4

ifadesini

6 2 + 5 \* 8 4 / -

şekline çeviren bir program yazınız.

Programınız ifadeyi, ortaek karakter dizisine yazsın ve ifadeyi sonek karakter dizisine sonek

gösteriminde yazmak için bu ünitedeki değiştirilmiş yığın fonksiyonlarından yararlansın.

sonek gösterimine çevirmenin algoritması aşağıdaki gibidir.

1) yığına bir sol parantez ‘(‘ it(push).

2) ortaek fonksiyonunun sonuna sağ parantez ‘)’ ekle.

3) yığın boş olmadığı sürece, soldan sağa ortaek’i oku ve aşağıdakileri gerçekleştir.

eğer ortaek’te o anda bulunan karakter bir rakamsa sonek’in bir sonraki elemanına

kopyala.

eğer ortaek’te o anda bulunan karakter bir sol parantezse, yığına it(push).

eğer ortaek’te o anda bulunan karakter bir operatör ise

Operatörleri(eğer varsa) eğer o anki operatörle aynı öncelik sırası ya da daha

yüksek öncelik sırası varsa yığının en üstüne çek(pop) ve çekilen operatörleri

sonek’e ekleyin.

ortaek’te o andaki karakteri yığına it(push).

eğer ortaek’teki o andaki karakter sağ parantez ‘)’ ise

yığının üstündeki operatörleri çek(pop) ve yığının üstünde sol parantez

kalana kadar

sonek’e ekle

yığından sol parantez çek(ve ihmal et).

Bir ifadede aşağıdaki aritmetik operatörlere izin verilmektedir:

499

+ ekle

- çıkarma

\* çarpma

/ bölme

^ üssel

% mod

yığın için aşağıdaki tanımlamalar yapılmalıdır.

struct yiginDugum {

char veri;

struct yiginDugum \*sonrakiPtr;

};

typedef struct yiginDugum YIGINDUGUM;

typedef YIGINDUGUM \*YIGINDUGUMPTR;

Programınız main fonksiyonu ve aşağıda başlıkları verilen sekiz fonksiyonu içermelidir.

void sonekeCevir(char ortaek[], char sonek[])

ortaek gösterimini sonek gösterimine çevir.

int operator(char c)

c’ nin operatör olup olmadığına karar ver.

int oncelik(char operator1, char operator2)

Operator1’in öncelik sırası operatör2’den küçük, büyük ya da operator2’ye eşit olup

olmadığına karar verir ve –1,0 ya da 1 geri döndürür.

void push(YIGINDUGUMPTR \*ustPtr, char deger)

Yığına bir değer at

char pop(YIGINDUGUMPTR \*ustPtr)

Yığından bir değer al

char ustYigin(YIGINDUGUMPTR ustPtr)

Yığından alma işlemi yapmadan, en üst değeri döndür.

int bos(YIGINDUGUMPTR ustPtr)

Yığının boş olup olmadığına karar ver.

void yiginYazdir(YIGINDUGUMPTR ustPtr)

Yığını yazdır.

12.13 Bir sonek ifadesini hesaplayan program yazınız.

6 2 + 5 \* 8 4 / -

Programınız bir karakter dizisine rakamlardan ve operatörlerden oluşan bir sonek ifadesini

almalıdır. Bu ünitede daha önceden değiştirilmiş olan yığın fonksiyonlarını kullanarak ifadeyi

tarayın ve çalıştırın. Algoritma aşağıdaki gibidir:

1) sonek ifadesinin sonuna (‘\0’) NULL karakterini ekleyin. Böylece null karakteri

okunduğuna işlem duracaktır.

2) ‘\0’ karakteri okunmadığı sürece, ifade soldan sağa okunmalıdır.

Eğer okunan karakter bir rakamsa

500

rakamın tamsayı değerini yığına it(push)(bir rakam karakterinin tamsayı

değeri, bilgisayardaki karakter kümesi değerinden ‘0’ ın karakter

kümesindeki değerin çıkarılmasına eşittir.)

Eğer okunan karakter bir operatör ise

Yığını en üstteki iki elemanını x ve y değişkenlerine çek(pop).

y operatör x işlemini yap.

Sonucu yığına it(push).

3) NULL karakteri işlendiğinde ise yığındaki değeri alın Bu sonek ifadesinin

sonucudur.

Not: 2.Şıkta, eğer operatör ‘/’ ise yığının en üstü 2’dir ve yığında bir sonraki eleman 8’dir.

Yani x’e 2’yi çekin(pop) ve y’ye 8’i çekin(pop). 8/2 işlemini gerçekleştirin ve sonucu yani

4’ü yığına itin(push). Bu not ‘-‘ operatörünede uygulanabilir. Bu ifadede kullanılmasına izin

verilen aritmetik operatörler aşağıdaki gibidir:

+ ekle

- çıkarma

\* çarpma

/ bölme

^ üssel

% mod

yığın için aşağıdaki tanımlamalar yapılmalıdır.

struct yiginDugum {

int veri;

struct yiginDugum \*sonrakiPtr;

};

typedef struct yiginDugum YIGINDUGUM;

typedef YIGINDUGUM \*YIGINDUGUMPTR;

Programınız main fonksiyonu ve aşağıda başlıkları verilen sekiz fonksiyonu içermelidir.

int sonekIfadesiniCalistir(char \*ifade)

sonek ifadesini çalıştır.

int hesapla(int op1, int op2, char operator)

op1 operator op2 ifadesini çalıştır.

void push(YIGINDUGUMPTR \*ustPtr, int deger)

Yığına bir değer at

int pop(YIGINDUGUMPTR \*ustPtr)

Yığından bir değer al.

int bos(YIGINDUGUMPTR ustPtr)

Yığının boş olup olmadığına karar ver.

void yiginYazdir(YIGINDUGUMPTR ustPtr)

Yığını yazdır.

12.14 Alıştırma 12.13’deki sonek çalıştırma programını 9’dan daha büyük tamsayılarda

kullanabilecek şekilde değiştiriniz.

501

12.15 (Süper market simülasyonu) Bir süper marketteki para ödeme sırasının simülasyon

programını yazınız. Sıra kullanmalısınız. Müşteriler 1 ile 4 dakika, rasgele tamsayı aralığında

gelmektedirler. Tabiki müşterilere 1 ile 4 dakika rasgele tamsayı aralığında servis

verilmektedir. Sonuç olarak, bu oranlar dengelenmelidirler. Eğer ortalama müşteri gelme

oranı, ortalama servis oranından büyükse, sıra sonsuza gidebilir. Ancak, oranlar dengeli olsa

bile, rasgelelikten dolayı sıra yine sonsuza gidebilir. Süpermarket simülasyonunuzu 12-saatlik

bir gün(720 dakika) için aşağıdaki algoritmayı kullanarak çalıştırın:

1) 1 ile 4 arasında rasgele bir tamsayı üreterek ilk müşterinin geldiği dakikayı

hesaplayın.

2) İlk müşterinin geldiği dakikada:

Müşterinin servis süresini hesaplayın(1 ile 4 arasında rasgele tamsayı)

Müşteriye servis vermeye başlayın.

Bir sonraki müşterinin gelme zamanını hesaplayın(o andaki zamana 1 ile 4

arasında rasgele bir tamsayının eklenmesi)

3) Günün her dakikası için

Eğer bir sonraki müşteri gelirse

Sıraya sokun

Bir sonraki müşterinin gelme zamanını hesaplayın

En son müşteri için servis süresi bittiyse

Servis yapılacak bir sonraki müşteriyi alın

Müşterinin servis süresini hesaplayın(o andaki zamana 1 ile 4 arasında

rasgele bir tamsayının eklenmesi)

Şimdi simülasyon programınızı 720 dakika için çalıştırınız ve aşağıdakilere cevap veriniz.

a) Her hangi bir zamanda sırada en fazla kaç kişi bulunabildi?

b) En uzun süre bekleyen müşteri ne kadar beklemiştir?

c) Müşteri gelme zaman aralığı 1 ile 4 arasından 1 ile 3 arasına düşürülürse ne olur?

12.16 Şekil 12.19’daki programı ikili ağacın aynı değerleri içerebileceği şekilde değiştiriniz.

12.17 Şekil 12.19’daki programdan yararlanarak, bir metin girdisi yaptıran ve bu metni

kelimelerine ayırıp ikili arama ağacına yerleştirdikten sonra ağacı oncesol, oncedugum,

oncecocuk ilerleme ile yazdıran bir program yazınız.

İpucu: Metni bir diziye yazınız. strtok kullanarak kelimelerine ayırınız. Her kelime

bulunduğunda ağaçta yeni bir düğüm yaratınız ve strtok tarafından döndürülen göstericiyi

string üyesine atayınız yeni düğümü ağaca ekleyiniz.

12.18 Bu ünitede ikili bir arama ağacı yaratırken kopya değerlerin elenmesinin oldukça basit

olduğunu gördük. Tek belirteçli bir dizide kopya eleme işlemini nasıl

gerçekleştirebileceğimizi tanımlayınız. Diziler kullanılarak gerçekleştirilen kopya eleme ile

ikili ağaç kullanılarak gerçekleştirilen kopya elemenin performansını karşılaştırınız.

12.19 derin isminde bir ikili ağaç alan ve ağacın kaç seviyesi olduğunu bulan bir fonksiyon

yazınız.

502

12.20 (Yinelemeli olarak bir listeyi tersten yazdırmak) listeyiTerstenYazdir isminde listeyi

ekrana tersten yazdıran bir fonksiyon yazınız. Fonksiyonunuzu sıralı bir liste oluşturan bir

programda listeyi tersten yazdırarak deneyiniz.

12.21 (Bir listeyi yinelemeli olarak aramak) listeAra isminde, bir bağlı listede belli bir değeri

yinelemeli arayan bir fonksiyon yazınız. Eğer değer bulunduysa fonksiyon ilgili göstericiyi

döndürmeli. Eğer bulunmadıysa, NULL döndürmeli. Bir tamsayı listesi oluşturan bir

programda, belli bir değeri aramak için fonksiyonunuzu test ediniz.

12.22 (İkili ağaç silme) Bu alıştırmada ikili ağaçtan parçalar sileceğiz. Silme algoritması,

ekleme kadar kolay değil. Silme işleminde uygulanacak üç durum vardır. – parçanın yaprak

düğümünde saklanmış olması (çocuğu olmaması), bir çocuğu olan bir düğümde saklanmış

olması ve iki çocuğu olan bir düğümde saklanmış olması.

Eğer silinecek parça bir yaprak düğümünde ise, düğüm silinmeli ve ebeveyn düğümünün

göstericisi NULL yapılmalıdır.

Eğer silinecek parça bir çocuğu olan bir düğümde ise, ebeveyn düğüm göstericisi, çocuğa

atanmalı ve ilgili parça silinmelidir. Bu, çocuk düğümünün silinen düğümün yerini almasını

sağlayacaktır.

Son durum ise en zor durumdur. İki çocuğu olan bir düğüm silinirse, mutlaka onun yerini

başka bir düğüm almalıdır. Ancak, ebeveyn göstericisini, silinecek çocuğun birine atamak

kolay değildir. Bir çok durumda sonuçta elde edilen ağaç, ağaçların şu karakteristiğine

uymamaktadır: Her hangi bir sol ağaççıktaki değer, ebeveyn düğümdeki değerden küçük

olmalı ve sağ ağaççıktaki değer ebeveyn düğümdeki değerden büyük olmalıdır.

Hangi düğüm bu karakteristiğe uyacak şekilde yer değiştirme düğümü olarak kullanılacaktır?

Ağaçtaki en büyük değeri ve silinecek düğümden küçük olan değeri içeren düğüm mü yoksa

ağaçtaki en küçük değeri ve silinecek düğümden büyük olan değeri içeren düğüm mü

silinmelidir ? Daha küçük olan değeri içeren düğümü ele alalım. Bir ikili arama ağacında,

ebeveyn değerinden küçük olan en büyük değer, ebeveyn düğümünün sol ağaççığında ve

ağaççığın sağ düğümünde olduğu kesindir. Bu düğüm, sol ağaççıktan sağa aşağı doğru o anki

düğümün sağ çocuğunun göstericisi NULL olana dek ilerleyerek bulunabilir. Şu anda, yer

değiştirme işleminde kullanacağımız, bir yaprak düğümü ya da bir çocuklu bir düğümün

kendisini göstermekteyiz. Eğer yer değiştirme işleminde kullanacağımız düğüm bir yaprak

düğüm ise silme işleminin basamakları aşağıdaki gibi olmalıdır.

1) Silinecek düğümün göstericisini, geçici bir gösterici değişkeninde saklayınız.( bu

değişken dinamik hafıza tahsisini serbest bırakmak için kullanılacaktır.)

2) Silinecek düğümün ebeveyn düğümünün göstericisini, yer değiştirme işleminde

kullanılacak düğümü gösterecek şekilde atayınız.

3) Yer değiştirme işleminde kullanılan düğümün ebeveyn düğümüne NULL atayınız.

4) Yer değiştirme işleminde kullanılan düğümün sağ ağaççık göstericisini silinecek

düğümün sağ ağaççığını gösterecek şekilde atayınız.

5) Geçici göstericinin gösterdiği düğümü siliniz.

Sol çocuğu olan yer değiştirme işleminde kullanılacak olan düğümün silinme basamakları

çocuğu olamayan yer değiştirme işleminde kullanılacak olan düğümün silinme basamaklarına

benzerdir ancak algoritma, çocuğu yer değiştirme işleminde kullanılan düğümün konumuna

503

taşımalıdır. Eğer yer değiştirme işleminde kullanılacak olan düğüm sol çocuğu olan bir

düğüm ise izlenecek basamaklar aşağıdaki gibi olmalıdır.

1) Silinecek düğümün göstericisini geçici bir gösterici değişkeninde saklayınız.

2) Silinecek düğümün ebeveyn düğümünün göstericisini yer değiştirme işleminde

kullanılacak olan düğümü gösterecek şekilde atayınız.

3) Yer değiştirme işleminde kullanılacak olan düğümün ebeveyn göstericisini, yer

değiştirme işleminde kullanılacak olan düğümün sol çocuğunu gösterecek şekilde

atayınız.

4) Yer değiştirme işleminde kullanılacak olan düğümün göstericisini sağ ağaççığını

gösterecek şekilde atatınız ki silinecek düğümün sağ ağaççığını göstersin.

5) Geçici göstericinin gösterdiği düğümü siliniz.

silDugum fonksiyonunu, ağacın kök düğümünü göstericisini ve silinecek değeri argüman

olarak alacak şekilde yazınız. Fonksiyon, silinecek değeri içeren düğümün konumunu bulmalı

ve yukarıda anlatılan algoritmaların uygulanmasıyla ilgili düğümü silmelidir. Eğer aranan

değer bulunamazsa, fonksiyon ekrana ilgili değerin bulunamadığını belirten bir mesaj

yazdırmalıdır. Şekil 12.19’daki programı bu fonksiyonu kullanacak şekilde değiştiriniz. Bir

parçayı sildiğinizde,oncesol, oncedugum, oncecocuk fonksiyonları çağırarak silme işleminin

doğru yapılıp yapılmadığını kontrol ediniz.

12.23 (İkili Arama Ağacı) Bir ikili arama ağacında istenen değerin konumunu bulan bir

ikiliAramaAgaci fonksiyonu yazınız. Fonksiyonunuz kök düğümü göstericisini ve konumu

bulunacak arama anahtarını argüman olarak almalıdır. Eğer aranan anahtar kelimeyi içeren

düğüm bulunursa ilgili gösterici döndürülmeli, bulunamazsa NULL karakteri

döndürülmelidir.

12.24 (Seviye sıralı ikili ağaçta ilerleme) Şekil 12.19’daki program ikili ağaçta ilerlemenin,

yinelemeli üç yöntemini göstermektedir. Bu alıştırma, düğüm değerlerinin, kök düğüm

seviyesinden başlanarak, seviye seviye yazdırılmasını gerçekleştiren seviye sıralı ilerlemeyi

sormaktadır. Her seviyedeki düğümler, soldan sağa yazdırılmaktadır. Seviye sıralı ilerleme

yinelemeli bir algoritma değildir. Düğümlerin yazdırılmasını sağlamak için sıra veri yapısını

kullanmaktadır. Algoritma aşağıdaki şekildedir:

1) Kök düğümü sıraya ekle

2) Sırada düğüm kaldığı sürece

Sıradaki bir sonraki düğümü al

Düğümün değerini yazdır

Eğer düğümün sol çocuğunu gösteren gösterici NULL değilse

Sol çocuk düğümünü sıraya ekle

Eğer düğümdeki sağ çocuğu gösteren gösterici NULL değilse

Sağ çocuk düğümünü sıraya ekle

Tüm algoritmayı gerçekleştiren seviyeSirali fonksiyonunu yazınız. Fonksiyon, ikili ağacın

kök düğümünü gösteren bir göstericiyi argüman olarak almalıdır. Şekil 12.19’daki programı

bu fonksiyonu kullanacak şekilde değiştiriniz. Bu fonksiyonun çıktısıyla diğer ilerleme

algoritmalarının çıktılarını karşılaştırarak fonksiyonunuzun doğru bir şekilde çalışıp

çalışmadığını kontrol ediniz.

504

12.25 (Ağaçları yazdırmak) agacYazdir isminde bir ikili ağacı ekrana yazdıracak olan

yineleme fonksiyonunu yazınız. Fonksiyonunuz satır satır, ağacın en üstü ekranın en solunda

ve en altı ekranın en sağında olacak şekilde ağacı yazdırmalıdır. Her satır, dikey olarak ifade

edilecektir. Örneğin Şekil 12.22’deki ikili ağacın ekrana yazdırılması şu şekilde olur.

99

97

92

83

72

71

69

49

44

40

32

28

19

18

11

En sağdaki yaprak düğüm, çıktıda en sağ sütunda ve en üste görülmektedir. Kök düğümü ise

en solda görülmektedir. Sütunlar arasında beşer satır boşluk vardır. agacYazdir fonksiyonu

argüman olarak, ağacın kökünü gösteren göstericiyi ve toplamBosluk adında, yazdırılacak bir

değeri geriden takip eden kaç boşluk olduğunu gösteren tamsayı değişkenini alır(değişkenin

ilk değeri sıfır olmalıdır ki kök düğümü ekranın en solundan başlasın). Fonksiyon, sonradan

değiştirilmiş,oncesol ilerleme çıktı için kullanmalıdır. – ağacın en sağdaki düğümünden

başlamalı ve en sola kadar gelmelidir. Algoritma aşağıdaki gibidir:

O anda üzerinde bulunulan düğümün göstericisi null olmadığı sürece

agacYazdir fonksiyonunu, o anki düğümün sağ ağaççığı ve topamBosluk + 5 ile

yinelemeli çağır.

1’den topamBosluk’a kadar sayacak ve boşluk bırakacak bir for döngüsü kullanın.

O anda üzerinde bulunulan düğümün değerini yazdırın.

O anda üzerinde bulunulan düğümün göstericisini, aynı düğümünü sol ağaççığını

gösterecek şekilde atayın ve topamBosluk değişkenini 5 artırın.

ÖZEL KISIM: KENDİ DERLEYİCİNİZİ YAZMAK

Alıştırma 7.18 ve 7.19’da Simpletron Makine Dilini(SMD) ve SMD’de yazılan programları

çalıştırabilmek için Simpletron bilgisayar simülasyon programını yazmıştık. Bu kısımda ise,

yüksek seviyeli bir programlama dilinde yazılmış olan programı, SMD’ye çeviren bir

derleyici yazacağız. Böylece, yeni yüksek seviyeli dilimizde programlarımızı oluşturacağız,

derleyeceğiz ve Alıştırma 7.19’da oluşturduğumuz simülasyon programıyla çalıştıracağız.

