

Sıfırdan başlayarak algoritma ve programlama öğrenme

Emre YAZICI

**Yapay Zeka Mühendisi
Programlama Uzmanı - MCP
2008**

Lütfen, kitabın herhangi bir sayfasını açıp okumaya başlamayınız. Bunun yerine, açıklamalar bölümünü okuyunuz.

Önsöz

Programlama artık dünyanın vazgeçilmezi haline geldi. Nereye gidersek gidelim, hangi işi yapacak olursak olalım bilgisayarsız ve bilgisayarları yöneten programlırsız bir sistemle karşılaşamıyoruz. Tabi, ülkemiz için bu durum söz konusu değildir. Ülkemiz daha tam olarak, bilişim sektörüne girememiştir. Ülkemiz, bilişim sektörünü, bilişim sektöründeki değişiklikleri, yaklaşık olarak 5-6 yıl geriden izlemektedir. Bilime önem veremediğimiz için ***bilimi kullanan değil bilimin kullandığı bir toplum*** haline geliyoruz ve bilime sonradan sahip olmak için çok büyük bedeller ödüyoruz. Sonuç olarak bilim, bize, olduğundan çok pahalıya satılıyor, üstelik bu ürünleri geliştirebilecek kapasiteye sahip olmamıza rağmen...

Yurtdışında geliştirilen sistemlerin (sadece) bazıları Türkler tarafından geliştiriliyor olsa da, hemen hemen hepsi Türkler tarafından geliştirilebilir. İşte beyin göçünü önleyip, teknolojileri Türkiye’de geliştirebilmek için, mümkün olabilecek en yüksek bilgi ve donanımlara sahip olmalıyız.

Türkiye’de hemen hemen hiçbir (yabancı) büyük şirketin, büyük bir Araştırma - Geliştirme ekibi yoktur. Yabancı büyük şirketlerin bazıları, Türkiye’yi sadece ürünleri satabilecekleri bir pazar olarak görmektedirler. Aynı Osmanlı İmparatorluğu’na uygulanan kapitilasyonlarda olduğu gibi...

Dünyada ki en zeki milletler arasında; Türkler, Hindular, Ruslar bulunur. Ancak malesef en çalışkanlar arasında çok gerideyiz. İşte burada, düşmanımızı, "**çalışmamazlık - tembellik**" olarak belirlemeli ve ona karşı savaşmalıyız...

Hedef Kitle

Bu kitabın hedef kitlesinde bir limitleme - kısıtlama bulunmamaktadır. Ancak, kitabı okuyacak kişilerin, bilgisayar kullanmayı (bilgisayarı açıp kapamayı, ofis programlarını kullanmayı, bazı ayarları ve dosya işlemlerini yapabilmeyi) bildiklerini varsayıyorum. Eğer, üniversitede bu bölümü (programlama - yazılım mühendisliği) okuyacaksanız, tavsiyem bu kitabı okuyarak, üniversiteye gitmenizdir. Eğer ikinci bir meslek sahibi olmak istiyorsanız veya daha sonra programlamayı okumayı planlıyorsanız, bu kitap sizin için en uygun olanıdır.

Ayrıca, bu kitabı okuyacakların, matematik konusunda bilgili olması – sorun yaşamıyor olması gerekmektedir. Aslında, matematikte bilgili olunması bu kitabın değil programlamanın şartıdır. Matematikte bilinmesi gereken konular; Sayılar, Kümeler, Dört İşlem, Üssü ifadeler, Mantık, Fonksiyonlar, Sayı tabanları, Toplam sembolü, Çarpım sembolü, Matris, Diziler - Seriler Eğer bu konularda çok ciddi probleminiz var ise, size bu konular üzerinde göz gezdirmenizi tavsiye ederim. Önemli olan o konu üzerinde soru çözmeniz değil, konuyu anlamış olmanızdır.

Yazılım Mühendisi: Normalde,

- bir projeyi tasarlayan,
- algoritmik (mantık – uğraş gerektiren) kodları yazan,
- veritabanını tasarlayan,
- sistemin akış şemasını hazırlayan,
- test işlemlerini gerçekleştiren
- benzeri planlama işlemlerini gerçekleştiren

kişiye verilen ünvanıdır.

Kodcu:

- Tasarlanmış olan programın kodunu yazan
- Hazırlanmış şemaya ve tasarım yapan
- Veri girişi – kontrolü yapan

kişiye verilen isimdir.

Programlamacı: Yazılım mühendisi ile kodcu arasında olan ve her ikisinde yapmış olduğu işlemleri gerçekleştiren kişiye verilen isimdir. Bu kitap, yukarıda belirtilen üç kategorideki kişi için tasarlanmıştır.

Yazar Hakkında

Uzun süredir programlama yapmaktayım. Benim, lisede başlayan programlama aşkım şimdiye kadar hiç sönmedi. Liseden sonra, İngiltere'de Yapay Zeka üzerine lisans öğrenimi aldım. Bu sırada beynin çalışma yapısı ile ilgili öğrendiklerimi geliştirdim. Daha sonra, bu kitabı bir okuyucunun nasıl daha kolay öğreneceğini hesaplayarak, planlayarak tasarladım.

Birçok farklı dilde, yüzlerce, proje geliştirdim. Hep farklı sistemler üzerinde çalıştığım için, çok iyi tecrübeye sahip oldum. Tecrübe, aynı konu, aynı program, aynı iş üstünde yıllarca çalışmak değildir. Bu yıllarca çalışma esnasında, aynı kategoride, farklı konuları işlemek, farklı programlar yazmak ve farklı işler üzerinde çalışmaktır.

Uzun süre, bazı programlama ile alakalı forumlarda moderatörlük, bazılarında yazarlık yaptım. Ancak bu süre zarfında gördüm ki, insanların birçoğu programlamayı bilmiyorlar. MSN'imi her gün biri ekliyor, soru soruyordu. Sorunun cevabını versem bile, "ben bunu yapamam, siz yapıp gönderebilir misiniz" diyenler oluyordu. Programlamayı mantıksal olarak değil yüzeysel olarak öğrendiklerinden dolayı, bir sistem kurabilmek onlar için zor oluyordu. Benim amacım, Çin atasözündeki gibi, balık tutmayı öğretmektir.

Türkiye'ye, hem bilgisayarın, hemde alanım olan yapay zekanın faydalarını, önemlerini anlatmak için çok uğraştım. Hâla da uğraşıyorum. Bunun için yaptıklarımın, yazdığım makalelerin haricinde sonunda bir kitap yazmaya karar verdim.

Ünlü yazar Balzac'ın dediği gibi, "bilginin efendisi olmak için, çalışmanın kölesi olmak gerekir". Ama ben sizi, köle olmaktan, bu kitabı yazarak kurtarıyorum. Sayfa sayısının az olması bu kitabın içinde çok fazla bilgi olmadığını düşündürür. Ancak kitapta, çok fazla örnek bulunmadığı için sayfa sayısı azdır. Birçok kitapta yüzlerce sayfa bulunur ama çoğu örnektir. Size bir konuyu tam anlatamadıkları için bol bol örnek sunarlar ki aradaki fark kapansın, ayrıca, kitap kalın olunca, okuyucular da, "bu iyi bir yazar" ifadesi oluşsun.

Emre YAZICI

Kitap Hakkında

Size garanti edebilirim ki, eğer bu işi yapabilecekseniz, en iyisi, bu kitabı okuyarak başlamaktır. Daha önceden öğrenenlerin de, öğrendiklerini, unutmasını ve baştan sağlam bir şekilde tekrar anlamalarını öneririm, tabi eğer yeterli olmadıklarını düşünüyor iseler.

Teknolojinin ayaklarımızın altında, bize hizmet için beklediği bir çağda, onu en iyi şekilde kullanabilmek, en büyük gereksinimlerden biridir. Unutmayalım ki, herşeyi - her bilgiyi bilemeyiz ama her gördüğümüzü (konumuzla alakalı) anlayabiliyorsak, o zaman her konuya adapte olabiliriz ve her işi yapabiliriz demektir.

Kitabın, teorik ağırlıklı olduğunu söyleyebilirim ve “hadi hemen kod yazmaya başlayalım” şeklinde düşüncelere sahip olan okurların olacağını tahmin edebiliyorum. Ancak, eğer pratik tabanlı öğrenmeye başlarsanız, sistem kuramazsınız. Bu nedenle, teori + pratik şeklinde yapılacak olan ilerleme ile birlikte, en iyi sonuca ulaşabilirsiniz.

Teşekkür

Bu kitabı yazarken, destek aldığım..... teşekkür ederim.

Telif hakkı

Bu kitabın, tüm telif hakları, xxxx numaralı koruma ile, Emre YAZICI'ya aittir.....

Web sitesi

Kitapın içinde geçen örnekler, dillerle ilgili güncel ve ayrıntılı bilgi, hazır algoritmalar v.b. dökümanlar, kitabın web sitesinde, www.zehirbeyin.com adresinde bulunmaktadır.

İçindekiler

Bölüm 1: Algoritma

Birinci bölüm, programlama ve birçok mühendislik sisteminin tabanı olan, algoritmanın ne olduğunu, yapısını, nasıl ve ne amaçla kullanıldığını, nasıl algoritma kurulabildiğini, programlamanın ne olduğunu nasıl yapılabileceğini anlatmaktadır.

Anahtar sözcükler: algoritma, programlama, program, programlayabilme, programlanabilme, programlama şeması, analiz, lowest limitation, operatör, analiz, planlama, WBS, AND, flow chart, pseudo kod

Bölüm 2: Programlama ilkeleri

İkinci bölüm, temel programlama ilkelerini, (öyle ki bu temel programlama ilkeleri; değişkenler, matematiksel ifadeler, koşul ifadeleri, parantezler, döngü, toplam sembolü, çarpım sembolü, fonksiyonlar, önermeleri içermektedir) anlatmaktadır.

Anahtar sözcükler: matematik, değişken, parantez, toplam sembolü, çarpım sembolü, döngü, mantık, önermeler, koşul ifadeleri, fonksiyonlar

Bölüm 3: Beyin ve programlama

Üçüncü bölüm, beynin çalışma yapısını, bu çalışma yapısına göre programlamanın nasıl mümkün olabileceğini, sınıf ve nesne kavramını, inheritance – sub – super class ifadelerini anlatmaktadır.

Anahtar sözcükler: beyin, class, object, sınıf, nesne, inheritance, super class, sub class

Bölüm 4: Bilgisayar sistemleri

Dördüncü bölüm, bilgisayar sistemlerini; bit, byte, 1-0, boolean mantığını, işlemcinin çalışma yapısını, programlama dillerinin kategorilerini – özelliklerini, programlama dili ağaçlarını, programlama dili yapısını anlatmaktadır.

Anahtar sözcükler: işlemci, byte, bit, boolean, dil, hafıza, ram, rom, hex

Bölüm 5: Programlama için gerekli diğer bilgiler

Beşinci bölüm, programlama için gerekli diğer bilgiler olan; veri türlerini, programsal blokları, syntaxı, operatörleri, prosedürleri, argümanları, fonksiyonları, işaretleri, değişken tanımlamayı anlatmaktadır.

Anahtar sözcükler: syntax, veri türleri, blok, operatör, prosedür, argüman, fonksiyon, işaret, değişken, enum, structure, interface

Bölüm 6: Programsal sınıflar

Altıncı bölüm, birçok programlama dilinde ortak kullanılan programsal sınıfları; ağaçlar, listeler, sepetler... anlatmaktadır.

Anahtar sözcükler: list, liste, ağaç, tree, queue, sıra, stack

Bölüm 7: Veritabanı

Yedinci bölüm, veritabanı yapısını, veritabanı yönetim sistemlerini, tablo – alan ilişkisini anlatmaktadır.

Anahtar sözcükler: sql, tablo, alan, kolon, e-r, entity, relation, dbms

Bölüm 8: Test ve Hata ayıklama

Sekizinci bölüm, programların test edilmesini, test yöntemlerini, test işleminde dikkat edilmesi gereken hususları, hata ayıklamayı anlatmaktadır.

Anahtar sözcükler: blackbox, whitebox, debug

Bölüm 9: Tasarım

Dokuzuncu bölüm, programların kullanıcı arabiriminin tasarlanmasını, kontrolleri formları anlatmaktadır.

Anahtar sözcükler: form, kontrol, event, olay

Bölüm 10: Mimari

Onuncu bölüm, birçok programlama dili tarafından ortak kullanılan mimarileri ve yapıları anlatmaktadır.

Anahtar sözcükler: threading, çoklu işlem, sockets

1 Algoritma

İçerik

- 1 Algoritma
 - 1.1 Programlam Öğretimi
 - 1.2 Algoritma
 - 1.2.1 Algoritmaya giriş
 - 1.2.2 Kimler algoritma kurabilir?
 - 1.3 Programlama
 - 1.3.1 Programlama - Program
 - 1.3.3 Programlar ne işe yarar?
 - 1.3.4 Programlanabilme & Programlayabilme
 - 1.3.5 Programlama için gerekenler
 - 1.3.6 Programlama nasıl mümkündür?
 - 1.4 Programlama şeması
 - 1.4.1 İşlem şeması
 - 1.4.2 Analiz
 - 1.4.3 Operatör
 - 1.4.4 Lowest Limitation
 - 1.4.5 Anlama
 - 1.4.6 Kod yazımı
 - 1.5 Planlama
 - 1.5.1 Geniş ve kapsamlı sistemler
 - 1.5.2 WBS
 - 1.5.3 AND
 - 1.5.4 Flow chart
 - 1.5.5 Pseudo Kod
 - 1.6 Programlama kısa tarihi ve yapısı
 - 1.7 Efektif Programlama
 - 1.8 Örnek Programlama

Amaçlar

- Birinci bölüm, programlama ve birçok mühendislik sisteminin tabanı olan, algoritmanın ne olduğunu, yapısını, nasıl ve ne amaçla kullanıldığını, nasıl algoritma kurulabilindiğini,
- programlamanın ne olduğunu, nasıl yapılabileceğini
- programlamada kullanılan tasarım elemanlarını anlatmaktadır.

Anahtar sözcükler

algoritma, programlama, program, programlayabilme, programlanabilme, programlama şeması, analiz, lowest

limitation, operatör, analiz, planlama, WBS, AND, flow chart,
pseudo kod

1.1 Programlama öğretimi

Ön bilgi

Lütfen, herhangi bir bölümü, sizi ilgilendirmediğini veya yeterince önemli olmadığını düşünerek, okumadan geçmeyiniz. Bütün bölümler eksiksiz ve sırayla okunulmalıdır. Bununla birlikte, bir konuyu tamamen anlasanızda, o bölümü yinede sonuna kadar okuyunuz ve belirtilen örneklerin hepsini aklınızda canlandırınız.

Kitabın amacı, daha sonra öğreneceğiniz bir programlama dilinde, istediğiniz her türlü işlemi gerçekleştirecek alt yapıyı öğretmektir.

Öğretmen

Bir konuyu biliyor olmak, bir konu üzerinde çalışıyor olmak veya bir konu üzerinde tecrübe sahibi olmak, o konunun iyi anlatılabileceği – öğretilebileceği anlamına gelmemektedir. Çünkü öğretebilmek, farklı bir yetenektir, farklı bir bilgidir. Uzun süreden beri yazılım – programlama üzerinde çalışmakta olan bir yazılım mühendisi olabilirsiniz, ancak bu, bildiklerinizi iyi bir şekilde öğretebileceğiniz anlamına gelmemektedir. Yani mühendis olmanız, öğretmen olduğunuz anlamına gelmez. Her mühendis öğretmendir diye, dersanelerin çoğunda, 2-3 senelik tecrübeye sahip yazılım uzmanları, öğretmen olarak çalışabilmektedir.

Halbuki, yazılım mühendisleri, bir konunun nasıl öğretileceğini, anlatım tekniklerini bilmek zorunda değildirler. Zaten, yazılım mühendisliği müfredatı içinde öğretim, öğrenci psikolojisi, anlatma teknikleri, beynin çalışma yapısı ve benzeri konuları anlatan dersler bulunmamaktadır.

Bir konuyu çok iyi anlatabilmek için, beynin çalışma yapısını bilmeniz gerekmektedir. İşte bunun için ya kendinizi bu konuda bilgilendirmiş olmalısınız ya da beynin çalışma yapısıyla ilgilenen bir bilim dalı olan yapay zeka uzmanı olmalısınız.

Programlama öğretimi – Dil ilişkisi

Bu kitabı diğerlerinden ayıran en önemli özellik; her bir programlama dili için bir veya iki kitap alarak o dilleri öğrenmektense, sadece bu kitabı okuyarak, birçok farklı dili çok kısa sürede öğrenebiliyor hâle geleceğinizdir. Piyasada, “C ile programlama”, “Delphi ile programlama”, “Java ile programlama”, “Basic ile programlama”.... gibi, birçok dilde, programlama yapmak üzere hazırlanmış kitaplar bulunur. Programlama dillerinin isimleri farklı olsada, hepsi aynı işlevi gerçekleştirir: **Programlama**. Eğer hepsi programlama kitabı ise, içeriklerinde ortak olan büyük bir bölümün olması gerekmiyor mu? Hatta bu ortak bölümün, kitapların en az %70’ini oluşturması gerekmiyor mu? Çünkü hepsi programlama kitabı ve kullandıkları platform aynı. Peki, varsayalım ki, on tane programlama kitabı aldık (Java ile programlama, C ile programlama). Hesabımıza göre bu kitapların %70’i aynı olacaktır. 70×10 kitap, $700 = 7$ kitap. Yani alacağınız on kitabın yedi tanesi aynıdır. On kitabın yedi tanesi programlamayı anlatmaktadır. Ancak hepsi programlamayı farklı yöntemlerle anlattıkları için, birinin %70’lik bölümünü okuduğunuzda, diğerlerinin %70’lik bölümünü okumuş olmazsınız.

Sonuç olarak, bu şekilde, para ve vakit kaybetmektense, bu kitabı okuyarak programlamayı öğrenip, daha sonra diğer birkaç dili, başka bir kitap alarak, o dili, programlamayı bilerek öğrenebilirsiniz.

Çok karşılaştığım bir soru olan: "Hangi dili seçerek programlamaya başlamalıyım?" sorusunun cevabına gelince: Bu sorunun herhangi bir cevabı yoktur. Birçok dersanede, size programlamayı öğretirken bir dili kullanarak öğretirler. Örneğin: C ile programlama, Java ile programlama.... Programlama ayrı bir ifadedir, ve programlama bir dili kullanarak mümkün olmaz! Size bu durumu anlattıktan sonra birde örnekle pekiştireyim. Bir sürücü kursuna yazıldığınızı varsayalım. Alacağınız dersler içinde, "Nissan ile araç kullanma", "Honda ile araç kullanma", "Toyota ile araç kullanma" ... olmayacaktır. Önemli olan otomobil kullanabilme yeteneğidir. Bu yeteneği öğrendikten sonra, her otomobili kullanabilirsiniz. Otomobiller arasında sadece birkaç fark bulunur (kullanım açısından: pedalların sertliği, kalkış hızı, boyutları, vites...).

Başka bir örnek olarak; (varsayalım ki) bir iş yerine, şoför olarak iş başvurusu yapıyorsunuz. İnsan kaynakları uzmanı, CV'nize bakıyor, sonra size şöyle bir soru soruyor: "Sizin ehliyetinizin sınıfı B, güzel.. Ancak bizim firmamızın araçları Honda markadır. Siz bu araçları kullanabilir misiniz?". Bu soruya hepimiz gülerек cevap verir. İşte aynı komik durum, programlamanın bir dilde ifade edilmesi içinde geçerlidir. Nasıl, Mercedes ile araç sürme şeklinde bir ifade olamayacağı gibi, bir dille (örneğin C, Delphi..) programlama ifadesi de olmayacaktır. Programlama tektir, programlamayı öğrendikten sonra hemen hemen her dili öğrenmeniz ve o dillerle programlama yapabilmeniz mümkün olacaktır.

Programlamayı öğrendikten sonra, sadece o dili konuşmayı öğrenmek gerekir ki, bu işlem çok kısa zaman alır. O yüzden, dile göre iş yapmayın, işinize göre dili seçin. **Alacağınız ayakkabıya, ayağınızı sığdırmaya çalışmazsınız, ayağınızın sığabileceği ayakkabı alırsınız.**

Programlamayı beyin, bir dil bilmeyi de el (veya diğer duyu organları – göz, kulak, ...) olarak düşünelim. El olmadan beyin yine düşünebilir, diğer işlemleri gerçekleştirebilir. Önemli olan beyindir. Ancak el, beyin olmadan hiçbir şey yapamayacaktır. Her ne kadar iki organda çok önemli gibi görünsede, beynin önemi ve görevi çok büyüktür.

Stephen Hawking, şu an bir tekerlekli sandalyede yaşayan ve dış dünya ile iletişimini sadece özel bir bilgisayar ile yapabilen ve çağımızın en büyük yaşayan fizikçilerinden biri olarak kabul edilir (bir hastalık nedeniyle).

University College ve Cambridge'deki Trinity College'da öğrenim gören Hawking, daha sonra Caius College'da araştırma görevlisi seçildi. 21 yaşında iken tedavisi olmayan bir hastalık olan amiotrofik lateral skleroz hastalığına yakalandı ve tekerlekli sandalyeye mahkum oldu. Hastalık Hawking'in konuşma yeteneğini de alıp götürdü. Ancak, dahi bilim adamı için özel olarak dizayn edilen bir bilgisayar, Hawking'in bir klavye aracılığıyla kurduğu cümleleri seslendirerek konuşmasına yardımcı oluyor. Hastalığın giderek ağırlaşan etkisine ve 60 yaşına basmasına rağmen Hawking halen çalışmalarına devam etmektedir. [e1]

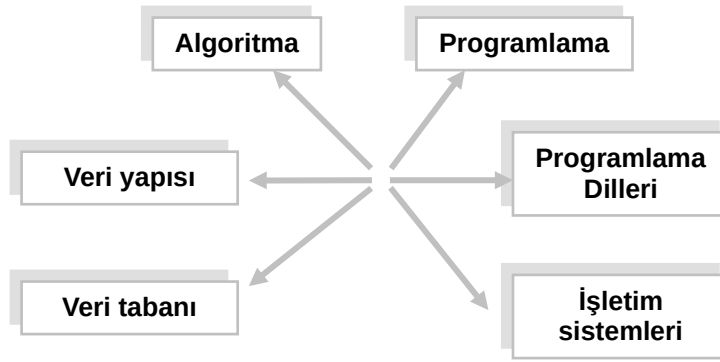
Hawking, ellerini kullanamasa da, beynini kullanarak, kuramlar

oluşturmaktadır.

Konu seçimi

Kitap içerisinde birçok farklı konu bulunmaktadır. Bu kitapla, programlamanın geçtiği ve kullanacak olduğunuz birçok konuya giriş yaptım. Ancak kitap içinde bahsi geçen, İşletim sistemleri, Veritabanı sistemleri, Programlama dilleri, Yazılım mühendisliği .. v.b. konular başlı başına onlarca kitap ile ifade edilmektedir. Kitapta, bu konuların temelini anlattım. Aksi takdirde bu konuların hepsi zaten onlarca kitap kadar yer tutardı.

Birçok algoritma veya programlamaya giriş kitabı, içerik bakımından bu kitaptan farklıdır. Yada bu kitap birçok algoritma ve programlamaya giriş kitabından içerik olarak farklıdır. Bu kitaplarda, temel alınan programlama ise, bu konu üzerine yoğunlaşmış ve ilerlenmiştir. Temel alınan konu algoritma ise, algoritma üzerine yoğunlaşmış ve bu konuda ilerlenmiştir. Bu tip kitaplarda, size algoritma veya programlama öğretilemez! Çünkü programlama; işletim sistemi ile de veritabanları ile de, programlama dilleri ile de, algoritma ile de ilgilidir. Dolayısıyla, kitabı bir yönde ilerlettikleri için diğer konulardan bilgisiz kalacağınız için, konuları düzgün bir şekilde öğrenemezsiniz.