505

12.26 (Simple Dili) Derleyiciyi yazmaya başlamadan önce, basit, hala güçlü ve yüksek

seviyeli dilimizden bahsedeceğiz. Kullanacağımız dil, BASIC dilinin eski sürümlerine

benzemektedir. Dilimize simple diyeceğiz. Her Simple ifadesi, bir satır numarasından, ve bir

simple komutundan oluşur. Satır numaraları mutlaka artan sırada kullanılmalıdır. Her komut

şu emirlerin herhangi biriyle başlamaktadır: rem, input, let, printf, goto, if/goto, yada end

(Şekil 12.23’e bakınız). end hariç diğer komutlar programda tekrar tekrar kullanılabilir.

Simple sadece +, -, \* ve / aritmetik operatörleri ile işlem yapabilir. Bu operatörler C’de aynı

önceliğe sahiptirler ve parantez kullanılarak öncelik sıraları değiştirilebilir.

Simple derleyicimiz sadece küçük harflerden anlamaktadır. Bir Simple dosyasındaki bütün

karakterler küçük harf olmak zorundadır(rem ifadesinin sonuna yazılan hatırlatma komut

satırları haricinde, çünkü bunlar ihmal edileceklerdir). Değişken isimleri bir harf olmalıdır.

Simple daha açıklayıcı değişken isimleri kullanmaya izin vermemektedir. Bu yüzden

değişkenler için hatırlatma satırları kullanılmalıdır. Simple dilinde, değişken bildirmeye

gerek yoktur. Programınızda bir değişken kullandığınızda otomatik olarak tanımlanır ve ilk

değeri sıfır atanır. Simple yazımı, stringler ile uygulamalar yapmaya izin vermez(string

okuma, yazma, karşılaştırma vb.) rem haricinde bir komuttan sonra eğer bir string girilirse,

derleyici yazım hatası mesajı verir.

Komut Örnek ifade Açıklama

rem 50 rem bu hatırlatmadır rem komutundan sonra yazılan sadece

bilgi vermek içindir ve derleyici

tarafından ihmal edilirler.

input 30 input x Ekrana bir ? işareti yazdırır ve

kullanıcının bir tamsayı girmesini bekler.

Bu tamsayıyı x değişkenine atar.

let 80 let u = 4 \* ( j – 56) u’ya 4 \* (j – 56) değerini ata demektir.

eşitliğin sağ tarafında istenildiği gibi

karışık bir ifade bulundurulabilir.

printf 10 print w w’nin değerini ekrana yazdırır.

goto 70 goto 35 program kontrolünü satır 45’e taşır.

if/goto 35 if i == z goto 80 i ve z’yi eşit olup olmadıklarını anlamak

için karşılaştır. Eğer koşul doğru ise

program kontrolünü satır 80’e taşı, değilse

bir alt satırdan devam et.

end 99 end Programı sonlandır.

Şekil 12.23 Simple komutları

Derleyicimiz, programın doğru girildiğini kabul etmektedir. Alıştırma 12.29’da derleyicinin

hata kontrolü yapması da istenecektir.

Simple, program akışını kontrol etmek için if/goto koşullu ifadesini ve goto koşulsuz ifadesini

içermektedir. Eğer if/goto koşulu doğru ise program istenilen satırdan çalışmaya devam eder.

<, >, <=, >=, ==, ya da != operatörlerini C ile aynı öncelik sırasını içerecek şekilde

kullanabilirsiniz.

506

Şimdi Simple’ın özelliklerini gösteren bir kaç Simple programı yazalım. İlk program(Şekil

12.24), klavyeden iki tamsayı alır, bunları a ve b değişkenlerinde saklar. Bu iki tamsayının

toplamını hesaplar ve ekrana yazdırır(c değişkeninde de saklar).

Şekil 12.25’deki program ise iki tamsayının büyüğünü yazdırır. Tamsayılar, klavyeden

alınarak s ve t değişkenlerinde saklanmaktadır. if/goto ifadesi s >= t koşulunu kontrol

etmektedir. Eğer koşul doğru ise, program kontrolü satır 90’a taşınır. Eğer doğru değilse, çıktı

olarak t’nin yazdırılması gerekmektedir. t yazdırılır ve program kontrolü satır 99’a taşınarak

program sonlandırılır.

1 10 rem iki tamsayının toplamının hesaplanması ve yazdırılması

2 15 rem

3 20 rem iki tamsayı girişi

4 30 input a

5 40 input b

6 45 rem

7 50 rem tamsayıları topla ve c değişkeninde sakla

8 60 let c = a + b

9 65 rem

10 70 rem sonucu yazdır

11 80 print c

12 90 rem program sonlandı

13 99 end

Şekil 12.24 İki tamsayının toplamının hesaplanması

1 10 rem iki tamsayının büyüğünün bulunması

2 20 input s

3 30 input t

4 32 rem

5 35 rem s>=t koşulunu kontrol et

6 40 if s >= t goto 90

7 45 rem

8 50 rem t, s’den büyüktür. O halde t’yi yazdır

9 60 print t

10 70 goto 99

11 75 rem

12 80 rem s, t’den büyük ya da t’ye eşittir. O halde s’i yazdır.,

13 90 print s

14 99 end

Şekil 12.25 İki tamsayının büyüğünün bulunması

Simple, döngü yapılarını içermez(C’de ki for, while ya da do/while gibi). Buna rağmen

if/goto ve goto ifadelerini kullanarak bu yapıların oluşturulması mümkündür. Şekil 12.26,

nöbetçi kontrollü bir döngü ile bir kaç sayının karesini hesaplamaktadır. Her tamsayı

klavyeden alınarak j değişkeninde saklanmaktadır. Eğer girilen tamsayı –9999 nöbetçi

değeriyse, program kontrolü satır 99’a geçer ve program sonlanır. Aksi takdirde, k

değişkenine j değişkeninin karesi atanır. k ekrana yazdırılır ve program kontrolü satır 20’ye

taşınır ve diğer tamsayı klavyeden alınır.

Şekil 12.24, Şekil 12.25 ve Şekil 12.26’taki örnek programlardan yararlanarak aşağıdakileri

gerçekleştiren bir program yazınız.

507

a) Üç tamsayı girişi yaptırın, ortalamalarını buldurun ve sonucu yazdırın.

b) Nöbetçi kontrollü bir döngü kullanarak 10 tamsayı girişi yaptırın toplamını hesaplayıp

ekrana yazdırın.

c) Sayıcı kontrollü bir döngü ile girişi yapılan, bazıları pozitif, bazıları negatif olan

tamsayıların ortalamalarını hesaplayınız.

d) Bir kaç tamsayı girişi yaptırarak bu tamsayıların en büyüğünü bulunuz. Girilen ilk

tamsayı, kaç sayı girileceğini göstersin.

e) 10 tamsayı girişi yaptırın ve en küçüğünü hesaplatın.

f) 2’den 30’a kadar olan çift tamsayıların toplamını bulduran bir program yazınız.

g) 1’den 9’a kadar olan tek tamsayıların çarpımını bulan bir program yazınız.

12.27 (Derleyici yazmak; Daha önceden Alıştırma 7.18, 7.19, 12.12, 12.13 ve 12.26 ‘yı

yapmanız gerekmektedir.) Artık simple dilini öğrendik, şimdi, derleyicimizi nasıl

yazacağımızı öğreneceğiz. Öncelikle, bir simple programının SMD’ye nasıl dönüştürüleceğini

ve Simpletron simülasyon programı tarafından nasıl çalıştırılacağını öğreneceğiz(Şekil

12.27’ye bakınız). Simple programını içeren dosya, derleyici tarafından okunmuş ve SMD

koduna çevrilmiştir. SMD kodu, her satırda bir komut olacak şekilde bir dosyaya yazılmıştır.

SMD dosyası saha sonra Simpletron simülasyon programına yüklenmiştir ve sonuçlar ekrana

ve bir dosyaya yazılmıştır. Alıştırma 7.19’daki program girişleri klavyeden yapacak şekilde

yazılmıştır. Bu program girişleri dosyadan alacak şekilde değiştirilerek derleyici tarafından

yazılan programları çalıştırabilmelidir.

Derleyici Simple programı SMD’ye çevirmek için program içersinde iki tur atar. İlk turda bir

sembol tablosu oluşturulur. Bu tabloda, tüm satır numaraları, değişken isimleri ve sabitleri

tipleriyle ve son SMD kodunda bulunacakları konumlarıyla beraber saklanırlar(Sembol

tablosu daha ayrıntılı bir şekilde aşağıda anlatılmıştır.). İlk turda aynı zamanda her Simple

ifadesi için SMD komutları da oluşturulur. Eğer, Simple programı kontrol transferi içeren

komutlar içeriyorsa ilk turda program tamamıyla SMD koduna dönüştürülemeyecektir. Eksik

kalan kısımlar ikinci turda çevrilecektir.

Şekil 12.27 Simple dilinde bir programın yazılması, derlenmesi ve çalıştırılması

İlk Tur

Derleyici Simple programının bir ifadesini hafızaya alarak başlar. Her satır mutlaka

işlenebilmesi için kelimelerine ayrılmalıdır(strtok standart kütüphane fonksiyonu bu iş için

kullanılabilir). Her ifadenin bir satır numarası ile başladığını sonra da komut içerdiğini

biliyoruz. Derleyici bir ifadeyi kelimelerine ayırdığında, eğer kelime satır numarası, değişken

Simple

Dosyası derleyici

SMD

dosyası

ekrana

çıktı al

diske

çıktı al

Simpletron

simülasyon

programı

508

ya da bir sabit ise sembol tablosuna yerleştirilir. Bir satır numarası eğer ifadenin ilk kelimesi

ise sembol tablosuna dahil edilir. sembolTablosu, programdaki sembolleri içeren tabloGirisi

yapısının bir dizisidir. Programda yer alabilecek sembol sayısı kısıtlı değildir. Yani, sembol

tablosu oldukça büyük olabilir. Şimdilik sembolTablosu dizisini 100 elemanı olacak şekilde

tanımlayınız. Program çalışırken bu uzunluğu değiştirebilirsiniz.

tabloGirisi yapısının tanımlaması aşağıdaki gibidir:

struct tabloGirisi {

int sembol;

char tip; /\* ‘C’, ‘L’, ya da ‘V’ \*/

int konum; /\* 00 ile 99 arasında \*/

}

Her tabloGirisi yapısı üç üye içerir. sembol üyesi, bir değişkenin, bir satır numarasının ya da

sabitin ASCII gösterimlerinin içerir. (değişken isimlerinin tek karakterde olduğunu

hatırlayınız), tip üyesi ise, sembolün tipini içerir: Sabit için ‘C’, değişken için ‘V’ ve satır

numarası için ‘L’. konum üyesi ise ilgili sembolün, Simpletron hafıza konumunu içerir(00 ile

99 arasında). Simpletron’un hafızası SMD komutlarının ve programının saklandığı 100

karakterlik bir dizidir. Bir satır numarası için hafızadaki konum, Simpletron hafıza dizisindeki

simple ifadesinin başladığı elemandır. Bir değişken ya da sabit için konum, Simpletron hafıza

dizisindeki değişkenin ya da sabitin saklandığı elemandır. Değişkenler ve sabitler Simpletron

hafızasına sondan başlayarak yukarı doğru yerleşirler. İlk değişken ya da sabit 99

konumunda, ikincisi 98 konumunda vs. saklanır.

Sembol tablosunun, simple programını SMD’ye çevrilmesinde birleştirici bir rolü vardır.

7.Ünitede bir SMD komutunun, 4 basamaklı bir tamsayı olduğunu öğrenmiştik – İki byte

işlem kodu ve iki byte operand. İşlem kodu, Simple komutlarıyla anlaşılmaktadır. Örneğin

input simple komutu, 10 SMD(oku) işlem koduna karşılık gelmekte ve print simple komutu

11 SMD(yaz) işlem koduna karşılık gelmektedir. Operand ise, işlem kodunun görevini

yapmak için kullanacağı hafıza konumunu içermekteydi(Öreğin 10 işlem kodu, klavyeden bir

değer almakta ve bunu operandın gösterdiği hafıza konumuna yazmaktaydı). Derleyici,

sembolTablosunda her sembolün ait olduğu hafıza konumunu arar ve ilgili hafıza konumu

SMD komutunun tamamlanmasında kullanılır.

Bir simple ifadesinin derlenmesi içerdiği komuta bağlıdır. Örneğin, bir rem ifadesinde satır

numarasında sonrası sembol tablosuna eklenir ve geri kalanı derleyici tarafından ihmal edilir.

Çünkü rem sadece, yorum satırları içerir. input, print, goto ve end ifadeleri SMD’de oku,

yaz, dallan(belli bir hafıza konumuna) ve bitir komutlarına karşılık gelmektedir. Simple

komutlarını içeren ifadeler, doğrudan SMD’ye çevrilirler( derleme sırası goto ifadesine

geldiğinde bu ifade, henüz işlenmemiş ilerideki bir satır numarasını içeriyor olabilir. Buna

bazen karar verilmemiş referans denir)

goto ifadesi karar verilmemiş referans ile derlendiğinde, SMD komutuna çevrimin ikinci

turda tamamlanması için mutlaka işaretlenmelidir. İşaretler, int tipindeki isaretler dizisinde

saklanmaktadır ve dizi elemanlarının ilk değerleri –1 olmalıdır. Simple programından gelen

bir satır numarasının hafıza konumu henüz bilinmiyorsa(örneğin, sembol tablosunda yoksa)

bu satır numarası, isaretler dizisinde, tamamlanmamış komutun belirteciyle saklanmalıdır.

Henüz çevirimi tamamlanmamış komutun operandı geçici olarak 00 yapılır. Örneğin bir

509

koşulsuz dallanma komutu(programın ileriye zıplamasını sağlıyor) iki tura dek +4000 olarak

bırakılmalıdır. İkinci tur kısaca açıklanacaktır.

if/goto ve let ifadelerinin derlenmesi diğerlerinden daha zor olacaktır, çünkü bu komutlar

birden fazla SMD komutu oluştururlar. Bir if/goto ifadesi için, derleyici, koşulu kontrol

edecek ve gerekirse başka bir satıra dallanacak kodu üretmelidir. Bu dallanma ise karar

verilmemiş referans içeriyor olabilir. Bütün karşılaştırma ve eşitlik operatörleri, SMD’nin

sıfırsa dallan ve negatifse dallan komutlarıyla(ya da ikisi ile beraber) ifade edilebilirler.

Bir let ifadesi için, derleyici değişkenler ve/veya sabitler içeren karışık bir aritmetik işlem

ifadesini gerçekleştirecek kodu yazmalıdır. Aritmetik işlem ifadelerinde operandlar ve

operatörler arasında boşluk olmalıdır. Alıştırma 12.12 ve 12.13 ortaek ve sonek çevirim

algoritmalarını içermektedir. sonek algoritması derleyiciler tarafından aritmetik işlemlerin

hesaplanmasında kullanılır. Derleyici yazma işleminde başlamadan önce bu alıştırmaları

çözmüş olmanız gerekmektedir. Derleyici bir aritmetik işlem ifadesiyle karşılaştığı zaman bu

ifadeyi ortaek gösteriminden sonek gösterimine çevirmeli ve daha sonra sonek gösterimindeki

ifadeyi hesaplamalıdır.

Bir derleyici, değişkenler içeren bir ifadenin makine dili kodunu nasıl oluşturur? sonek

ifadesinin hesaplanma algoritması bir “kanca” içerir. Bu kanca, SMD komutlarının aslında

aritmetik ifadeyi çalıştırmadan oluşturulmasını sağlar. Bu kancanın derleyici tarafından

kullanılabilmesi için, sonek ifadelerini hesaplama algoritmasını, karşılaştığı her sembolü

sembol tablosunda arayacağı(belki ekleyeceği) ve ilgili sembolün hafıza konumu bulup

sembolün aksine bu konumu yığına iteceği(push) şekilde değiştirmek gerekmektedir. sonek

ifadesinde eğer bir operatörle karşılaşılırsa yığının en üst iki hafıza konumu çekilmeli(pop) ve

bu işlemi oluşturmak için gereken makine dili kodu, bu hafıza konumlarının operand olarak

kullanılmasıyla oluşturulmalıdır. Her iki ifadeciğin sonucu hafızada geçici bir konumda

saklanmalı ve sonek ifadesinin hesaplanmasına devam edilebilmesi için yığına

itilmelidir(push). sonek ifadesinin hesaplanması bittiğinde, sonucu içeren hafıza konumu,

yığında kalan tek hafıza konumudur. Bu konum çekilir (pop) ve sonucu let ifadesinin solunda

yer alan değişkene atanması için gereken makine dili komutları üretilir.

İkinci Tur

Derleyicinin ikinci turda iki görevi vardır: karar verilmemiş referanslar için karar vermek ve

SMD kodunun çıktısını bir dosyaya almak. Karar verilmemiş referanslar için karar vermek

aşağıdaki gibi olur:

1) isaretler dizisinde karar verilmemiş referansı ara(-1’den farklı bir değer içeren eleman)

2) isaretler dizisinde saklanan sembolü içeren yapıyı sembolTablosu dizisinde

bul(sembolün tipinin, satır numarası anlamına gelen ‘L’ olduğundan emin ol)

3) Yapının konum üyesinden alından hafıza konumunu komuta karar verilmemiş referans

ile birlikte ekle(karar verilmemiş referans içeren komutun 00 operandını içerdiğini

hatırlayınız).

4) isaretciler dizisinin sonuna gelinceye dek 1,2 ve 3. adımları tekrarlayın.

Bu işlemler bittikten sonra SMD kodunu içeren tüm dizi, her satırda bir SMD komutu olacak

şekilde bir dosyaya yazılmalıdır. Simpletron bu dosyayı çalıştırmak için okuyacaktır

(Simülasyon programının girdilerinin klavye yerine dosyadan yapılabilmesi için gerekli

değişiklikler yapıldıktan sonra)

510

Tam bir Örnek

Sıradaki örnek, bir simple programının SMD’ye Simple derleyicisi tarafından tamamıyla

dönüşümünü göstermektedir. Program bir tamsayı girişi yaptıracak ve 1’den bu tamsayıya

kadar olan sayıların toplamını hesaplayacaktır. İlk turda oluşturulacak SMD komutları Şekil

12.28’de gösterilmiştir. İlk turda oluşturulan sembol tablosu ise Şekil 12.29’da gösterilmiştir.

rem, 20.satırdaki if/goto ifadesi ve let ifadeleri haricinde diğer Simple komutları, SMD

komutlarına doğrudan çevrilmiştir. Hatırlatma satırları makine diline çevrilmezler. Buna

rağmen satır numaraları sembol tablosuna dahil edilir. Çünkü bu satır numarasına if/goto ya

da goto ifadeleriyle tekrar geri dönülebilir. Programda satır 20, eğer y == x koşulu doğruysa

program kontrolünün, satır 60’a taşınması gerektiğini belirtmektedir. Programın derlenmesi

sırasında henüz satır 60’a gelinmediği için, ilk turda sembol tablosuna bu satır numarası

eklenmez(Satır numaraları sadece ifadelerin başında bulunduklarında sembol tablosuna

eklenirler). Sonuç olarak, SMD komutları dizisinin 03 konumundaki SMD sıfırsa dallan

komutunun operandının şu anda belirlenmesi mümkün değildir. Derleyici isaretler dizisinin

03 konumuna 60 koyarak, bu işlemin ger kalanını ikinci turda tamamlayacaktır.