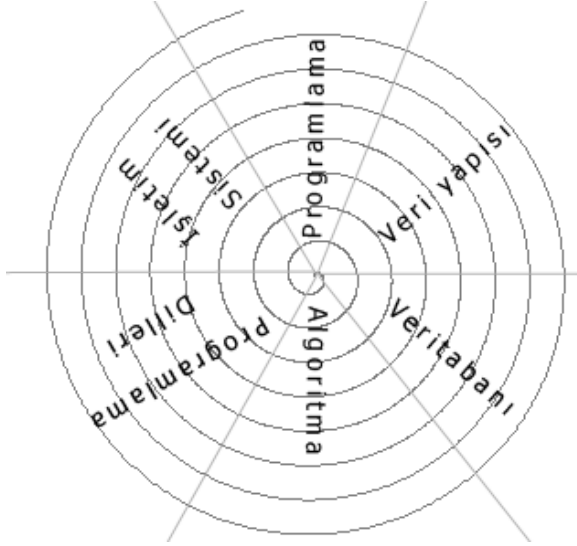


Figür 1.1 A : Genelde kitapların öğretim şekli

Yukardaki diagram, piyasadaki birçok kitabın öğretim şeklini anlatmaktadır. Her kitap kendine, bir yön seçer ve o yönde ilerler. Bir yönde ilerlerken, diğer konulardan neredeyse hiç bahsetmez ve sizin bu konudaki bilginiz az veya hiç olmadığından anlatılan kavramları anlamakta zorluk çekersiniz veya yüzeysel öğrenirsiniz. Bir yönde çok bilgi diğer yönde az veya hiç bilgi vermedikleri için tek bir yönde ilerlersiniz ve öğrendiğiniz konunun diğer konularla arasındaki bağlantısını anlayamazsınız.

Yukarıda bahsi geçen 6 konuyu giriş – orta ve ileri seviye olarak üçe bölelim. Algoritma kitabı alırsanız, bu kitapta, algoritmanın, giriş – orta – ileri seviyesi anlatılır ve programlamanın giriş seviyesi anlatılabilir.

Bu kitapta ise, birbiriyle ilgili olan bütün bu konuların öncelikle “giriş” seviyesi, daha sonra, algoritmanın orta seviyesi, programlamanın orta seviyesi anlatılmıştır.



Figür 1.1 B : Kitabın öğretim şekli

Yukarıdaki şekilde görüldüğü gibi, sırayla, her bölümden belirli miktarda bilgiler öğrenip, sonra, yine ilk bölümden daha ayrıntılı bilgiler öğrenecek ve daha sonra diğer bölümlerden daha ayrıntılı bilgiler..... öğreneceksiniz.

Kitap içinde bazı kısımlardan sonra **[Ara verme bölümü]** ifadesi kullanılmıştır. Kitabı okurken, sadece bu belirtilen kısımlarda ara vermenizi (günlük ara verme veya bir gün içinde ara verme olarak) tavsiye ederim. Çünkü, bir konuyu ortadan bölerseniz, birbiriyle bağlantılı konuların bütünlüğünü bozduğunuzdan bazı konuları anlamakta güçlük çekebilirsiniz. Bunun haricinde, bir günde, 1'den fazla bölüm okumamanızı tavsiye ederim.

Dersaneler - Süre

Programlamayı öğrenmek için, bir dersaneye gidip, (orta veya ileri düzey) 4-5 bin ytl den başlayan fiyatlarla kayıt olup, dersaneyi bitirdiğinizde; bu kitabı okuduğunuzdaki kadar bilgi öğrenmeden mezun olursunuz. Amacım dersaneleri kötülemek değil. Ancak, birçok dersane, size, normalde 100-200 saat sürecektir, 500 saatte verir, 1-1,5 yıl boyunca kursa gitmek zorunda kalırsınız. Hem paranız boşa harcanmış olur, ama en önemlisi, zamanınızın boşa harcanmasıdır. Çünkü zamanın parasal bir karşılığı yoktur. Eğer var diyorsanız ben size o kadar ödeyeyim, bana zamanınızı verin...

Anlatmaya çalıştığım, dersanelerin birçoğunun "*öğrenci bankası*" haline geldiğidir. Normalde, bir öğretmeni uzmana, bir dersin ne kadar sürede öğrencilere öğretilbileceğini sorulur. Uzman hesaplar, ve örneğin cevap olarak "200 saati" verir. Ancak finansal işlerle ilgilenen kişi, bu süreyi yetersiz bulur. Çok iyi kâr edilebilmesi için sürenin 500 saat olması gerektiğini söyler ve bu şekilde karar alınır. Peki uzman (öğretici) nasıl olacakta, 200 saatlik konuyu 500 saate yayacak?

Eğer normal bir şekilde anlatılsa, 200 saat sonunda öğrenciler, dersaneyi (konuyu öğrendikleri için) bırakıp gidebilir. O halde, öğrencilere, kodu kullanmayı, programlamayı, algoritmayı, mantığı değil de, sadece örnekleri öğretmekten başka çözüm kalmayacaktır. Yani, konuyu öğretip öğrencilerin, ileride kendi kendilerine daha iyi öğrenebilmelerini, karşılaştıkları sorunları çözebilmelerini ve sistemlere adapte olmalarını değilde, en sık karşılaşılabileceği programlarla alakalı örnekler

göstermeyi amaçlarlar. Öğrenciler, öğrenemediği için, örneklerle bağımlı kalacaklardır. Böylece, uzman, bol bol örnek gösterecektir ve öğrenciler programlama mantığını kavrayamadığı için örneklerle mahkum kalacaklardır.

Tabi birde, ucuz olsun, daha çok öğrenci gelsin diye, programlamaya girişi sadece 10 saatte öğreten dersanelerde vardır. Bu dersaneler sadece zaman kaybetmenize neden olur. Bu anlatılan malesef bir hikaye değildir. Üstüne üstlük, size herhangi bir bilimsel metotla değil, kendilerinin de öğrendiği gibi anlatırlar.

Öğretim Tekniği

Bu konu üzerine yazılmış birçok kitap, benim tabirimle, “sample based” yani “örnek tabanlı” kitaptır. Örnek tabanlı kitaplarda, size bir terimin, tanımını yaptıktan sonra hemen bir örnekle anlatmaya çalışırlar. Ancak bu yöntem öğrenmenizi zorlaştırır. Doğru olan ise, önce tanımlı önemsemeyip, gerçek hayattan benzetme yaparak, o terimin ne anlama geldiğini bilimsel ifadelerle anlattıktan sonra pekiştirmek amaçlılığıyla örnek kullanmaktır. Yani örnek amaç değil araç olmalıdır. Konuyu düzgün anlatamadıkları için, bol bol örnek verip aradaki mesafeyi kapatmaya çalışırlar. Şimdi bunu size daha iyi açıklama için “***bir örnek verelim***”.

2, 4, ? . Bu dizide, soru işareti yerine hangi sayı gelmelidir şeklinde bir soru sorulsaydı, bu soruyu çözmek mümkün olamazdı. Çünkü, 2 ile 4 arasındaki olası ilişkiler

- 4, 2’den 2 fazladır
- 4, 2’nin 2 katıdır
- 4, 2’nin karesidir.
- 4, 2’nin, 5 katından 6 eksiktir
-

şeklinde sıralanabilir. Bu gibi bir durumda, size daha çok örnek verilmesi gerekmektedir.

1, 2, 4, 8....

Yukardaki seriden anlaşacağı gibi, ikinci sayı, ilk sayının 2 katı, sonraki sayılar bir önceki sayının iki katı şeklinde devam etmektedirler. İşte karmaşık olan bir terimi, size bol bol örnek göstererek bu şekilde öğretmeye kalkarlar. Halbuki, terim - örnek göstermektense;

“Elimizde bir dizi sayı bulunmaktadır. Bu sayılar, bir kurala göre dizilmiştir. İlk sayı 1’dir. Sonraki sayılar, bir önceki sayının, iki katıdır”

ifadesi kullanılsa, ve akabinde, örnek gösterilip zaten belirtilen kuraldan anlaşılmış olan örnek pekiştirilse, durum daha kolay kavranır.

İlkokulda, ödevimizi yapmadığımızda, öğretmenimiz konuyu öğrenmemiz için (ayrıca ceza olarak), ödev metnini 5 sayfa - 10 sayfa yazmamızı söyler. Böyle bir durumda yani bir metni 10 sayfa yazdığımızda işin mantığını kavramadığımız (veya çalışmadığımız için) beynimizi değil beyinciğimizi kullanarak bir tür refleks haline getirmeye çalışırız. Yani işin formülünü – anlamını öğrenmemiş oluruz. Sadece sonradan bu kadar çok tekrar sonucunda, işin mantığını kavrayabiliriz (her zaman

değil!).

Meslektaşlarımızı (aslında tamda meslektaş sayılmayız) eleştirmek istemem ancak, bazı, "programlama nedir, nasıl yapılır", "algoritma", "programlamaya giriş" v.b. konulu makalelerde - kitaplarda, derslerde (üniversite - lise - dersane), daha programlamayı öğrenmeden bir örnek gösterilir. Yani siz daha programlama nedir bilmezken, C ile programlama gösterilir veya "Merhaba Dünya (merhaba dünya mesajı gösteren bir program)" şeklinde bir giriş yapılır. Peki, yeni doğan bir bebeğin, doğum öncesinde hiç bir gelişim evresi yok mudur? Yada yeni dünyaya gelen bir bebek nasıl olurda, "Merhaba Dünya" diyebilir. Bence bu öğretim şekli yanlıştır, bu yüzden, önce algoritmayı anlatmak adına, bu kitabı yazdım ve sizlere sunuyorum.

```
#include<stdio.h>
int main( void )
{
    printf("Merhaba Dünya");
}
```

Figür 1.1 C : Merhaba dünya örneği

Kitap içinde, bazı yerlerde, gerekli bilgilerin hafızanızda daha uzun süre kalabilmesi için, olağan dışı örnekler kullandım. Beynimiz içinde, yüz binlerce mantıklı bilgi bulunmaktadır. İşte bu yüzden, eğer kurulan örnekler mantıklı olursa hatırlamanız daha uzun, mantıksız olursa, hatırlamanız daha kısa süre alacaktır. Böylece hatırlanması gereken olayları, ilginç ve mantıksız oldukları için daha kolay hatırlayabileceksiniz.

Annem, 1977-81 arasında liseyi okurken arkadaşlarıyla kimsayal formülleri hafızalarında tutmak için bir kısaltma bulmuşlar (bu bilgiyi başka birilerinden öğrenmişler). Kimyada sülfürik asit olarak nitelendirilen ve formülü H_2SO_4 olan bu maddenin formülünü hafızalarında tutmak için, "Hasan 2, Salak Osman 4" gibi bir ifade kullanmışlar. Aradan 26 yıl geçmiş olmasına rağmen, nerede konusu geçerse geçsin, annem bu bilgiyi halen hatırlayabilmektedir. Üstelik, bu bilgiyi 26 yıl içinde hiç kullanmamasına rağmen.

İşte amacım da, hem kavramanızı kolaylaştıracak hem de hafızanızda uzun süre kalacak örnekler ve açıklamalar kullanarak kitabı ve programlamayı sizin için kolaylaştırmaktır.

Kitabın Farkı

Bu kitabın farkı: benim, bu yanlışlıkların farkında olabilmem, ve size, X saatte öğretilmesi gereken konuyu X saatte öğretmem ve amacımın ticaret değil öğretim olmasıdır. Yani, bu kitabı aldıktan sonra bana veya yazdığım kitaplara bağımlı kalmayacaksınız (bazı dersanelerin yaptığı gibi), aksine, artık kendi kendinize her türlü programı tasarlayabilecek kapasiteye (biraz da deneyimle) geleceksiniz. Günlük olayları bile programsal olarak ifade etmeye başlayacaksınız. Bazen kendinizi programlamaya öyke kaptıracaksınız ki, çok geveze bir arkadaşınızı susturmak için bir program bile yazmayı düşüneceksiniz.

Bu kitabın okunması için en önemli gereksinim, programlamayı bilmemektir. Çünkü, öğrenmeyi ve inanmayı en zorlaştıran etken bilgidir. Programlama öğrenmiş kişilerde

okuyamaz deęiller, fakat, řimdiye kadar öğrendiklerini unutmak řartıyla...

Birçoęunuz “kod yazmaya bařlayalım artık” düşüncesinde olabilir. Ancak, kod ile öğretilenler sonucunda sadece o kodları kullanmayı öğrenirsiniz ve kendiniz bir “řeyler” keřfedemezsiniz.

Eęer amacımız programlama öğrenmekse, doęal olan, programlamayı, programlama dilleriyle yapmamız gerektięi ve programlama dili öğrenmemiz gerektięidir. Dolayısıyla, aslında bu kitap bir DİL kitabıdır. Yani bu kitabı, bir yabancı dil (ingilizce, ispanyolca...) kitabı okuyormuř gibi okumalıyız.

Dil kitaplarının dayandıęı temel,

“1. dilde ifade edilen X terimi, 2. dilde Y řeklinde ifade edilir”

“1. dilde ifade edilen Z terimi, 2. dilde T řeklinde ifade edilir”

řeklindedir.

Bu kitap benzeri işlemleri anlatacaktır ancak, bu ifadelerden ibaret deęildir. Yani aslında bu kitapta bir çeřit gremer öğreneceksiniz. Piyasada, yabancı dil öğreten binlerce kitap bulunur. Ancak hiçbir kitap, herhangi bir yabancı dili nasıl öğreneceęinizi anlatmaz!!! Sadece o dili öğretir. Ben size, herhangi bir dili öğrenebileceęiniz bir kitap veriyorum.

İKİ AMACIMIZ BULUNUYOR:

İngilizce öğretilirken, nasıl ařaęıdaki gibi herhangi bir metni, ingilizcedeki haline

“Orada birřey var”

“There is something over there”

çeviriyorsak, bir programıda işte bu řekilde metinleri çevirerek, yapacaęız! İşte bunun için önce (1. amacımız) hangi metinleri nasıl yazacaęımızı öğrenmeliyiz sonrasında ise bu (2. amacımız) metinleri nasıl bilgisayar dillerine çevireceęimizi öğrenmeliyiz.

Kitap içinde, “ÇEVİRME” ifadesi geçen kısımlara dikkat ediniz. Bu ifadelerin geçtięi bölümlerde, bir metnin nasıl programlama diline çevrileceęi anlatılmıştır.

1.2 Algoritma

1.2.1 Algoritmaya giriş

Bilgisayar Bilgisayarlar; onlara anlatılabilen (yada ifade edilebilen) belirli emirleri; işleyen, belirli verileri depolayabilen, gerçekleştirdiği bu işlemleri insanlara göre daha hızlı ve hatasız tamamlayabilen (*bilgisayarlar hata yapmamaktadır – sadece onları programlayan programcılar hata yaparsa, bilgisayarlarda bu hatalı emirleri gerçekleştirmiş olduğu için hata yaparlar*) elektronik makinelerdir.

Bilgisayarları, kontrol etmek için bilgisayar programlarını kullanırız ve programlarla bilgisayarları yönetiriz. Yani programlar, bilgisayarların **direksiyonudur** veya **dizginidir** diyebiliriz.

Konuya başlamadan önce, inanmanız gereken, bu işin aslında düşündüğünüzden daha kolay olduğudur. İşin zor kısmı, sizi, programlama ve algoritmanın kolay olduğuna inandırmaktır, programlama öğretmek değil!!!

Programlama öğrenmek için okuduğunuz bir kitabın algoritma konusu ile başlaması olağandır. Çünkü, her tür planlama ve programlamanın temeli, algoritma kurmakla mümkün olacaktır. Bu kitabı okuyan herkes algoritma kurabilecek yeteneğe sahip olduğundan dolayı, okuyucular için; “programlama yapabileceği garantisi” verilebilir (tabi bu hemen her programı yazabileceğiniz anlamına gelmemektedir). Çünkü, “**okuma-yazma**” bile algoritma gerektirmektedir, ve siz okurlar, **okuma** bildiğiniz için algoritma kurabilir, dolayısıyla programlama yapabilirsiniz.

Algoritmaya giriş yapmadan önce, bir soruya cevap vermeye çalışalım: “**Neden düşünüyoruz?**”. Şuan, “Neden düşünüyoruz” sorusunun cevabını bulmak için **düşünüyorsunuz, çünkü bu sorunun cevabını bilmiyorsunuz**. O halde, düşünmenin nedenlerinden birinin, bir **sorunun** cevabı bulmak olduğunu söyleyebiliriz. Çünkü eğer bu sorunun cevabını biliyor olsaydınız, düşünmenize gerek kalmazdı ve direkt olarak cevaplardınız.

Algoritma **Algoritma**, "algorithm is a procedure or formula for solving problems [e2]" yani; "problemleri çözmek için tasarlanmış bir prosedür veya formül" anlamında gelmektedir.

Daha açıkcası, **bir problemin çözümü için izlenecek yol** (yöntem - metod) **yani** (bu yöntem içerisinde bulunan) **bir dizi işleme verilen addır**. Bir dizi işlem: tek bir işlem veya birden fazla işlem olabilir. Eğer yapılacak bir dizi işlem yok ise, ortada problem olan bir durum söz konusu değildir.

Algoritma Kısa Tarihi

Algoritmanın kurucusu dokuzuncu yüzyıl başlarında yaşayan **Müslüman-Türk** matematik alimi Ebu Abdullah Muhammed bin Musa el-Harezmi'dir. Matematikçiler için temel olan Kitab-ül Muhtasar fi Hesab-il Cebri ve'l-Mukabele adlı eseri meşhurdur. Kitabın aslı, Oxford'daki Bodliana kütüphanesindedir. Matematikteki şöhreti on altıncı yüzyılda Avrupa'yı etkisi altına almıştır. Harezmi'nin ismi Avrupa'da türlü şekillerle söylenmiştir.

Latince’de “Alkhorismi” şeklinde söylenerek bulunduğu metod Algoritma (algorizme) olarak literatüre geçmiştir. [e3]

Bir problemi çözmek için, düşünme eylemi gerçekleştirdiğimiz her evrede, algoritma kurarız veya daha önceden (bizim veya başkası tarafından) kurulmuş olan algoritmaları kullanırız. Bazı durumlarda da, geçerli olan durumlar için algoritma kuramayabiliriz. Gelişen dünya – bilişim sayesinde artık her tür problem çözülmeye, çözüm yolları da herkes tarafından öğrenilmeye (internet veya kitaplar tarafından) başlandı. Artık bu noktadan sonra, önemli olan, algoritma kurabilme değil, kuracağınız algoritmaların; **ne kadar hızlı, ne kadar kapsamlı, ne kadar ucuz ve ne kadar yüksek performanslı** olduğudur.

Algoritma kurmanın nedeni, **düşünmektir** ve düşünmenin nedeni ise, **problem çözmektir (dört paragraf öncesinde, düşünmenin nedenlerinden birinin, bilinmeyen bir sorunun cevabını bulmak olduğunu söylemiştim, bir sorunun cevabının bilinmemesi de bir problemdir).**

Problem Problem; istenilmeyen bir durumdur[**durum: Bir zaman kesiti içinde bir şeyi belirleyen şartların hepsi, vaziyet, hâl, keyfiyet, mevki, pozisyon**] (bir kullanıcı, kişinin veya topluluğun istemediği bir durum – değiştirilmesi gerekli olan bir durum) ve algoritma kurularak, bir dizi işlem oluşturulur. Bu işlemler, o anki durumu, problem durumundan çıkarır ve istenilen hâle getirir.

İlk okul çağlarından hatırladığımız “matematik problemleri” de aynı esasla çözülür. Problemlerde, bilinmeyen bir durum söz konusudur. Bir işçi problemi düşünelim... Evimizi boyatacağız ve boyacı ustasının günlüğü 40 YTL (veya artık 2009’da isek, 40 Lira) olsun ve boyacı bir günde sadece bir oda boyayabilsin.. Bizde buradan oda sayısı ile günlük masraf olan 40 lirayı çarpıp boyacıya vereceğimiz ücreti hesaplarız...

Yapay zeka (insan gibi düşünebilen – çözüm üretebilen – sistemler geliştiren bir bilim dalı) sistemlerinde de; insan gibi düşünen sistemler geliştirmede ilk temel alınan yöntemlerden biri, başlangıç durumundan, istenilen duruma ulaşan bir formül bulmaktır. Algoritma kurma evresi aşağıdaki diagramdaki gibi gerçekleşmektedir.



Figür 1.2.1 A : Problem ve Çözüm durumu

Şimdiye kadar ki bölüm tamamen teorikti, bu öğrendiklerimizi pratiğe dökelim.

Bu anlatılanların aklınızda daha uzun süre kalmasını ve anlaşılır olmasını sağlamak amacıyla, geçmişe çocukluğunuza dönelim. Hepimiz, Hansel ve Gretel hikayesini hatırlarız. Bu hikaye içinde geçen 1-2 cümleyi yazayım. (*Hansel ile Gretel iki kardeştir ve üvey anneleri onların kaybolmasını ister.*)

“.... Hansel zeki bir çocukmuş. Sabah ormana doğru yürürlerken, akşam yemeğinde cebine sakladığı kuru ekmeğin kırıntılarını (yere iz bırakıp kaybolmamak ve daha sonra bu izi takip ederek evin yolunu bulmak için için) yere saçıp arkasında bir iz bırakmış.....”

Çok basit bir örnek olarak; Hansel, evin yolunu kaybetmemek için bir algoritma yani çözüm bularak, ekmek kırıntılarını yere işaret bırakmak için kullanmıştır.

Algoritma üstünde neden bu kadar yoğun olarak durulduğunu ileri ki bölümlerde daha iyi kavrayacaksınız. Birçoğunuzun, **“bu konuları zaten herkes biliyor”** dediğini tahmin ediyorum. Bazılarınızda, **“bu konular aslında zor olduğu için mi üzerinde bu kadar duruluyor?”** şeklinde düşünüyor olabilir. Bu düşüncenin aksine algoritma konusu inanılmaz derecede kolaydır. Ancak, algoritmanın öneminin sürekli olarak aklımızda olmasını sağlamak gerekmektedir. İşte bu yüzden bir kaç örnek daha vererek, algoritmanın hayatımızın bir parçası olduğunu göstermek daha doğru olacaktır.

Öğrenmenizi daha kolaylaştırmak için, ilginç bir olayı aklınızda tutmanızı öneririm. Örneğin, yanınızda, bir uzaylı(geri kalmış bir gezegenden gelen – ilk kez dünyamıza gelen) veya medeniyetten uzak (eğitilmemiş) kalmış 12-15 yaşlarında bir çocuk olduğunu, ben kitapta algoritma konusunu anlatırken, sizde sanki onun yanındaymışsınız ve öğrettiklerimi tasdik edermişcesine ona destek ve yardımcı olduğunuzu düşünün. Siz algoritma kurmasını biliyorsunuz ve bu konu ile ilgili anlatacaklarımı da duyduğunuzda çok kolay bir şekilde anlayabilirsiniz. Yani ben sadece bir kitap yazdım, siz oradan (müfredat gibi) bakıp, aynı bir öğretmen gibi birisine anlatıyormuşsunuz gibi düşünün. Hatta içinizden “bunları da gördük biz bu hayatta... kolay terimler... bunları çok rahat yapabilirim...” ifadelerini geçirebilirsiniz. Bu durum sizi hem rahatlatacak hemde konuları daha kolay şekilde HATIRLAMANIZI sağlayacaktır. Hatırlamanızı diyorum çünkü, siz zaten algoritmanın ne olduğunu biliyorsunuz, tek bilmediğiniz, **“algoritmayı bildiğiniz ve algoritma kurmanın ne kadar kolay olduğunu bildiğiniz”** dir.

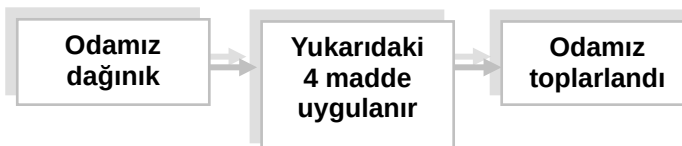
İnsan beyni, yukarıdaki figürde gösterildiği gibi; bir başlangıç durumundan, sonuç durumu elde etmek için işlemler bulmak üzere çalışmaktadır. Tabi ki, algoritma kurma öncesinde, problemin oluşması, ve kurulan algoritmanın sorunu çözmesinden sonra, sorunun çözülmesi durumları da eklediğimizde daha ayrıntılı bir figür elde ederiz.