Adım adım Derleme İşlemleri

Şekil 12.28’deki programın derlenmesini satır satır ele alalım. Derleyici programın ilk satırını

hafızaya okur:

5 rem 1’den x’e kadar topla

strtok fonksiyonu ile ilk kelime(satır numarası) belirlenir (Stringlerle ilgili fonksiyonlar için

8.Üniteye bakınız).

Simple program SMD konumu Açıklama

ve Komutlar

5 rem 1’den x’e kadar topla yok rem ihmal edildi

10 input x 00 +1099 99 konumuna x’i oku

15 rem y==x koşulunu kontrol et yok rem ihmal edildi

20 if y == x goto 60 01 +2098 y(98)’i akümülatöre yükle

02 +3199 x(99)’u akümülatörden çıkart

03 +4200 sıfırsa karar verilmemiş referans

25 rem y’yi bir artır yok dallan

30 let y = y + 1 04 +2098 y’yi akümülatöre yükle

05 +3097 akümülatöre 1(97)

06 +2196 geçici bir hafıza konumunda

sakla(97)

07 +2096 96 geçici konumundan yükle

08 +2198 akümülatörü y’ye sakla

35 rem y’yi toplama ekle none rem ihmal edildi

40 let t = t + y 09 +2095 t(95)’yi akümülatöre al

511

10 +3098 y’yi akümülatöre ekle

11 +2194 94 geçici konumunda sakla

12 +2094 94 geçici konumundan yükle

13 +2195 akümülatörü t’de sakla

45 rem y’ye git none rem ihmal edildi.

50 goto 20 14 +4001 01 konumuna dallan

55 rem sonucun çıktısını al none rem ihmal edildi.

60 print t 15 +1195 t’yi ekrana yazdır.

99 end 16 +4300 programı sonlandır

Şekil 12.28 Derleyicinin ilk turundan sonra oluşan SMD komutları

strtok ile döndürülen kelime atoi fonksiyonun kullanılmasıyla sembol tablosuna aranmak

üzere tamsayıya çevrilir. Eğer sembol bulunamazsa tabloya eklenir. Programın henüz başında

olduğumuz için ve bu satır ilk satır olduğu için tablo henüz boştur. Böylece, 5 sembol

tablosuna L(satır numarası) tipinde eklenir ve SMD dizisinin ilk konumuna atanır(00). Bu

satır bir hatırlatma satırı olmasına rağmen satır numarası sembol tablosuna eklenmiştir( çünkü

daha sonra if/goto yada goto ifadeleriyle bu satıra dönüş mümkün olabilir). rem için her

hangi bir SMD komutu üretilmez ve komut sayıcısı artırılmaz.

Daha sonra

10 input x

ifadesi kelimelerine ayrılır. Satır numarası 10, L tipinde sembol tablosuna eklenir ve SMD

dizisinin ilk konumuna atanır.(00 çünkü, program hatırlatma satırı ile başladı ve komut

sayıcısı hala 00). input komutu, bundan sonra gelecek kelimenin bir değişken olduğunu

gösterir(bir input ifadesinde sadece bir değişken yar alır). Çünkü input doğrudan bir, SMD

işlem kodunu gösterir. Derleyici kolayca x’in, SMD dizisindeki konumunu bulmalıdır.

Sembol Tip Konum

5 L 00

10 L 00

‘x’ V 99

15 L 01

20 L 01

‘y’ V 98

25 L 04

30 L 04

1 C 97

35 L 09

40 L 09

‘t’ V 95

45 L 14

50 L 15

55 L 15

99 L 16

Şekil 12.29 Şekil 12.28’teki programın sembol tablosu

512

x sembolü sembol tablosunda bulunmadığı için ASCII gösterimi V tipinde tabloya eklenir ve

SMD dizisinde konum 99’a atanır.(veri depolama 99’dan başlar ve yukarı doğru tahsis

edilmiştir) Artık bu ifade için SMD kodu oluşturulabilir. İşlem kodu 10( SMD oku işlem

kodu) 100 ile çarpılır ve x’in konumu(sembol tablosundan bulunur) komuta eklenir. Daha

sonra bu komut SMD dizisinin 00 konumunda saklanır. Komut sayıcı ise 1 artırılır, çünkü bir

SMD komutu oluşturulmuştur.

Daha sonra

15 rem y==x koşulunu kontrol et

ifadesi kelimelerine ayrılır. Sembol tablosunda satır numarası 15 aranır(bulunamaz). Satır

numarası L tipinde sembol tablosuna eklenir ve dizideki bir sonraki konuma, 01 atanır. (rem

ifadeleri için kod üretilmez, yani komut sayıcısı artırılmaz.

Daha sonra

20 if y == x goto 60

kelimelerine ayrılır. Satır numarası 20, sembol tablosuna, L tipinde yazılır ve SMD dizisinde

01 konumuna atanır. if ifadesinden sonra bir koşul gelecektir. y değişkeni sembol tablosunda

bulunmadığı için V tipinde eklenir ve SMD dizisinde 98 konumuna atanır. Daha sonra, koşul

için SMD komutları hazırlanacaktır. if/goto ifadesi için doğrudan bir SMD komutu yoktur.

Ancak bu ifade, x ve y arasında bazı hesaplamaların yapılması ve dallanma sayesinde

gerçekleştirilebilir. Eğer y, x’e eşitse, farkları sıfırdır. Böylece sıfırsa dallan komutu

kullanılabilir. İlk adımda y(SMD dizisinde konum 98) akümülatöre yüklenmelidir. Bu 01

+2098 komutunun oluşmasını sağlar. Daha sonra, x, akümülatörden çıkarılmalıdır. Böylece

02 +3199 komutu oluşturulur. Akümülatördeki değer, pozitif, negatif veya sıfır olabilir.

Operatör == olduğu için sıfırsa dallan kullanılacaktır. İlk önce, sembol tablosunda, dallanma

konumu aranır(bu durumda 60) ve bulunamaz. Bu yüzden, 60, isaretler dizisinin, 03

konumuna atanır ve 03 +4200 komutu oluşturulur(dallanma konumunu ekleyemeyiz, çünkü

SMD dizisinde satır 60 için bir konum henüz belirtilmemiştir). Komut sayıcı artırılarak 04

olur.

Derleyici,

25 rem y’yi bir artır

ifadesine geçer. Satır numarası 25, L tipinde sembol tablosuna eklenir ve 04 SMD konumuna

atanır. Komut sayıcısı artırılmaz.

30 let y = y + 1

ifadesi kelimelerinde ayrıldığında, satır numarası 30, L tipinde sembol tablosuna yazılır ve 04

SMD konumuna atanır. let komutu bu satırın bir atama ifadesi içerdiğini gösterir. İlk olarak o

satırda bulunan bütün semboller sembol tablosuna eklenirler(daha önce eklenmedilerse).1

tamsayısı, C tipinde sembol tablosuna eklenir ve SMD dizisi 97 konumuna atanır. Sonra,

atama ifadesinin sağ tarafı, ortaek gösteriminden sonek gösterimine çevrilmiştir ve sırada bu

ifadenin(y 1 +) hesaplanması vardır. y sembolünün sembol tablosunda konumu bulunmuştur

ve ona ait olan hafıza konumu yığına itilir(push). 1 sembolünün de sembol tablosunda

513

konumu bulunmuştur ve ona ait olan hafıza konumu yığına itilir(push). + operatörü ile

karşılaşıldığında yığını operatörün sağ operandına çeker(pop) ve tekrar yığını operatörün sol

operandına çeker(pop). ve aşağıdaki SMD komutları oluşturulur.

04 +2098 ( yukle y )

05 +3097 (topla 1)

Bu ifadenin sonucu ve

06 +2196 (geçici olarak sakla)

komutu (96)geçici hafıza konumunda saklanır ve bu geçici olarak kullanılan konum yığına

itilir(push). sonek ifadesinin sonucun hesaplanmıştır ve y değişkeninde saklanmalıdır(=’in

solundaki değişken). Böylece, geçici konum, akümülatöre yüklenir ve aşağıdaki komutlarla

akümülatör y’de saklanır.

07 +2096 (yukle geçici konum)

08 +2198 (sakla y)

Bu komutların fazladan yazıldığını, yani bu işlemlerin daha kısa kodlarla yapılabileceğini fark

etmiş olmalısınız. Bu konuyu ileride kısaca anlatacağız.

Daha sonra,

35 rem y’yi toplama ekle

kelimelerine ayrılacaktır. satır numarası 35, L tipinde sembol tablosuna eklenir ve 09

konumuna atanır.

40 let t = t + y

ifadesi satır 30’a benzer. t değişkeni, sembol tablosuna V tipinde yazılır ve 95 SMD

konumuna atanır. Bu komut, satır 30’la aynı mantıkta derlenir ve 09 +2095, 10 +3098, 11

+2194, 12 +2094, 12 +2094 ve 13 +2195 komutları oluşturulur. t + y’ nin sonucu t(95)’e

atanmadan önce 94 geçici hafıza konumuna atanır. 11 ve 12 hafıza konumundaki komutların

yine fazladan yazıldığını, yani bu işlemlerin daha kısa kodlarla yapılabileceğini fark etmiş

olmalısınız. Bu konuyu ilerde kısaca anlatacağız.

45 rem y’ye git

ifadesi de bir hatırlatma satırıdır. Böylece satır numarası 45, sembol tablosuna L tipinde

eklenir ve SMD dizisi konum 14’e atanır.

50 goto 20

ifadesiyle kontrol, satır 20’ye taşınır. Satır numarası 50, sembol tablosuna L tipinde eklenir

ve SMD dizisi konum 14’e atanır. goto komutunun SMD karşılığı koşulsuz dallanmadır(40).

Koşulsuz dallanma, program kontrolünü, belli bir SMD konumuna taşır. Derleyici, sembol

tablosunda satır numarası 20’yi arar ve SMD dizisinde konum 01’de olduğunu bulur. 40 işlem

numarası, 100 ile çarpılır ve 01 konumu bu komuta şu şekilde eklenir: 14 +4001.

514

55 rem sonucun çıktısını al

ifadesi bir hatırlatma ifadesidir. Böylece satır numarası 55, sembol tablosuna L tipinde eklenir

ve SMD dizisi konum 15’e atanır.

60 print t

çıktının alındığı ifadedir. Satır numarası 60, sembol tablosuna L tipinde eklenir ve SMD dizisi

konum 15’e atanır. print’in SMD karşılığı işlem kodu 11’dir(yaz). t’nin konumu sembol

tablosunda bulunur ve 100 ile çarpılan işlem koduna eklenir.

99 end

programın son satırıdır. Satır numarası 99, sembol tablosunda L tipinde saklanır ve SMD

dizisi konum 16’ya atanır. end komutu, +4300(43 SMD’de bitir anlamına gelir) SMD

komutunu oluşturur. Bu komut SMD hafıza dizisine yazılan son komuttur.

Bu şekilde derleyicinin ilk turu biter. Şimdi ikinci tura geçeceğiz. isaretler dizisi, -1’den

farklı değerler için aranır. 03 konumu, 60 içermektedir. Böylece derleyici, 03 komutu ile

işinin henüz bitmediğini anlar. Derleyici, sembol tablosundan satır numarası 60’ı arar ve

konumunu bularak bu konumu henüz SMD’ye tam olarak çevirmediği komuta ekler. Bu

arama işleminde, satır numarası 60, konum 15’te bulunur ve komut son haline getirilir.Yani

03 +4200 yerine 03 +4115 yazılır. Simple programı artık başarılı olarak derlenmiştir.

Derleyiciyi oluşturmanız için aşağıdaki görevleri yerine getirmeniz gerekmektedir:

a) Alıştırma 7.19’da yazdığınız Simpletron simülasyon programını, giriş olarak

kullanıcının belirttiği bir dosyayı alacağı şekilde değiştiriniz(11.Üniteye bakınız).

Simülasyon programınız, çıktı sonuçlarını ekran çıktısı ile aynı biçimde olacak

şekilde bir dosyaya yazmalıdır.

b) Alıştırma 12.12’deki ortaek gösteriminin sonek gösterimine çevrildiği algoritmayı

birden fazla basamaklı tamsayılarla ve tek karakterlik değişken isimleriyle işlem

yapılacak şekilde değiştiriniz. İpucu: strtok standart kütüphane fonksiyonu bir

ifadedeki sabitleri ve değişkenleri bulmada kullanılabilir. Sabitler ise string

durumunda bulundukları için atoi fonksiyonu ile tamsayılara dönüştürülmelidirler.

(Not: sonek ifadelerinin veri gösterimleri değişken isimlerinin ve sabitleri

destekleyecek şekilde değiştirilmelidir.)

c) Sonek ifadelerini hesaplama algoritmasını, operand olarak birden fazla basamaklı

tamsayılarla ve değişken isimleriyle işlem yapabilecek şekilde değiştiriniz. Artık

bu algoritma SMD komutlarını doğrudan oluşturmak yerine daha önce

bahsettiğimiz “kanca” metoduyla oluşturmalıdır. İpucu: strtok standart kütüphane

fonksiyonu bir ifadedeki sabitleri ve değişkenleri bulmada kullanılabilir. Sabitler

ise string durumunda bulundukları için atoi fonksiyonu ile tamsayılara

dönüştürülmelidirler. (Not: sonek ifadelerinin veri gösterimleri değişken

isimlerinin ve sabitleri destekleyecek şekilde değiştirilmelidir.)

d) Derleyiciyi oluşturun. (c) ve (b) şıklarını birleştirerek let ifadelerindeki aritmetik

işlemlerin hesaplanmasını sağlayın. Programınız, derleyicinin ilk turu için bir

fonksiyonu, ikinci tur için ise başka bir fonksiyonu çağırmalı ve her iki

fonksiyonda görevlerini yapmak için diğer fonksiyonları çağırmalıdırlar.

515

12.28 (Simple Derleyicisini iyileştirme) Bir program derlendiğinde ve SMD’ye çevrildiğinde

bir çok komut üretilir. Genellikle üçlü bir grup oluşturan bazı komutlar kendilerini tekrar

ederler. Buna prodüksiyon denir. Normal olarak bir prodüksiyon üç komuttan oluşur.

Örneğin, yukle, topla ve sakla.Örneğin Şekil 12.30’da, Şekil 12.28’deki programın derlenmesi

sırasında oluşan beş SMD komutu gösterilmektedir. İlk üç komut y’ye 1 ekleyen bir

prodüksiyondur. 06 ve 07 komutları, akümülatörü geçici hafıza konumu 96’da

saklamaktadır. Daha sonra bu değer akümülatöre geri yüklenerek 08 komutuyla hafıza

konumu 98’de saklanmaktadır. Bir prodüksiyonu genellikle, henüz saklanmış olan hafıza

konumunu yükleyen bir komut takip eder. Bu kod, sakla komutunun ve aynı hafıza konumunu

yükleyen komutun elenmesiyle iyileştirilebilir. Bu işlemlerle artık Simpletron daha az komutu

derleyeceği için daha hızlı çalışacaktır. Şekil 12.31, Şekil 12.28’deki programın bu

işlemlerden geçtikten sonraki halini göstermektedir. Dört komutunun bu şekilde çıkarılması

%25’lik bir hafıza alanının kurtarılması anlamına gelir.

Derleyiciyi, oluşturacağı kod için bir iyileştirme seçeneği içerecek şekilde değiştiriniz.

İyileştirilmemiş kodları, iyileştirilmiş kodlarla karşılaştırınız.

12.29 (Simple Derleyicisinde değişklikler) Simple derleyicisinde aşağıdaki değişiklikleri

yapınız. Bu değişikliklerin bazılarını yapabilmek için, Alıştırma 7.19’da yazdığınız

Simpletron Simülasyon programınızda da değişiklikler yapmak zorundasınız.

a) let ifadelerinde % operatörünün de kullanılmasını sağlayınız. Bunun için

Simpletron makine dili, mod komutunu içerecek şekilde değiştirilmelidir.

b) let ifadelerinde üssel ifadelerin ^ işareti ile beraber kullanılmasını sağlayınız.

Simpletron makine dili, üs komutunu içerecek şekilde değiştirilmelidir.

c) Derleyicinin, simple ifadelerindeki büyük ve küçük harfleri tanımasını sağlayınız

(‘A’nın eşiti ‘a’dır). Simpletron simülasyon programınızda bir değişikliğe gerek

yoktur.

d) input ifadelerini birden fazla değişkene değer okuyabilecek şekilde değiştiriniz.

Örneğin input x, y. Simpletron simülasyon programınızda bir değişikliğe gerek

yoktur.

e) Derleyicicin print ifadesiyle birden fazla değerin çıktısını yazdırabilmesini

sağlayınız. Örneğin print a, b, c. Simpletron simülasyon programınızda bir

değişikliğe gerek yoktur.

f) Derleyicinize, simple programında yazım hataları olduğu zaman bunları hata

mesajlarıyla kullanıcıya gösterebilmesi için yazım hatası kontrolünü ekleyiniz.

Simpletron simülasyon programınızda bir değişikliğe gerek yoktur.

g) Tamsayı dizilerinin kullanılabilmesini sağlayınız. Simpletron simülasyon

programınızda bir değişikliğe gerek yoktur.

04 +2098 (yukle)

05 +3097 (topla)

06 +2196 (sakla)

07 +2096 (yukle)

08 +2198 (sakla)

Şekil 12.30 Şekil 12.28’deki iyileştirilmemiş kodlar.

516

Simple program SMD konumu Açıklama

ve Komutlar

5 rem 1’den x’e kadar topla yok rem ihmal edildi

10 input x 00 +1099 99 konumuna x’i oku

15 rem y==x koşulu kontrol et yok rem ihmal edildi

20 if y == x goto 60 01 +2098 y(98)’i akümülatöre yükle

02 +3199 x(99)’u akümülatörden çıkart

03 +4200 sıfırsa karar verilmemiş referans

25 rem y’yi bir azalt yok dallan

30 let y = y + 1 04 +2098 y’yi akümülatöre yükle

05 +3097 akümülatöre 1(97)

06 +2198 akümülatörü y(98)’de sakla

35 rem y’yi toplama ekle none rem ihmal edildi

40 let t = t + y 07 +2096 t(96)’yı akümülatöre al

08 +3098 y’yi akümülatöre ekle

09 +2196 akümülatörü t(96)’da sakla

45 rem y’ye git none rem ihmal edildi.

50 goto 20 10 +4001 01 konumuna dallan

55 rem sonucun çıktısını al none rem ihmal edildi.

60 print t 11 +1196 t’yi ekrana yazdır.

99 end 12 +4300 programı sonlandır

Şekil 12.31 Şekil 12.28’deki programın iyileştirilmiş kodu komutları.

h) gosub ve return Simple komutlarının uygulanacağı alt yordamların

kullanılabilmesini sağlayınız. Gosub komutu program komutunu alt yordama

geçirirken return komutuda program kontrolünü gosub’tan bir sonra gelen ifadeye

geçirir. Bu C’de fonksiyon çağırmaya benzer.

i) Aşağıdaki biçimdeki tekrar yapılarının kullanılmasını sağlayınız.

for x = 2 to 10 step 2

Simple ifadeleri

next

for ifadesi, 2’den 10’a kadar ikişer ikişer artan bir döngü kurar. next satırı for

ifadesinin gövdesinin sonunu gösterir. Simpletron simülasyon programınızda bir

değişikliğe gerek yoktur.

j) Aşağıdaki biçimdeki tekrar yapılarının kullanılmasını sağlayınız.

for x = 2 to 10

Simple ifadeleri

next

for ifadesi, 2’den 10’a kadar birer birer artan bir döngü kurar. next satırı for

517

ifadesinin gövdesinin sonunu gösterir. Simpletron simülasyon programınızda bir

değişikliğe gerek yoktur.

k) Derleyicinin string giriş ve çıkışları yapabilmesini sağlayın. Bunun için Simpletron

Simülasyon programının saklayabileceği ve işleyebileceği şekilde değiştirilmesi

lazımdır. İpucu: Her Simpletron wordü iki basamaklı tamsayılardan oluşan iki ayrı

gruba ayrılabilir. Her iki grup tamsayıda bir karakterin ASCII karşılığını içerir.