Figür 1.2.1 B : Bir problemin gerçekleşmesi ve çözümü

Örneğin; (varsayalım ki) çalışma odamız dağınık ve odamızın dağınık olması bizim için bir problem, çünkü biz istenilen durum olarak odamızın düzenli olmasını istiyoruz. İşte, odamızın dağınık durumundan → düzenli durumuna geçebilmesi için bir algoritma kuruyoruz;

- Eşyalarımızı yerine yerleştiriyoruz
- Gereksiz kağıtları çöpe atıyoruz
- Açık kitapları kapatıp rafa kaldırıyoruz
- Odamızı temizliyoruz ve süpürüyoruz



Figür 1.2.1 C : Örnekteki algoritma işlemleri, ilk ve son durum

Bu adımlar sonucunda (öyle ki bu adımlar, yukarda bahsettiğimiz bir dizi işlemidir) odamız düzenli hale geliyor. Sonuçta diyebilirizki, algoritma istenilmeyen bir durumdan istenilen bir duruma **mantıklı** geçiş için bir yöntemdir.

Birçoğunuzun aklında, “**böyle bir durum için neden algoritma kurduk?**” sorusu oluşmuştur. Belki hergün, belki her hafta yapıyor olduğumuz ve **BASİT** olan bir problem için neden algoritma kuralım? Düşüncemiz kesinlikle **doğrudur**. Çünkü, algoritma;

- **ya daha önceden kurulmamış durumlar için**
- **ya daha önceden kurulmuş ancak unutulmuş durumlar için**
- **ya daha önceden kurulmuş ancak başarıya ulaşamamış durumlar için**

kurulur. Hepimiz, daha önce ya odamızı topladığımız yada toplayanları gördüğümüz, odamızı toplamamız gerektiğinde neler yapacağımızı düşündüğümüz için, böyle basit durumda algoritma kurmaya gerek görmeyiz. Tabi ki haklıyız da. Burada belirtilmek istenen, algoritma kurmanın günlük hayatta önemsenmeyecek bir ayrıntı olduğu ve daha önce onlarca hatta yüzlerce algoritma kurduğumuz veya öğrendiğimizdir.

Şimdi de, daha önce algoritmasını kurmadığımızı düşündüğüm bir konu hakkında, algoritma kurma örneği vereyim: Elimizde, 3 ve 5 litrelik birer su bidonu olduğunu ve bir çeşme başında olduğumuzu düşünelim. Varsayalım ki, görevimiz, 5 litrelik su bidonu içine, 2 litre su doldurmak olsun. (elimizde bir tartı veya benzeri bir araç bulunmamaktadır)

Şuan bu problemi çözmek için gerekli yolun ne olduğunu bilmiyoruz. Ancak, bu soruyu hepimiz rahatlıkla çözebiliriz. Bidonlara ve suya uygulayabileceğimiz işlemlerin listesini çıkaralım:

- bir bidona su ekleyebiliriz (çeşmeden)
- bir bidondaki suyu dökebiliriz
- bir bidondan diğer bidona su dökebiliriz

TRT’de yayınlanan “Bir kelime, bir işlem” yarışmasını hatırlayalım. Bu yarışma programında “işlem” bölümünde, amaç; verilen bir dizi rasgele sayıyı kullanarak, bir dizi işlem sonrasında, istenilen (hedef) sayıya ulaşmaktır. Yarışmacılar, belirli bir algoritma kurarak, hedef sayıya ulaşmaya çalışırlar.

Bu sorunu çözmek için algoritma kurarken;

- 5 litrelik bidonda, 2 litre su olabilmesi için, öncelikle 5 litrelik bidonu doldururuz
- Sonra, 5 litrelik bidondaki suyu yavaş yavaş, 3 litrelik bidona dökeriz. 3 litrelik bidon dolduğu an ($5-3=2$), 5 litrelik bidonda, 2 litre su kalmış olacaktır.



Figür 1.2.1 D : Örnekteki algoritma işlemleri, ilk ve son durum

Fonksiyonlar konusunda ayrıntılı bilgiler ileriki bölümlerde verilecektir. Ancak, biz matematiksel fonksiyonları hatırlayalım. Bir kaset çaları yani teybi düşünelim. Teybin görevi, içine konulan kasedin içindeki şarkıları çalmaktır. Dikkat edersek teybin görevi hiç değişmemektedir. İçine hangi kaseti koyarsak o kasedi çalabilmektedir. Her bir şarkıyı çalan ayrı ayrı araçlar keşfetmektense, aynı işlevi gerçekleştirdiği için, bir teyp alırız ve bu teybe yerleştireceğimiz kaset ile istediğimiz şarkıyı çalarız. Fonksiyonları bir teybe benzetebiliriz. İçine hangi değeri (kasedi) yerleştirirsek o değeri işlem yapar ve ona göre çıktı (şarkıyı) verir.



Figür 1.2.1 E : Fonksiyon örneği

$$y = f (x)$$

Aslında günlük hayatta, bir şeyleri girdi olarak alan ve sonra bir tür çıktı veren herşeyi bir fonksiyon gibi düşünebilirsiniz. Bir devlet dairesine gittiniz, bir resmi evrak (girdi) verdiğiniz, bu evrağa numara atıldı, imza atıldı, mühür basıldı ve size geri (çıkıtı) verildi. İşte orada, bu işlemleri gerçekleştiren memur, bilgisayarda fonksiyon olarak adlandırılabilir.

Burada, kaseti x (girdi), teybi, fonksiyon (f) ve çalan müziği y (çıkıtı) olarak ifade edebiliriz. Unutmayalım ki, yerleştirilen kasete göre bir müzik çalınacağı için, giren x değerine göre bir y değeri çıkacaktır, ve fonksiyon görevini gören teyp hep aynı işlemi (kaset çalma) gerçekleştirecektir. Aynı şekilde ; $y=f^{-1}(x)$ şeklinde bir ifadede de, (üssü -1 ifadesi olduğu için) ters işlem yapılacaktır. Yani bu sefer teyp dışarıdaki bir ses kaynağından sesi alıp kasete kayıt edecektir.

$$y = f (x)$$

şeklinde bir fonksiyon ve eşitlik düşünelim. Bu eşitlikte, X istenilmeyen (problem durumu) durumunu, y ise, sonuç (istenilen) durumunu simgelesin. F olarak ifade edilen terim ise, matematikten hatırladığımız bir fonksiyonu simgelesin. F fonksiyonu, girdi olan X (yani istenilmeyen durum) durumuna belirli işlemler uygular ve sonuç olarak, bize, Y (yani istenilen durum) durumunu çıkarır

Son durum = algoritma işlemleri (ilk durum)



Figür 1.2.1 F : Fonksiyonların, algoritmik gösterimi

Başka bir örnek olarak; (varsayalım ki) otomobilimizin lastiği patladı, ve bu durum (lastiğin patlak olması) bizim istemediğimiz bir durum – yani bir problemdir. Çözüm durumu olarak; lastiğin patlak olmamasını veya, otomobildeki patlak lastiği başka bir yedek lastikle değiştirilmiş olmasını verebiliriz.

Lastik sağlam = işlemler (Lastik patlak)

İşlemler adlı fonksiyon içerisindeki yapılacak olan işlemleri, hepimiz tahmin ederiz. Fakat, bu işlemi bir robota öğrettiğimizi düşünelim. Bilim kurgu filmlerinde, robotların olduğu durumlarda, bazı sahneleri, robotun göz kesitinden görürüz. Bu sahnelerde, robota, “düşmanı öldür”, “aracı sür” gibi emirlerin verildiğini görmüşüzdür. Varsayalım ki, yıl 2020, otomobilimizi sürüyoruz, yanımızda yardımcı robotumuz bulunuyor. Lastik patlıyor ve bu durumda, robotun gözü önüne yine emirler geliyor. Robotun bu emirleri anlayarak, işlemi gerçekleştirdiğini düşünelim.

Durum: Sahibin kullandığı aracın lastiği patladı

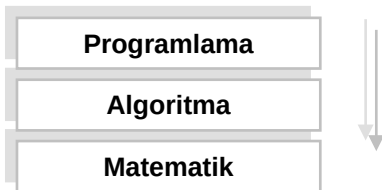
- Kriko kullanarak otomobili kaldır
- Patlak lastiği çıkar
- Yedek lastiği tak
- Krikoyu gevşeterek otomobili yere indir

Gerçek hayatta her gün, onlarca, yüzlerce algoritma kuruyor veya kurulmuş olan algoritmaları görüyor – kullanıyor olabiliriz. [Bu olayı, yani robotun yukarıdaki işlemleri, gözü önünde görmesini, gözlerinizi kapatarak zihninizde canlandırınız]

Algoritma, matematiksel olarak ifade edilebilir. Aslında dış dünyadaki olayların, eylemlerin bile matematiksel tanımı bulunmaktadır. Programlamanın kökü algoritmadır. Algoritmanın kökünde, matematik olduğunu bir önceki cümlede ifade etmiştik. Buradan, programlamanın kökünün, **matematik** olduğunu söyleyebiliriz.

[Ara verme bölümü]

Aşağıdaki diagrama bakarak, programlamanın **kökünün**, algoritma olduğunu, algoritmanın **kökünde** matematik olduğunu söyleyebiliriz. Bu nedenle, matematik konusunda iyi bilgiye ve anlama kabiliyetine sahip olmanız programlama yapmayı kolaylaştıracaktır.



Figür 1.2.1 G : Matematik, algoritmanın, algoritma, programlamanın tabanıdır

Birkaç paragraf önce, neden **mantıklı** (...*algoritma istenilmeyen bir durumdan istenilen bir duruma mantıklı geçiş için bir yoldur...*) kelimesini kalın ifade ettiğimize gelince; bir algoritmanın ürettiği çözümün mantıklı olması gerekmektedir. Mantık, aslında bilinenin aksine, akılla alâkalı değildir. Bugüne kadar Zeka-Akıl-Mantık ifadeleri birçok yerde yanlış kullanılmıştır.

Zeka Bir insan ne kadar çok bilgi biliyorsa ve/veya ne kadar çok düşünürse, ne kadar FARKLI ortamlarda ve durumlarda çalışırsa (deneyim kazanırsa) o kadar zekidir, o kadar **zekaya** sahiptir. Öğrendikçe veya deneyim sahibi oldukça; insanların bilgileri artar, buna bağlı olarak zekileşirler. Zeka arttıkça, problemleri çözme yeteneği artar. Sonuçta, zekayı, bir aracın benzini ve hızı olarak düşünebiliriz. Aşağıdaki iki örnekte, gerçek hayattan bir benzetme ile zekanın nasıl kavranabileceği ifade edilmiştir.

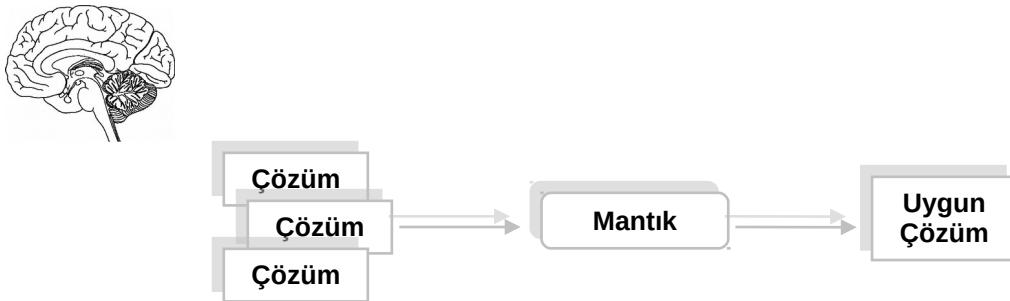
→ Ne kadar benzinimiz var ise o kadar uzağa gidebiliriz, o kadar farklı yerleşim birimine gidebiliriz

→ Ne kadar zeki isek, o kadar farklı çözüm üretebiliriz, o kadar farklı problemleri çözebiliriz.

→ Ne kadar fazla hıza sahipsek, (otomobilimiz ne kadar hızlı gidebiliyorsa) mesafelere o kadar hızlı (kısa sürelerde) ulaşırız.

→ Ne kadar zeki isek, çözümlere o kadar hızlı ulaşırız, o kadar çabuk ve mantıklı çözümler üretiriz.

Mantık **Mantık**, zeka'nın bulduğu çözümlerden en iyisini veya en iyilerini seçip karar mekanizmasına gönderen sistemdir. Buradaki, **iyi** kriterini belirleyen, insan duygularıdır. Mantık, bir çözümün daha iyi olmasını, insan duygularını temel alarak belirler. Yani, bir durum için, birden fazla çözüm yöntemi geliştirilebilir. Daha sonra bu geliştirilen yöntemlerden “hangisi daha mantıklıdır” sorusuna cevap buluruz. Bu durumu; hafızamızda, elekten geçen bir dizi çözümü canlandırarak daha iyi kavrayabiliriz.



Figür 1.2.1 H : Beynin bir sorun için birden fazla çözüm üretebilmesi, ve mantığın bu çözümleri eleyerek en iyisini seçmesi

→ Örneğin, İstanbul'dan, Ankara'ya gitmek için, zekamız bize

- İstanbul -> Ankara

- İstanbul -> Bursa -> Ankara

- İstanbul -> Edirne -> Ankara

...

olmak üzere rotalar çizebilir. Zeka için çözümün ne kadar iyi olduğu önemli değildir. İşte burada, zeka tarafından bulunan, çözümleri filtrelemek ve en iyisini seçmek üzere, mantık devreye girmektedir.

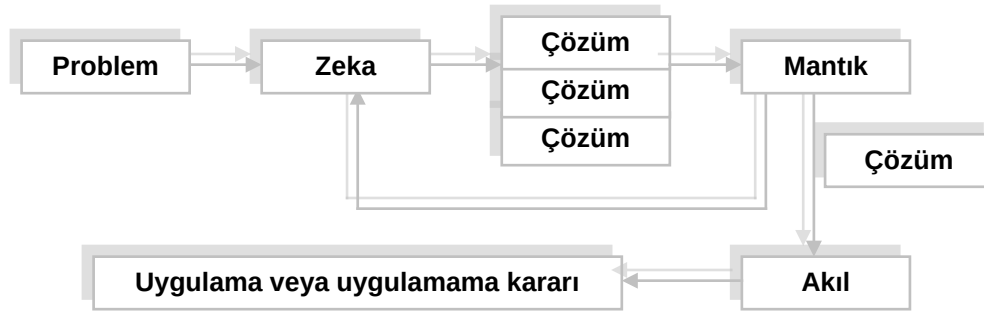
Örnekte, İstanbul'dan, Edirne'ye gittikten sonra Ankara'ya gitmek sorunu çözmektedir. Ancak mantıksal olarak, bu çözüm iyi bir çözüm yöntemi değildir. Çünkü, ekstra olarak; hem zaman, hem para harcamamız gerekmektedir.

Ekstra para harcamamız neden **mantıksal** değildir, veya mantığımız neden bu çözümle tatmin olmamaktadır.

- Çünkü para kazanmak için çalışılıyor, zaman ve emek harcanıyor. Zaman, Para ve Emegimizi istediğimiz, hoşlandığımız, zevk aldığımız aktivitelerde harcamak istiyoruz (para)
- Çünkü zamanımızı istediğimiz, hoşlandığımız, zevk aldığımız aktivitelerde harcamak istiyoruz. (zaman)

Tabiki eğer benzeri bir olay daha önce yaşanmamışsa, yani daha önce bir yerden başka bir yere gitmek için plan yapmışsak, zekamız, ilk olarak, en çok kullanılmış olanı (yani, İstanbul – Ankara) seçecektir.

Akıl Akıl ise, direksiyon başında aracı kullanan “kişi”dir. Yani ne yapılacağına karar veren sistemdir. İnsanların zekaları veya mantıkları tamamen aynı olabilir fakat hiçbir kimsenin aklı bir başkasıyla aynı değildir. Akıl her kişiye verilmiş olan karar verme yeteneğidir.



Figür 1.2.1 I : Beynin bir sorun için birden fazla çözüm üretebilmesi, ve mantığın bu çözümleri eleyerek en iyisini seçmesi, daha sonra aklın karar vermesi

Yukarıdaki figürde görüldüğü gibi, mantık uygun çözümü bulamazsa tekrar zekaya dönüp, yeni bir çözüm bulmasını istiyor. Eğer uygun bir çözüm bulursa, bu çözümü akla iletiyor. Akılda bu bulunan çözümün uygulanıp uygulanmayacağına karar veriyor. Yani bulduğumuz her çözümü uygulayamayız. Bu kararı verende, insan aklıdır. Aslında bu konu beynin çalışma yapısı ile ilgili olmasına karşın, bu konu hakkında bilgiye sahip olmamız gerekiyor. Bu hiyerarşiden daha sonrada yararlanacağız.

Bilgisayar dünyasında da kuracağımız algoritmalarının, mantıksal; yani bir problemi, düzgün ve uygun bir şekilde çözen bir sonuç olması gerekmektedir. Bu nedenle uygulanacak

olan algoritma düzgün tasarlanmalıdır. Düzgün algoritma tasarlamamanın ilk kuralı, algoritmayı kuracağınız sistem ve algoritmasını kuracağınız olayın ayrıntılarını ve özelliklerini yani etkenlerini iyi bilmek, bu sistem veya olayı iyi tanımanızdır.

1.2.2 Kimler algoritma kurabilir?

1.2.1 Bölümünde verdiğim örneklerle bakıldığında herkesin algoritma kurabileceği sonucu zaten çıkmaktadır. Algoritma kurmanın derecesi veya seviyesi bulunmamaktadır. **Kişiler arasındaki algoritma kurabilme farkı; bilgiye ve deneyime dayanmaktadır.** Daha çok algoritma kuran bir kişi, kurmayan bir kişiye göre, başka algoritmaları daha hızlı kurabilir, ayrıca; daha önceden kurmuş olduğu algoritmalarla benzer algoritmalarla karşılaştığında, tecrübeli olduğu için, düşünmeye vakit ayırmaz ve daha önceden kurmuş olduğu algoritmayı kullanır. Ancak unutmayalım ki tecrübe herşey değildir. Daha önce hiç karşılaşılmayan bir durumda tecrübe kadar, kişinin ne kadar zeki olduğu da öne çıkacaktır.

Ek Makale

Hiçbir kurala, hiçbir kitaba uymasa bile ben yapay zeka'yı 3'e ayırıyorum.(Bu düşünce bana aittir.)

- 1 - Yetersiz Zeka
- 2 - Yapay Zeka
- 3 - İnsan Zekası

Yetersiz zeka; sadece belirli durumlar için küçük bir çapta işlem yapabilen bir sistemdir.

Örnek olarak; bir sisteme 1 değeri girildiğinde A değeri çıkıyor 2 değeri girildiğinde B değeri çıkıyor. Bu sistem sadece bu iki değer için organize edilmiş olduğundan dolayı, her değer için teker teker işlem yapmak gerekmektedir. Yani bu sistem, fonksiyonel bir sistem değildir. Eğer bu sisteme 4 değeri girilirse ne olur? Hiç bir değer çıkmaz. Ayrıca Yetersiz zeka çok çok basittir. Yapay zekanın en basit hali olarak nitelendirilebilir. Bunu size şöyle bir örnekle açıklayayım.

Bu konuda gözlem yapabilmek için bilgisayarda üç tane otomobil yarışı oyunu oynadım. 1. otomobil yarışında benim kullandığım otomobil ve 5 tane bilgisayar tarafından kontrol edilen otomobil bulunuyordu. Yarış esnasında ben tam viraja yaklaşırken, bilgisayarın kontrol ettiği bir otomobile yandan hızlıca çarptım. Fakat, bilgisayarın kullandığı otomobil yavaş olmasına (viraja yaklaştığı için) ve ben, kullandığım otomobille son sürat o otomobile çarpma rağmen, otomobil yoluna devam ederken benim kullandığım araç yoldan çıktı.

Bunu yapan programcı, böyle bir durumda eğer bilgisayarın kullandığı otomobiller yoldan çıkarsa, bunu yeniden yola dönmesinin zor olacağını ve o kadar uğraşmak istemediği için, kullanıcının sürdüğü otomobili yoldan çıkartıyor. Nedeni ise, "kullanıcı insan, insan yolunu bulabilir, otomobilini toplayıp yeniden yola devam eder, fakat bilgisayarın kullanacağı otomobil yoldan çıkarsa, onu bir daha yola sokup devam ettirmek için çok kodlama yazmak gerekiyor (hız, açı, gelen otomobilin kuvveti, momentum, kayma ve lastik dayanıklılığı, sürtünme.....)" düşüncesidir (tabi ki her oyun mükemmel değil).

Başka bir otomobil yarışında aynı işlemi yaptığımda ise, ilk denememde benim kullandığım otomobili, bir-iki hamle ile toplayıp yola devam ettim. Gördüm ki yapay zeka, kullanıcının otomobili için güzel programlanmış.

İkinci denememde ise tam çarpma anından sonra durup, çarptığım otomobilin durumuna baktım. Tabiki çok yüksek süratle geldiğim ve hiç yavaşlamadan otomobile çarptığım için (normal olarak) 1-2 tur döndü. O sırada biraz hız veriyor olduğu için (virajı yavaş yavaş geçmeye çalışırken) spin atmaya başladı ve yoldan çıkıp çimlere girdi. Daha sonra frene bastı otomobilin direksiyonunu düzeltti. Yavaş yavaş hızlanarak yoluna devam etti.

Diğer bir otomobil yarışında ise, aynı şekilde son sürat gelip araca çarpmadan önce(çarpma 1-2 metre kala), araç bunu farkettiler ve yavaşladı, frene bastı. Ben her ne kadar süratli çarpsamda çok yavaş olduğu için fazla etkilenmedi. Sonra çarpmanın etkisinin geçmesine yakın yavaş yavaş hızlanarak devam etti ve benim kullandığım araçtan daha iyi bir çıkış bile yaptı.

Şimdi sizden 3 oyunu karşılaştırmanızı rica ediyorum.

İlkinde ki sistem yapay zeka değil yetersiz zeka, ikincisi ve üçüncüsü yapay zeka.

Fakat aralarında yine de fark bulunuyor. Yani aynen insanların arasında ki zeka farkı gibi...

İnsan zekası, kendi arasında, farklı bölümlere ayrılabilir. “Çok gezen mi bilir, çok okuyan mı” sözünü hepimiz duymuşuzdur. Bu sözden çıkarabileceğimiz: (1) gezmek, görmek daha görsel ve sanatsaldır, yani beynin sağ tarafını ilgilendirir. Çok okumakta, bilimsel temelli olduğundan dolayı, beynin sol tarafını ilgilendirir. Zekanın bu şekilde, biçime göre, bilimsel ve görsel olarak bölündüğünü söyleyebiliriz.

Bunun haricinde, insanlar için: “çok usta”, “çok deneyimli”, “çok iyi biliyor”.... terimleri kullanıldığında, gerçekleştirdikleri işi yıllardır yaptıkları anlamı ortaya çıkmaktadır. Birde, “çok zeki”, “hızlı çözüm üretir”, “hazır cevap”, “hemen çözer” terimlerinin kullanıldığı insanlar vardır. Bu tip ifadelerden de, kişilerin çok “hızlı” bir zekaya ve daha önceden görmede, yeni karşılaştığı bir durumu çözebilecek yeteneğe sahip oldukları ifade edilir. Buradanda, “hızlı” ve “kapsamlı” zeka türleri olabileceği sonucunu çıkarabiliriz.

Diğer bir kritere göre ayrılmış zeka türleri [e28]

- Müziksel Zeka: Müzik alanlarındaki beceri.
- Bedensel Kinestetik Zeka: Tüm bedeninin veya çeşitli bölümlerinin bir problemin çözümünde, bir üretim veya gösteri sırasında kullanılması ile ilgili becerilerdir; dans etme, atletizm, aktörlük, operatörlük gibi beceriler buna örnek gösterilebilir
- Mantık-Matematik Zekası: Problem çözme ve bilişsel düşünmedeki beceriler.
- Dilsel Zeka: Bir dilin kullanımı ve o dilde eserler üretme ile ilgili beceriler.
- Uzaysal-Konum Zeka: Mimarların, ressamların, heykeltıraşların veya uzay-konum durumlarını anlamadaki becerileri.
- Kişiler Arası İletişim: Diğer kişilerle etkileşimde diğerinin ruh halini, isteklerini, niyetlerini anlamadaki beceriler.
- İçerik yönelik Zeka: Bir kişinin iç dünyasındaki yönelimlerini anlaması, duygularına erişebilmesi becerisidir.