Belli bir simpletron hafıza konumundan başlayan stringi yazdıran bir makine dili

komutu ekleyiniz. O konumdaki wordun ilk yarısı stringin kaç karakter

içereceğini(karakter uzunluğunu) belirtir.Art arda gelen her word iki basamaklı

rakamlarla ifade edilen bir karakterin ASCII karşılığıdır. Makine dili komutu,

stringin uzunluğunu kontrol etmeli ve iki basamaklı sayıları karakter eşleniklerine

çevirerek stringi yazdırmalıdır.

l) Tamsayılara ek olarak derleyicinin, ondalıklı sayılarla da işlem yapmasını

sağlayınız. Simpletron simülasyon programınızda değişiklikler yapmanız

gerekmektedir.

12.30 (Basit bir çevirici) Bir çevirici, yüksek seviyeli bir dilde yazılmış olan ifadeyi okuyup,

bu ifadeyle gerçekleştirilecek olan işlemi anlayıp, bu işlemi hemen çalıştıran programdır.

Program makine diline çevrilmez. Çeviricler yavaş çalışırlar, çünkü programda karşılaşılan

her ifade önce deşifre edilmelidir. Eğer bu ifadeler bir döngü içerisindeler ise döngünün her

turunda tekrar deşifre edilirler. BASIC programlama dilinin eski sürümler bu şekilde

çalışmaktaydı.

Alıştırma 12.26’daki Simple dili için bir çevirici yazınız. Program, let ifadelerinin

hesaplanması için Alıştırma 12.12’deki ortaek gösterimlerini sonek gösterimlerine çeviren

algoritmayı ve Alıştırma 12.12’deki ortaek gösterimindeki ifadelerin hesaplanma

algoritmasını içermelidir. Alıştırma 12.26’daki Simple Dilindeki kısıtlamalar bu program

içinde geçerlidir. Alıştırma 12.26’da yazılan Simple programlarını yazdığınız çeviriyle

deneyiniz. Bu programların çeviriciyle çalıştırıldığında, derlendiğinde ve Alıştırma 7.19’daki

Simpletron Simülasyon programıyla çalıştıklarındaki sonuçları karşılaştırınız.

518

C ÖNİŞLEMCİSİ

AMAÇLAR

 Büyük programlar geliştirirken #include kullanabilmek

 #define kullanarak makrolar ve argüman içeren makrolar yaratabilmek.

 Koşullu derlemeyi anlamak

 Koşullu derleme esnasında hata mesajları yazdırabilmek.

 Deyimlerin değerlerinin doğru olup olmadığını belirlemek için bildiriler (assertion)

kullanabilmek.

BAŞLIKLAR

13.1 GİRİŞ

13.2 #include ÖNİŞLEMCİ KOMUTU

13.3 #define ÖNİŞLEMCİ KOMUTU:SEMBOLİK SABİTLER

13.4 #define ÖNİŞLEMCİ KOMUTU:MAKROLAR

13.5 KOŞULLU DERLEME

13.6 #error ve #pragma ÖNİŞLEMCİ KOMUTLARI

13.7 # ve ## OPERATÖRLERİ

13.8 SATIR NUMARALARI

13.9 ÖNCEDEN TANIMLANMIŞ SEMBOLİK SABİTLER

13.10 BİLDİRİLER (ASSERTIONS)

Özet \* Genel Programlama Hataları \* İyi Programlama Alıştırmaları \* Performans İpuçları

\* Cevaplı Alıştırmalar \* Cevaplar \* Alıştırmalar

13.1 GİRİŞ

Bu ünite, C önişlemcisini tanıtmaktadır. Önişleme, bir program derlenmeden önce gerçekleşir.

Yapılabilecek bazı işlemler; derlenen dosyanın içine başka dosyaları eklemek, sembolik

sabitler ve makrolar tanımlamak, program kodlarını koşullu olarak derlemek ve önişlemci

komutlarının koşullu çalıştırılmasıdır. Bütün önişlemci komutları # ile başlar ve bir satırda

önişlemci komutundan önce yalnızca boşluk karakterleri bulunabilir.

13.2 #include ÖNİŞLEMCİ KOMUTU

Şu ana kadar #include önişlemci komutunu kullanmıştık. #include önişlemci komutu,

belirlenen dosyanın kopyasının, komutun bulunduğu yere eklenmesini sağlar. #include

komutunun iki biçimi vardır:

#include <dosyaismi>

#include “dosyaismi”

Bu ikisi arasındaki fark, önişlemcinin dahil edilecek dosyayı aradığı konumdur. Eğer dosya

ismi tırnak içinde ise, önişlemci dosyayı, aranan dosya eklendikten sonra derlenecek dosyanın

bulunduğu dizin içinde arar. Bu yöntem, genellikle programcı tarafından tanımlanmış öncü

519

dosyaları eklemek için kullanılır. Eğer dosya ismi açılı parantezler (< ve >) içinde ise, bu

yöntem standart kütüphane öncü dosyalarını eklemek için kullanılır. Arama, uygulamaya

bağımlı bir şekilde ve genellikle daha önceden tasarlanmış dosyalar içinde yapılır.

#include komutu, stdio.h ve stdlib.h gibi öncü dosyaları eklemek için kullanılır (bakınız

Şekil 5.6). #include komutu ayrıca birlikte derlenecek bir çok kaynak dosyayı içeren

programlarda kullanılır. Ayrı program dosyalarının tümünde geçerli bildirimler içeren öncü

dosyalar, genellikle dosya içinde yaratılır ve dosyaya eklenir. Bu tarzda bildirimlerin

örnekleri, yapı ve birlik bildirimleri, sayma sabitleri ve fonksiyon prototipleridir.

13.3 #include ÖNİŞLEMCİ KOMUTU

#define komutu, sembolik sabitler ( sembollerle temsil edilen sabitler) ve makrolar ( işlemleri

sembol olarak tanımlanır) yaratır. #define komutunun biçimi aşağıdaki şekildedir:

#define tanıtıcı yer değiştirme\_metni

Bu satır bir dosyada yer aldığında, program derlenmeden önce tanıtıcının dosyada yer aldığı

tüm yerlerde, tanıtıcı yer değiştirme metni ile otomatik olarak değiştirilir. Örneğin,

#define PI 3.14159

satırından sonra PI sembolik sabitiyle karşılaşılan her yerde PI , 3.14159 nümerik sabitiyle

değiştirilir. Sembolik sabitler, programcının bir sabit için isim verebilmesini ve bu ismi tüm

program boyunca kullanabilmesini sağlar. Eğer programda sabitin değerinin değiştirilmesi

gerekirse, değiştirme #define komutu içinde yapılır ve program derlendiğinde sabitin

programda yer aldığı tüm yerlerdeki değeri otomatik olarak değiştirilir ( Not: Sembolik sabit

isminin sağındaki her şey sembolik sabitle değiştirilir). Örneğin, #define PI = 3.14159

önişlemcinin PI’yi =3.14159 ile değiştirmesine sebep olur. Bu, çoğu mantık ve yazım

hatasının sebebidir. Bir sembolik sabiti yeni bir değerle yeniden tanımlamak da bir hatadır.

İyi Programlama Alıştırmaları 13.1

Sembolik sabitler için anlamlı isimler kullanmak, programın kendiliğinden daha okunur bir

hale gelmesini sağlar.

13.4 #define ÖNİŞLEMCİ KOMUTU:MAKROLAR

Bir makro, #define önişlemci komutu içinde tanımlanmış bir işlemdir. Sembolik sabitlerde

olduğu gibi, program derlenmeden önce makro tanıtıcısı programda yer değiştirme metni ile

değiştirilir. Makrolar, argümanlarla ya da argümansız olarak tanımlanabilir. Argümansız bir

makro, sembolik bir sabit gibi işlenir. Argümanları olan bir makroda, argümanlar yer

değiştirme metnindekilerle değiştirilir. Daha sonra makro genişletilir (yani, yer değiştirme

metni programda tanıtıcıyı ve argüman listesini değiştirir.)

Bir çemberin alanı için kullanılan tek argümanlı bir makro tanımını ele alalım:

#define CEMBER\_ALANI ( x ) ( ( PI ) \* ( x ) \* ( x ) )

520

Programda, CEMBER\_ALANI ( y ) ile karşılaşıldığında y’nin değeri, yerdeğiştirme

metnindeki x ile , PI sembolik sabiti de değeri ile ( daha önceden tanımlanmıştı )

değiştirilecek ve böylece makro genişletilecektir. Örneğin,

alan = CEMBER\_ALANI ( 4 ) ;

ifadesi

alan = ( ( 3.14159 ) \* ( 4 ) \* ( 4 ) );

şeklinde genişletilir ve deyimin değeri hesaplanarak, alan değişkenine atanır. Yerdeğiştirme

metnindeki her x’i içine alan parantezler, makro argümanı bir deyim olduğunda hesaplama

sırasının doğru bir şekilde yapılmasını sağlatır. Örneğin,

alan = CEMBER\_ALANI ( c + 2 ) ;

ifadesi

alan = ( ( 3.14159 ) \* ( c + 2 ) \* ( c + 2 ) ) ;

şeklinde genişletilir ve böylece hesaplama doğru bir şekilde yapılır, çünkü parantezler

hesaplama sırasını doğru bir biçime sokmuştur. Eğer parantezler konmasaydı, makro

genişletildiğinde

alan = 3.14159 \* c +2 \* c + 2 ;

olacak ve operatör önceliği kuralları yüzünden yanlış bir biçimde

alan = (3.14159 \* c) +(2 \* c) + 2 ;

olarak hesaplanacaktı.

Genel Programlama Hataları 13.1

Yerdeğiştirme metninde makro argümanlarını parantez içine almayı unutmak

CEMBER\_ALANI makrosu fonksiyon olarak da tanımlanabilir. cemberAlani

fonksiyonu;

double cemberAlani ( double x )

{

return 3.14159 \* x \* x ;

}

CEMBER\_ALANI makrosuyla aynı hesaplamayı yapar. Fakat cemberAlani fonksiyonu ile

ilgili olan bir fonksiyon çağrısı kullanılmalıdır. CEMBER\_ALANI makrosunun avantajı,

makroların kodu programa doğrudan eklemesidir ve program okunabilir halde kalır, çünkü

CEMBER\_ALANI hesaplaması ayrıca yapılmış ve anlamlı bir şekilde isimlendirilmiştir.

Dezavantaj ise argümanın iki kez hesaplanmasıdır.

521

Performans İpuçları 13.1

Makrolar bazen, çalışma zamanından öncelikli olarak ,fonksiyon çağrısı yerine programa

doğrudan kod eklemek için kullanılır.

Aşağıdaki iki argümanlı makro tanımı, dikdörtgenin alanını hesaplamaktadır:

#define DIKDORTGEN\_ALANI (x , y ) ( ( x ) \* ( y ) )

Programda DIKDORTGEN\_ALANI (x , y ) ile karşılaşıldığında, x ve y’nin değerleri makro

yerdeğiştirme metni içinde değiştirilir ve makro, makro isminin bulunduğu yerde genişletilir.

Örneğin,

diktAlani = DIKDORTGEN\_ALANI ( a + 4 , b + 7 ) ;

ifadesi

diktAlani= ( ( a + 4 ) \* ( b + 7 ) );

şeklinde genişletilir. Deyimin değeri hesaplanır ve diktAlani değişkenine atanır.

Bir makro ya da sembolik sabit için yerdeğiştirme metni, #define komutu içindeki tanıtıcıdan

sonra o satırdaki herhangi bir metindir. Eğer makro ya da sembolik sabit için kullanılacak

yerdeğiştirme metni satırın geri kalanından daha uzunsa, satırın sonuna, yerdeğiştirme

metninin sonraki satırda da devam ettiğini belirten bir ters çizgi ( \ ) yerleştirilmelidir.

Sembolik sabitler ve makroların geçerliliğini sonlandırmak için #undef önişlemci komutu

kullanılır. #undef komutu, bir sembolik sabiti ya da makro ismini tanımsız hale getirir. Bir

sembolik sabitin ya da makronun faaliyet alanı, tanımlandığı noktadan #undef ile tanımsız

hale getirildiği yere ya da dosya sonuna kadardır. Bir kez tanımsız hale getirildikten sonra, bir

isim #define ile yeniden tanımlanabilir.

Standart kütüphanedeki fonksiyonlar bazen başka kütüphane fonksiyonlarına dayalı olarak

makro biçiminde tanımlanırlar. stdio.h öncü dosyasında tanımlanmış bir makro aşağıda

gösterilmiştir:

#define getchar( ) getc ( stdin )

getchar‘ın makro tanımı, getc fonksiyonunu standart girişten bir karakter almak için

kullanmaktadır. stdio.h içindeki putchar fonksiyonu ve ctype.h içindeki karakter işleme

fonksiyonları sıklıkla makro olarak da uygulanır. Yan etkileri olan ( örneğin, değişken

değerleri değiştirilmiştir ) deyimlerin, makroya geçirilmemesi gerekir çünkü makro

argümanları birden fazla kez hesaplanabilir.

13.5 KOŞULLU DERLEME

Koşullu derleme, programcının önişlemci komutlarının çalışmasını ve program kodunun

derlenmesini kontrol edebilmesini sağlar. Koşullu önişlemci komutlarının her biri, sabit bir

tamsayı deyimini değerlendirir. Dönüşüm operatörleri, sizeof deyimleri ve sayma sabitleri

önişlemci komutlarında değerlendirilemez.

522

Koşullu önişlemci oluşturmak, if seçim yapısına oldukça benzer. Aşağıdaki önişlemci kodunu

inceleyelim:

#if !defined ( NULL )

#define NULL 0

#endif

Bu komutlar, NULL’un tanımlı olup olmadığına karar vermektedir. defined ( NULL ) deyimi

NULL tanımlı ise 1, değilse 0 üretir. Eğer sonuç 0 ise, !defined ( NULL ) 1 olur ve NULL

tanımlanır. Aksi takdirde, #define komutu atlanır. Her #if mutlaka #endif ile sonlanır.

#ifdef ve #ifndef komutları, #if defined ( isim ) ve #if !defined ( isim ) için kısaltma olarak

kullanılır. Çok kısımlı bir koşullu önişlemci oluşumu, #elif ( if yapısındaki else if kısmıyla

eşdeğerdir) ve #else (if yapısındaki else kısmıyla eşdeğerdir) komutları kullanılarak

denenebilir.

Program geliştirirken, programcılar kodun yorumlanmasında yardımcı olan bazı kısımların

derlenmemesini tercih edebilirler. Eğer kod yorumlar içeriyorsa, /\* ve \*/ bu görev için

kullanılamaz. Bunun yerine, programcı aşağıdaki önişlemci oluşumunu kullanabilir:

#if 0

derlenmesi engellenen kod

#endif

Kodun derlenebilmesi için az önceki oluşumdaki 0 yerine 1 yazılmalıdır.

Koşullu derleme genellikle bir hata ayıklama aracı olarak kullanılır. Çoğu C uygulaması,

koşullu derlemeden daha güçlü özelliklere sahip hata ayıklayıcılar ( debugger ) içerir. Eğer bir

hata ayıklayıcı bulunamıyorsa, printf ifadeleri değişkenlerin değerlerini yazdırmak ve akış

kontrolünü onaylamak için kullanılır. Bu printf ifadeleri, koşullu önişlemci komutları arasına

yazılarak yalnızca hata ayıklama sonlanmadıkça yazdırılmaları sağlanabilir. Örneğin,

#ifdef DEBUG

printf(“değişken x=%d\n”,x);

#endif

eğer DEBUG sembolik sabiti #ifdef DEBUG komutundan önce tanımlanmışsa, printf

ifadesinin derlenmesine sebep olur. Hata ayıklama sonlandıktan sonra, #define komutu

kaynak koddan çıkartılır ve hata ayıklama amacıyla yerleştirilmiş printf ifadeleri derleme

esnasında atlanır. Büyük programlarda, kaynak kodun değişik kısımlarında koşullu derlemeyi

kontrol edebilmek için daha fazla sayıda sembolik sabit tanımlanması gerekebilir.

Genel Programlama Hataları 13.2

C’nin tek bir ifade beklediği konumlara, hata ayıklama amacıyla, koşullu olarak derlenen

printf ifadeleri yerleştirmek. Bu durumda, koşullu olarak derlenen ifade birleşik bir ifade

içine yerleştirilmelidir. Bu sayede ,program hata ayıklama ifadeleri ile derlendiğinde,

programın akışı değişmemiş olur.

13.6 #error ve #pragma ÖNİŞLEMCİ KOMUTLARI

523

#error komutu olan

#error atomlar

komutta belirlenen atomları içeren mesajları uygulama-bağımlı olarak yazdırır. Atomlar,

boşluklarla birbirinden ayrılmış karakter serileridir. Örneğin,

#error 1 – Aralık dısında hatasi

5 atom içermektedir. Bazı sistemlerde #error komutu işlendiğinde, komuttaki atomlar bir hata

mesajı olarak yazdırılır, önişleme durur ve program derlenmez.

#pragma komutu olan

#pragma atomlar

uygulama-bağımlı bir işlem gerçekleştirir. Uygulama tarafından tanınamayan bir pragma

ihmal edilir. #error ve #pragma hakkında daha fazla bilgi için, C uygulamanızdaki

dokümanları inceleyiniz.

13.7 # ve ## OPERATÖRLERİ

# ve ## önişlemci operatörleri, standart C içinde mevcuttur. # operatörü, bir yerdeğiştirme

metni atomunun, tırnak içine alınmış bir string haline dönüştürülmesini sağlar. Aşağıdaki

makro tanımını inceleyiniz:

#define HELLO ( x ) printf ( “ Merhaba, ” #x ” \ n”);

Bir program dosyasında HELLO (John) ile karşılaşıldığında , aşağıdaki şekilde genişletilir:

printf ( “ Merhaba , ” “John” “\n” );

“John” stringi yerdeğiştirme metnindeki #x yerine geçer. Boşluk ile ayrılan stringler

önişleme esnasında birleştirilir. Bu sebepten, az önceki ifade

printf(“Merhaba , John\n”);

ile eşdeğerdir.

# operatörü, makro içinde argümanlarla kullanılmak zorundadır çünkü # operatörünün

operandı makro içindeki bir argümanı belirtir.

## operatörü iki atomu birleştirir. Aşağıdaki makro tanımını inceleyiniz:

#define ATOMEKLE(x,y) x ## y

Programda ATOMEKLE ile karşılaşıldığında, argümanları birleştirilir ve makro yerine

kullanılır. Örneğin, ATOMEKLE ( O,K ) programda OK ile değiştirilir. ## operatörünün iki

operandı olmak zorundadır.

524

13.8 SATIR NUMARALARI

#line önişlemci komutu, kendinden sonra gelen tüm kaynak kod satırlarının, belirlenen

tamsayı sabitinin değerinden başlanarak numaralandırılmasını sağlar.