Algoritma kurma ile; algoritma kurabilme yeteneđi deđil, hızı ve kapsamı geliřir. İnsan beyninin alıřma yapısı ok karmařıktır. Örneđin, karanlık bir ortamda kaldıđınıza, belirli bir süre, normalde karanlık olduđundan dolayı ayrıntılarını seemeyeceđiniz cisimlerin ayrıntılarını görebilecek seviyeye gelirsiniz. ünkü, beyninizdeki **nöronlar** (beyin hücreleri), bütün duyu organlarınızı ve beyninizi, “o anki” duruma uygun hale getirmek için yeniden düzenler. Bu duruma ortama adapte olma da denilebilir.

Üniversitedeki bir hocam, bir kitabı 8-10 dakika içinde okuyabiliyordu (500 sayfalık orta boyutta). Bu başarıya nasıl ulařtıđını sorduđumuzda, yıllardır hergün 1 saat (1 sayfa veya belirli sayıda sayfa deđil: 1 saat. ünkü, 1 saatte hergün daha fazla miktarda okursunuz) kitap okuduđunu söyledi. Bu işlemleri yıllarca yapmış ve artık sadece sayfa çevirmesi vakit alıyordu. Aynı řekilde, Türkiye’de, karřılařtıđım bir hızlı okuma kitabının kapađında yazan slogan “300 sayfayı anlayarak 2 saatte okuyun”du. Bu řekilde, insan beynindeki nöronlar, sık olarak yapılmakta olan bu işlem için daha ok bađlantı kurar ve böylece, işlemler daha hızlı gerekleşir. İşte algoritma kurma ile, algoritma kurma hızınız bu örnekteki gibi artar.

Ünlü düşünürler, genelde, alıřma ortamının rahat olduđu, yařamının zengin ve kolay olduđu bir durumda ortaya ıkmıřtır.

Biz yani Türkler, genelde rahat bir yařam yařamazsakda, ok pratik zekaya sahibizdir. Bu sürekli düşündüğümüz veya düşünmek zorunda kaldığımız için olabilir. Ay sonunda kirayı nasıl öderim, işi nasıl tamamlarım, gibi sürekli problemler oluşur ve biz bir řekilde özmeye alıřırız. Avrupadaki insanlar bizim kadar pratik deđildir... Örneđin, bir markete gittik, hesap 11 YTL tuttu cebimizde 20 YTL banknot ve 3-4 YTL bozuk para bulunuyor. Hemen herkes, 20 YTL’nin yanında, 1 YTL bozuk para verir ve tam 10 YTL para üstü alır. Bu aslında günde 2-3 kere yaptığımız bir işlem olsada, Avrupa’da uygulanmamaktadır. Üniversite yıllarında, İngiltere’deyken, aynı olay başıma 4-5 kere geldi. Markete gittim ve hesap tam 11 sterlin gibi bir rakam tuttu. Bende, 20 sterlin verdim, üstüne birde 1 sterlin bozuk para verdim. Kasiyer, ilgin bir ifade ile yüzüme baktı.. Bana, “20 sterlin, zaten 11 sterlinden büyük, bir daha 1 sterlin vermenize gerek yok” dedi.

Devam etmeden Hertz kavramı ile ilgili ek bilgiye göz atabilirsiniz.

Hertz, bir sıklık birimidir. Bir saniyede bir kere yapılan işlem anlamına gelir. Yani bir saniye de bir işlem yapılıyorsa, o işlemi yapan kiři veya makine, 1 heartz hızla alıřıyor anlamına gelir. Yapılan işlemin cinsi veya kapsamı önemli deđildir.

Bilgisayar monitörlerinin bazılarının, 60 Hertz hızla alıřtıđını görmüşüzdür. 60 Hertz, saniyede 60 kere (1 saniye, 1000 salise olduđundan, $1000 / 60 \approx 16.7$ salise) yenileme yapıyor (ekranı tekrar baştan çizip görüntölüyor) anlamına gelmektedir.

Eđer bir saniye de 1000 işlem yapılıyorsa, 1 KiloHertz (KHz), bir saniye de 1 milyon işlem yapılıyorsa 1 MegaHertz (MHz), 1 saniye de 1 milyar işlem yapılıyorsa 1 GigaHertz (GHz), 1 saniyede 1 trilyon işlem yapılıyorsa 1 TeraHertz (THz) olarak adlandırılır.

İnsan beyninde, yaklaşık 100 milyar nöron vardır. Beynimizdeki bir nöronunun hızı 1 MHz bile deđildir (gerçek hız, saniyede 200 kere yani 200 Hertz). Öyleki, řuandaki bilgisayarlar 4GHz (bir saniyede 4 milyar elektriksel - matematiksel işlem) hızı yani

4096Mhz'yi geçmiştir. Yani bir bilgisayar, beynimizdeki nöronlardan en az 4096 kat daha hızlı çalışmaktadır. Dünyanın en hızlı bilgisayarı ise; bir nöronun hızından yüz binlerce kat daha hızlı çalışmaktadır. Ancak burada dikkat etmemiz gereken bir durum söz konusudur. **Bir bilgisayarla, bir nöronun kıyaslamasını yaptık, hâlbuki, insan beyninde 100 milyar nöron olduğunu söylemiştik. İşte durum böyle olunca, dünyanın en hızlı bilgisayarı beynimiz yanında sadece bir “hiç” olarak ifade edilebilir.**

Bu durumu bir örnekle pekiştirelim: Birkaç karıncanın gücü bize hiçbirşey ifade etmezken, bir-iki milyon karınca, toprak bir alanda yatmakta (toprak üzerine uzanmış) olan bir insanı, 2-3 saniye içinde toprağa gömebilmektedir. Başka bir örnek olarak, bir karınca suda boğulabilirken, birlikte hareket ettiklerinde bir sal şekli alır ve hayatta kalabilirler. Nöronların tek tek çalışması pek bir anlam ifade etmesede, birleştiklerinde inanılmaz derecede karmaşık işlemleri bile çözebilirler ve inanılmaz hızlara ulaşabilirler.

Beynimizin, milyarlarca bilgisayardan oluşan birbirine bağlı bir ağ olduğunu düşünebiliriz.

Aynı dünyadaki insanlar gibi.... Eğer bütün veya bir bölüm insanlar aynı amaçlar doğrultusunda çalışırsa inanılmaz derecede güçlü olurlar.

Nöronların hızları (belirli limitler içinde) sabittir. Yani hızları artmaz veya azalmaz. Bu verilere göre, beynimizin daha hızlı çalışması için daha çok nöronu kullanmamız gerekmektedir, işte bunun için pratik yapmak ve düşünmek önemlidir. Pratik, kullanılmakta olan nöronların arasındaki ilişkilerin ağırlıklarını uygun hale getirir, düşünmek ise, kullanılmakta olan bir bilgiyi diğer nöronlarla ilişkilendirir ve böylece o bilgiyi kullanmak istediğinizde, diğer bağlı olduğu nöronları da kullandığımız için beyniniz çok daha hızlı düşünür ve çözüm üretir.

Algoritma kurabilme yeteneği, kişinin IQ'su (internetdeki veya gazetelerdeki IQ testleri gerçek çözüm üretebilme kapasitesini ölçmez! Bu nedenle IQ'um X seviyede diye sevinmeyin veya üzülmeyin.) ile ilgilidir ve direkt olarak değiştirilemez, yukarda da belirtildiği gibi deneyim ve bilgi ile hızlandırılabilir.

Bizim için algoritmanın önemi, sadece programlama algoritması konusunda geçerlidir, yani programlamadır. Ancak, algoritma kurarak, daha zeki bir hâle gelebiliriz, ayrıca, bilgisayar dünyasında yazmış olduğumuz programlar gerçek hayatta daha hızlı düşünebilmemizi sağlayacaktır.

[Ara verme bölümü]

Alıştırmalar

- Algoritma nedir?
- Günlük hayatta ne gibi algoritmalar kurarız?
- Günlük hayatta kurduğumuz algoritmaların evreleri nelerdir?
- Neden algoritma kurarız?
- Kimler algoritma kurabilir?
- Problem nedir?
- Problemlerin gerçekleşme nedenleri ve çözümleri nasıl bulunur?
- Hangi durumlarda algoritma kurulur?
- Fonksiyon nedir? Fonksiyonlar nasıl ifade edilebilir?
- Günlük hayatta fonksiyon olarak gösterebileceğimiz bir örnek veriniz.

- Programlamanın kökeni nedir?
- Zeka nedir? Akıl nedir?
- Algoritma kurma yeteneđi neye bađlıdır?
- İnsan beyni nasıl bilgisayardan daha hızlı çalışır?
- Daha hızlı nasıl düşünebiliriz?
- Hertz nedir?

1.3 Programlama

1.3.1 Programlama - Program

Program Program, “a sequence of instructions that a computer can interpret and execute” [e4]; yani, “bilgisayarın, tercüme edip çalıştırabileceği bir dizi talimat” anlamına gelmektedir. Daha açıkcası, program, bir talimat listesidir. Algoritmanında, bir sorunu çözmek için uygulanacak olan bir dizi talimat olduğunu söylemiştik.

Buradan, program’ın, bir algoritmadan neredeyse hiç farkı olmadığını çıkarabiliriz. Tek farkı, algoritma herhangi bir ortamdaki bir sorunu çözmek için kullanılan yöntemdir, programlama ise, bilgisayar ortamı içinde (veya bilgisayar ortamı içine aktarılan) bir sorunu çözmek için izlenen yoldur.

```
public ZipFile(File file, int mode) throws ZipException,
IOException
{
    if ((mode & OPEN_DELETE) != 0)
    {
        throw new IllegalArgumentException
            ("OPEN_DELETE mode not supported yet in
java.util.zip.ZipFile");
    }
    this.raf = new RandomAccessFile(file, "r");
    this.name = file.getPath();
}

private int readLeShort(DataInput di, byte[] b) throws
IOException
{
    di.readFully(b, 0, 2);
    return (b[0] & 0xff) | (b[1] & 0xff) << 8;
}
```

Figür 1.3.1 A : Bir programlama dilinde yazılmış program örneği.

Yazılım Yazılım, “Computer software is a general term used to describe a collection of computer programs, procedures and documentation that perform some tasks on a computer system [e5]” yani “Bilgisayar yazılımı, bir dizi yazılmış programı veya prosedürü, bunlarla birlikte dökümantasyonu ifade eden, bilgisayar sistemleri üzerinde belirli bir işlemi gerçekleştiren, genel bir terimi anlatmak için kullanılır” anlamına gelmektedir.

Bilgisayar üzerinde bir işlem gerçekleştirmek üzere tasarlanmış programlar veya prosedürler, ve onları açıklayan dökümantasyon... Basit bir tanımla, yazılım bir program ve o programın diğer ayrıntılarıdır yani; dökümantasyonu, kurulumu, kullanım şartları, lisansı

İki tip yazılım bulunmaktadır: (bespoke) Özel ve (generic) Genel

Genel yazılımlar, büyük kitleleri hedef alan ve herkes tarafından kullanılabilecek yazılımları içermektedir. Örneğin; Messenger, Winamp, Winzip

Özel yazılımlar, firmalara özel olarak yapılan yazılımları içermektedir. Örneğin, bir fabrikaya özgü üretim takip programı.

Bu tip tanımların (programlama, algoritma veya yukarıdaki program) sözlük anlamlarının önemi yoktur. Çünkü, burada öğrenmek istediğimiz Türkçe terimler değil, programlamadır (yani dersimiz edebiyat değil, programlamadır). O yüzden, tanımın, gremer olarak düzgünlüğünün önemi bulunmamaktadır. Size, “programlama” nedir diye sorulsa, hemen sözlükte anlamına baktığınız herhangi bir kelimenin açıklaması formatında bir cümle kalıbı aklınıza gelir ve siz bu kalıba uygun kelimeleri yerleştirerek tanımlı yaparsınız.

Örneğin;

Temizleme:

- 1 . Temizlemek işi.
- 2 . Yüzeylere yapışmış leke ve kirlerin giderilmesi, çözelti veya asıltı durumuna getirilmesi olayı. (Türk Dil Kurumu web sitesinden alıntı)

Sizde burada belirtilen tanımlara benzer bir şekilde;

“Bilgisayarların bir işlemi gerçekleştirilmesi için ayarlanması – kurulması için.....” gibi bir tanım kullanırsınız. Ancak, tanım ne kadar doğru olsa da bizim işimize yaramamaktadır.

Basit olarak diyebiliriz ki, **programlama, program yapmaktır**. Birçok kişi, özellikle genç kesim, programlamanın bilgisayar bilimine ait bir kelime olduğunu düşünür. Halbuki, **program**, bilgisayar keşfedilmeden öncede kullanılan bir kelimeydi. **Programın ne olduğunu daha iyi kavrayabilmek için, bilgisayar ile olan ilişkisini koparmalıyız**. Bu yüzden bilgisayarın keşfedilmediği bir zamanı düşünerek, böyle bir zamanda, programın ne anlama geldiğini düşünmeliyiz (örneğin iki - üç yüz yıl önce veya daha öncesinde). Bu şekilde düşünmeliyiz ki programı bilgisayardan ayıralım. Belkide program bizim daha önceden bildiğimiz ve kullandığımız bir tanımdır...

Böyle bir zamanda, **programlama**; belirli bir durumda, ne gibi işlemler yapacağımızın prosedürünü oluşturma anlamına gelmektedir. Yani, programlama, gerçek hayatta planlamadır veya organize etmedir.

Günlük hayatımızda kullanılan ve bilgisayarla alakası olmayan, içinde program geçen ifadeler;

- “Yarın boş musun ?”
- “**Programı**ma bakmam lazım”
- Haftalık ders **programı**
- “**Programım** çerçevesinde yardımcı olmaya çalışacağım”
- Davetin açılış **programı**

- “Bu iş böyle olmaz, bir **program** yapalım herkes ona uysun”

- “Saatleri ayarlayın, **programa** göre geç kalmamalıyız”

Dikkat ettiyseniz program kelimesi, birçok cümlede, bir tür **plan** anlamına gelmektedir. Peki, bugüne kadar, hiç birimiz, herhangi bir konuda plan yapmadık mı? Yani bir işi planlamadık mı? Tabiki planladık. Peki, eğer programlama, planlama ise, ve biz bugüne kadar plan yaptı isek, o zaman programlama da yapabiliyoruz demektir. Çünkü **plan** da zaten bir problemi izlenecek yolun tasarımıdır (**plan**: Bir işin, bir eserin gerçekleştirilmesi için uyulması tasarlanan düzen [e6]).

Programlama yapabilme yetisi sizde şuan mevcuttur ancak siz bunun farkında değilsiniz. Bu kitabın amacı da, size zaten biliyor olduğunuz “programlamayı” saklandığı yerden nasıl çıkaracağınızı ve kullanacağınızı göstermektir. Bu durum aslında felsefenin önemli bir konusudur. **Bilgi doğuştan mıdır yoksa sonradan mı?**

Sürekli olarak kullandığınız, ve “**çok**” diye ifade ettiğimiz her tür; nesne – konsept – tanım bize yabancı yada önemsiz gelecektir. Örneğin, bazen kendi yüzümüzü hayal edemeyebiliriz, telefon numaramızı unutabiliriz, cebimizde kaç para olduğunu unutabiliriz. Bazen bir konu için verilebilecek çok sayıda örnek bulunuyorsa bu örnekleri hatırlayamayabiliriz (çok sayıda olmasına karşın). Eğer bir maddeden çok bulunuyor ise, o madde değersiz olur. Örneğin demir gibi. Halbuki, altın, daha az bulunduğu için çok değerlidir. Bizde günlük hayatta onlarca – yüzlerce algoritma kurduğumuz, onlarca – yüzlerce plan yaptığımız için, plan yaptığımızın farkında bile olmayız, veya olsak bile bizim için plan kurma önemsizdir.

İnsanlar genelde, her olay ve her durum için **önceden** plan yapmazlar, çünkü insanların **karar verme yeteneği (akılları)** vardır ve birçok durumda o an karar verip uygulayabilirler. Varsayalım ki, sinemaya gidiyorsunuz. Evden çıkarken,

“eğer sinemada yangın çıkarsa, bende sinemadan çıkarım”

“eğer sinemada film izlerken deprem olursa önce kendimi korur, sonra deprem bitince dışarı çıkarım”

“istediğim macera filminde yer yoksa, o zaman savaş filmi var ona gidebilirim”

.....

gibi durumlar için plan yapmayız. Çünkü bu konularda plan yapmaya ihtiyacımız yoktur. Aklımız yani karar verme yeteneğimiz olduğu için, böyle durumlarda, kararı o an verebilir ve uygulayabiliriz, bu tip durumları her sefer planlamak için zaman harcamamıza gerek bulunmamaktadır. Ayrıca yukarıdakilere benzer tip olayların sonunda yoktur.

Tabi ki, insanlar önceden planlayarak davrandıklarında, daha düzgün davranışlar sergilerler - daha iyi karar verir ancak, her durum için, plan yapmakta çok büyük bir zaman kaybı olacağı için, sadece önemli durumlar için plan yapılması daha uygundur.

Örneğin Japon’lar, çok fazla sayıda plan yaparlar. Bir duvara matkapla delik açacaklarında bile hangi konumdan, nasıl, ne kadar derinlikte, hangi uçla, bir delik açmaları gerektiğini planlarlar (bir keresinde benzer bir olaya şahit olmuştum).

Ülkemizde (malesef) ise en önemli işler bile ayrıntılı planlanmadan yapılmaktadır. Programlamada eğer iyi planlama yaparsanız, yazdığınız programın her bölümünü test etmeniz gerekmez. Bu şekilde tasarım yaptığınızda (ayrıntılı - planlayarak), test sonucunda, değişmesi gerekebilecek kısımların sayısının çok az veya sıfır olduğunu yani daha az hata yaptığınızı görürsünüz (ayrıntılı bilgi ileriki bölümlerde mevcuttur).

Programlama (Bilgisayar bilimindeki) Programlama: bir sistemin; bilgisayar ortamına, o sistemin özelliklerini, işlemlerini, davranışlarını, olaylarını ve spesifikasyonlarını tanımlayacak şekilde aktarılması anlamına gelmektedir. Bu işlem; bilgisayara aktarılacak olan sistemin, diğer bilgisayara aktarılmış (yani programlanmış) sistemlerle, gerçekte olan, ilişkilerinin kurulmasını da içermektedir.

Programlama, terim olarak, 'creating a sequence of instructions to enable the computer to do something [e7]' yani, 'Bilgisayarın, bir işlemi gerçekleştirmesini sağlayacak bir dizi talimat oluşturmak' anlamına gelmektedir. Daha basit bir tanım olarak, bir yazılımın oluşturulmasında geçen evrelerin toplamına programlama denilebilir.

Bilgisayarların karar verme yeteneği (yani akı) olmadığından dolayı, karşılaşılabilecekleri her farklı durum için, onları, "programlarız". Yani ileride karşılaşılabileceği (kullanım sırasında) her durum için onları **tembihleriz** ve

"şöyle" bir durumda, "böyle" bir işlemi,
"x" durumunda, "a" işlemini,
"y" durumunda, "b" işlemini gerçekleştir, şeklinde direktifleri önceden belirleyip programı yazarız (ayrıntılı bilgi 2.bölümde mevcuttur).

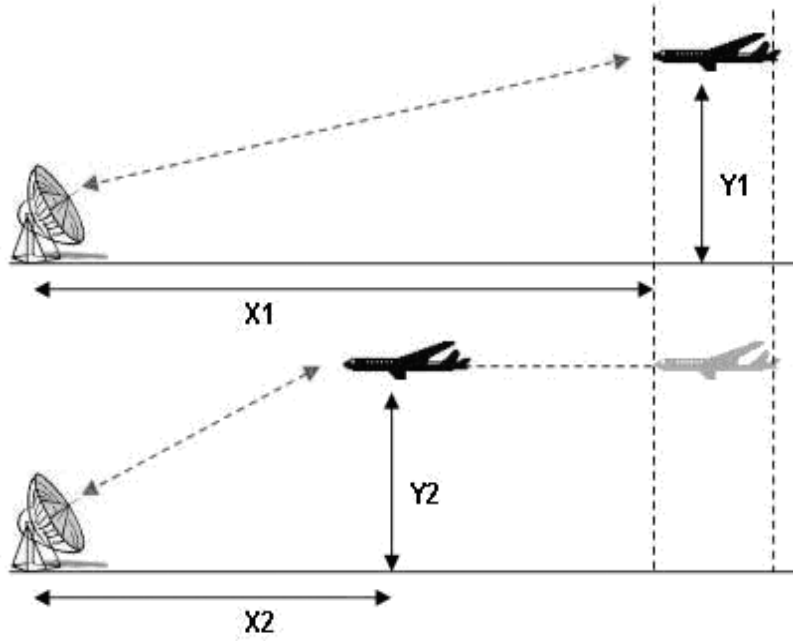
Şüphesiz ki (yapay zeka kullanan sistemler hariç), bir program, daha önceden belirtilmeyen bir durumda, bir insanın hiç bilmediği bir konuda hiçbir işlem yapamadığı gibi, bir işlem gerçekleştiremeyecektir. İşte bu yüzden hiç bir zaman her işlemi yapan bir program yazılamayacaktır, bunun yerine: muhasebe programı, ofis programı, müzik çalma programı..... gibi binlerce farklı alanda farklı durumda eylem gerçekleştiren ayrı programlar yazılacaktır. Zaten her tür işlemi gerçekleştiren bir programın yazılması bilimsel olarakta imkansızdır. Herbert Simon ve Allen Newell isimli bilim adamları, 1957 yılında, Genel Problem Çözücü (General/Global Problem Solver) fikrini ortaya attılar ve bu teoride, her sorunu çözebilecek bir sistemin olduğunu savundular. Ancak daha sonra kendileri, böyle bir teorinin gerçek olamayacağını ifade ettiler (ve ödüllendirildiler. Bir yöntem keşfetmeseler bile, böyle bir yöntemin yanlış olduğunu keşfettiler).

1.3.2 Veri - Bilgi

Veri **Veriler** gerçeklerdir. Bir makinenin verdiği çıktı, bir deneyin sonucu, bir aracın ölçüm sonucu, veri olarak nitelendirilebilir. Tek başına bir anlam ifade etmemektedir. 3cm, 5kg, 7, sıcak..... gibi ifadeler veri olarak nitelendirilebilir.

Bilgi Bilgi ise, bir veya birden fazla veriyi değerlendirerek, elde edilen, ve anlamı olan sonuçlardır.

Örneğin; bir sınavdan 100 üstünden 43 puan aldınız. Bu bir veridir ve işe yaramayacaktır. Ancak başka bir veri olarak, sınavın geçiş notunun 40 olduğunu bildiğimizi varsayalım. Sınavın geçiş notunun 40 olması da bir veridir ve yine tek başına bir anlamı yoktur. Ancak bu iki veriyi birleştirdiğimizde, (eğer geçiş notu 40 ise ve ben 43 almışsam) sınavdan geçtiğimiz **bilgisini** elde ederiz. Başka bir örnek olarak, bir öğrencinin notu 56 (veri) olabilir ve bizim için bir anlam ifade etmezken, bütün öğrencilerin ortalamasını hesapladıktan sonra, o öğrencinin ortalamanın üstünde mi yoksa altından mı (yani çalışkan bir öğrencimi yoksa değilmi) olduğunu tespit edebiliriz ve bu tespit bir bilgidir.



Figür 1.3.2 A : Veri - Bilgi arasındaki ilişki

Örneğin, bir uçağın yönünü tespit etmek için, yaptığınız ölçümler sonucunda, havaya göndermiş olduğunuz sinyal, belirli bir süre sonra geri gelmektedir. Gönderdiğiniz bölgenin koordinatları, gönderdiğiniz sinyalin X° ve Y° açısı, giden sinyalin geri gelme süresi, bu ölçümdeki verilerdir. İki ölçüm yaptığınızı varsayalım. İki ölçüm sonucunda, 2 farklı (veya aynı) veri elde etmiş oluruz. Bu verileri kullanarak, uçağın nerede olduğunu, hızını ver rotasını (yönünü) öğrenebiliriz, öyle ki, bu 2 sonuç, verileri kullanarak elde ettiğimiz, bilgilerdir. Yani bilgiler, verilerin işlenip daha anlaşılır hale getirilmesi ile elde edilmektedir.

İlk baştaki sinyalin gönderilmesinden X saniye sonra sinyalin geri gelir (bir nesneye çarparak - aksi taktirde, sinyal geri gelmez). “X saniye” bir veridir. X saniye, bir anlam ifade etmemektedir. Ancak, gönderilen sinyalin ne kadar hızlı olduğunu (metre/sn) kullanarak, Y metre uzaklıkta bir cismin olduğunu bulabiliriz. İşte bu bulduğumuzda bilgidir ve bir anlam ifade etmektedir.

1.3.3 Programlar ne işe yarar?

Matematikteki “+”, “-”, “×”, “÷” sembolleri, sağına ve soluna yazılan sayıları işler ve sonuç olarak verir. Yani yukarıdaki matematiksel sembollerden birinin öncesine ve sonrasına birer sayı yazdığımızda, bu sembol, bir işlem gerçekleştirecek ve sonucu bize verecektir. Yani sayıları işleyip, yeni bir sonuç verecektir.

Bilgisayarda hemen hemen bütün işlemlerin veriler kullanarak yapıldığını belirtmiştik. Veri ekleme, veri silme, veri düzenleme (değiştirme), veri okuma ile birlikte, beşinci işlem olan **veri işleme**, yukarıdaki örnektende anlayabileceğimiz gibi “**bilgi**” elde etmek için tasarlanmıştır.

Bilgisayarların dolayısıyla programların temel amacı, işlemleri hızlandırmaktır, bu hızlandırma, programların, verileri işleyerek, bilgi ve hatta karar çıktısı sağlamasıyla mümkün olmaktadır.

Örneğin, yazdığımız bir program, bir şirketin mali işlemlerinin takibini, istatistiğini ve yorumlamasını yapıyor ve daha sonra bu verileri işleyip, bilgi, bilgileri de işleyip, karar elde ediyor ve bize, “geçen aylara bakarak; bu ay, X ürünümüzü tanıtmak için reklam kampanyasına Y lira yatırım yapmalıyız” şeklinde bir çıktı veriyor olabilir.

Şimdi de bu anlattıklarımı ayrıntılı açıklayayım.

Programlar, bildiğimiz gibi, işlemciye sahip makineler veya bilgisayarlar üzerinde çalışırlar. Bilgisayarların bir tür makine olduğunu belirtmiştim. Peki, (örnek olarak) ürün üreten bir makine (ürün üreten bir fabrikadaki makineyi düşünelim) genelde ne yapar? İçine yerleştirilen, hammaddeyi veya yarı-mamülü işler ve sonunda işe yarar ürünler haline getirir. Programlarda, bilgisayarlar veya işlemcileri olan makineler üzerinde çalışabildiklerine göre, aynı tür işlemler gerçekleştirirler.

Programlar, genellikle bilgisayar üzerindeki veriyi **işler**, veri girişi (kayıt) ve görüntülemesi (okuma) yaparlar. Örneğin Winamp (müzik dinleme programı – www.winamp.com), bilgisayar üzerindeki MP3 (şarkı) dosya verisini işler, ve bu dosyanın sesini ses kartını kullanarak, kullanıcıya iletir. Muhasebe programları, ofis programları, çizim programlarından birini mutlaka kullanmışızdır. Bu tip ofis programları; üzerine girdiğimiz veriyi işlerler. Veri olarak tabir edilen bir şirket verisi, resmi kayıt veya benzeri bir türde olmak zorunda değildir. Messenger gibi programlar yazdığınız chat mesajını işler ve karşı tarafa gönderir, daha sonra, başkalarından gelmiş olan verileri işleyip kullanıcıya gösterirler. Antivirus programları, dosyalar içindeki verileri okuyarak virus olup olmadıklarını anlarlar. Grafik programları (PhotoShop, Paint shop Pro...) resimleri işler ve üzerine eklemeler - değişiklikler yapılmasına olanak sağlar.

Programlar genelde, hayatımızda olan işlemleri kısaltmak için kullanılır. Yani bir program, gerçek hayatta yapılamayan bir olayı yapabilmek için kullanılmaz. Programlar, gerçek hayatta yapılan işlemleri hafızada tutmak, sorgulamak, yorumlamak ve bizim manuel olarak yaptığımız işlemleri otomatik yapmak, **süreden ve masraftan tasarruf sağlamak amacıyla kullanılır**. Aynı bir hesap makinesi gibi...

Yukarıdaki paragrafta geçen, “süreden ve masraftan tasarruf sağlamak amacına” uygun olarak bir örnek verelim:

1645 yılında Fransız filozof **Blaise Pascal**, vergi tahsildarı olan babasına yardımcı olmak için bir hesap makinesi tasarladı [e8]

Bir hesap makinesinin yaptığı işlemlerin hepsini insanlar çok kolay bir şekilde yapabilir. Ancak hem doğru hemde çok hızlı sonuç almak için hesap makineleri

keşfedilmiştir. Fakat hiçbir şekilde bir hesap makinesinin içine kodlanmayan bir işlemi (örneğin türev) gerçekleştirmesi beklenemez!

Basit anlamda baktığınızda, programlar, bilgisayara bağlı olan donanımları hareket ettirir, onları çalıştırır, veri okur – veri yazar v.b. işlemleri gerçekleştirir. Tabi bilgisayarları küçümsememeliyiz. Zira aynı mantıkla düşündüğümüzde, insanlarda gün içinde, kollarındaki, bacaklarındaki ve vücutlarındaki diğer organlarda bulunan kasları çalıştırır. Ancak bunlar bir düzen içinde olduğunda ortaya bir “iş” yani “eylem” çıkar. Örneğin, kaslarımızı dolayısıyla vücudumuzu belirli bir düzende belirli bir sırada hareket ettirdiğimizde dans figürleri ortaya çıkmaktadır. Aslında, elde edilen dans figürleri, kaslarımıza gönderilen elektriksel sinyallerden ibarettir.



Figür 1.3.3 A : Bütün figürlerin aslında elektriksel sinyallerdir

Programlar, insanların onlara emrettiklerini yaparlar ancak insanlara göre çok düşük yeteneklere sahiptirler. Bilgisayarları ne kadar mükemmel tasarlasanız tasarlayın, bazı noktalarda eksik kalacaklardır. Ancak bir yönde bilgisayarlar insanların yapamadığı bir işlemi gerçekleştirebilmektedir: Rasgele sayı üretmek [aslında bir bilgisayar bile rasgele sayı üretemez]. İnsanlar hiçbir zaman bir görüntüye, düşünceye, ilişkiye, olaya, bağımsız kalarak rasgele bir sayı üretemezler. Mutlaka belirtilen sayı birşeyle ilgilidir. Siz her ne kadar, “3.540.192”, “6.340.746” gibi uzun sayılar söyleyerek rasgele olduğunu düşünmeye çalışsanızda, aslında belirttiğiniz sayıdaki rakamların dizilişi belirli bir sıradadır ve yine rasgele değildir. Mutlaka bilinçaltınızda bu sayıyla ilgili bir bilgi bulunmaktadır ve bu bilgi böyle bir sayıyı söylemenizi tetiklemektedir. Düşüncelerden bağımsız rasgele sayı üretilemez. Bilgisayarlar bu noktada insanları geçiyor.

Başka bir nokta; bilgisayarlar, bizim ona öğrettiğimizden başka hiç bir işlem gerçekleştirememektedirler. Yani diyoruz ki biz bilgisayara, toplama işlemini öğretirsek, çıkarma işlemini yapamaz. Sadece ve sadece toplama işlemini yapabilir...

Evet, doğru, bu kadar ilkel bir araç, ancak birde şu yönden bakalım. Eğer toplama işlemi tamamen doğru düzgün öğretilmişse, bütün toplama işlemlerini %100 doğru yapacaktır. Çıkarma işlemini yapma özelliği olmamasına rağmen toplama işleminde hiç bir zaman hata yapamayacaktır. Yapamayacaktır diyorum çünkü biz, bilgisayara hata yapabilme özelliği eklemedik. Yani isteğimiz dışına asla çıkamaz, sonuçta 50 – 60 haneli toplama işlemlerinde insanlar hata yapabilirler ancak bilgisayar ASLA...

Madem dediğimizden çıkamıyor, o zaman hata da yapamaz... Çünkü hata yapabilme özelliği yok... Hata yapabilme özelliği insana ait bir özelliktir.

Bazı bilgisayar oyunlarında oyunlara, BİLEREK hata yapabilme özelliği konulur. Örneğin bir otomobil yarışında... Eğer otomobillere hata yapabilme özelliği eklenmemiş olsa hiç bir şekilde bilgisayarın kontrol ettiği otomobillerden daha iyi bir süre de, o yarışı bitirmemiz söz konusu olamaz. Bu da kullanıcının hoşuna gitmez. O yüzden bazı virajlarda bilgisayarın kullandığı otomobiller bile yavaşlar (ekstra olarak) veya rasgele bir sayı üretip, belirli bir değerden küçükse, virajda yoldan çıkar yoksa, düzgün geçebilirler...

Sonuç olarak bilgisayarlar hata yapamaz ve bu bizim için en önemli özelliktir. İşte insanlarda bilgisayarların hata yapamam özelliğini ve hızlı işlem yapabilme özelliğini kullanır.

Ek Makale: Kaos Teorisi ve Rasgelelilik

Böyle bir makaleden ne beklediğinizi belki birçoğunuz bilmiyor. Belki ismi ilginç geldi diye, belki de, bu makaleyi okumaya harcayacak kadar boş vaktiniz bulunuyor.

Aslında bu makalenin, ne anlattığı değil, makaleyi okuduktan sonra hayatınıza ne katacağı önemlidir. Çok uzatmadan konumuza girelim. Makale içinde, üzerinde duracağım kuram, kaos kuramına benzer olan, evrende hiçbir eylemin, hiç bir işlemin, rasgele olmadığı gerçeğidir.

İlk olarak bilimsel çalışmalara bakabiliriz. Bilimsel çalışmaların %90'ının üzerine kurulu olduğu kuram, eğer yanlış değilse, büyük ihtimalle doğrudur kuramıdır. Yani, ortaya bir tez sunduk, sonra bu tezin doğru olduğunu ispatlamaya değil, yanlış olduğunu ispatlamaya çalışırız. Evet, kendi tezimizi, sanki başkasının teziymiş ve yanlış bir tezmiş gibi, yalanlamaya, çürütmeye çalışırız. Eğer birçok bilimsel yöntem ile bu çürütme işlemi başaramamışsak ve kendimizi “bu tez yanlış değildir” şeklinde tatmin etmişsek (bilimsel otoriteler tarafından kabul görürse), bu tez doğrudur demektir. Daha doğrusu bu teori, doğru olarak kabul edilir.

Bu olayı bir örnek ile açıklayayım. Örneğin, 100 kişilik bir ekiptesiniz ve ekipte bazı casuslar olduğunu biliyorsunuz. Casusların kim olabileceğini bilmiyorsunuz, ancak, kim olmayacağını biliyorsunuz. Böyle bir durumda, casus olamayacakları seçeriz. İşte böyle durumlarda %100 doğru sonuç elde etmesenizde, elde edeceğiniz sonuç genelde doğrudur. Aslında, casus olabilir diye seçtiğiniz kişilerin sadece (Belki) bir tanesi bile casus diğerleri masum olabilir. Ancak bilim bu kısmı önemsemez. Önemli olan, seçtiğiniz insanların casus olmamasıdır.

Yani sonuç olarak, bu yöntemi, “eğer yanlış değilse, doğrudur” şeklinde ifade edebiliriz.

Konumuza dönelim: Evrende herşeyin – her sistemin – her işlemin, rasgelemi değil mi olduğunu bulmaya çalışıyoruz. Benim tezim, evrende hiçbir şeyin rasgele olmadığıdır. O zaman, evrendeki, herşeyin, rasgele olduğunu ispatlamaya çalışalım, eğer başarısız olursak, o zaman anlarız ki, evrendeki herşey bir düzen üzerine kurulmuştur.

Bu işlemi yaparken, dikkat edeceğimiz konu, uç noktaları önemseyecek olmamızdır. Yani zaten belirli bir düzen içinde olan, sıradan – her gün yaptığımız eylemleri değilde, rasgele olduğuna inandığımız olaylar – eylemler üzerine yoğunlaşmalıyız.

İnsanların pek merak etmediği konulardan biri olan rasgele oluşum, aslında, evrendeki herşeyin bir düzen içinde olmadığını veya olduğunu kanıtlamanın en iyi yoludur diyebiliriz. Rasgele oluşumun içinde; rasgele bir sayı üretmek, rasgele bir yere gitmek, rasgele bir yemek yemek.... gibi eylemler olabilir. Bunlardan herbirinin, hiç bir amaca dayanmaksızın olduğuna inanırız. İşte bu durum tamamen yanlıştır.

Bu örneklerin hepsinde, amaç, amaca dayanmayan bir durum oluşturmak olduğu için, bu örneklerden sadece bir tanesini açıklayacağım: rasgele sayı üretmek. Malesef, bilim, bize yalan söylüyor. “Rasgele sayı” diye bir kavramın olması söz konusu olamaz. Bunu duyduğunuzda hemen hemen hepiniz, “ben rasgele sayı atarım kafamdan, hiçbir şeye de bağlı olmaksızın” ifadesini kullanmıştır.

Ancak, insan, rasgele sayı üretme konusunda inanılmaz derecede acizdir. Bu konuyu ayrıntılı açıklayayım (Hem bilimsel hemde örnek tabanlı olarak). Diyelim ki sayısal loto oynuyorsunuz. Bildiğiniz üzere, bu iş için, 1’den 49’a kadar 6 rakam seçmeliyiz. Bu da, 49’un 6’lısı olarak yaklaşık 14 milyonda bir doğru tutturma olanağı çıkıyor. Peki, şimdi varsayalımki, ilk kolonu dolduruyoruz. 1,2,3,4,5,6 veya 3,4,5,6,7,8 veya 44,45,46,47,48,49,....

Birçoğunuz, “böyle peşpeşe hayatta çıkmaz, araya farklı rakamlar seçilmeli” demiştir. Ancak, oluşturulabilecek her varyasyon aynı değil midir? Yani yukarıda belirtilen varyasyonlardan herhangi birinin doğru çıkma olasılığı, bizim genelde oynadığımız gibi karışık (4,20,23, 37, 41, 44... gibi) bir varyasyonla aynı orana sahip değil midir: 1/14milyon. Yani 14 milyonda 1 olasılık.

Başka bir örnek verelim: 28 hafta boyunca, aynı sayılara oynuyorsunuz. [1, 6, 19, 25, 35, 40]. Sonrasında “yok bu sayılara çıkmayacak” şeklinde bir ifade kullanabilirsiniz. Hatta bırakın 28 haftayı, 2 kere aynı sayılara oynamayız bile. Peki, ilk haftada çıkmadı diye, bu hafta o varyasyonun çıkma olasılığı sıfırlanıyor mu? Hayır. Dikkat ettiyseniz böylesine basit bir konuda bile çok aciz kalıyoruz.

Başka bir örnek, fiyatı aynı olan üç – dört – beş ürün olsun (markaları farklı). Siz her ne kadar ben raftan bakıp rasgele aldım desenizde, reklamlarını gördüğünüz firma bilinçaltınıza girip sizi çoktan etkilemiştir bile. Aksini söylemeye çalışmayın. Zaten eğer, bu reklam aklınız seviyesinde kalmışsa, o reklam etkili değildir. Bilinçaltınıza girmeli ve sizi oradan, siz farkında olmadan etkilemeli. Eğer, etkilendiğimiz farkındaysak zaten bilinçaltında değildir. Aksi takdirde, bir sponsorluk reklamına 4 milyon dolar gitmezdi. Türkiye’nin en çok izlenen (akşam saatlerinde) programlarında, bir gecelik sponsor ve ara reklam bedelidir bu rakam. Birde, bir kanalın bütün gün içi reklamlarını düşünün, sonra buna bütün kanalları da dahil edin. Elde edeceğimiz rakam, 100’e yakın bir rakam. Bu kadar büyük paraların döndüğü bir piyasada, sakın bu reklamlar beni etkiliyor demeyin.

Evet rasgele sayı üretme konusuna dönelim. Baştan söylüyeyim insanlar, rasgele sayı üretemez. Rasgele sayı dediğiniz, birinin daha önce söylediği bir sayı, gördüğünüz bir plakadaki sayı, telefon numaranızın son hanesi, arkadaşınızın toka sayısı, gibi

binlerce etkenden biri olabilir. Yine birçoğunuz, hemen aklında, “3232424”, “3490902”, “9038943”, “232”, “8583”.... gibi sayılar üretmeye başlamıştır. Ancak bunlar her ne kadar siz, kendinizi tatmin etmiş olsanızda, rasgele sayılar değildir. Herbiri bir dizi rakamdan oluştuğu için, aslında bunlar için iki durum söz konusudur... İlk olarak uzunluk: Yani kaç tane rakam söyleyince (artık bu rasgele bir sayıdır) diye tatmin olacağınız. İkinci olarakta, her bir hanede bir rasgele sayı seçmeye çalışmak.

Ancak bu yöntemde de hiç bir şekilde rasgele bir sayı seçemeyiz. Aslında sayının uzun olması bir anlam ifad etmez. Sadece karşdakini “işte bu kadar uzunlukta bir sayı söylüyorsa bu kesinlikle rasgeledir” şeklinde tatmin etmeye yarar.

Rasgele sayı üretmek veya buna bağlı olarak rasgele hareket etmek diye belirttiğimiz durum, insanlarda (ve hayvanlarda) gerçekleştirilemediği gibi, bilgisayarlarda da gerçekleşmemektedir. Bilgisayarlarda rasgele sayı üretmek veya rasgele davranmak çok büyük olmasada önemli bir rol oynar. Bazı bilgisayar efektleri, bilgisayar oyunları, yapay sinir ağları, v.b. uygulamalarda rasgele davranışlar çok önemlidir. Örneğin bir bilgisayar oyununu düşünelim. Bu oyunda iki otomobil olsun ve bir tanesi bir insan tarafından bir taneside bilgisayar tarafından yönetiliyor olsun ve bu oyunda amaç yarışı daha önce bitirmek olsun. Eğer, rasgele hareket olmasa, bilgisayar oyununda, insan tarafından kullanılan otomobilin, bilgisayar tarafından kullanılan otomobili geçmesi durumu söz konusu olamayacaktır. Çünkü her insan ufakta olsa bir hata yapacaktır. Ancak bilgisayarlar asla! İşte bu nedenden dolayı, bilgisayarın kullandığı otomobil, her zaman birinci olacaktır. Bu durumda, oyunu satın alan ve oynayan kişiyi sürekli kızdıracaktır. Bu nedenle, rasgele davranışlar kullanılır ve bilgisayarın biraz hata yapması sağlanmaya çalışılır. Şöyleki; varsayalım bir viraja giriliyor ve bilgisayar bu viraj sonunda kullandığı otomobilin iyi bir viraj dönüşü yapacağı veya yapamayacağı kararını verecek. Bu durumda, 1 ile 100 arasında rasgele bir sayı oluşturuyor, sonra bu sayıyı kullanarak şöyle bir ifadeyi işliyor: eğer, rasgele üretilen sayı 30’dan küçükse, o zaman bu virajdan iyi bir dönüş yapma (yoldan çık), eğer 30’a eşit veya büyükse, o zaman güzel bir dönüş yap ve devam et...

İşte bu şekilde, %30 hata toleransı verebileceğiniz bir oyun üretmiş oluyorsunuz. Bazı oyunlarda, zorluk derecesi vardır: Zor / Kolay / Normal.. İşte bu seçenek, rasgele hata yapma oranını belirler. Kolay bir oyunda, hata yapma oranı %40’a çıkabilir zorda ise %20’ye kadar düşer.

Peki, bilgisayar rasgele sayıyı nasıl üretiyor. Aslında rasgele sayı diye bir terim yoktur. Bilgisayar, tarih – saat – cputime gibi sürekli değişen bilgileri kullanarak rasgele sayı üretir. CPU-Time, açıldığından beri o âna kadar, işlemcinin kaç işlem geliştirdiğini ifade eden bir sayıdır. Saniyede yaklaşık olarak 2 milyon adet artabilir. Her seferinde çok farklı bir sayı verir. Aynı şekilde, o andaki salise değeri de kullanılabilir.

Kaldı ki, en basit ve tamamen rasgele olduğunu düşündüğümüz bir olay bile, rasgele olamıyorsa, diğer bütün olayların ve durumların rasgele olması durumu söz konusu olamaz.

Herşeyin ama herşeyin, üst üste konulan taşlar olduğunu düşünelim, ve bir tanesi bile eksik olursa yaptığınız kulenin yıkılacağını düşünelim. Böyle bir durumda herşey denge üzerine kurulmuştur ve her taşın bir anlamı vardır.

Dolayısıyla, yaşadığınız her olay, yaptığınız her yanlış, karşılaştığınız en kötü – yararsız kişi, çalıştığınız her iş, gerçekleştirdiğiniz en aptalca işlemler bile ne tesadüftür, ne de gereksiz. Hepsi yaşamınızın hatta başkalarının yaşamında birer parçasıdır.

Bilgisayar insanların söylediklerini nasıl anlar?

Bilgisayar, insanların ifadelerini, bir aracı sayesinde anlayabilir. Aynı dili bilmeyen (konuşmayan) iki insanı düşünelim, bu insanların konuşması için arada, bu iki dili de anlayabilen ve birbirine çevirebilen bir tercümanın olması gerekmektedir. Gerçek hayattaki tercümanlar bir konuşma dilindeki metni başka bir dile çevirir. Bilgisayarda ise işler biraz karışıktır. Çünkü, bilgisayar bir konuşma dili kullanmaz, programlama dili kullanır. Bu tercümana derleyici (compiler) denilir. Derleyici ile ilgili ayrıntılı bilgiler ileriki bölümlerde verilecektir.

Peki programlar neleri yapamaz?

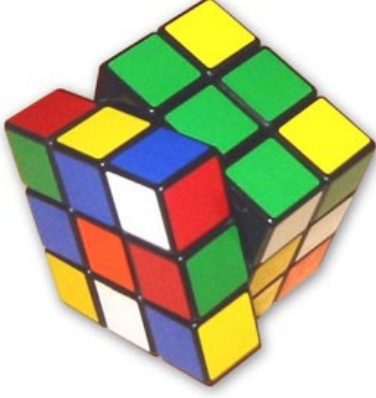
Lisedeyken bir arkadaşımın programlama konusunda yaptığı bir espiriyi anlatayım. Bana, “Emre, karakollar için bir program yap, üst tarafta suçlunun ismi ve bilgileri olsun, alt tarafta iki buton olsun: Yakala ve Bırak. Yakalaya bastığında, gidip suçluyu yakalasın, bırak butonuna basınca da suçluyu serbest bıraksın.”

Keşke hayat bu kadar kolay olsaydı.. Bu program yapılabilinse, karakollar büyük bir dertten kurtulurdu..

Tabiki, bu örnekteki gibi, programların gerçek olayları gerçekleştirmesi mümkün değildir. Programlardan bu şekilde işlevleri gerçekleştirmesini beklemek zaten komik olacaktır. Ancak bilgisayar programları, bu işlevlerin yapıldığına dair kayıtları tutmak üzere tasarlanabilir. Yani, bir polis memuru, bir suçluyu yakaladığında, programa, o suçlunun yakalandığına ilişkin kayıt girişini gerçekleştirirse, o suçlunun, durumunu ve bilgilerini takip edebilir. Bununla birlikte teknolojinin sınırı olmadığından, isterseniz, nezarethane kapısını bilgisayarla otomatik açan bir sistem kurarak bu sistemin kontrolünü programdan yapabilirsiniz.

Sonuç olarak, bilgisayarlar öğrettiğinizden yada programladığınızdan ilerisini yapamazlar.

Her bilimin bir kökü vardır ve ayrıca her bilimin kök biliminin kökü felsefedir. Bilgisayar biliminin kökü, Matematiğe dayanmaktadır. Matematik ve Bilgisayar Bilimi, dolayısıyla programlama iç içedir.



Figür 1.3.3 B : Zeka küpü oyuncacı

Yukarıdaki zeka küpünü ele alalım. Nereye çevirirsek çevirelim, bize içinde olmayan bir renk göstermeyecektir, içinde olmayan bir desen ortaya çıkarmayacaktır. İşte programlarında, her ne kadar değiştirsek, aslında yaptığı, elektriksel verileri işlemektir.

Matematikte temelde dört işlem vardır (artı, eksi, çarpı, bölü). Bilgisayarda temelde 4+1 işlem vardır. Veri ekleme, veri silme, veri düzenleme, veri görüntüleme, veri işleme. Dikkat ederseniz içinde veri olmayan neredeyse hiç bir program yoktur. Çünkü, başka türlü, bilgisayarlar bir amaca hizmet etmiş olmaz.