#line 100

komutundan sonraki kaynak satırı, 100’den başlanarak numaralandırılır. #line komutu içinde

bir dosya ismi yer alabilir.

#line 100 “file1.c”

komutu, kendinden sonraki satırların 100’den başlanarak dosya ismiyle birlikte

numaralandırılacağını ve derleyici mesajlarının “file1.c” için verileceğini belirtir. Komut,

genellikle yazım hatalarının ve derleyici mesajlarının daha anlamlı olmasına yardımcı olması

için kullanılır. Satır numaraları kaynak kod içinde gözükmez.

13.9 ÖNCEDEN TANIMLANMIŞ SEMBOLİK SABİTLER

ANSI C, daha önceden tanımlanmış sabitlere ( Şekil 13.1) sahiptir. Daha önceden

tanımlanmış sembolik sabit tanıtıcılarının her biri, 2 adet alt çizgiyle başlar ve 2 adet alt

çizgiyle sonlanır. Bu tanıtıcılar ve defined tanıtıcısı (Kısım 13.5’te kullanılmıştır) #define ya

da #undef komutları içinde kullanılamaz.

Sembolik Sabit Açıklama

\_\_LINE\_\_ Kullanılan kaynak kodun satır numarası(bir tamsayı sabiti)

\_\_FILE\_\_ Kaynak kodun varsayılan ismi(bir stringtir)

\_\_DATE\_\_ Kaynak kodun derlendiği tarihtir(“Aaa gg yyyy” biçiminde bir

stringtir.Örneğin,”Mar 19 2002”

\_\_TIME\_\_ Kaynak dosyanın derlendiği zamandır.(“ss:dd:sn” biçimindeki bir

string bilgisidir.)

Şekil 13.1 Önceden tanımlanmış bazı sembolik sabitler.

13.10 BİLDİRİLER (ASSERTIONS)

assert makrosu, ( assert.h öncü dosyası içinde tanımlanmıştır) bir deyimin değerini test eder.

Eğer deyimin değeri 0 (yanlış) ise assert bir hata mesajı yazdırır ve abort fonksiyonunu

(genel kullanımlı kütüphane stdlib.h içindedir) programı sonlandırması için çağırır. Bu, bir

değişkenin doğru bir değere sahip olup olmadığını test etmek için kullanışlı bir hata ayıklama

aracıdır. Örneğin, x değişkeninin bir programda asla 10 değerini geçmemesi gerektiğini

525

düşünelim. x’in değerini test etmek için bir bildiri kullanılabilir ve eğer x’in değeri hatalı ise

bir hata mesajı yazdırılır. İfade,

assert (x < = 10);

biçiminde olacaktır.

Eğer x 10’dan büyükse, programda az önceki ifadeyle karşılaşıldığında, satır numarasını ve

dosya adını içeren bir hata mesajı yazdırılacak ve program sonlanacaktır. Programcı hatayı

gidermek için kodun bu kısmını inceleyebilecektir. Eğer NDEBUG sembolik sabiti

tanımlanmışsa, daha sonraki bildiriler ihmal edilir. Bu sebepten, bildirilere gerek

kalmadığında

#define NDEBUG

satırı kullanılarak, tüm bildirileri elle silmek yerine otomatik olarak geçersiz yapmak mümkün

olur.

ÖZET

 Tüm önişlemci komutları # ile başlar.

 Bir satırda önişlemci komutundan önce yalnızca boşluk karakterleri bulunabilir.

 #include komutu, belirlenen dosyanın bir kopyasını ekler. Eğer dosya ismi tırnak

içinde ise, önişlemci dosyayı, aranan dosya eklendikten sonra derlenecek dosyanın

bulunduğu dizin içinde arar. Eğer dosya ismi açılı parantezler (< ve >) arama

uygulamaya bağımlı bir şekilde yapılır.

 Bir sembolik sabit, bir sabitin ismidir.

 Bir makro, #define önişlemci komutu içinde tanımlanmış bir işlemdir. Makrolar

argümanlarla ya da argümansız olarak tanımlanabilir.

 Bir makro ya da sembolik sabit için yerdeğiştirme metni, #define komutu içindeki

tanıtıcıdan sonra o satırdaki herhangi bir metindir. Eğer makro ya da sembolik sabit

için kullanılacak yerdeğiştirme metni satırın geri kalanından daha uzunsa, satırın

sonuna, yerdeğiştirme metninin sonraki satırda da devam ettiğini belirten bir ters çizgi

( \ ) yerleştirilmelidir.

 Sembolik sabitler ve makroların geçerliliğini sonlandırmak için #undef önişlemci

komutu kullanılır. #undef komutu, bir sembolik sabiti ya da makro ismini tanımsız

hale getirir.

 Bir sembolik sabitin ya da makronun faaliyet alanı, tanımlandığı noktadan #undef ile

tanımsız hale getirildiği yere ya da dosya sonuna kadardır.

 Koşullu derleme, programcının önişlemci komutlarının çalışmasını ve program

kodunun derlenmesini kontrol edebilmesini sağlar.

 Koşullu önişlemci komutlarının her biri, sabit bir tamsayı deyimini değerlendirir.

Dönüşüm operatörleri, sizeof deyimleri ve sayma sabitleri önişlemci komutlarında

değerlendirilemez.

 Her #if oluşumu #endif ile sonlanır.

 #ifdef ve #ifndef komutları, #if defined(isim) ve #if !defined(isim) için kısaltma

olarak kullanılır.

526

 Çok kısımlı bir koşullu önişlemci oluşumu, #elif ve #else komutları kullanılarak

denenebilir.

 #error komutu komutta belirlenen atomları içeren mesajları uygulama-bağımlı olarak

yazdırır.

 #pragma komutu uygulama-bağımlı bir işlem gerçekleştirir. Uygulama tarafından

tanınamayan bir pragma ihmal edilir.

 # operatörü, bir yerdeğiştirme metni atomunun, tırnak içine alınmış bir string haline

dönüştürülmesini sağlar. # operatörü makro içinde argümanlarla kullanılmak

zorundadır çünkü, # operatörünün operandı makro içindeki bir argümanı belirtir.

 ## operatörü iki atomu birleştirir. ## operatörünün mutlaka iki operandı olmalıdır.

 #line önişlemci komutu, kendinden sonra gelen tüm kaynak kod satırlarının, belirlenen

tamsayı sabitinin değerinden başlanarak numaralandırılmasını sağlar.

 \_\_LINE\_\_ sabiti, kullanılan kaynak kodun satır numarasıdır ( bir tamsayı sabiti ).

\_\_FILE\_\_sabiti, kaynak kodun varsayılan ismidir ( bir string ). \_\_DATE\_\_ sabiti

kaynak kodun derlendiği tarihtir (bir string). \_\_TIME\_\_ sabiti, kaynak dosyanın

derlendiği zamandır (string bilgisidir.)

 assert makrosu, bir deyimin değerini test eder. Eğer deyimin değeri 0(yanlış) ise

assert bir hata mesajı yazdırır ve abort fonksiyonunu programı sonlandırması için

çağırır.

ÇEVRİLEN TERİMLER

C preprocessor............................ C önişlemcisi

conditional compilation.............. koşullu derleme

debug......................................... hata ayıklama

expand a macro ........................ bir makronun genişletilmesi

predefined symbolic constants.. önceden tanımlanmış sembolik sabitler

replacement text....................... yerdeğiştirme metni

scope of a symbolic constant... bir sembolik sabitin faaliyet alanı

symbolic constant................... sembolik sabit

GENEL PROGRAMLAMA HATALARI

13.1 Yer değiştirme metninde makro argümanlarını parantez içine almayı unutmak

13.2 C’nin tek bir ifade beklediği konumlara, hata ayıklama amacıyla, koşullu olarak

derlenen printf ifadeleri yerleştirmek .Bu durumda, koşullu olarak derlenen ifade birleşik

bir ifade içine yerleştirilmelidir. Bu sayede, program hata ayıklama ifadeleri ile

derlendiğinde, programın akışı değişmemiş olur.

İYİ PROGRAMLAMA ALIŞTIRMALARI

13.1 Sembolik sabitler için anlamlı isimler kullanmak, programın kendiliğinden daha okunur

bir hale gelmesini sağlar.

PERFORMANS İPUÇLARI

527

13.1 Makrolar bazen, çalışma zamanından öncelikli olarak, fonksiyon çağrısı yerine

programa doğrudan kod eklemek için kullanılır.

Cevaplı Alıştırmalar

13.1 Aşağıdaki boşlukları doldurunuz.

a) Her önişlemci komutu, \_\_\_\_\_\_\_ ile başlamalıdır.

b) Koşullu derleme oluşturmak, çoklu ifadeler test edebilecek şekilde \_\_\_\_\_\_\_ ve

\_\_\_\_\_\_\_ komutlarının kullanılmasıyla genişletilebilir.

c) \_\_\_\_\_\_\_ komutu makro ve sembolik sabit oluşturmada kullanılır.

d) Bir ön işlemci komutu satırından önce sadece \_\_\_\_\_\_\_ karakterleri görünür.

e) \_\_\_\_\_\_\_ komutu, sembolik sabitleri ve makro isimlerini geçirsiz hale getirir.

f) \_\_\_\_\_\_\_ ve \_\_\_\_\_\_\_ komutları, #ifdefined(isim) ve #if!defined(isim) komutlarının

kısa gösterimleridir.

g) \_\_\_\_\_\_\_, programcının önişlemci komutlarının çalışmasını ve program kodunun

derlenmesini yönetebilmesini sağlar

h) \_\_\_\_\_\_\_ makrosu, eğer çalıştırdığı ifade 0 ise bir mesaj yazdırır ve programı

sonlandırır.

i) \_\_\_\_\_\_\_ komutu bir dosyanın içine başka bir dosya ekler

j) \_\_\_\_\_\_\_ operatörü, iki argümanını birbirine bağlar.

k) \_\_\_\_\_\_\_ operatörü, operandını stringe çevirir.

l) \_\_\_\_\_\_\_ karakteri, sembolik bir sabit için metin yazılmasının ya da makronun diğer

satırda devam edeceğini gösterir.

m) \_\_\_\_\_\_\_ komutu, kaynak kodu satırlarının belirli bir değer ile bir sonraki kaynak

kodu satırından itibaren numaralandırılmasını sağlar.

13.2 Şekil 13.1de listelenen, daha önceden tanımlanmış sembolik sabitlerin değerlerini ekrana

yazdıran bir program yazınız.

13.3 Aşağıdakiler gerçekleştirecek birer ön işlemci komutu yazınız.

a) EVET sembolik sabiti 1 değerini alsın.

b) HAYIR sembolik sabiti 0 değerini alsın.

c) common.h öncü dosyasının dahil edilmesini sağlayın. Öncü dosya, derlenecek

dosya ile aynı dizinde bulunmaktadır.

d) Dosyanın içindeki 3000 ile başlayan satırdan sonraki satırları tekrar numaralandırın.

e) Eğer DOGRU sembolik sabiti tanımlanmışsa, tanımsız hale getirin ve değeri 1

olacak şekilde tekrar tanımlayın, #ifdef kullanmayın.

f) Eğer DOGRU sembolik sabiti tanımlanmışsa, tanımsız hale getirin ve değeri 1

olacak şekilde tekrar tanımlayın, #ifdef kullanın.

g) Eğer DOGRU sembolik sabiti 0 değilse, YANLIS sembolik sabitini 0 olacak

şekilde tanımlayın. Aksi takdirde YANLIS’ ı 1 olacak şekilde tanımlayın.

h) KUP\_HACIM makrosunu bir küpün hacmini hesaplayacak şekilde tanımlayınız.

Makronuz sadece bir argüman alsın.

528

ÇÖZÜMLER

13.1 a) #. b) #elif, #else. c) #define. d) whitespace. e) #undef. f) #ifdef, #ifndef g)

Koşullu derleme. h) assert i) #include. j) # #. k) #. l) /. m) #line.

13.2

1 /\* Önceden tanımlanmış makroların değerlerini yazdır \*/

2 #include <stdio.h>

3 main( )

4 {

5 printf (“\_\_LINE\_\_ = %d\n”, \_\_LINE\_\_);

6 printf (“\_\_FILE\_\_ = %d\n”, \_\_FILE\_\_);

7 printf (“\_\_DATE\_\_ = %d\n”, \_\_DATE\_\_);

8 printf (“\_\_TIME\_\_ = %d\n”, \_\_TIME\_\_);

9 printf (“\_\_STDC\_\_ = %d\n”, \_\_STDC\_\_);

10 }

\_\_LINE\_\_ = 5

\_\_FILE\_\_ = macros.c

\_\_DATE\_\_ = Mar 08 1993

\_\_TIME\_\_ = 10:23:47

\_\_STDC\_\_ = 1

13.3

a) #define EVET 1

b) #define HAYIR 0

c) #include “common.h”

d) #line 3000

e) #if defined (DOGRU)

#undef DOGRU

#define DOGRU 1

#endif

f) #ifdef DOGRU

#undef DOGRU

#define DOGRU 1

#endif

g) #if DOGRU

#define YANLIS 0

h) #else

#define YANLIS 1

#endif

h) #define KUP\_HACIM(x) (x) \* (x) \* (x)

ALIŞTIRMALAR

529

13.4 Kürenin hacmini hesaplayan ve bir argümanı olan bir adet makro tanımlayan programı

yazınız. Programınız, yarı çapı 1’den 10’a kadar olan kürelerin hacmini hesaplamalıdır ve

sonucu düzgün bir çizelge şeklinde ekrana yazdırmalıdır. Kürenin hacim formülü:

(4 / 3 ) \* π \* r3

π = 3.14159

13.5 Aşağıdaki çıktıyı veren bir program yazınız.

x ve y toplamı 13

Programınız, x ve y isminde iki argümanı olan TOPLAM makrosunu içermeli.

13.6 ENKUCUK2 isminde, iki sayısal değerin en küçüğünü bulan bir makro yazınız. Sayı

girişlerini klavyeden alınız.

13.7 ENKUCUK3 isminde, üç sayısal değerin en küçüğünü bulan bir program yazınız.

ENKUCUK3 makrosu, Alıştırma 13.6’daki ENKUCUK2 makrosunu kullanarak sayıların en

küçüğünü bulmalıdır. Sayı girişlerini klavyeden alınız.

13.8 YAZDIR isminde bir makro kullanarak, ekrana string yazdıran bir program yazınız.

13.9 DIZIYAZDIR makrosunu kullanarak, bir tamsayılar dizisini ekrana yazdıran bir

program yazınız. Makro, diziyi ve dizinin eleman sayısını argüman olarak almalıdır.

13.10 DIZITOPLA adında bir makro kullanarak, sayı değerleri içeren bir dizinin

elemanlarını toplayan bir program yazınız. Makro, diziyi ve dizinin eleman sayısını argüman

olarak almalıdır.

530

İLERİ C KONULARI

AMAÇLAR

 Klavye girişini dosyadan gelecek şekilde değiştirebilmek.

 Ekran çıktısının dosyaya yerleştirilebilmesini sağlamak

 Uzunluğu değişebilen argüman listesine sahip fonksiyonlar yazabilmek.

 Komut satırı argümanlarını işleyebilmek.

 Nümerik sabitlere belli tipler atayabilmek.

 Geçici dosyaları kullanabilmek.

 Bir programda beklenmeyen durumları işleyebilmek.

 Diziler için dinamik hafıza tahsisi yapabilmek.

 Daha önceden dinamik olarak tahsis edilmiş hafıza miktarını değiştirebilmek.

BAŞLIKLAR

14.1 GİRİŞ

14.2 UNIX VE DOS SİSTEMLERİNDE GİRİŞ/ÇIKIŞI YENİDEN YÖNLENDİRMEK

14.3 UZUNLUĞU DEĞİŞEBİLEN ARGÜMAN LİSTELERİ

14.4 KOMUT SATIRI ARGÜMANLARINI KULLANABİLMEK

14.5 ÇOK KAYNAK DOSYALI PROGRAMLARI DERLEME HAKKINDA NOTLAR

14.6 exit VE atexit İLE PROGRAM SONLANDIRMA

14.7 volatile TİP BELİRTECİ

14.8 TAMSAYI VE ONDALIKLI SAYI SABİTLERİ İÇİN SONEKLER

14.9 DOSYALAR ÜZERİNE NOTLAR

14.10 SİNYAL İŞLEME

14.11 DİNAMİK HAFIZA TAHSİSİ: calloc VE realloc FONKSİYONLARI

14.12 KOŞULSUZ DALLANMA: goto

14.1 GİRİŞ

Bu ünite, giriş kurslarında genellikle işlenmeyen birkaç ileri konu başlığını tanıtmaktadır.

Burada tartışılan yeteneklerin çoğu, yalnızca belirli bazı işletim sistemleri için özellikle de

UNIX ve DOS için geçerlidir.

14.2 UNIX VE DOS SİSTEMLERİNDE GİRİŞ/ÇIKIŞI YENİDEN YÖNLENDİRMEK

Normalde, bir programa giriş klavyeden (standart giriş) ve bir programın çıktıları ekrandan

(standart çıkış) yapılır. Çoğu bilgisayar sisteminde (özellikle UNIX ve DOS sistemlerinde),

girişin klavye yerine bir dosyadan gelecek biçimde ve çıkışın ekran yerine dosyaya

yerleştirilecek biçimde değiştirilmesi mümkündür. İki biçimdeki değiştirmede standart

kütüphanenin dosya işleme yetenekleri kullanılmadan gerçekleştirilebilir.

Giriş ve çıkışı, UNIX komut satırından değiştirmenin birkaç yolu vardır. Çalıştırılabilir bir

program olan toplam dosyasının, tamsayıları bir seferde alarak, dosya sonu belirteciyle

karşılaşıncaya kadar tamsayıların toplamını tuttuğunu, daha sonrada yazdırdığını düşünelim.

Normalde, kullanıcı tamsayıları klavyeden girecek ve en sonunda da daha fazla veri

531

girmeyeceğini belirtmek için dosya sonu belirtecini girecektir. Giriş değiştirilirse, giriş bir

dosyada saklanabilir. Örneğin, eğer veri giris dosyası içinde saklanacaksa

$ toplam<giris

komut satırı, toplam programının çalıştırılmasını sağlar ve girişi değiştir sembolü ( < ) , giris

dosyası içindeki verilerin programda giriş değerleri olarak kullanılacağını belirtir. DOS

sistemlerinde girişi değiştirmek aynı biçimde yapılır.

$ , UNIX komut satırı başlangıcını belirtir ( bazı UNIX sistemleri % kullanır). Öğrenciler

genellikle, giriş/çıkış değiştirmenin C‘nin bir özelliği değil de bir işletim sistemi fonksiyonu

olduğunu anlamakta güçlük çekerler.

Girişi değiştirmenin ikinci yöntemi piping’dir. Bir pipe ( | ), bir programın çıktısının diğer

programın girişi olarak kullanılmasını sağlar. rasgele programının, rasgele sayılar ürettiğini

düşünelim; rasgele programının çıktılarını toplam programında giriş olarak kullanmak için

aşağıdaki UNIX komut satırı kullanılır:

$ rasgele | toplam

Bu, rasgele programı tarafından oluşturulan sayıların toplamının hesaplanmasını sağlar.

Piping, UNIX ve DOS sistemlerinde yapılabilir.

Program çıktısı, çıkış yönlendirme sembolü ( > ) ( UNIX ve DOS sistemlerinde aynı sembol

kullanılır) ile dosyaya yönlendirilebilir. Örneğin, rasgele programının çıktısını out dosyasına

yönlendirmek için

$ rasgele > out

kullanılır.

Son olarak , programın çıktısı daha önceden var olan bir dosyanın sonuna çıktı ekle sembolü

(>>) (UNIX ve DOS sistemlerinde aynı sembol kullanılır) ile eklenebilir. Örneğin, rasgele

programının çıktılarını az önceki komut satırında yaratılan out dosyasının sonuna eklemek

için

$ rasgele >> out

kullanılır.

14.3 UZUNLUĞU DEĞİŞEBİLEN ARGÜMAN LİSTELERİ

Belirlenmemiş sayıda argüman alabilecek fonksiyonlar yazmak mümkündür. Kitaptaki

programların çoğu, bildiğiniz gibi, değişken sayıda argüman kullanan standart kütüphane

fonksiyonu printf‘i kullanıyordu. En azından, printf ilk argümanı olarak bir string almak

zorundadır ancak printf fazladan herhangi bir sayıda argüman alabilir. printf fonksiyonunun

prototipi

int printf ( const char \* format , . . . ) ;

532

biçimindedir. Fonksiyon prototipi içindeki üç nokta ( ... ), fonksiyonun değişken sayıda

argüman alabileceğini belirtir. Üç noktanın her zaman parametre listesinin en sonuna

yerleştirilmesi gerektiğine dikkat ediniz.

Değişken argüman öncü dosyası stdarg.h’ın makroları ve tanımlamaları (Şekil 14.1),

değişken uzunlukta argüman listeleri kullanan fonksiyonları oluşturmak için gerekli

yetenekleri sağlar. Şekil 14.2, değişken sayıda argüman alan ortalama (satır 25)

fonksiyonunu göstermektedir. ortalama fonksiyonunun ilk argümanı her zaman ortalaması

alınacak değerlerin sayısıdır.

Tanıtıcı Açıklama

va\_list va\_start, va\_arg ve va\_end makrolarının ihtiyaç duyduğu bilgiyi

tutmak için uygun bir tiptir. Değişken uzunlukta argüman listesindeki

argümanlara erişmek için va\_list tipinde bir nesne bildirilmelidir.

va\_start Değişken uzunlukta bir argüman listesindeki argümanlara erişmeden

önce çağrılan bir makrodur. Makro, va\_arg ve va-end makrolarının

kullanması için va\_list ile bildirilmiş nesneye ilk değer atar.

va\_arg Değişken uzunlukta argüman listesi içindeki bir sonraki argümanın tipi

ve değerinde bir deyime genişleyen bir makrodur. va\_arg için yapılan

her çağrı, va\_list ile bildirilmiş nesneyi listedeki bir sonraki argümanı

gösterecek biçimde değiştirir.

va\_end Değişken uzunluktaki argüman listesi va\_start ile belirlenmiş bir

fonksiyondan, normal geri dönüş yapılmasını sağlayan bir makrodur.

Şekil 14.1 stdarg.h içinde tanımlanmış tip ve makrolar.

1 /\* Şekil 14.2: fig14\_02.c

2 Değişken uzunlukta argüman listeleri kullanmak \*/

3 #include <stdio.h>

4 #include <stdarg.h>

5

6 double ortalama( int, ... );

7

8 int main( )

9 {

10 double w = 37.5, x = 22.5, y = 1.7, z = 10.2;

11

12 printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",

13 "w = ", w, "x = ", x, "y = ", y, "z = ", z );

14 printf( "%s%.3f\n%s%.3f\n%s%.3f\n",

15 "w ve x’in ortalaması : ",

16 ortalama( 2, w, x ),

17 "w, x ve y ‘in ortalaması : ",

18 ortalama ( 3, w, x, y ),

19 "w, x, y ve z ‘in ortalaması : ",

533

20 ortalama( 4, w, x, y, z ) );

21

22 return 0;

23 }

24

25 double ortalama( int i, ... )

26 {

27 double toplam = 0;

28 int j;

29 va\_list ap;

30

31 va\_start( ap, i );

32

33 for ( j = 1; j <= i; j++ )

34 toplam += va\_arg( ap, double );

35

36 va\_end( ap );

37 return toplam / i;

38 }

w = 37.5

x = 22.5

y = 1.7

z = 10.2

w ve x’in ortalaması : 30.000

w, x ve y‘in ortalaması : 20.567

w, x, y ve z‘in ortalaması : 17.975

Şekil 14.2 Değişken uzunlukta argüman listeleri kullanmak.

ortalama fonksiyonu, stdarg.h içindeki tüm makroları ve tanımlamaları kullanmaktadır.

va\_list tipindeki ap nesnesi, va\_start, va\_arg ve va\_end makroları tarafından değişken

uzunlukta argüman listesine sahip olan ortalama fonksiyonunu işlemek için kullanılmıştır.

Fonksiyon, va\_arg ve va\_end’in ap nesnesini kullanabilmesi için bu nesneye ilk değer atayan

va\_start makrosunu çağırmaktadır (satır 31). Makro iki argüman almaktadır: ap nesnesi ve

argüman listesinde üç noktadan önce en sağda yer alan tanıtıcı ( bu durum için tanıtıcı i ‘dir )

va\_start , i’yi değişken uzunluktaki argüman listesinin nereden başladığına karar vermek için

kullanmaktadır. Daha sonra, ortalama fonksiyonu değişken uzunluktaki argüman listesindeki

argümanları toplam değişkenine eklemektedir (satır 33-34). Argüman listesinden toplam

değişkenine eklenecek değer, va\_arg makrosu çağrılarak elde edilmektedir. va\_arg makrosu

iki argüman almaktadır: ap nesnesi ve argüman listesinden beklenen değerin tipi ( bu durum

için tip double‘dır) Makro, argümanın değerini döndürür. ortalama fonksiyonu, main

fonksiyonuna normal bir şekilde dönmek için va\_end makrosunu ap nesnesi ile çağırmaktadır

(satır 36). Sonuç olarak, ortalama hesaplanmakta ve main‘e döndürülmektedir.

Genel Programlama Hataları 14.1

Fonksiyonun parametre listesinin ortasına üç nokta yerleştirmek. Üç nokta yalnızca parametre

listesinin sonuna eklenebilir.

534

Okuyucu, printf ve scanf fonksiyonlarının va\_arg makrosunda kullanılacak tipleri nereden

bildiğini sorabilir. Bu sorunun cevabı, printf ve scanf’in, bir sonra işlenecek argümanın tipine

karar vermek için biçim kontrol dizesi içindeki biçim dönüşüm belirteçlerini taraması olarak

verilebilir.

14.4 KOMUT SATIRI ARGÜMANLARINI KULLANABİLMEK

Çoğu sistemlerde, özellikle de DOS ve UNIX sistemlerinde, main fonksiyonunun parametre

listesinde int argc ve char \*argv[ ] parametrelerini kullanarak, komut satırından main

fonksiyonuna argüman geçirmek mümkündür. argc parametresi, komut satırı argümanlarının

sayısını alır. argv parametresi, gerçek komut satır argümanlarının depolandığı bir string

dizisidir. Komut satırı argümanlarının genel kullanımları, argümanları yazdırmak, programa

seçenekler ve dosya isimleri geçirmektir.

Şekil 14.3, bir dosyadaki karakterleri teker teker diğer dosyaya kopyalamaktadır. Program

için çalıştırılabilecek dosya mycopy olarak adlandırılmıştır. mycopy programı için UNIX

sistemlerinde komut satırı şu şekilde kullanılır:

$ mycopy giris cikis

Komut satırı, giris dosyasının cikis dosyasına kopyalanacağını belirtmektedir. Program

çalıştırıldığında eğer argc 3 değilse ( mycopy argümanların biri olarak kabul edilir ), bir hata

mesajı yazdırılır ve program sonlanır. Aksi takdirde, argv dizisi “mycopy”, ”giris” ve

“cikis” stringlerini içerir. Komut satırındaki ikinci ve üçüncü argümanlar program tarafından

dosya ismi olarak kullanılır. Bu dosyalar fopen fonksiyonu kullanılarak açılır. Eğer iki

dosyada başarılı bir şekilde açılmışsa, giris dosyasındaki karakterler dosya sonu belirteciyle

karşılaşılıncaya kadar okunup, cikis dosyasına yazdırılır. Bütün bilgisayar sistemlerinin

komut satırını UNIX ya da DOS kadar basit bir şekilde desteklemediğine dikkat ediniz.

Örneğin, Macintosh ve VMS sistemleri komut satırı argümanlarını işlemek için özel ayarların

yapılmasına gereksinim duyarlar. Komut satırı argümanlarını işlemek için sisteminizin

kılavuzuna bakınız.

1 /\* Şekil 14.3: fig14\_03.c

2 Komut satırı argümanlarını kullanmak \*/

3 #include <stdio.h>

4

5 int main( int argc, char \*argv[ ] )

6 {

7 FILE \*girisDosyaPtr, \*cikisDosyaPtr;

8 int c;

9

10 if ( argc != 3 )

11 printf( "Kullanım:: copy girişDosyası ÇıkışDosyası\n" );

12 else

13 if ( ( girisDosyaPtr = fopen( argv[ 1 ], "r" ) ) != NULL )

14

15 if ( ( cikisDosyaPtr = fopen( argv[ 2 ], "w" ) ) != NULL )

16

17 while ( ( c = fgetc( girisDosyaPtr ) ) != EOF )

535

18 fputc( c, cikisDosyaPtr );

19

20 else

21 printf( "\"%s\" dosyası açılamadı\n", argv[ 2 ] );

22

23 else

24 printf( "\"%s\" dosyası açılamadı\n", argv[ 1 ] );

25

26 return 0;

27 }

Şekil 14.3 Komut satırı argümanlarını kullanmak

14.5 ÇOK KAYNAK DOSYALI PROGRAMLARI DERLEME

HAKKINDA NOTLAR

Daha önceden belirtildiği gibi, birden çok kaynak dosya içeren programlar oluşturmak

mümkündür (16.Ünite’ye bakınız) Birden çok dosya içinde program yaratırken göz önünde

tutulacak birkaç husus vardır. Örneğin, bir fonksiyon tanımının tamamı bir dosya içinde yer

almalı, başka dosyaların içine yayılmamalıdır.

5.ünitede, depolama sınıfı ve faaliyet alanı kavramlarını tanıtmıştık. Tüm fonksiyon

tanımlarının dışında bildirilen değişkenlerin aksi belirtilmedikçe statik depolama sınıfında

bulunduğunu ve global değişkenler olarak adlandırıldıklarını öğrenmiştik. Global değişkenler,

değişkenler bildirildikten sonra o dosyada tanımlanan tüm fonksiyonlar tarafından

kullanılabilirler. Ayrıca diğer dosyalardaki fonksiyonlarda global değişkenlere erişebilirler.

Ancak, global değişkenler kullanıldıkları her dosya içinde bildirilmelidirler. Örneğin, eğer bir

dosya içinde bayrak tamsayı global değişkenini tanımlarsak, ikinci dosya

extern int bayrak;

bildirimini, değişkenin kullanıldığı yerden önce içermek zorundadır. Az önceki bildirimde,

extern depolama sınıfı belirteci derleyiciye bayrak değişkeninin aynı dosya içinde daha

sonra ya da başka bir dosya içinde tanımlandığını bildirmektedir. Derleyici, bağlayıcıya dosya

içinde bayrak değişkeni için çözülemeyen referanslar bulunduğunu bildirir. (derleyici

bayrak değişkeninin nerede tanımlandığını bilmez, bu sebepten bayrak‘ı bulmayı

bağlayıcıya bırakır) Eğer bağlayıcı bayrak için bir tanımlama bulamazsa, bir bağlayıcı hatası

oluşturulur ve çalıştırılabilir bir program üretilmez. Eğer uygun bir global tanımlama

bulunursa, bağlayıcı bayrak’ın nerede bulunduğunu belirterek referansları çözer.

Performans İpuçları 14.1

Global değişkenler performansı arttırır çünkü her fonksiyon tarafından doğrudan

erişilebilirler, verinin fonksiyonlara geçirilmesi yükü ortadan kaldırılmış olur.

Yazılım Mühendisliği Gözlemleri 14.1

Global değişkenleri, uygulamanın performansı kritik olmadıkça kullanmamak gerekir çünkü

global değişkenler en az yetki prensibine uymazlar.

extern bildirimlerinin diğer program dosyaları için global değişkenler bildirmede kullanılması

gibi, fonksiyon prototipleri de bir fonksiyonun faaliyet alanını tanımlandığı dosyanın dışına

genişletebilir (fonksiyon prototipi içinde extern belirtecine gerek yoktur). Bu, fonksiyon

536

prototipini fonksiyonun çağrıldığı her dosya içine yazmak ve dosyaları birlikte derlemek

(bakınız Kısım 13.2) sayesinde gerçekleştirilir. Fonksiyon prototipleri derleyiciye, belirlenen

fonksiyonun aynı dosya içinde daha sonra ya da başka bir dosya içinde tanımlandığını

bildirmektedir. Derleyici yine böyle bir fonksiyon için referansları çözmeye kalkmaz, bu

görev bağlayıcıya bırakılır. Eğer bağlayıcı uygun bir fonksiyon tanımlaması bulamazsa bir

hata üretilir.

Fonksiyonların faaliyet alanlarını genişletmek için fonksiyon prototiplerinin kullanılmasına

bir örnek olarak, #include<stdio.h> önişlemci komutunu içeren bir program düşünelim. Bu

komut, printf ve scanf gibi fonksiyonların prototiplerini programa dahil eder. Dosyadaki

diğer fonksiyonlar, görevlerini yapmak için printf ve scanf ‘i kullanabilirler. printf ve scanf

fonksiyonları bizim için ayrıca tanımlanmıştır. Bu fonksiyonların nerede tanımlandıklarını

bilmemiz gerekmez. Biz programlarımızda, kodu yeniden kullanıyoruz. Bağlayıcı, bu

fonksiyonlara yaptığımız referansları otomatik olarak çözer. Bu süreç, standart kütüphanedeki

fonksiyonları kullanabilmemizi sağlar.

Yazılım Mühendisliği Gözlemleri 14.2

Programları birden çok dosya içinde yaratmak, yazılımın yeniden kullanılabilirliğini sağlar ve bu

sebepten iyi bir yazılım mühendisliğidir. Fonksiyonlar birçok uygulama için genel olabilir. Bu

durumlarda, ortak olarak kullanılacak fonksiyonlar kendi dosyaları içinde depolanmalı ve her kaynak

dosya, fonksiyon prototiplerini içeren ilgili bir öncü dosyaya sahip olmalıdır.Bu, programcıların

farklı uygulamalarda uygun öncü dosyayı ekleyerek ve uygulamalarını ilgili kaynak dosya ile birlikte

derleyerek aynı kodu yeniden kullanmalarını sağlatır.

Taşınırlık İpuçları 14.1

Bazı sistemler 6 karakterden uzun global değişken isimleri ve fonksiyon isimlerini desteklemez.Bu,

başka platformlara taşınacak programlar yazılırken göz önünde tutulması gereken bir husustur.

Bir global değişkenin ya da fonksiyonun faaliyet alanını, tanımlandığı dosya içinde kısıtlamak

mümkündür. static depolama sınıfı belirteci, global bir değişken ya da fonksiyona

uygulandığında, aynı dosyada tanımlanmamış bir fonksiyon tarafından kullanılmaları

engellenmiş olur. Buna iç bağlama (internal linkage) denir. Tanımlarında static içermeyen

global değişkenler ve fonksiyonlar dış bağlamaya (external linkage) sahiptir yani eğer

kullanılacakları dosyalar uygun bildirimleri ya da fonksiyon prototiplerini içeriyorsa, başka

dosyalar içinde de kullanılabilirler.

static double pi = 3.14159 ;

global değişken bildirimi, double tipteki pi değişkenini yaratmakta, 3.14159 değerine

atamakta ve pi’nin yalnızca tanımlandığı dosya içindeki fonksiyonlar tarafından

kullanılabileceğini belirtmektedir.

static belirteci, belli bir dosyadaki fonksiyonlar tarafından çağrılan görev fonksiyonları ile

kullanılır. Eğer bir fonksiyonun belli bir dosyanın dışında kullanılması gerekmiyorsa, static

kullanılarak en az yetki prensibine uyulmalıdır. Eğer bir fonksiyon dosyada kullanılmadan

önce tanımlanmışsa, static fonksiyon tanımına uygulanmalıdır. Aksi takdirde, static

fonksiyon prototipine uygulanmalıdır.

Birden çok kaynak dosya içinde geniş programlar yaratırken, programı derleme eğer bir

dosyada küçük bir değişiklik yapılırsa ve tüm programın yeniden derlenmesi gerekirse

usandırıcı olabilir. Çoğu sistem, yalnızca değişiklik yapılan dosyayı yeniden derleyen özel

537

hizmetler sağlar. UNIX sistemlerinde bu hizmet, make olarak bilinir. make hizmeti,

programın derlenmesi ve bağlanması hakkında emirler içeren makefile adındaki dosyayı

okur. Borland C++ ve Microsoft Visual C++ gibi ürünler de make hizmetlerini sağlar. make

hizmetleri hakkında daha fazla bilgi için sisteminizin kılavuzuna bakın.

14.6 exit VE atexit İLE PROGRAM SONLANDIRMA

Genel amaçlı kütüphane (stdlib.h), main fonksiyonundan geri dönmek yerine başka program

sonlandırma yöntemleri sağlar. exit fonksiyonu, programın normal bir şekilde çalışmış gibi

sonlanmasını sağlatır. Fonksiyon, genellikle girişte bir hata tespit edildiğinde ya da programda

işlenecek bir dosya açılamadığında programı sonlandırmak için kullanılır. atexit fonksiyonu

program içinde programın başarılı bir şekilde sonlandırılmasında (yani program main sonuna

ulaşarak sonlandığında ya da exit çağrıldığında) çağrılan bir fonksiyonu kaydeder.

atexit fonksiyonu, argüman olarak fonksiyonu gösteren (fonksiyon ismini) bir gösterici alır.

Program sonlandırmada çağrılan fonksiyonlar argümana sahip olamazlar ve değer

döndüremezler. Programın sonlanmasında çalıştırılacak en fazla 32 fonksiyon kaydedilebilir.

exit fonksiyonu bir argüman alır. Argüman normalde EXIT\_SUCCESS ya da

EXIT\_FAILURE sembolik sabitidir. Eğer exit fonksiyonu EXIT\_SUCCESS ile çağrılırsa,

çağırıcı ortama başarı için sisteme bağımlı olarak tanımlanmış değer döndürülür. Eğer exit

EXIT\_FAILURE ile çağrılırsa, sistemde başarısız sonlanma için tanımlanmış değer

döndürülür. exit fonksiyonu çağrıldığında, atexit ile kaydedilmiş fonksiyonlar kayıt

sıralarının tersine bir biçimde çağrılırlar, program ile ilgili akışlar kapatılır ve kontrol esas

ortama döndürülür. Şekil 14.4, exit ve atexit fonksiyonlarını test etmektedir. Program

kullanıcıya programın exit ile mi yoksa main sonuna ulaşılarak mı sonlandırılacağını sorar.

yaz fonksiyonunun programın sonlandırılmasında her durumda çalıştırıldığına dikkat ediniz.