İnsanların beyni de aynı sistemle çalışmaktadır. Dış ortamdaki gelişen olaylara göre karar alırız ve alınan bu karara göre hareket ederiz. İlk bölümde belirttiğimiz gibi dış dünyada oluşan problemler, beynimiz için birer girdidir ve beynimiz sonuç olarak, algoritma kurup bu kurduğu algoritmayı çıktı olarak dışarı verir.

Ağırlıklı olarak şirketlerin kullandığı programlarda veri takibi ve veri işleme yapılır. Önemli olan neyin nerede olduğu, ne zaman kimin tarafından işlem gördüğüdür. Yani şirketlerde genelde, yapılan işlemler, ve o işlemlerin ayrıntılarını tutmak, görmek, işlemek amacıyla programlar kullanılır. Veri işleme ise, genelde, (şirketler için) istatistiki iş takip veya istatistiksel işlemler için ya da, belirli bir girdiye göre, bir çıktı alabilmek için kullanılmaktadır. Tabi ki, “genelde” kelimesinden de anlaşılacağı gibi, takip ve işleme haricinde işlem yapan programlarda mevcuttur.

Binlerce farklı tipte program bulunmaktadır. Örneğin, bir otomobili süren, bir makineye çalışma emri veren, dosya sıkıştıran, cep telefonunuzdan wap’a bağlanmanızı sağlayan...

1.3.4 Programlanabilme – Programlayabilme

Programlanabilme Bir sistemin, bir işlemin, bir sürecin, bilgisayar ortamına aktarılıp, sistemin fonksiyonlarının işlenmesinin, sistemin iş akışının gerçekleşmesinin sağlanabilmesine verilen addır. Daha açık olarak, bir sistemin sahip olduğu özellikleri, olayları, işlemleri, metodları bilgisayar ortamında ifade edebilmektir.

Programlayabilme Programlanabilen bir sistemin, bir kişi veya ekip tarafından bilgisayara aktarılacak, mantık, bilgi ve deneyime, programın yazılacağı editöre

- derleyiciye ve donanıma sahip olmasına verilen addır. Yani, programlama yapabilecek kapasiteye sahip olmaktır.

Neler programlanabilir?

Bir akış diagramı, pseudo kodu veya bir şema olarak gösterilebilen her sistem, programlanabilir. Yani açıkça, tasarımı yapılabilen her sistem, bilgisayar ortamına aktarılabilir. Nasıl projesi çizilebilen (kurallarına göre) her bina inşa edilebiliyorsa, mantığı – şeması oluşturulan bir sistemde, bilgisayar programı olarak ifade edilebilir (belirtili sistem, bilgisayar programı halinde yazılabilir).

Kimler programlayabilir?

Bu sorunun cevabı aslında kimler algoritma kurabilir sorusunun cevabı ile aynıdır. Yeterli; bilgiye - ilgiye, araştıma yetisine, analiz yapma yeteneğine sahip olan, ve bu kitabı okuyan hemen hemen herkes programlayabilir.

Ben, sizin, programlama yapabildiğinizi, keşfetmenizi sağlayacağım yada en azından çalışacağım.

1.3.5 Programlama için gerekenler

Programlayabilmek için, öncelikle bir sistemin programlanabilir olması gerekmektedir. Daha sonra, sistemin çalışma ilkelerini, iş akış diagramını, özelliklerini, metodlarını ve olaylarını bilmemiz, anlamamız, analiz etmemiz gerekmektedir. Eğer bu işlemleri yapabiliyorsak, programlama yapabiliyoruz demektir.

Özet olarak programlayabilmek için;

- 1 – Programlanabilir bir sistem
- 2 – Sistemin çalışma prensibi (anlama ile)
- 3 – Sistemin çalışma ilkeleri, iş akış diagramı
- 4 – Özellikleri, olayları, metodları, ayrıntıları...

gerekmektedir. Tabiki bunun haricinde programı yazacak bir kişi veya ekip, programın yazılacağı bir ortamın bulunması gerekmektedir.

[Ara verme bölümü]

1.3.6 Programlama nasıl mümkündür?

Programlamanın nasıl mümkün olabileceğini anlatmadan önce aslında hepimizin bildiği ancak tanımlamakta zorluk çektiği bir terimi hatırlamalıyız.

Sistem Sistem veya düzenek, birbiriyle etkileşen veya ilişkili olan, bir bütün oluşturan cisim veya varlıkların, ki bunlar soyut veya somut olabilirler, bileşkesidir **[e9]**

Günlük hayatımızda sistem diye adlandırabileceğimiz bir çok kavram bulunmaktadır. Sistem, birbirleriyle bağlantılı olmayan ancak birbirleriyle

etkileşen ve koordineli bir şekilde çalışan soyut veya somut varlıklardan oluşan bir bileşkedir. Örneğin, trafik lambaları, birbirlerinden ayrıdır, ancak birbirleriyle ilişkili olarak çalışmaktadırlar aksi takdirde, aynı anda yeşil ışık yanabilir ve bunun sonunda, trafik kazaları oluşabilir.

Hemen hemen bütün bilim dalları doğa da bulunmaktadır (matematik hariç), yani doğayı incelemek için, doğadan keşfedilmişlerdir. Örneğin fizik: doğada bulunan maddeyi inceler, biyoloji: doğada bulunan canlıları inceler, jeoloji: dünyanın katı maddesinin, içeriğinin, yapısının, fiziksel özelliklerinin, tarihinin ve onu şekillendiren süreçlerin incelenmesini içerir... Yani bilim dalları, genellikle, doğadan esinlenerek keşfedilmişlerdir. Örneğin makine mühendisliği, insandan esinlenerek oluşturulmuştur. İnsan kol ve bacakları, makine mühendisliğinde, pistonlara, insan kalbi pompaya, karaciğer bir tür aküye, gözler kameraya, kulaklar mikrofona denk gelir diyebiliriz.

Dikkat ederseniz, gerçekte varolanlardan esinlenerek, yeni keşifler yapılır. İşte bilgisayar dünyasındaki hemen hemen her sistem de, gerçek dünyadakilerden esinlenerek yapılmıştır.

Bu konuya daha derin girmeden önce, iki farklı **dünya** olduğunu varsayalım. İlk dünyayı, şuan içinde bulunduğumuz dünya, ikincisini de bilgisayar ortamı içindeki sanal dünya olarak ifade edelim. Bilgisayar dünyası içinde olan bir nesne gerçek değildir. Yani bilgisayar dünyasında - bilgisayar içinde oluşturduğumuz dosyayı elle tutamayız, görünür veya dokunulabilir bir nesne değildir, yapaydır. Bir işlem yapmanın ücreti yoktur. İstedığımız kadar dosya oluşturabilir veya işlem gerçekleştirebiliriz ve olaylar gerçeğine göre çok hızlı gelişmektedir.

Gerçek Dünya	Yapay Dünya Bilgisayar Dünyası
Her nesne gerçek Yoktan nesne var edilemez. Varolan nesne yok olmaz.	Her nesne sanal – yapaydır. Yoktan nesne var edilebilir veya var olan nesne yok edilebilir
Her nesnenin bir ücreti vardır. İşlemler uzun sürebilir	Her nesne – işlem – olay ücretsizdir. İşlemler hızlı gerçekleşir

Figür 1.3.6 A: Gerçek dünya ile bilgisayar dünyasının karşılaştırılması

Programlama, gerçek dünyadaki bir sistem, bilgisayar içine transfer edilerek yani, gerçek dünyadaki bir olayın, (bilgisayar içinde) aynı işleyişine sahip bir sistemin oluşturulması ile mümkündür. Yani, dış ortamda olan bir ‘kavramı, sistemi’, yapay ortamda (bilgisayar içinde), oluşturarak programlama yapabiliriz. Yaptığımız bu işlem bir tür “kopyalama” dır, örneğin bir tüp bebek gibi... Bunlar haricinde, kopya tabloları(resim tabloları), kopya ürünleri, kopya cd ve kasetleri düşünebiliriz. Bir anahtarcıda anahtarın kopyalanmasını gördüyseniz, öncelikle orjinele benzer bir kopya anahtar makineye yerleştirilir, sonrasında orjinalinden bakılarak kopyası makine tarafından otomatik olarak oluşturulur.

Tabi ki bilgisayar ortamı içinde oluşturuyor olduğumuz ‘sistem’ sanal yani yapay olduğu için, hem değişiklikler hemde çoğaltma çok kolay bir şekilde yapılabilir. Ayrıca,

inşa ettiğiniz sistem, herhangi bir ‘gerçek’ materyal harcamadığı için, masrafı bulunmamaktadır. Tek masrafı emek yani **zamandır**.

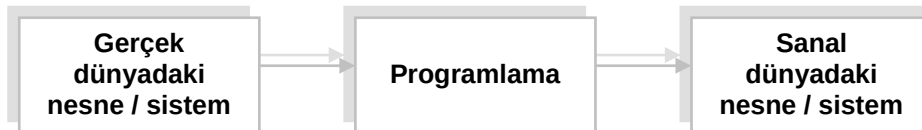
Öncelikle, bir “boş” ortam olduğunu (gerçek dünyada : evren, bina, oda veya çalışma ortamınızın olacağı konum) düşünelim. Bu boş ortama gerçek dünyadan programlayacağımız bir sistemin, o sistemin gerçek dünyadaki özelliklerini – işlemlerini – olaylarını sanal ortama aktararak, sanal bir kopyasını oluştururuz.

Nasıl bir ressam, bir kişinin portresini çizerken, önce, boş bir kağıt alıyor (yada tuval), daha sonra portresini çizeceği kişiye bakarak, gerekli ayrıntıları, kağıda aktarıyorsa programlama da aynı şekilde gerçek dünyadan, gereken ayrıntıların, sanal ortama (yani bu örnekteki kağıda – tuvale) aktarılmasıyla mümkün olmaktadır. Dikkat etmemiz gereken nokta, ressamın, portre yapmak için, portresini yapacağı insanı, kağıdın veya tuvalin içine sokmadığıdır! Sadece, önemli – gerekli ayrıntılarına, gerçekteki nesne olan kişiden yararlanarak (referans alarak) çizmektedir.

Gerekli ayrıntıların önemsenmesi: Bildiğimiz gibi portrelerde, göğüs seviyesinden aşağısı olmaz (çizilmez). İşte bu nedenle, kişinin ne renk pantolon giydiği, veya pantolonun çizimi, ressam için önemli olmayacaktır.

Gerçek dünyadaki nesnelerin ve sistemlerin TRİLYARLARCA ayrıntısı bulunmaktadır. Ancak, bu ayrıntıların, programlamada sadece, 100-200 tanesini bazen daha fazlasını, bazende daha azını, ancak, gerçekte bulunan özelliklere göre, çok düşük orandaki bir kısmını kullanırız. Örneğin, ressam için, portresini yaptığı kişinin, gözünün retina kodunun (retina kodu: göz içinde sadece özel optik cihazlar tarafından okunabilen ve her insanda farklı olan – parmak izi gibi- bir tür koddur) önemi bulunmamaktadır.

Gerçek dünyadaki sistemlerin ve nesnelerin bütün özelliklerini önemse ve programlasaydık, o zaman, evrenin bütün özelliklerini baştan tasarlamamız gerekirdi, yani herhangi bir sistemi programlamamız mümkün olmazdı.



Figür 1.3.6 B : Programlama sistemi

Not: Aşağıdaki örnekler, yukarıda anlatılmış kavramı pekiştirmek için kullanılmıştır. Her bir örneğin amacı aynıdır. Anlatımla birlikte, pekiştirmeninde iyi olabilmesi için, aşağıda bol sayıda örnek verilmiştir.

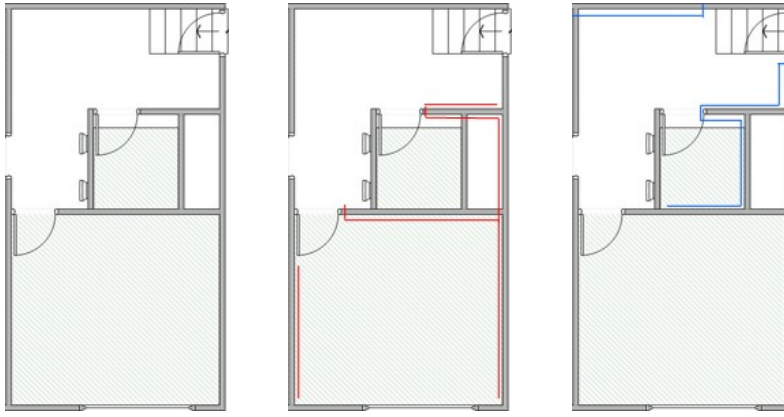
Bir örnek olarak; bir madeni parayı düşünelim. Çocukken hepimiz bir madeni parayı kağıdın altına yerleştirip, üstünden kurşun kalemle, paranın kalıbını çıkarmışızdır. Aşağıdaki resimde dikkat etmemiz gereken, soldaki madeni paranın, renginin ve bütün ayrıntılarının olduğu, 3 boyutlu olduğu hangi maddeden yapıldığı açıkça bellidir. Ancak yapmış olduğumuz karalamada, ne renk, ne hangi maddeden yapıldığı, ne tam şekli ve ayrıntıları belli değildir. Dikkat ettiysek, gerçek dünyadaki nesnelerin veya sistemlerin özelliklerinden sadece bir kısmını (işimize yarar kısmını) alıp kullanıyoruz.



Figür 1.3.6 C : Madeni para ve kağıt altından taranmış hali

Kargas hali bitmiş bir inşaatı düşünelim. Bundan sonraki evrede, su tesisatı, elektrik tesisatı yapılacaktır. Bunun için, inşaatı gelen bir elektrik tesisatçısı, binada, kolonların nereden gittiğini, ne kadar demir kullanıldığını, kapı ve pencere sayısını önemsemeyecektir. Elektrik tesisatçısının önemseyecekleri: buvatların nerede bulunduğu, odalarda kaç priz, kaç lamba bulunacağı, telefon ve uydu bağlantılarının olup olmayacağıdır.

Su tesisatçısı AYNI binaya su tesisatı yapcak olsa bile, yapacağı plan değişik olacaktır. Çünkü, su tesisatçısı, su tesisatının nereden geçeceğini, kolonları, banyo ve mutfakın nerede olduğunu önemseyerek belirleyecektir.

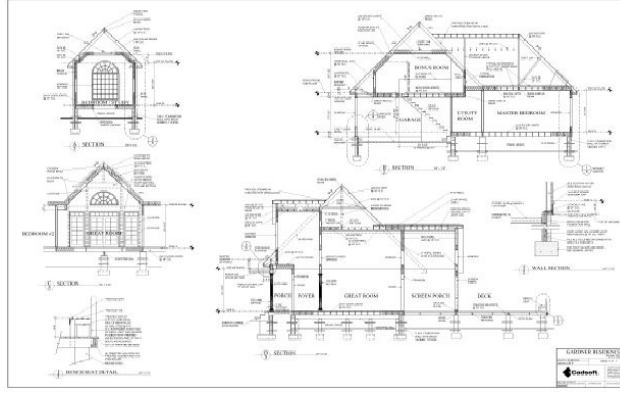


Figür 1.3.6 D : Bir evin planı ve tesisatları

Yukarıdaki ilk resim binanın bir katının bir bölümünü göstermektedir. İkinci resimde, elektrik planı, son resimde de su tesisat planı gösterilmiştir.

→ Varsayalımki, bir inşaat firmasında çalışan bir teknik ressam bulunuyor. Bu inşaat şirketinin, genel müdürünün, şehri gezerken bir binayı gördüğünü, ve çok beğendiğini varsayıyoruz. Daha sonra teknik ressama, bu binanın aynısından yapmak için, binanın projesini çizmesini istiyor.

Yani, teknik ressamın, binanın projesini çizmesi gerekmektedir. Peki bu nasıl mümkün? **Teknik ressam, gördüğü binayı, kağıdın içine yerleştiremez!** Bunun yerine, eline bir kağıt alır ve o binanın önemli ayrıntılarını (BELİRLİ ORANDA KÜÇÜLTEREK) çizmeye başlar. Eğer belirli oranda küçültmezse, binanın cephe yüzeyleri kadar kağıt kullanması gerekmektedir. Bu hepimizin bildiği gibi saçma + masraflı olacaktır. Plan, bir sistemin, özetidir. Kendisi kadar bir büyük özet olsa, zaten gerçeği gibi olacaktır.



Figür 1.3.6 E : Bir binanın teknik çizim örneği

Bu anlatılanlardan çıkarabileceklerimiz;

- (A) Teknik ressamın yaptığı çizim sanaldır. Çizim içinde gerçek bir bina yoktur. Dolayısıyla bütün programlar, projeler ve planlar , **SANAL**'dır.
- (B) Gerçek dünyadaki herhangi bir nesne – sistem, sanal ortama direkt olarak aktarılamaz. Belirttiğimiz gibi, binayı alıp kağıt içine sokmamız durumu söz konusu olamayacaktır.
- (C) İşte o yüzden, sanal ortamda; gerçek ortamdakine benzeyen, yeni baştan başlayarak çizdiğimiz bir proje geliştiririz. Bu projeyi geliştirirken, referans olarak “gerçek” dünyadaki nesne olan binayı alırız.
- (D) Dikkat edilmesi gereken nokta, binanın bütün ayrıntılarının dikkate alınmamasıdır. Eğer biz bina içindeki çöp tenekelerini, binanın çatısındaki antenleri ve binayı inşa etmek için gerekli olmayan ayrıntıları da çizim içine katacak olursak, bu çizim yaklaşık 20-30 senede bitecektir.
- (E) Yapılan bu plan, gerçek binanın özetidir. İstenilen konu üzerine yoğunlaşmıştır.

Sonuç olarak diyebiliriz ki;

Programlamanın uygulanmasında:

- **Sanal, boş bir ortam oluşturulur.**
- **Gerçek dünyadaki sistemi veya nesneyi referans alarak, sadece hedef işlemde kullanılacak ayrıntıları göz önünde bulundurularak, gerçek dünyadaki nesne veya sistemin yapay olanı, sanal ortam içinde baştan inşa edilir.**

→ Başka bir örnek vereyim. Varsayalım ki, bir makine mühendisisiniz ve çalıştığınız firmanın müdürü, sizi, yeni geliştirilen makineleri görmek amacıyla, bir fuara gönderdi. Fuarda gördüğünüz bir makinenin aynısını yapmak için, o makinenin neler yaptığını not almamız, şeklinin nasıl olduğunu çizmemiz gerekir.

Yine öncelikle, boş bir kağıda, bu makinenin çizimini yaparız. Daha sonra bir not defterine, makinenin özelliklerini, işlevlerini yani gerçekleştirdiği işlemleri yazarız.

Tabiki, (D) bölümünde bahsettiğimiz gibi, bir sistemin – nesnenin planı – programı yapılırken, o program, daha sonra ne amaçla kullanılacaksa, o konu ile ilgili ayrıntıları göz önünde tutulur. Örneğin, bu konuda, makinenin paneli üzerindeki düğmelerin hangi sırada olduğunun veya renklerinin ne olduğunun önemi yoktur.

→ Başka bir örnek vereyim. Varsayalım ki bir anketörüz, bizden, değişik kişilerle anket yapmamızı isteyerek, bu kişilerin hangi siyasi partiyi desteklediğini saptamamamız istendi. Anketlerin amacı, istatistik belirlemektir. Anket formunda, İsim, Soyisim, Yaş, Meslek gibi bilgilerden sonra, desteklediği siyasi parti bilgileri bulunacaktır. Ancak, yine, anketi yaptığımız kişi ile ilgili olan bilgilerden ikisi olan “kan grubu”, “tuttuğu futbol takımı” gibi özellikler ankette olmayacaktır. Çünkü, bu özellikler, belirtilen konu kapsamında değildir. Yani, kan grubu 0 rh – olanlar, X partisini, AB rh + olanlar, Y partisini destekliyor şeklinde bir istatistik çıkaramayız. İşte bu yüzden sadece konu ile alakası olan bilgileri toplamalıyız. Yaş ve meslek özellikleriyle istatistiki sonuçlara ulaşmanız mümkündür. Örneğin; genç kesim (18-25), X partisini, orta yaştaki kesim (25-35) Y partisini destekliyor, veya üniversite mezunları C partisini, mühendisler D partisini destekliyor şeklinde sonuçlara ulaşabilirsiniz. Peki, diyebilirsiniz ki, isim ve soyisim de, ankette bir istatistiği etkilemeyen özelliklerdendir ancak, anket formunda bulunmaktadır, **neden?**

Bu sorunun cevabı çok basittir. İstersek, isim ve soyisim özelliklerini kaldırabiliriz. Çünkü, bu özellikler (hatta bazen telefon adres bilgileri) sadece anketörün, formları kendi başına mı doldurduğu yoksa gerçekten kişilerle mi anket yaptığını belirtmek için kayıt edilir. Daha sonra, bu kişilerden rasgele 1-2 kişi seçilip aranarak, böyle bir ankete katılıp katılmadığı sorularak anketörün doğru söyleyip söylemediği doğrulanmış olunur.

Her planın – projenin – programın bir ismi yada tanımlayıcısı bulunmaktadır. Bu tanımlayıcı olmadan projenin – planın veya programın bir anlamı olmayacaktır. Hepimiz, üstünde sadece numara yazılı kağıtları biliriz. O an için birisinin telefon numarasını yazmışızdır ancak isim yazmayı o an gerekli görmeyiz. Daha sonra, aradan belirli bir vakit geçince, o kağıda yazdığımız numaranın kime ait olduğunu unuturuz ve o bilgi artık anlamsız hale gelir.

Varsayalım ki iki farklı firma fabrikalarında ürünler üretiyor. Ürettikleri ürünlerin isimlerinin hepsi birbirinden farklı olsun. Bu firmalar farklı isimde ürün üretiyor olsa bile, aynı ürünü üretiyor olabilirler. İşte bu durumda, fabrikalarda üretilen ürünlerin aynı olup olmadığını anlamak için, üretilen ürünlerin özelliklerinin aynı olup olmadığını, ürünlerin aynı işlevleri gerçekleştirip gerçekleştirmediğini kontrol ederiz. Sonuç olarak, ürünlere verilen isimler önemli değildir. Önemli olan yaptıkları işlevlerdir.

Yavaş yavaş gerçek dünyada olan bir sistemin, bilgisayar ile ilişkilendirilmiş haline geçiyoruz.

→ Bu konuya bir örnek olarak;

Gerçek dünyada : Bir defter; defter içinde sayfalar, sayfalar içinde yazılar bulunur, ve yazılar karakterlerden meydana gelir. Bir defter, içinde veri saklamaya yarayan bir tür dökümandır.

Yukarıdaki durumu ele alırsak, bilgisayar dünyasındaki bir “kavram” ile bir defterin ayrıntılarını inceleyelim.

★ Bir deftere uygulanabilecek işlemler :

- 1 Bir deftere yazı yazılır
- 2 Bir defterdeki yazı silinir
- 3 Bir defterdeki yazı silinip-değiştirilir

- 4 Bir defterdeki yazı okunur
- 5 Bir deftere yeni sayfa eklenir
- 6 Bir defter çöpe atılabilir

.....

★ **Bir defterin özellikleri :**

- 1 (Tanımlayıcı) Bir defterin isim özelliği (Matematik Defteri – Fizik Defteri) bulunur.
- 2 (Boyut) Bir defterin boyutu (sayfa sayısı – orta sayısı) bulunur.
3. (Tür) Bir deftere yazılabilecek – çizilebilecek verileri kapsayan küme (örneğin çizgili bir deftere teknik resim çizilemez, aynı zamanda kareli bir deftere kompozisyon yazılamaz) bulunur.

Bilgisayar dünyasında, bir **Word** dökümanı; dosya içinde sayfalar, sayfalar içinde yazılar bulunur, yazılar karakterlerden meydana gelir.

★ **Bir Word dökümanına uygulanabilecek işlemler:**

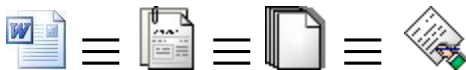
- 1 Word dökümanını açıp yazı yazma
- 2 Word dökümanını açıp yazı silme
- 3 Word dökümanını açıp yazı değiştirme
- 4 Word dökümanını açıp yazı okuma
- 5 Word dökümanını açıp CTRL+Enter tuşlarına basma – Yeni sayfa
- 6 Word dökümanını silme – Çöp Kutusuna atma

....