1 /\* Şekil 14.4: fig14\_04.c

2 exit ve atexit fonksiyonlarını kullanmak.\*/

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 void yaz( void );

7

8 int main( )

9 {

10 int cevap;

11

12 atexit( yaz );

13 printf( "Programı exit fonksiyonu ile bitirmek için 1 giriniz"

14 "\nProgramdan normal çıkış için 2 giriniz\n" );

15 scanf( "%d", &cevap );

16

17 if ( cevap == 1 ) {

18 printf( "\nProgramdan exit fonksiyonu

19 kullanılarak çıkılıyor\n" );

20 exit( EXIT\_SUCCESS );

21 }

22

538

23 printf( "\nmain fonksiyonunun sonuna

24 ulaşılarak programdan çıkılıyor\n" );

25 return 0;

26 }

27

28 void yaz( void )

29 {

30 printf( "Program sonlanırken yaz fonksiyonu çalıştı\n

31 “Program sonlandı\n" );

32 }

Programı exit fonksiyonu ile bitirmek için 1 giriniz

Programdan normal çıkış için 2 giriniz

: 1

Programdan exit fonksiyonu kullanılarak çıkılıyor

Program sonlanırken yaz fonksiyonu çalıştı

Program sonlandı

Programı exit fonksiyonu ile bitirmek için 1 giriniz

Programdan normal çıkış için 2 giriniz

: 2

main fonksiyonunun sonuna ulaşılarak programdan çıkılıyor

Program sonlanırken yaz fonksiyonu çalıştı

Program sonlandı

Şekil 14.4 exit ve atexit fonksiyonlarını kullanmak.

14.7 volatile TİP BELİRTECİ

6 ve 7. ünitede, const tip belirtecini tanıtmıştık. ANSI C, ayrıca çeşitli iyileştirmelerin

yapılmasını engellemek için volatile tip belirtecini kullanmamıza izin verir. ANSI standardı

(An90) bir tipin belirtmesi için volatile kullanıldığında, o tipteki nesneye erişimin sistem

bağımlı olacağını belirtmiştir.

14.8 TAMSAYI VE ONDALIKLI SAYI SABİTLERİ İÇİN SON EKLER

C, tamsayı ve ondalıklı sayı sabitlerin tiplerinin belirlenmesi için sonekler kullanılmasına izin

vermektedir. Tamsayı sonekleri unsigned bir tamsayı için u ya da U, long tamsayılar için l ya

da L ve unsigned long tamsayı için ul, lu, UL ya da LU olarak belirlenmiştir. Aşağıdaki

sabitler sırasıyla unsigned, long ve unsigned long tiptedir:

174u

8358L

539

28373ul

Eğer bir tamsayı sabitine sonek eklenmemişse, tipi o büyüklükteki bir değeri tutabilecek ilk

tip olarak belirlenir (önce int, sonra long int ve sonrada unsigned long int)

Ondalıklı sayı sonekleri ise float için f ya da F, long double için l ya da L olarak

belirlenmiştir. Aşağıdaki sabitler sırasıyla float ve long double tipindedir:

1.28f

3.14159L

Sonek almamış bir ondalıklı sayı sabit otomatik olarak double tipinde olacaktır.

14.9 DOSYALAR ÜZERİNE NOTLAR

11 ünite, metin dosyalarını sıralı erişim ve rasgele erişimle işleme yeteneklerini tanıtmıştı. C, ayrıca

ikili dosyaları (binary file) işleme yeteneğine de sahiptir ancak bazı bilgisayar sistemleri ikili

dosyaları desteklemez. Eğer ikili dosyalar desteklenmiyorsa ve dosya ikili dosya modunda açılırsa

(Şekil 14.5), dosya metin dosyası olarak ele alınır. İkili dosyalar metin dosyaları yerine yalnızca hız,

depolama ve/veya uyum koşulları ikili dosyalara dayandığı durumlarda kullanılmalıdır. Aksi takdirde,

metin dosyaları her zaman doğal taşınılabilirlikleri ve dosyadaki veriyi araştırmak ve işlemek için

standart araçları kullanabilme yetenekleri için tercih edilir.

Performans İpuçları 14.2

İkili dosyaları metin dosyaları yerine yalnızca yüksek performansa gerek duyan uygulamalarda

kullanın.

Taşınırlık İpuçları 14.2

Taşınılabilir programlar yazarken metin dosyalarını kullanın.

11.ünitede anlatılan dosya işleme fonksiyonlarına ek olarak, standart kütüphane “wb+”

modunda geçici bir dosya açan tmpfile fonksiyonunu da sunmaktadır. Bu mod, ikili dosya

modu olsa da bazı sistemler geçici dosyaları metin dosyaları olarak işlerler. Geçici bir dosya

fclose ile kapatılana kadar ya da program sonlanana kadar var olur.

Şekil 14.6, bir dosyadaki tab karakterlerini boşluk karakterleriyle değiştirmektedir. Program,

kullanıcıya değiştirilecek dosyanın adını girmesini belirten bir mesaj yazdırır. Eğer kullanıcı

tarafından girilen dosya ve geçici dosya başarılı bir şekilde açılırsa, program değiştirilecek

dosyadaki karakterleri okur ve geçici dosyaya yazar. Eğer tab ( ‘\t’ ) karakteri ile karşılaşılırsa

boşluk ile değiştirilir ve geçici dosyaya yazdırılır. Değiştirilen dosyanın sonuna ulaşıldığında

her dosyanın dosya göstericileri rewind ile dosyaların başlarını gösterecek şekilde yeniden

konumlandırılır. Daha sonra, geçici dosya orijinal dosyaya karakter karakter kopyalanır.

Program, orijinal dosya karakterleri geçici dosyaya kopyalarken yazdırmaktadır ve

karakterlerin yazıldığını göstermek için yeni dosyayı geçici dosyadan karakterleri

kopyalarken yazdırmaktadır.

Mode Açıklama

rb İkili bir dosyayı okumak için aç

540

wb Yazmak için ikili bir dosya yarat. Eğer dosya daha önceden varsa,o andaki içeriğini

siler.

ab Ekle;ikili bir dosyayı okumak için ya da dosyanın sonuna yazma yapmak için aç

rb+ İkili bir dosyayı güncellemek için aç (okuma ve yazma yapmak için)

wb+ Güncellemek için ikili bir dosya yarat. Eğer dosya daha önceden varsa,o andaki

içeriğini sil.

ab+ Ekle; ikili bir dosyayı güncellemek için aç ya da yarat. Tüm yazma dosyanın sonuna

yapılır.

Şekil 14.5 İkili dosya açma modları

1 /\* Fig. 14.6: fig14\_06.c

2 Geçici dosyaları kullanmak. \*/

3 #include <stdio.h>

4

5 int main( )

6 {

7 FILE \*dosyaPtr, \*geciciDosyaPtr;

8 int c;

9 char dosyaAdi[ 30 ];

10

11 printf( "Bu program tab boşluklarını tek boşluğa çevirir.\n"

12 "Değiştirilecek dosyayı giriniz: " );

13 scanf( "%29s", dosyaAdi );

14

15 if ( ( dosyaPtr = fopen( dosyaAdi, "r+" ) ) != NULL )

16

17 if ( ( geciciDosyaPtr = tmpfile( ) ) != NULL ) {

18 printf( "\nDosya değişmeden önce:\n" );

19

20 while ( ( c = getc(dosyaPtr) ) != EOF ) {

21 putchar( c );

22 putc( c == '\t' ? ' ': c, geciciDosyaPtr );

23 }

24

25 rewind( geciciDosyaPtr );

26 rewind( dosyaPtr );

27 printf( "\n\nDosya değiştirildikten sonra:\n" );

28

29 while ( ( c = getc(geciciDosyaPtr) ) != EOF ) {

30 putchar( c );

31 putc( c, dosyaPtr );

32 }

33

34 }

35 else

36 printf( "Geçici dosya açılamadı\n" );

37 else

38 printf( " %s dosyası açılamadı\n", dosyaAdi );

39

541

40 return 0;

41 }

Bu program tab boşluklarını tek boşluğa çevirir.

Değiştirilecek dosyayı giriniz: veri

Dosya değişmeden önce:

0 1 2 3 4

5 6 7 8 9

Dosya değiştirildikten sonra:

0 1 2 3 4

5 6 7 8 9

Şekil 14.6 Geçici dosyaları kullanmak.

14.10 SİNYAL İŞLEME

Beklenmeyen bir olay ya da sinyal, programın beklenenden daha önce sonlanmasına yol açabilir. Bazı

beklenmeyen olaylar kesme (UNIX ve DOS sitemlerinde <ctrl>-c yazmak), illegal emirler, segment

kısıtlamaları, işletim sistemi tarafından gönderilen sonlandırma emirleri ve ondalıklı sayı istisnaları

(sıfıra bölme yada büyük ondalıklı sayıları çarpma) olabilir. Sinyal işleme kütüphanesi, beklenmeyen

olayları signal fonksiyonu ile yakalayabilme yeteneğini sunar. signal fonksiyonu iki argüman alır:

tamsayı olan bir sinyal sayısı ve sinyal işleme fonksiyonunu gösteren bir gösterici. Sinyaller argüman

olarak tamsayı olan bir sinyal sayısı alan raise fonksiyonu tarafından üretilebilirler. Şekil 14.7,

signal.h içinde tanımlanmış standart sinyalleri özetlemektedir. Şekil 14.8, signal ve raise

fonksiyonlarını göstermektedir.

Şekil 14.8, signal fonksiyonunu etkileşimli bir sinyali (SIGINT) yakalamak için kullanmaktadır.

Program signal fonksiyonunu, SIGINT ve sinyal\_isleyici fonksiyonunu gösteren bir göstericiyle

(fonksiyon isminin fonksiyonun başlangıcını gösteren bir gösterici olduğuna dikkat ediniz) çağırarak

başlamaktadır ( satır 14 ). SIGINT tipinde bir sinyal üretildiğinde, kontrol sinyal\_isleyici

fonksiyonuna geçer, bir mesaj yazdırılır ve kullanıcıya programın normal çalışmasını devam ettirme

seçeneği verilir. Eğer kullanıcı programın çalışmasının devamını istiyorsa, signal yeniden çağırılarak

sinyal işeyiciye yeni değer verilir. (bazı sistemler sinyal işleyicinin yeniden ilk değere atanmasına

ihtiyaç duyarlar) Daha sonrada kontrol, programda sinyalin yakalandığı noktaya döndürülür. Bu

programda raise fonksiyonu (satır 21), etkileşimli bir sinyal üretmek için kullanılmıştır. 1 ile 50

arasında rasgele bir sayı seçilmiştir. Eğer sayı 25 ise raise, sinyali üretmek için çağrılmıştır.

Normalde etkileşimli sinyallere ilk değerler programın dışında verilir. Örneğin, UNIX ya da DOS

sistemlerinde program çalışırken <ctrl>-c yazmak programın çalışmasını sonlandıran etkileşimli bir

sinyal yaratır. Sinyal işleme, etkileşimli sinyali yakalamak ve programın sonlanmasını engellemek için

kullanılabilir.

Sinyal Açıklama

SIGABRT Programın normal olmayan şekilde sonlanması (abort fonksiyonuna yapılan

bir çağrıdaki gibi)

SIGFPE Hatalı bir aritmetik işlem, örneğin sıfıra bölme ya da taşma ile sonuçlanacak

bir işlem

SIGILL İllegal emrin tespiti

542

SIGINT Etkileşimli sinyalin alınması

SIGSEGV Depolamaya geçersiz erişim

SIGTERM Programı sonlandırma isteği

Şekil 14.7 signal.h öncüsünde tanımlı sinyaller.

1 /\* Şekil 14.8: fig14\_08.c

2 Sinyal işleme kullanmak \*/

3 #include <stdio.h>

4 #include <signal.h>

5 #include <stdlib.h>

6 #include <time.h>

7

8 void sinyal\_isleyici( int );

9

10 int main( )

11 {

12 int i, x;

13

14 signal( SIGINT, sinyal\_isleyici );

15 srand( clock( ) );

16

17 for ( i = 1; i <= 100; i++ ) {

18 x = 1 + rand( ) % 50;

19

20 if ( x == 25 )

21 raise( SIGINT );

22

23 printf( "%4d", i );

24

25 if ( i % 10 == 0 )

26 printf( "\n" );

27 }

28

29 return 0;

30 }

31

32 void sinyal\_isleyici( int sinyalDegeri )

33 {

34 int yanit;

35

36 printf( "%s%d%s\n%s",

37 "\nKesme sinyali ( ", sinyalDegeri, " ) alındı.",

38 "Devam etmek istiyor musunuz ( 1 = evet ya da 2 = hayır )? " );

39

40 scanf( "%d", &yanit );

41

42 while ( yanit != 1 && yanit != 2 ) {

43 printf( "( 1 = evet ya da 2 = hayır )? " );

44 scanf( "%d", &yanit );

45 }

46

47 if (yanit == 1 )

48 signal( SIGINT, sinyal\_isleyici );

49 else

543

50 exit( EXIT\_SUCCESS );

51 }

1 2 3 4 5 6 7 8 9 10

11 12 13 14 15 16 17 18 19 20

21 22 23 24 25 26 27 28 29 30

31 32 33 34 35 36 37 38 39 40

41 42 43 44 45 46 47 48 49 50

51 52 53 54 55 56 57 58 59 60

Kesme sinyali ( 2 ) alındı.

Devam etmek istiyor musunuz ( 1 = evet ya da 2 = hayır )? 1

60

61 62 63 64 65 66 67 68 69 70

71 72 73 74 75 76 77 78 79 80

81 82 83 84 85 86 87 88 89 90

91 92 93 94 95 96 97 98 99 100

Şekil 14.8 Sinyal işleme kullanmak

14.11 DİNAMİK HAFIZA TAHSİSİ: calloc VE realloc FONKSİYONLARI

12. ünitede malloc fonksiyonu kullanarak dinamik hafıza tahsisi yapmayı anlatmıştık. 12. ünitede

belirttiğimiz gibi, diziler hızlı sıralama, arama ve veri erişiminde bağlı listelere göre daha iyidir.

Ancak, diziler genellikle statik veri yapılarıdır. Genel amaçlı kütüphane (stdlib.h) dinamik hafıza

tahsisi için iki fonksiyon daha sunar: calloc ve realloc. Bu fonksiyonlar, dinamik dizilerin yaratılması

ve değiştirilmesi için kullanılır. 7. ünitede gösterildiği gibi diziyi gösteren gösterici, dizilerde olduğu

gibi belirteçlerle kullanılabilir. Bu sebepten, calloc ile yaratılan ve birbirine bitişik bir hafıza alanını

gösteren gösterici bir dizi gibi yönetilebilir. calloc fonksiyonu bir dizi için dinamik olarak hafıza

tahsisi yapar. calloc fonksiyonunun prototipi

void \*calloc (size\_t nmemb , size\_t size);

şeklindedir. Bu prototip iki argüman alır: eleman sayısı (nmemb) ve her elemanın boyutu

(size). Ayrıca dizinin elemanlarını sıfıra atar. Fonksiyon, tahsis edilen alanı gösteren bir

gösterici ya da hafıza tahsis edilemezse NULL döndürür. malloc ile calloc arasındaki temel

fark calloc fonksiyonunun tahsis ettiği hafızayı temizlemesi ancak malloc’un

temizlememesidir.

realloc fonksiyonu kendinden önce malloc,calloc ya da realloc çağırılarak tahsis edilmiş

nesnenin boyutunu değiştirir. Orijinal nesnenin içeriği, eğer tahsis edilen yeni alan daha

önceden tahsis edilen alandan daha büyükse değiştirilmeden korunur.Aksi takdirde, yeni

nesnenin boyutuna ulaşılıncaya kadar içerik değiştirilmeden korunur.realloc fonksiyonunun

prototipi aşağıda verilmiştir:

void \*realloc(void \*ptr,size\_t size);

realloc fonksiyonu iki argüman alır ; orijinal nesneyi gösteren bir gösterici ( ptr ) ve nesnenin

yeni boyutu ( size ). Eğer ptr NULL ise, realloc fonksiyonu malloc ile eşdeğer biçimde

çalışır. Eğer size 0 ve ptr NULL değilse, nesne için kullanılan hafıza serbest bırakılır. Eğer

544

ptr NULL değil ve size sıfırdan büyükse, realloc nesne için yeni bir hafıza alanı tahsis

etmeye çalışır. Eğer yeni alan tahsis edilemezse, ptr tarafından gösterilen nesne değiştirilmez.

realloc fonksiyonu yeniden tahsis edilmiş alanı gösteren bir gösterici ya da NULL gösterici

döndürür.

14.12 KOŞULSUZ DALLANMA: goto

Tüm kitap boyunca hata ayıklaması, değiştirmesi ve geliştirmesi kolay güvenilir programlar

oluşturabilmek için yapısal programlama teknikleri kullanmanın önemini anlattık. Bazı

durumlarda performans yapısal programlama tekniklerine bağlı kalmaktan daha önemlidir. Bu

durumlarda, yapısal olmayan bazı programlama teknikleri kullanılabilir. Örneğin, döngü

devam koşulu yanlış olmadan döngüden çıkmak için break kullanabiliriz. Bu, döngü

sonlanmadan önce istenen görev yerine getirilmiş olursa döngünün kalan kısımlarının

tekrarını engeller.

Yapısal olmayan programlamanın başka bir örneği koşulsuz bir dallanma olan goto ifadesidir.

goto ifadesinin sonucu programın akışındaki kontrolün değişimidir. Bu değişim, goto

ifadesinde belirtilen etiketten sonraki ilk ifadeye gidilmesi ile sağlanır. Bir etiket, tanıtıcıdan

sonra iki nokta üst üste konarak oluşturulur. Bir etiket, kontrolü kendisine gönderen goto

ifadesi ile aynı fonksiyon içinde bulunmalıdır. Şekil 14.9, on kez tekrarlanan ve her tekrarda

sayıcının değerinin yazdırıldığı bir döngü yaratmak için goto ifadeleri kullanmaktadır.

sayici’ya ilk değer olarak 1 verildikten sonra program sayici’nin 10’dan büyük olup

olmadığına karar vermektedir ( basla etiketi atlanır çünkü etiketler her hangi bir işlem

yaptıramazlar ). Eğer sayici 10’dan büyükse, kontrol goto ifadesinden son etiketinden sonra

yer alan ilk ifadeye aktarılır. Aksi takdirde ise sayici yazdırılır, arttırılır ve kontrol goto

ifadesinden alınıp basla etiketinden sonraki ilk ifadeye aktarılır.

3.ünitede herhangi bir programı yazmak için yalnızca üç kontrol yapısının (sıra,seçim ve

döngü yapıları) yeterli olacağını söylemiştik. Yapısal programlamanın kurallarına

uyulduğunda içinden verimli bir şekilde çıkılması güç yuvalı kontrol yapıları yaratmak

mümkündür. Bazı programcılar bu durumlarda yuvalı yapıdan hızlıca çıkabilmek için goto

ifadelerini kullanabilirler. Bu, bir kontrol yapısından çıkmak için bir çok koşulun test edilmesi

gerekliliğini ortadan kaldırır.

Performans İpuçları 14.3

goto ifadeleri yuvalı kontrol yapılarından verimli bir çıkış yapılması için kullanılabilir.

Yazılım Mühendisliği Gözlemleri 14.3

goto ifadeleri yalnızca performansın ön planda tutulduğu uygulamalarda kullanılmalıdır.goto ifadesi

yapısal değildir ve hata ayıklaması, geliştirmesi ve değiştirmesi oldukça güç programlar yazılmasına

yol açabilir.