★ **Bir Word dökümanının özellikleri :**

- 1 (Tanımlayıcı) Bir dökümanın isim özelliği (.doc uzantılı dosya) bulunur.
- 2 (Boyut) Bir dökümanın boyutu (kaç bytelık alan işgal ettiği) bulunur.
3. (Tür) Bir dosyaya yazılabilecek – çizilebilecek verileri kapsayan küme (örneğin çizgili bir word dökümanına yazı yazılır, AutoCad te, teknik resim çizilir) bulunur.

Görüldüğü üzere, gerçek dünyadaki bir defter, bilgisayar dünyasındaki bir word dökümanı ile eşleşebilir (word dökümanı bir dosyadır). Aynı şekilde, bu word dökümanını yazıcıdan çıkardığımızda bir dosya olacaktır. Tabi bahsettiğimiz gibi her elemanın, sistemin, yapının bilgisayarda eşleşen bir benzeri olmayabilir. Bunun için kendi sınıfımızı tanımlamalıyız. Ayrıntıları ileriki bölümlerde göreceğiz.



Figür 1.3.6 F : Aynı amaca hizmet eden çeşitli dosyaların birbirlerine denk olduğunun gösterilmesi

Aynı işlevi gerçekleştiren farklı SANAL veya GERÇEK nesneler, birbirlerine mantıksal olarak denktir.

Konuya devam etmeden önce Tümevarım ve Tümden gelim kavramlarını açıklayayım.

Tümdengelim; genelden özele ya da yasalardan olaylara geçiş şeklindeki akıl yürütmedir. Örn: “Su bir akışkandır.”, “Akışkanlar hareket ettikçe basıncı düşer. O halde, suyu hareket ettirdikçe basıncı düşer” çıkarımı, tümdengelim türü bir akıl yürütmedir.

Tümevarım; özelden genele ya da olaylardan yasalara geçiş şeklindeki, akıl yürütmedir. Ör: “Gözlediğim bütün filler maviydi. O halde bütün filler mavidir” çıkarımı, tümevarım şeklinde bir çıkarmadır. Özel gözlemlerden genel bir sonuca ulaşılmıştır. [e10]

(Tümdengelim) Dikkat ettiysek -Örn: “Su bir akışkandır.”, “Akışkanlar hareket ettikçe basıncı düşer.”, “O halde, suyu hareket ettirdikçe basıncı düşer”-örneğinde, verilen temel iki bilgiden, başka bir alt bilgi elde ederiz. İlk iki ifadeye baktığımızda, iki tane kesin hüküm görürüz : “akışkandır” ve “düşer”. Bu iki kesin olarak bilinen bilgiyi kullanarak başka bir bilgi oluştururuz ve bu oluşturduğumuz bilgi kesindir.

(Tümevarım) “Gözlediğim bütün filler maviydi” örneğine baktığımızda, bir kesinliğin olmadığını anlayabiliriz. “Gözlediğim” kelimesinden de anlaşılacağı gibi, şahıs, bütün filleri gözlemiş olamayacaktır. Burada elde edilen sonuç bir teori olacaktır. Yani bilimsel gerçek olarak kabul edilse bile, daha sonradan yanlış olduğu (daha fazla fil gözlemlendiğinde sonuç değişebilir) ispatlandığında değişebilmektedir. Dikkat etmemiz gereken “Gözlediğim bütün filler maviydi” örneğinde, elde edilen bilginin birden fazla fil tarafından doğrulandığıdır yani bu yargının sadece bir fil için değil birden fazla fil için doğru olduğudur (Kural 1).

$Y = F(X)$ şeklinde bir fonksiyon ve eşitlik düşünelim. Bu fonksiyonun ne işlem gerçekleştirdiğini ve x değerini biliyorsak, yani iki kesin yargı biliyorsak, tümdengelim metodunu kullanır, fonksiyon içindeki işlemlerini x değerine uygulayarak y değerini elde ederiz.

$$Y = F(X) \quad F(X) = 3.X + 3 \quad X=2 \quad Y=?$$

Bu örnekte, F fonksiyonunun ne işlem yaptığı biliniyor, x değeri biliniyor ve y değeri soruluyor. İşlemleri uyguluyoruz ve sonuca ulaşırız.

$$Y = F(2) = 3.2 + 3 = 9$$
$$Y = 9$$

Halbuki tümevarımda, fonksiyonun ne yaptığı bilinmemekte, buna karşın x ve y değerleri bilinmektedir. Bu x ve y değerlerini kullanarak, fonksiyonun içinde ne olduğu tahmin etmeye çalışılır.

$$Y = F(X) \quad Y=3 \quad X=1$$

Böyle bir durumda bütün bilimsel teorilerin başında olduğu gibi, çok fazla sayıda aynı sonucu çıkaran fonksiyonlar bulunabilir. Örneğin;

$$Y = F(X) \quad F(X) = X + 2$$
$$F(X) = 3.X$$
$$F(X) = 7.X - 4$$
$$.....$$

Önceki paragraflarda, Kural 1 olarak nitelendirdiğim bir paragraf bulunuyor. Bu paragrafta, bir bilimsel konu üstünde genelleme yapmak için birden fazla örneğin (yada durumun) test edilmesi ve ortaya attığınız teorinin bütün bu durumlar için doğru sonuç vermesi gerektiğinden bahsetmiştim. O halde, Y ve X için sadece 1 tane değil birden fazla örnek kullanırsak, fonksiyonun içinde olabilecek işlemler varyasyonlarını azaltabiliriz. Örneğin;

$$\begin{array}{ll} Y=3 & X=1 \\ Y=6 & X=2 \\ Y=9 & X=3 \end{array}$$

Bu örneklerin hepsi göz önünde tutulduğunda, fonksiyonun;

$$F(X) = 3.X \text{ şeklinde olabileceğini söyleyebiliriz.}$$

Tümdengelim, programlama da şu şekilde ifade edilebilir:

Hüküm 1 : Elimde bir kod bulunuyor ve bu kod bir dosyayı sıkıştıran bir kod

Hüküm 2 : Programlar verileri işlerler.

Hüküm 3: Dosyalar bir tür veri içeren şablonlardır.

Bu üç bilgiyi kullanarak;

O halde, dosya sıkıştıran programdan (1), bir dosya sıkıştırmasını istersek, öyleki dosya bir tür veri içeren şablonlardır (3) ve programlarda verileri işlerler (2), program, belirtilen dosyayı sıkıştıracaktır.

Tümevarımda, programlamaya benzettiğimizde, tasarlanmış olan bir programı analiz ederiz. Yani aslında tümevarımda, bir program yazmanın ters işlemini uyguluyoruz. Elimizde, bir girdi ve çıktı bulunur. Girdi olan, ilk baştaki dosya, çıktı olan ise dosyanın sıkıştırılmış halidir.

Sıkıştırılmış dosya = İşlemler (Girdi Dosyası – Orjinal Hali)

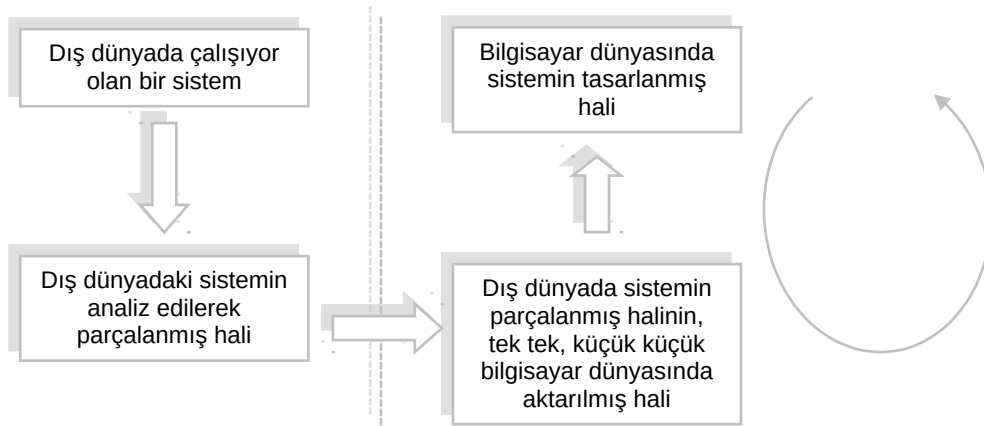
Tümevarımda olduğu gibi, girdi ve çıktıyı biliyoruz ancak işlemler kısmında neler yapıldığını bilmiyoruz ve girdi – çıktı ikililerini inceleyerek, işlemler içinde neler yapıldığını tahmin etmeye çalışırız.

Tümevarımı programlama ile ilişkilendirdiğimizde, önümüzde çalışmakta olan bir sistemin, nasıl çalıştığını tahmin etmeye çalışmak olarak ifade edebiliriz. Yani sonuçtan başlangıca gitmeye benzetebiliriz.

Bu konuları programlama içinde direkt olarak kullanmayacağız. Daha sonraki konuları anlamana kolaylık getirmesi açısından bu konuları işledik.

Şimdiye kadar öğrendiklerimizi toparlayalım. Dört evreden oluşan bir dizi işlem düşünelim. Bu evreler, gerçekteki bir sistemin programının yazılması için gerekli olan dört evreyi temsil etmektedir. Aşağıdaki diagramda sol kısım, gerçek dünyayı, sağ kısım bilgisayar

dünyasını temsil etmektedir. Amacımız, gerçek dünyada olan bir sistemi, bilgisayar dünyasına geçirmektir.

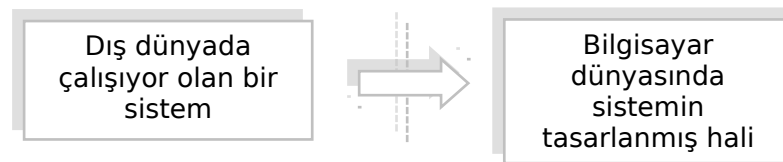


Figür 1.3.6 G : Bir sistemin bilgisayar dünyasına aktarılması

Dış dünyadaki sistemin bilgisayar içine geçebilmesi için bir tür elekten geçmesi, yani önce küçük parçalara ayrılması ve daha sonra elekten geçerek, bilgisayar ortamına aktarılması ve daha sonra yine birleştirilip bir program oluşturması gerekmektedir. Yani işlem yukarıdaki grafikte sağdaki işarete olduğu gibi saat yönünün tersine hareket etmektedir.

Programlamanın temeli, “sadece” yukarıdaki paragraftır. Bir sistemi komple tek bir bakışta programlamak mümkün değildir. O sistemi, parçalarına ayırıp irdeledikten sonra, bilgisayar içine aktarmalıyız.

Peki yukarıda verdiğimiz Tümevarım ve Tümdengelim tanımları ne işimize yarayacak? Eğer önümüzde, yapacağımız programın, daha önceden yapılmış – tasarlanmış bir örneği bulunuyorsa, böyle bir durumda bu örnekten yararlanabiliriz ve tümevarım kullanırız. Asıl amacın gerçek dünyada olan bir sistemin bilgisayar dünyasına aktarılması olduğunu söylemiştik.



Figür 1.3.6 H : Bir sistemin bilgisayar dünyasına aktarılması (2)

Varsayalım ki; X, Y, Z,... özellikleri olan bir sistemi bilgisayar dünyası içine aktaracağız. Ayrıca önümüzde daha önceden aktarılmış bir sistem örneği bulunuyor.

Gözlemlediğimiz kadarıyla, daha önce gerçekteki bir sistemdeki X özelliği, bilgisayar dünyasında A olarak ifade ediliyorsa,

Gözlemlediğimiz kadarıyla, daha önce gerçekteki bir sistemdeki Y özelliği, bilgisayar dünyasında B olarak ifade ediliyorsa,

Gözlemlediğimiz kadarıyla, daha önce gerçekteki bir sistemdeki Z özelliği, bilgisayar dünyasında C olarak ifade ediliyorsa,

.....

(bu sistemleri birleştirerek) O halde; “dış dünyadaki bir P sistemi, bilgisayar dünyası içine Q olarak ifade edilebilir” denilebilir. Burada kullandığımız ifadeler her ne kadar kesin olmasa bile(tümevarımdaki gibi), daha önceden örneğini gördüğümüz bir sistemi programlamak için, çoğu zaman yeterli olabilir.

Ancak eğer bir sistemi baştan tasarlıyorsak (ki biz kitap boyunca hep bu konu üzerinde duracağız) veya sistem kurmayı yeni öğreniyorsak, doğal olarak pratik yapmak daha doğru olacaktır. İşte bu yüzden, yukarıda bahsedilen yöntemi değilde, elekten geçirme yöntemini kullanmalıyız. Bu konu ile ilgili ayrıntılı bilgiler analiz bölümünde mevcuttur.

Örnek : Programlayabilme

Winzip programı: Winzip adlı program (veya diğer benzer programlar : Winrar, Winace, Gzip) dosyaları sıkıştırıp, boyutlarını küçültmek için kullanılan bir programdır. Bilindiği üzere, daha çok dosyaları saklarken veya internet üzerinden gönderirken kullanılır. Dosya boyutu küçültüldüğünde, gönderirken daha kısa sürede gönderileceği, saklarken daha az yer kaplayacağı için kullanılır.



Figür 1.3.6 I : Winzip ikonu

Tahminimce, **doğru hedef kitesindeyseniz**, Winzip benzeri bir programı yazamayacağınızı düşünüyor olabilirsiniz. Ancak, sizin bu yetiye sahip olduğunuzu ilerki paragraflarda ispatlayacağım.

Winzip’in ne yaptığını anlayabilmek için, sistemini ve olayı analiz etmek gerekmektedir. Bu şekilde işleyişe hakim olabiliriz. Winzip programının algoritmasını çıkartalım.

- 1 – Programlanabilir bir sistem: Bu sistem programlanabilir (Daha önce bir benzeri olduğu için hiçbir şekilde sorgulama ve araştırma ihtiyacı duymadan bu soruya ‘programlanabilir’ şeklinde cevap verebiliriz)
- 2 – Sistemin çalışma prensibi (Analiz ile) – Algoritma (İleride gösterilecektir)
- 3 – Sistemin çalışma ilkeleri, iş akış diagramı (İleride gösterilecektir)
- 4 – Özellikleri, olayları, metodları (İleride gösterilecektir)

Bir olayı analiz edebilmek; olay içindeki, bireyler arasındaki ilişkileri kurabilmek ve bir işleyiş şeması çıkartabilmektir. Analiz için, tam olarak kavranmayan (veya bilinmeyen) terimlerin anlaşılması gerekmektedir.

[Ara verme bölümü – Ancak 1.3.5 ve 1.3.6 bölümlerini tekrar gözden geçiriniz]

Alıřtırmalar

- Programlama nedir?
- Program nedir?
- Bilgisayar dünyası dışında programlama - program nedir?
- Yazılım nedir?
- Hangi konularda plan yapılır?
- Programlar ne işe yarar?
- Bilgisayar insanların söylediklerini nasıl anlar?
- Programlar neler yapamaz?
- Veri - Bilgi nedir? Aralarındaki fark nedir?
- Programlayabilme - Programlanabilme nedir?
- Programlama için gerekenler nelerdir?
- Sistem nedir?
- Programlama nasıl mümkündür?
- Tümevarım nedir? Tümdengelim nedir?

1.4 Programlama şeması

1.4.1 İşlem şeması

A - Programlanması gereken bir ‘Programlanabilir sistem’, programlamayı başlatır.

Bu madde de önemli olan, öncelikle bir problemin olması (öyleki bu problem bir sistemin programlanmasının istenmesidir) ve sistemin mutlaka programlanabilir olmasıdır (bu konudan ilk bölümde bahsetmiştik).

İlk bölümlerin özeti olarak;

Problemin olması aslında algoritma için çok önemli değildir. Basit olarak, problem olmayan bir durumu programlayamayız. Bu konu yapay zekada başlangıç noktasıdır, ve ‘düşünürüz çünkü problemleri çözmek isteriz’ ilkesinden yola çıkmıştır. Algoritma içinde, düşünme eylemini algortima kurarak bir sistemin programlanması için kullanıyoruz.

Bir sistem programlanabilir değilse, o sistemin yaptığı işlemleri yapabilecek, o sistem gibi davranabilecek bir program hazırlanamaz. Teorik olarak, hemen hemen her sistem programlanabilir. İnsanlar düşündükleri ve hayal edebildikleri birçok sistemi programlamayabilirler. Ancak programlanamayacak sistemlerde mevcuttur. Programlanabilecek en büyük sistemler yapay zeka, yapay mantık, yapay duygu sistemleridir.

B - Sistem anlaşılır.

Sistemin anlaşılma evresi en önemli iki evreden biridir. Sistemin anlaşılması için gerekli olan işlem adımları ileriki bölümlerde belirtilmiştir.

1.4.2 Analiz

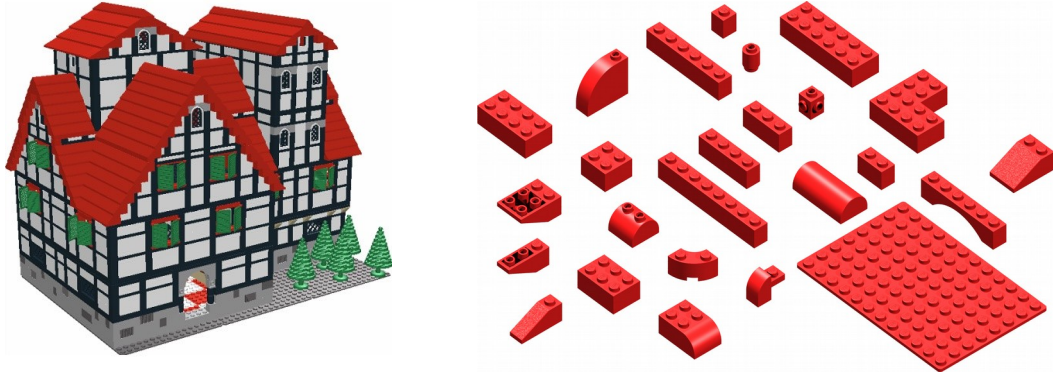
Analiz

İlk olarak sistem analiz edilir. Analizin sözlük anlamı “The ability to break down material into its component parts so that its organizational structure may be understood. This may include the identification of the parts, analysis of the relationships between parts, and recognition of the organizational principles involved [e11]” dir

Türkçesi: bir materyali, o materyalin bileşenlerine ayırarak, materyalin organizasyonel mimarisi anlaşılabilir. Bu işlem, bölümlerin tanımlanabilmesini, bölümler arasındaki ilişkilerin kurulabilmesini ve organizasyonel kuralların daha sonra kullanmak üzere hatırlanabilmesini içerebilir.

Daha sade bir dille belirtmek gerekirse, **analiz**, bir sistemi, o sistemi oluşturan parçalara ayırarak, bu parçaları incelemektir. Bu işlem içinde, parçalanmış olan parçaları daha küçük ve daha sade parçalara ayırabilme de olabilir.

Peki bir sistem nasıl analize edilir: Bir olayı analiz etmek, programlamanın da temelini oluşturur. Hemen hemen hepimiz Lego adı verilen oyuncakları bilir. Bu oyuncaklardaki parçaları birleştirerek, binalar, araçlar, evler inşa edebiliriz. Dikkat etmemiz gereken nokta; aynı parçalar kullanılsa bile, birbirinden çok farklı türde materyallerin oluşabileceğidir. Yani aynı olan hammaddeyi farklı şekilde birbiriyle birleştirerek, birbirinden çok farklı sonuçlar elde edebiliriz.



Figür 1.4.2 A: Lego örneği

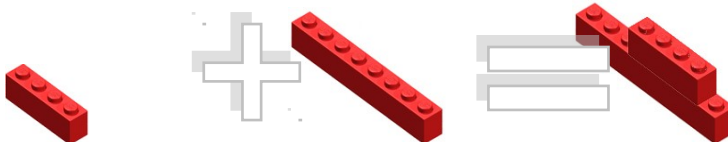
Yukarıdaki eve baktığınızda nasıl tasarlandığını, içinde neler olduğunu göremeye – anlayamayabilirsiniz. Ancak bu evi, onu oluşturan parçalara ayırırsanız, evin nasıl oluşturulduğunu çok kolay bir şekilde anlayabilirsiniz. Daha sonra dağılmış olan parçaları birleştirdiğinizde tekrar eski evi elde edersiniz.

ISO standartlarını hepimiz duymuşuzdur. International Standardisation Organisation (Uluslar arası Standartlandırma Örgütü – Organizasyonu), uluslar arası kabul görmüş standartları belirlemek için çalışmalar yapan bir örgüttür. Bu örgütün aldığı kararlar sonucunda, dünyadaki standartlar belirlenir. Örneğin, bir cd üretiyorsunuz, eğer bu ürettiğiniz cd'yi bir standarda göre üretmezseniz, cd okuyuculara (cd-rom) sığmayacaktır veya küçük gelecektir. Bir kablo, bir vida, bir conta bile standartlara göre üretilir ki, diğer parçalarla ve ürünlerle de birleştirilebilsin ve kullanılabilinsin. Yani ISO, bütün ürünler ve ürün parçaları için ortak bir platform oluşturur.

İnsanlar zor sistemleri anlamak için her zaman o sistemleri, parçalar, böler, daha önceden bildiği veya anlayabileceği hale getirir. Böylece, sistemler hem kolay bir şekilde hemde kısa sürede anlaşılmış olur.

Bir sistemi parçalarken dikkat etmemiz gereken, elde edeceğimiz parçaların kolayca anlaşılır boyutlarda olması gerektiğidir. Örneğin, dikkat ettiysek yukarıdaki figürde, kullanılan parçalar (anlaşılabilecek parçalar) sağ kısımda gösterilmiştir. Amacımız, evi ayırarak, sağda gösterilmiş olan parçalar haline getirmektir. Daha sonra tek tek bilgisayar dünyasına geçirerek, tekrar birleştirmektir.

Fonksiyonlar ve tümdengelim konularını hatırlayalım.



Figür 1.4.2 B : Lego örneği (2)

Elimizde yukarıdaki birinci parça bulunuyor, ve yukarıdaki ikinci parça ile birleştirirsek, üçüncü parçayı elde ederiz. Aynı şekilde, üçüncü parçayı, birinci ve ikinci parça olarak ayırabiliriz. Ayırma sonucunda elde edeceğimiz parçalar, belirli standartlarda olmalıdır (ISO daki gibi). Ayırma işlemi sonucunda elde edeceğimiz parçaların bilgisayar ortamına aktarılacak standartlara ulaşmış olması gerekmektedir. Yani bir sistemin parçası bilgisayar ortamında tanımlanabiliyorsa, yeterli seviyeye ulaşmış demektir.

Analiz etmek için **NE(ne veya neler)** ve **NASIL** sorularını sisteme sorarak aldığımız cevaplara göre sistemi parçalara ayırabiliriz. Nasıl ilkokuldan hatırladığımız gibi, bir cümlemin öznesini bulmak için, yükleme “kim/ne” sorusunu soruyorsak, analiz etmede de, sisteme ne ve nasıl sorularını peşpeşe bir veya birden fazla sorarız.

İlk olarak “Ne” sorusu ile başlarız ve sisteme, “bu sistem NE(LER) yapıyor?” sorusunu sorarız. Daha sonra bize;

- Sistem, A işlemini
- Sistem, B işlemini
- Sistem, C işlemini

yapıyor cevapları verilir.

Daha sonra, verilen bu cevaplar içindeki işlemlerin her birine, “peki bu işlem NASIL yapılıyor?” sorusu sorulur. Örneğin, “sistem A işlemini NASIL yapıyor?”. Bu işlemin sonrasında ise bize;

Sistem, A işlemini, önce X alt işlemini sonra Y alt işlemini sonra Z alt işlemini gerçekleştirerek yapıyor

cevabı verilir. Daha sonra bu işlemi bir NE bir NASIL sorusu sorarak devam ettiririz. Ancak bu aşamadan sonra NE sorusu sormaya gerek duymayabiliriz. Verilen cevaplara göre hep NASIL sorusunu sorarak devam edebiliriz.

→ Bu konuya bir örnek olarak;

Bir makine (fabrikalarda bulunan ürün üretim makinesi) düşünelim, bu makine; makineye gelen bir ürünü test ediyor, testten geçmeyen ürünleri geri gönderiyor, geçenleri paketliyor, seri numarası veriyor, ve çıktı olarak dışarı veriyor.