1 /\* Şekil 14.9: fig14\_09.c

2 goto kullanmak \*/

3 #include <stdio.h>

4

5 int main( )

6 {

545

7 int sayici = 1;

8

9 basla: /\* etiket \*/

10 if ( sayici > 10 )

11 goto son;

12

13 printf( "%d ", sayici);

14 ++sayici;

15 goto basla;

16

17 son: /\* etiket\*/

18 putchar( '\n' );

19

20 return 0;

21 }

1 2 3 4 5 6 7 8 9 10

Şekil 14.9 goto kullanmak.

ÖZET

 Çoğu bilgisayar sisteminde (özellikle UNIX ve DOS sistemlerinde), girişin klavye

yerine bir dosyadan gelecek biçimde ve çıkışın ekran yerine dosyaya yerleştirilecek

biçimde değiştirilmesi mümkündür.

 Giriş, UNIX ve DOS işletim sistemlerinde komut satırından girişi değiştir sembolü(< )

ve pipe sembolü( | ) ile yeniden yönlendirilebilir.

 Program çıktısı, UNIX ve DOS işletim sistemlerinde komut satırından çıkış

yönlendirme sembolü ( > ) ya da dosyanın sonuna çıktı ekle sembolü (>>) ile yeniden

yönlendirilebilir. Çıkış yönlendirme sembolü, program çıktısını bir dosya içinde saklar

ve dosyanın sonuna çıktı ekle sembolü (>>) , çıktıları bir dosyanın sonuna ekler.

 Değişken argüman öncü dosyası stdarg.h’ın makroları ve tanımlamaları, değişken

uzunlukta argüman listeleri kullanan fonksiyonları oluşturmak için gerekli yetenekleri

sağlar.

 Fonksiyon prototipi içindeki üç nokta ( ... ), fonksiyonun değişken sayıda argüman

alabileceğini belirtir.

 va\_list tipi, va\_start , va\_arg ve va\_end makrolarının ihtiyaç duyduğu bilgiyi tutmak

için uygun bir tiptir. Değişken uzunlukta argüman listesindeki argümanlara erişmek

için va\_list tipinde bir nesne bildirilmelidir.

 va\_list makrosu, değişken uzunlukta bir argüman listesindeki argümanlara erişmeden

önce çağrılan bir makrodur. Makro,va\_arg ve va-end makrolarının kullanması için

va\_list ile bildirilmiş nesneye ilk değer atar.

 va\_arg makrosu, değişken uzunlukta argüman listesi içindeki bir sonraki argümanın

tipi ve değerinde bir deyime genişleyen bir makrodur. va\_arg için yapılan her çağrı,

va\_list ile bildirilmiş nesneyi listedeki bir sonraki argümanı gösterecek biçimde

değiştirir.

546

 va\_end makrosu, değişken uzunluktaki argüman listesi va\_start ile belirlenmiş bir

fonksiyondan, normal geri dönüş yapılmasını sağlayan bir makrodur.

 Çoğu sistemlerde,özellikle de DOS ve UNIX sistemlerinde, main fonksiyonunun

parametre listesinde int argc ve char \*argv[ ] parametrelerini kullanarak, komut

satırından main fonksiyonuna argüman geçirmek mümkündür. argc parametresi,

komut satırı argümanlarının sayısını alır. argv parametresi, gerçek komut satır

argümanlarının depolandığı bir string dizisidir.

 Bir fonksiyon tanımının tamamı bir dosya içinde yer almalı , başka dosyaların içine

yayılmamalıdır.

 Global değişkenler kullanıldıkları her dosya içinde bildirilmelidirler.

 Fonksiyon prototipleri bir fonksiyonun faaliyet alanını, tanımlandığı dosyanın dışına

genişletebilir (fonksiyon prototipi içinde extern belirtecine gerek yoktur). Bu,

fonksiyon prototipini fonksiyonun çağrıldığı her dosya içine yazmak ve dosyaları

birlikte derlemek sayesinde gerçekleştirilir.

 static depolama sınıfı belirteci, global bir değişken ya da fonksiyona uygulandığında,

aynı dosyada tanımlanmamış bir fonksiyon tarafından kullanılmaları engellenmiş olur.

Buna iç bağlama (internal linkage) denir. Tanımlarında static içermeyen global

değişkenler ve fonksiyonlar dış bağlamaya (external linkage) sahiptir yani eğer

kullanılacakları dosyalar uygun bildirimleri ya da fonksiyon prototiplerini içeriyorsa,

başka dosyalar içinde de kullanılabilirler.

 static belirteci, belli bir dosyadaki fonksiyonlar tarafından çağrılan görev

fonksiyonları ile kullanılır. Eğer bir fonksiyonun belli bir dosyanın dışında

kullanılması gerekmiyorsa, static kullanılarak en az yetki prensibine uyulmalıdır.

 Birden çok kaynak dosya içinde geniş programlar yaratırken, programı derleme eğer

bir dosyada küçük bir değişiklik yapılırsa ve tüm programın yeniden derlenmesi

gerekirse usandırıcı olabilir. Çoğu sistem, yalnızca değişiklik yapılan dosyayı yeniden

derleyen özel hizmetler sağlar. UNIX sistemlerinde bu hizmet, make olarak

bilinir.make hizmeti, programın derlenmesi ve bağlanması hakkında emirler içeren

makefile adındaki dosyayı okur.

 exit fonksiyonu, programın normal bir şekilde çalışmış gibi sonlanmasını sağlatır.

 atexit fonksiyonu program içinde programın başarılı bir şekilde sonlandırılmasında

(yani program main sonuna ulaşarak sonlandığında ya da exit çağrıldığında) çağrılan

bir fonksiyonu kaydeder.

 atexit fonksiyonu, argüman olarak fonksiyonu gösteren (fonksiyon ismini) bir

gösterici alır. Program sonlandırmada çağrılan fonksiyonlar argümana sahip olamazlar

ve değer döndüremezler. Programın sonlanmasında çalıştırılacak en fazla 32

fonksiyon kaydedilebilir.

 exit fonksiyonu bir argüman alır. Argüman normalde EXIT\_SUCCESS ya da

EXIT\_FAILURE sembolik sabitidir. Eğer exit fonksiyonu EXIT\_SUCCESS ile

çağrılırsa, çağırıcı ortama başarı için sisteme bağımlı olarak tanımlanmış değer

döndürülür.Eğer exit EXIT\_FAILURE ile çağrılırsa, sistemde başarısız sonlanma

için tanımlanmış değer döndürülür. exit fonksiyonu çağrıldığında, atexit ile

kaydedilmiş fonksiyonlar kayıt sıralarının tersine bir biçimde çağrılırlar, program ile

ilgili akışlar kapatılır ve kontrol esas ortama döndürülür.

 ANSI standardı (An90) bir tipin belirtmesi için volatile kullanıldığında, o tipteki

nesneye erişimin sistem bağımlı olacağını belirtmiştir.

 C, tamsayı ve ondalıklı sayı sabitlerin tiplerinin belirlenmesi için sonekler

kullanılmasına izin vermektedir.Tamsayı sonekleri unsigned bir tamsayı için u ya da

U, long tamsayılar için l ya da L ve unsigned long tamsayı için ul, lu, UL ya da LU

547

olarak belirlenmiştir. Eğer bir tamsayı sabitine sonek eklenmemişse, tipi o

büyüklükteki bir değeri tutabilecek ilk tip olarak belirlenir (önce int , sonra long int ve

sonrada unsigned long int). Ondalıklı sayı sonekleri ise float için f ya da F, long

double için l ya da L olarak belirlenmiştir. Sonek almamış bir ondalıklı sayı sabiti

otomatik olarak double tipinde olacaktır.

 C, ayrıca ikili dosyaları işleme yeteneğine de sahiptir ancak bazı bilgisayar sistemleri

ikili dosyaları desteklemez. Eğer ikili dosyalar desteklenmiyorsa ve dosya ikili dosya

modunda açılırsa, dosya metin dosyası olarak ele alınır.

 Tmpfile fonksiyonu “wb+” modunda geçici bir dosya açar. Bu mod ikili dosya modu

olsa da bazı sistemler geçici dosyaları metin dosyaları olarak işlerler. Geçici bir dosya

fclose ile kapatılana kadar ya da program sonlanana kadar var olur.

 Sinyal işleme kütüphanesi beklenmeyen olayları signal fonksiyonu ile yakalayabilme

yeteneğini sunar. signal fonksiyonu iki argüman alır: tamsayı olan bir sinyal sayısı ve

sinyal işleme fonksiyonunu gösteren bir gösterici.

 Sinyaller argüman olarak tamsayı olan bir sinyal sayısı alan raise fonksiyonu

tarafından üretilebilirler.

 Genel amaçlı kütüphane(stdlib.h) dinamik hafıza tahsisi için iki fonksiyon daha sunar:

calloc ve realloc. Bu fonksiyonlar, dinamik dizilerin yaratılması ve değiştirilmesi için

kullanılır

 calloc fonksiyonu iki argüman alır: eleman sayısı (nmemb) ve her elemanın boyutu

(size). Ayrıca dizinin elemanlarını sıfıra atar. Fonksiyon tahsis edilen alanı gösteren

bir gösterici ya da hafıza tahsis edilemezse NULL döndürür.

 realloc fonksiyonu kendinden önce malloc, calloc ya da realloc çağırılarak tahsis

edilmiş nesnenin boyutunu değiştirir. Orijinal nesnenin içeriği, eğer tahsis edilen yeni

alan daha önceden tahsis edilen alandan daha büyükse değiştirilmeden korunur.Aksi

takdirde, yeni nesnenin boyutuna ulaşılıncaya kadar içerik değiştirilmeden korunur.

 realloc fonksiyonu iki argüman alır; orijinal nesneyi gösteren bir gösterici (ptr) ve

nesnenin yeni boyutu (size). Eğer ptr NULL ise, realloc fonksiyonu malloc ile

eşdeğer biçimde çalışır. Eğer size 0 ve ptr NULL değilse, nesne için kullanılan hafıza

serbest bırakılır. Eğer ptr NULL değil ve size sıfırdan büyükse, realloc nesne için

yeni bir hafıza alanı tahsis etmeye çalışır. Eğer yeni alan tahsis edilemezse, ptr

tarafından gösterilen nesne değiştirilmez.realloc fonksiyonu yeniden tahsis edilmiş

alanı gösteren bir gösterici ya da NULL gösterici döndürür.

 goto ifadesinin sonucu programın akışındaki kontrolün değişimidir. Programın

çalışması,goto ifadesinde belirtilen etiketten sonraki ilk ifadeden devam eder.

 Bir etiket, tanıtıcıdan sonra iki nokta üst üste konarak oluşturulur. Bir etiket, kontrolü

kendisine gönderen goto ifadesi ile aynı fonksiyon içinde bulunmalıdır.

ÇEVRİLEN TERİMLER

append output symbol.................. çıktı ekle sembolü (>>)

command line arguments............. komut satır argümanları

dynamic arrays............................ dinamik diziler

floating-point exception............. ondalıklı sayı istisnası

internal linkage......................... iç bağlama

interrupt................................... kesme

pipe........................................

piping.....................................

548

redirect input symbol............ girişi değiştir sembolü

redirect output symbol......... çıkış yönlendirme sembolü ( > )

temporary file...................... geçici dosya

variable length arument list.. uzunluğu değişebilen argüman listesi

GENEL PROGRAMLAMA HATALARI

14.1 Fonksiyonun parametre listesinin ortasına üç nokta yerleştirmek.Üç nokta yalnızca

parametre listesinin sonuna eklenebilir.

TAŞINIRLIK İPUÇLARI

14.1 Bazı sistemler 6 karakterden uzun global değişken isimleri ve fonksiyon isimlerini

desteklemez.Bu, başka platformlara taşınacak programlar yazılırken göz önünde

tutulması gereken bir husustur.

14.2 Taşınılabilir programlar yazarken metin dosyalarını kullanın.

PERFORMANS İPUÇLARI

14.1 Global değişkenler performansı arttırır çünkü her fonksiyon tarafından doğrudan

erişilebilirler, verinin fonksiyonlara geçirilmesi yükü ortadan kaldırılmış olur.

14.2 İkili dosyaları metin dosyaları yerine yalnızca yüksek performansa gerek duyan

uygulamalarda kullanın.

14.3 goto ifadeleri yuvalı kontrol yapılarından verimli bir çıkış yapılması için kullanılabilir.

YAZILIM MÜHENDİSLİĞİ GÖZLEMLERİ

14.1 Global değişkenleri, uygulamanın performansı kritik olmadıkça

kullanmamak gerekir

çünkü global değişkenler en az yetki prensibine uymazlar.

14.2 Programları birden çok dosya içinde yaratmak, yazılımın yeniden

kullanılabilirliğini sağlar ve bu sebepten iyi bir yazılım

mühendisliğidir.Fonksiyonlar birçok uygulama için genel olabilir. Bu

durumlarda , ortak olarak kullanılacak fonksiyonlar kendi dosyaları

içinde depolanmalı ve her kaynak dosya, fonksiyon prototiplerini içeren

ilgili bir öncü dosyaya sahip olmalıdır.Bu, programcıların farklı

uygulamalarda uygun öncü dosyayı ekleyerek ve uygulamalarını ilgili

kaynak dosya ile birlikte derleyerek aynı kodu yeniden kullanmalarını

sağlatır.

14.3 goto ifadeleri yalnızca performansın ön planda tutulduğu uygulamalarda kullanılmalıdır.

goto ifadesi yapısal değildir ve hata ayıklaması,geliştirmesi ve değiştirmesi oldukça güç

programlar yazılmasına yol açabilir.

549

ÇÖZÜMLÜ ALIŞTIRMALAR

14.1 Aşağıdaki boşlukları doldurunuz.

a) \_\_\_\_\_\_ sembolü, klavyeden yapılacak girişinin, bir dosyadan yapılmasını sağlar.

b) \_\_\_\_\_\_ sembolü, ekran çıktısını bir dosyaya yerleştirmek için kullanılır.

c) \_\_\_\_\_\_ sembolü bir programın çıktısını, bir dosyanın sonuna eklemek için kullanılır.

d) \_\_\_\_\_\_ bir programın çıktısının diğer bir programa giriş olarak kullanılmasını

sağlar.

e) Bir fonksiyonun parametre listesindeki \_\_\_\_\_\_, o fonksiyonun farklı sayılarda

argüman alabileceğini gösterir.

f) Uzunluğu değişebilen argüman listelerinde, argümanlara erişmeden önce \_\_\_\_\_\_

makrosu mutlaka çağırılmalıdır.

g) \_\_\_\_\_\_ makrosu, uzunluğu değişebilen argüman listelerinde argümanlara ayrı ayrı

erişmek için kullanılır.

h) \_\_\_\_\_\_ makrosu, uzunluğu değişebilen argüman listesi va\_start makrosu ile

ilişkilendirilmiş bir fonksiyondan geri dönmeyi kolaylaştırır.

i) main fonksiyonunun \_\_\_\_\_\_ argümanı komut satırındaki argüman sayısını alır.

j) main fonksiyonunun \_\_\_\_\_\_ argümanı komut satırı argümanlarını karakter stringleri

şeklinde saklar.

k) UNIX’te \_\_\_\_\_\_ hizmeti, çoklu kaynak dosyalarını derleme ve bağlama

işlemlerinde kullanılacak olan direktifleri içeren \_\_\_\_\_\_ dosyasını okur.Bu hizmet

sayesinde bir dosya, eğer en son derlenmesinden sonra dosyada bir değişiklik

yapıldıysa tekrar derlenir.

l) \_\_\_\_\_\_ fonksiyonu programın sonlanmasını sağlar.

m) \_\_\_\_\_\_ fonksiyonu bir diğer fonksiyonu program normal olarak sonlandığında

çalıştırır.

n) \_\_\_\_\_\_ tip belirteci, bir nesnenin ilk değeri verildikten sonra bir daha

değiştirilemeyeceğini gösterir.

o) Bir tamsayı ya da ondalıklı sayı \_\_\_\_\_\_, bir tamsayı ya da ondalıklı sayı sabitine

eklenerek sabitin esas tipini belirtir.

p) \_\_\_\_\_\_ fonksiyonu, program sonlanılana kadar ya da kullanıcı tarafından

kapatılıncaya kadar açık kalan geçici bir dosya açar.

q) \_\_\_\_\_\_ fonksiyonu beklenmeyen olayları yakalamada kullanılır.

r) \_\_\_\_\_\_ fonksiyonu bir programda sinyal üretir.

s) \_\_\_\_\_\_ fonksiyonu bir dizi için dinamik hafıza tahsisi yapar ve dizinin ilk

değerlerine sıfır atar.

t) \_\_\_\_\_\_ fonksiyonu daha önceden tahsisi yapılan hafıza bloğunun boyutunu

değiştirir.

ÇÖZÜMLER

14.1 a) (<) b) (>) c) (>>) d) (|) e) (...) f) va\_start g) va\_arg h) va\_end i) argc j) argv k)

make, makefile l) exit m) atexit n) const o) soneki p) tmpfile q) signal r) raise s) calloc t)

realloc

ALIŞTIRMALAR

550

14.2 carpim fonksiyonuna gönderilen bir grup tamsayının çarpımını, değişebilir uzunlukta

argüman listesi içeren bir fonksiyon yardımıyla hesaplayınız. Her seferinde farklı sayıda

argüman ile fonksiyonunuzu çağırarak test ediniz.

14.3 Komut satırı argümanlarını ekrana yazdıran bir program yazınız.

14.4 Bir tamsayısı dizisini artan yada azalan sırada sıralayan bir program yazınız.

Programınız, –a komut satırı argümanını artan sıralama yapmak için, -d argümanını ise

azalan sıralama yapmak için kullanmalıdır.(Not: Bu uygulama, UNIX işletim sisteminde

bir programa seçenek sunmanın standart biçimidir)

14.5 Bir dosyadaki bütün karakterlerin arasına boşluk koyan bir program yazınız.

Programınız önce dosyanın bütün karakterleri arasına boşluk konulmuş halini geçici bir

dosyaya yazsın ve daha sonra bu dosyayı orijinal dosya olarak kaydetsin.

14.6 Sisteminizin el kitaplarını okuyarak, sisteminizin sinyal işleme kütüphanesindeki

(signal.h) hangi sinyalleri desteklediğini öğreniniz. SIGABRT ve SIGINT standart

sinyalleri için sinyal işleyicileri içeren bir program yazınız. Programınız bu sinyallerin

yakalanmasını, SIGABRT sinyali üreten ve <ctrl>c tuş kombinasyonuna basıldığında

SIGINT tipinde bir sinyal üreten abort fonksiyonunu çağırarak test etmelidir.

14.7 Bir tamsayı dizisinin dinamik tahsisini yapan bir program yazınız. Dizi elemanlarına

klavyeden değerler alınmalıdır ve ekrana yazdırılmalıdır. Daha sonra, hafızayı dizinin

eleman sayısının yarısı için tekrar tahsis ediniz Geri kalan dizi elemanlarını, orijinal

dizinin ilk yarısıyla eşlendiklerini göstermek için yazdırınız.

14.8 Komut satırı argümanları olarak dosya isimleri alan bir program yazınız. Programınız ilk

dosyanın içeriğini bir seferde okusun ve ikinci dosyaya tersten yazsın.

14.9 goto ifadesi kullanarak bir yuvalı döngü yapısını gerçekleştiren program yazınız.

Programınız aşağıdaki gibi bir kare çizsin.

\*\*\*\*\*

\* \*

\* \*

\* \*

\*\*\*\*\*

Programınız sadece aşağıdaki printf ifadelerini içersin:

printf(“\*”);

printf(“ “);

printf (“\n”);