Test aşamasında, belirli testleri, yani (örneğin) dayanıklılık, sağlamlık testlerini uyguluyor. Belirtili sistemi analiz etmek için yukarıda belirttiğimiz gibi, ilk önce NE sorusunu soruyoruz.

Sistem NE yapıyor?

- Ürün testi
- Ürün paketlenme
- Ürünlere seri no yazma
- Çıktı olarak verme

‘Ne’ sorusuyla, yapılan işlemleri, yukarıdaki 4 maddeyi tespit ettik. Sistemin neleri yaptığının tespitinden sonra, bu yapılan işlemlerin nasıl yapıldığını öğrenmek için, bu sefer, bütün işlemlere **NASIL** sorusunu sorarız.

Sistemi bileşenlerine parçalayalım.
(Ürün Testi) **NASIL YAPILIYOR?**

-**Ürün Teste alınır**

- **Ürünün dayanıklılığı ölçülür** : Eğer yeterince dayanıklı değilse işlem iptal edilir, ürün defolu bölümüne gönderilir.
- **Ürünün boyu ölçülür** : Eğer belirli ölçülerde değilse işlem iptal edilir, ürün defolu bölümüne gönderilir.
- **Ürünün renk tonuna bakılır** : Eğer istenilen tonda değilse işlem iptal edilir, ürün defolu bölümüne gönderilir.

-**Eğer ürün bütün bölümlerden geçer not alırsa işlem tamamlanır ve ürün bir sonraki evreye aktarılır.**

İşte yukarıda gördüğümüz şekilde, bir işlemi, birden fazla daha küçük – basit işlemlere ayırdık. Genelde, Ne ve Nasıl soruları peşpeşe sorulur. Yani bir kere “Ne” bir kere “Nasıl” sorusu sorulur. Bu soruları sormaya “Ne” sorusundan başlarız ve bir “ne” bir “nasıl” şeklinde devam ederiz, aynı binom açılımındaki gibi...

$$(x-y)^n = + C(n,0) x^n - C(n,1) x^{n-1}y + C(n,2) x^{n-2}y^2 - \dots$$

Her bir +’da, “NE”, her bir –’de, “NASIL” sorunu sorabiliriz. Şüphesiz, işlemler bölündükçe, içlerinde gerçekleşen iş sayısı azaldıkça, anlaşılması ve planlanması dolayısıyla programlanması kolaylaşır. Einstein’ın dediği gibi, **“keep it simple, as simple as possible, but not simpler”**, yani, “bir işi olabildiğince basit şekilde yapmaya gayret et, mümkün olan en basit şekilde yapmaya çalış, ama ondan da daha basit yapma” anlamına gelmektedir. İşte bu ilke ile, bir sistemi olabildiğince basit parçalara ayırmalıyız. Tabi basit parçaları tasarladıktan sonra birleştirmeliyiz.

1.4.3 Operatör

Operatör Bir işlemi (veya alt işlemi – yani bir işi) gerçekleştiren bir alet/makine/araç/kişi(işçi) veya bir sistemdir. Bir operatör, belirtilen işlemi gerçekleştirir. Yukardaki örnekte, “Ürün teste alınır” işlemini yapan bir kişi olabilir. Yani bir işçi belirtilen ürünü kontrol aşamasına getirebilir (üretim bölümünden test bölümüne taşıyarak). Aynı şekilde, ürün renk tonunun kontrolü bir makine tarafından, ürün boyu da bir alet olan metre ile yapılabilir (Tabi bu sırada bu makineler ve aletler de bir veya birden fazla kişi tarafından yönetilebilir-kullanılabilir). Yani bir işlem birden fazla operatör tarafından yapılabilir ve bir operatör birden fazla işlem yapabilir.

İşte işlemleri gerçekleştiren varlıklara operatör denilir. Bir program içinde yazılmış olan işlemler, o işlemlerin operatörü olan “bilgisayar” tarafından yapılır. Sonuç olarak operatör, aslında, bir cümledeki, **öznedir**.

1.4.4 Lowest Limitation (en düşük limit)

Lowest Limitation Lowest Limitation (en düşük limit), algoritması kurulacak bir sistemin; işlemlerinin nereye kadar bölüneceği anlamına gelmektedir. Yukarıda bahsettiğimiz gibi, Ne ve Nasıl sorularıyla bir sistemi daha küçük ve basit parçalara bölebiliyorduk. Tabi ki bu ayırma işlemi sonsuza kadar devam etmeyecek, bir yerde sonuçlanacaktır.

Yukarıda, “Ürün Teste alınır” alt işleminin daha alt işlemlere ayrılmasına gerek bulunmamaktadır. Lowest, en düşük, limitation, limit anlamına gelmektedir. Bir işlemi o işlemi meydana getiren alt işlemlere böleriz, böleriz, böleriz.... Taa ki, en alt kademeye ulaşınca kadar.

Daha açık olarak, bir işlem, bir operatör tarafından tek başına yapılabiliniyorsa, daha fazla bölmeye gerek bulunmamaktadır. Çünkü bir noktadan sonrasını operatör eğer tek başına yapabiliyorsa zaten o işlemin ayrıntıları üstünde durmamıza gerek bulunmamaktadır.

→ Örneğin, Formula 1 yarışını düşünelim. Formula 1’de yarışmakta olan bir pilot, Pit-In’e (araçların, benzin almak, bakım yaptırmak, lastik değiştirmek için girdiği ve yarışa geçici olarak ara verdiği bölüm) girer ve daha sonra burada pit-in işlemi yapılır. Pit in işlemini, daha alt işlemlere böldüğümüzde: benzin alma, lastik değiştirme, pilota su verme alt işlemlerini elde ederiz.

Bu işlemlerin de alt işleme bölünebilenlerini bölelim. Örneğin lastik değiştirme, genelde bir ekip, lastikleri söker, diğer ekip onların yerine yeni lastikler takar. O halde, lastik değişimi, lastikleri sökme ve lastikleri takma olarak iki farklı alt işlemten meydana gelmektedir. Ancak pilotun su içme işlemi daha alt işlemlere bölünemez. Yani pilotun su içme işlemini, “pilot suyun kapağını açar”, “suyu içer” ve “suyun kapağını kapatır” şeklinde bölmeyiz. Çünkü bu işlemlerin her biri aynı operatör tarafından peşpeşe 1 kerede yapılabilir.

Benzeri durumlarda, işlemler tek bir operatör tarafından ve (çok önemli) peşpeşe yapılabiliniyorsa, o nokta bizim için en alt noktadır ve daha alt işlemlere bölünmesine gerek bulunmamaktadır.

Bir insana bir işi yapmasını tarif ettiğimizi düşünelim. Örneğin, bir işçiye, demir bir aleti (çekiç – tornavida) yapmasını söyleyelim. İşçinin bu aleti yapabilmek için gerekli bilgiye sahip olmadığını varsayalım. Böyle bir durumda: “10 cm x 40 cm lik bir demir levha (saç) hazırla, sonrasında, bu levhadan (orada bulunan bir) kağıda çizdiğim gibi bir kalıp kes, sonrasında, bu kalıba ek olarak tahtadan bir tutma sapı hazırla, tutma sapını hazırladığın demire monte et....” İşte bu şekilde, bir aletin meydana gelişi tarif edilmiş olurdu.

Burada dikkat ettiysek, işçiye direktif verirken; (ilk işlem için : 10 cm x 40 cm lik bir demir levha hazırla)

- öncelikle bir levha bul,
- sonrasında bir metre ile eni 10 cm ölçüp işaretle,
- sonra 40 cm ölçüp işaretle
- sonrasında işaretli yerlerden kes

şeklinde emirler vermedik. Çünkü, biz 10 cm x 40 cm lik bir demir levha hazırla dediğimizde, çalışan, bundan sonra o işlemin nasıl yapılacağını BİLİYOR. Bildiği bir iş için tekrardan ayrıntı vermemize gerek bulunmuyor. Aynı durum diğer işlemler içinde geçerlidir.

Lowest Limitation'a nasıl karar verilir?

Bir işlemin daha alt işlemlere parçalanıp parçalanmayacağı, operatörün kapasitesi ve yaptığı işleme göre belirlenir.

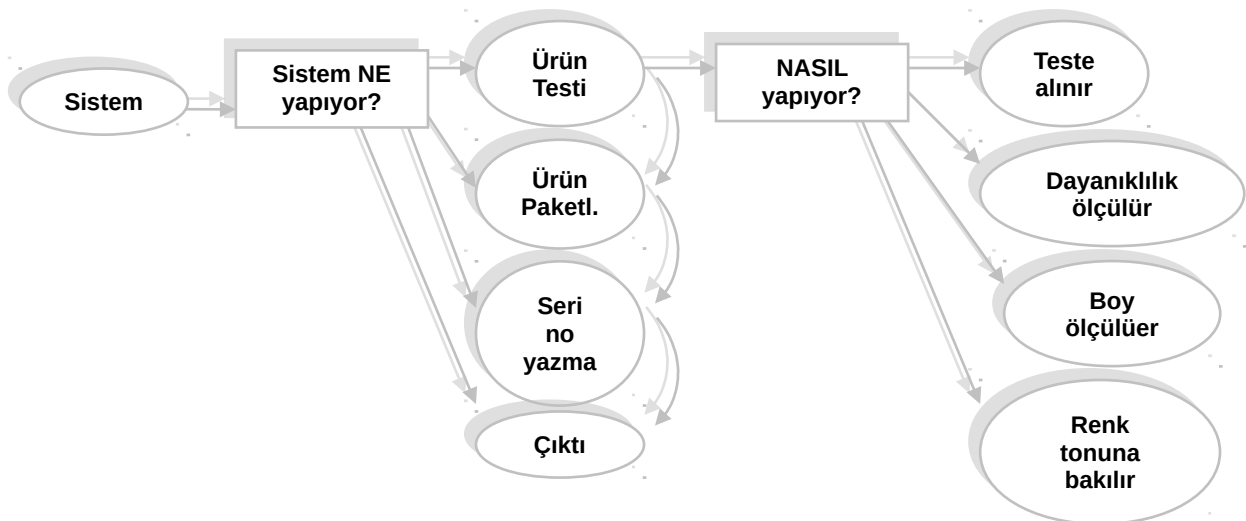
Yukarıdaki örnekte, “Ürünün renk tonu kontrol edilir” işlemi bir makine (operatör) tarafından yapılabilmektedir (öyle olduğunu varsayıyoruz – tabi ki bu işlem, algoritmasını kuracağımız sisteme göre değişmektedir.). Ancak, “Ürün paketlenme” evresi tek bir operatör tarafından yapılamıyor ise, bu işlemi daha alt bölümlere bölmek zorundayız. Çoğunlukla her bir operatör için bir işlem bulunmaktadır.

Aynı şekilde, yukardaki ikinci örnekte, su içme işlemini yapan pilot, operatördür ve başka kimseyi hiçbir şeyi kullanmaksızın, bu işlevi gerçekleştirebilmektedir. Ancak, lastik değiştirme için iki farklı ekip (sökme ve takma) çalıştığından bu işlemleri ikiye ayırdık.

Fakat bazı durumlarda bir operatör bir den fazla işlem yapabilir. Aynı makine (operatör olarak) birden fazla işlem yapabilme kapasitesine sahip olabilir. Örneğin, bir test makinesinin olduğunu ve bu test makinesinin birden fazla test işlemi yapabildiğini varsayalım. Bu makinenin, yukarıdaki saydığımız bütün test işlemlerini tek başına yapabilecek kapasitede olduğunu varsayalım. Sistemin bu bölümünü; ürün testini bir işlem olarak belirterek, geçebiliriz. Çünkü, bu işlemi yapan aynı operatördür ve bir dizi işlemi peşpeşe yapmaktadır.

Ancak kurduğumuz bu sistem yanlıştır! Çünkü, test sırasında bir testi geçemeyen, vasata giden ve orada tekrar onarılıp veya düzeltilip geri gönderilen ürün sadece, geçemediği teste girecektir. İşte bu yüzden, işlemler, aynı operatör tarafından gerçekleştirilecekte olsa, eğer farklı şekilde çağrılıyorsa farkı bir operatörmüş gibi düşünülerek ayrı bir işlem gibi çalışırlar.

Yani, bir işlem içindeki alt işlemler, aynı noktadan başlayıp aynı noktaya çıkıyorsa, o işlemi tek bir işlem gibi görebiliriz.



Figür 1.4.4 A : Bir sistemin işlemleri arasındaki ilişki

Örnekte de görüldüğü gibi, bir sistemin algoritması, sisteme NE ve NASIL soruları sorularak, işlemlere bölünerek kurulabilir.

Başka bir örnek verelim: Bir otomobil kullandığımızı düşünelim. Biz vites değiştirmek için, alt işlemler olarak;

- ayağımızı gazdan çekemiz
- debriyaja basarız
- vitesi o anki vitesten çıkarıp, istediğimiz vitese atarız
- ayağımızı debriyajdan çekemiz

yaparız. “vitesi o anki vitesten çıkarıp, istediğimiz vitese atarız” işlemi aslında biraz karmaşıktır. Bizim için kolay görünse bile içeride, vites dişlilerin yerleri değişmektedir. Ancak, bu işlemi biz bizzat yapmadığımız için, bize çok kolay gelir. Fakat, zaten önemli olan, “sürücü” yani bu işin operatörü olan bizim, sadece vitesin yerini elimizle değiştirmemizdir. Daha sonrası bizim için önemli değildir.

İleride, işlemleri bölme ile alakalı ek bilgiler bulunmaktadır. Bu bilgiler, aynı şekilde, lowest limitation’ı belirlemek içinde kullanılır.

1.4.5 Anlama

Anlama Bir olay veya önermenin daha önce bilinen bir kanunun veya formülün sonucu olduğunu görme [e12]. Yani, bir olayın rasgele değil bir formülün (fonksiyonun) sonucu olduğunu görme anlamına gelmektedir.

Anlama aslında yapay zekanın konusudur. ‘Anlama’ nın birçok çeşidi bulunmaktadır ancak biz, programlama algoritması için sadece bir tanesini yani ‘Classification’ (klasifiye etme – sınıflandırma) olanı kullanacağız.

En kolay anlama metodu benzetmedir. Benzetme yapılarak bir olayın çalışma prensibi kavranmış olunur. Daha önceden anlamış olduğumuz ifadelerden yararlanmak, anlama işlevinin temelini oluşturur.

Dünya üzerindeki milyonlarca kitabın içinde milyarlarca örnek bulunmaktadır. Örnek, benzetmenin temelidir ve daha önceden “nasıl olsa anlamış” olduğumuz ifadelerin kullanılmasını sağlar.

Bu sistem, yapay zekanın dallarından biri olan, “Machine Learning” yani **makine öğrenmesi** sisteminin temelini oluşturur. ML (Machine Learning)’de bilgisayara, daha doğrusu yapay sinir ağlarına (beynin çalışma yapsına benzer bir şekilde çalışan bir çeşit program olduğunu düşünelim) belirli girdi – çıktı ikilileri öğretilir ve bilgisayar bu girdi – çıktı çiftlerini öğrenir. Daha sonra başka durumlarda da yani başka girdilerin söz konusu olması durumunda, daha önceden öğrendiklerini kullanarak, verilen girdiye uygun bir çıktı elde etmeye çalışır.

Örnek:

Girdi	Çıktı
1	1
2	44
5	2
0	3
7	47

Figür 1.4.5 A : Örnek bir girdi çıktı tablosu

$\text{Çıktı}_1 = \text{Fonksiyon}(\text{girdi})$

ML’de amaç, bu örnekte gördüğümüz gibi, verilen girdiye göre, çıktı üreten bir fonksiyon yazmaya çalışmaktır.

(Basit bir örnek olarak) Varsayalım ki, müzik çalarımız bozuldu, bizde tamir etmek niyetiyle, içini açtık, biraz bakındıktan sonra, bir telin kopmuş olduğunu gördük, ve o teli bağladık (riskli bir işlem – doğru olduğunu bilmiyoruz) daha sonra, müzik setinin çalıştığını gördük. İşte bu şekilde, müzik setindeki bir problemin, içindeki bir telin kopmasından kaynaklanabileceğini öğrenmiş olduk. Bir süre sonra, varsayalım ki, televizyonumuz bozuldu, yine aynı şekilde tamir etmek niyetiyle içini açtık ve biraz bakındıktan sonra aklımıza, daha önce, müzik çalarda yaptığımız işlem geldi. Kopmuş bir tel aradık ve bulduk, son olarak, bu teli de bağlayıp tekrar denedik ve televizyonun çalıştığını gördük. (Bu işlem basit bir örnek olarak verilmiştir)

İşte, örnekleme ile öğrenme (Machine Learning) bu şekilde çalışır. Gelelim bu konunun programlama ile alakasına; yukarıdaki örnekte olduğu gibi, gerçek hayatta ürettiğimiz bir çözüm, programlama dünyasında da “**çoğunlukla ve benzer durumlar için**” çalışabilir. İşte bizde daha önceden bulduğumuz çözümleri, o sistemleri programlarken kullanacağız.

Uluslararası dillerdeki sözlüklerin nasıl yazıldığını veya dillerin ilk başta birbirlerine nasıl çevrildiğini düşünelim. İki farklı dili konuşan iki kişi, nesnelere veya eşyalara bakarak, bir nesnenin kendi dillerindeki söylenişi karşı tarafa söyleyerek anlaşırlar. Sözlüklerde bu şekilde oluşturulur. Önemli olan bir nesnenin ismi değil, kendisidir. Aynı şekilde bir İngilizce bilen bir Türk’ün bir İngiliz ile konuştuğunu düşünelim. Bu Türk kişi, aklında Türkçe düşünsede, kendisini İngilizce ifade etmektedir. Yani, aynı düşünceleri, İngilizce’ye denk gelen kelimelerle anlatmaktadır. İşte aynı şekilde, bir sistemi bilgisayar içinde (aynı anlama gelecek şekilde) ifade edebilecek ifadeler kullanarak bilgisayar içine yerleştirebiliriz.

Benzetme yapabilmek için, bilinmeyen terim olan “Dosya”nın tanımlanması gerekmektedir.

Dosya; içinde bilgi tutan bir tür veritabanıdır. Bilgisayar dünyasındaki, dosya, içinde (örnek olarak bir Word Dökümanında) sayfalar, sayfalar içinde yazılar bulunur.	Defter; Bir defterde, sayfalar, defter sayfalarında, yazılar bulunur. Yazılar, kelimelerden, kelimeler karakterlerden meydana gelmektedir.
---	---

kelimelerden, karakterlerden gelmektedir.	kelimeler meydana	
---	----------------------	--

O halde, bir defter, bilgisayar dünyasında dosya olarak tanımlanabilir. Tabi ki başka tip dosyalar da bulunmaktadır, diğer tip dosyalar hakkında detaylı bilgi daha ileri ki konularda verilecektir.

İşleyişe hakim olmak için, bilgisayar dünyasında yapacak olduğumuz işlemin, önce gerçek dünyada nasıl yapılacağını buluyoruz.

(Winzip gibi bir program nasıl yaparız örneğini hatırlamamızda tutalım)

Bir defteri sıkıştırmak (defterde daha az alan kullanmak) için neler yapılabilir? Lütfen okumaya devam etmeden bu işlemi gerçekleştirmek için ne/neler yapabileceğimizi düşünün. (Unutmayın ki amaç, daha az yaprak kullanmak)

• **Defter katlanabilir**, (bu yöntem sadece boyutları değiştirir, ne ağırlık ne de hacim değişmez) Yani boyut azaltma için bir anlam ifade etmez

• **Defterdeki yazılar daha küçük yazılabilir** (belirli bir standarta kadar yazılar küçültülebilir ancak bir noktadan sonra daha fazla küçültülemez. Ayrıca, biz, yazı boyutunu küçültmeden dosyayı sıkıştırmak istesek bu yöntem geçerli olmaz),

• **Kısaltmalar kullanılabilir**.

Kullanılabilecek en iyi yöntem, yazı içinde kısaltılabilecek kelimeler bulup, bu kelimeleri kısaltmalarıyla değiştirmektir. Tabi eğer defteri okuyacak kişilerin anlayamayacağı veya tahmin edemeyeceği kısaltmalar kullanıyorsak, bunları, sayfa sonunda veya defter sonunda (son sayfalarda) belirtmemiz gerekmektedir.

→ Örneğin;

0°C derecede 1 atm basınç altında deniz seviyesindeki koşula normal şartlar denir. **Normal şartlar altında** 1 mol gazın hacmi 22,4 lt dir. **Normal şartlar altında** 1 mol madde de $6,02 \cdot 10^{23}$ atom bulunur.

Bu yazıda, “normal şartlar altında” ifadesi iki kere geçmektedir ve 22 karakterli, “**normal şartlar altında**” ifadesi yerine 3 harfli “**nşa**” kısaltmasını kullanabiliriz. Öyle ki burada kullanılan **atm** ifadesi zaten kısaltmadır ve atmosfer anlamına gelmektedir.

0°C derecede 1 atm basınç altında deniz seviyesindeki koşula normal şartlar denir. **Nşa** 1 mol gazın hacmi 22,4 lt dir. **Nşa** 1 mol madde de $6,02 \cdot 10^{23}$ atom bulunur.

Böylelikle, daha az karakter, daha az satır daha az sayfa kullanılıyor ve boş sayfaları defterden attığımızda, daha az alan kaplamış olacaktır. Daha doğrusu defter içindeki yazılar daha az yer kaplamış oluyor. İşte gerçek dünyada bir dosya – defterin sıkıştırılmasının mantığı bu şekildedir.

İlkokula giden bir kardeşim (4.sınıf) var ve el yazısı çok çirkin. Yazıları okunamayacak kadar çirkin olan kişiler, yazı yazmaktan hoşlanmazlar ve mümkün olduğunca, hızlı – az yazmak isterler. Kardeşimin öğretmeni, sınıfa ilk “den den” (“”)’i getirenin kardeşim olduğunu söylemişti. Evde ödevlerine yardımcı olurken mutlaka kısaltmalar kullanır, hatta yazıyı öyle bir biçimlendirir ki, aynı kelimeler alt alta gelsin de, bir “den den” kullanabilsin.

Winzip ne yapıyor ?

Winzip, **Lempel-Ziv-Welch** adlı bir algoritmayı kullanmaktadır. Bu algoritmanın altında yatan temel fikir, bir dosya içinde işlenen verileri (okunan verileri) kullanarak olabildiğince uzun bir dizi, veri bulunması ve bu verilere birer kod verilerek, dosya içindeki bu verilen kodlarıyla değiştirilmesinden ibarettir. Tabi başka tür sıkıştırma algoritmaları da vardır. Onlarda, uzun veri dizileri bulmak yerine, çok sık geçen terimleri tespit etmektedir. Bulduğum başka bir sıkıştırma algoritmasının sitesini vereyim. Benzer bir sistem kullanılıyorlar. [e13]

Amaca, ben olsam, üçüncü yöntem olan, kısaltmaları kullanma sistemini uygulardım. Bu algoritmalara, program içinde çok sık geçen veya çok uzun olan ifadeleri, birer kod ile değiştirirler. Daha sonra aynı şekilde, UnZip (yani ziplenen – sıkıştırılan – bir dosyanın açılması – normal haline döndürülmesi) işlemi sırasında bu kısaltmalar orjinal hallerine döndürülür ve dosya bir bütün olarak sunulur.

Yani bir deftere kısaltmalar kullanarak bir metin yazıyoruz, sonra bu kısaltmaları orjinalleriyle değiştirip okuyucuya sunuyoruz. Winzip programı işte bu algorithmadan meydana gelmektedir.

Yukarıda belirtilen işlemleri düşünebiliyorsak, Winzip benzeri bir programı yazabiliriz. Bundan sonra, bir **Zip** programı yazabilmek için gerekenler sadece,

- Programı yazacağımız dilde dosya işlemlerinin nasıl yapıldığı,
- Kısaltma yapabilecek bir algoritma,
- Birkaç kısaltma kalıbıdır (veya uzun veriler bulmayamı yoksa çok kullanılan verileri bulmayamı önem vereceğimizdir).

Yani temel algoritma ve plan kurulduktan sonrası her zaman kod hamallığıdır.

Kod hamallığı Eğer bir sistemin algoritmasını çözebiliyorsak – anlayabiliyorsak, gerisinde sadece kod hamallığı kalmaktadır. Kod hamallığı, algoritması bilinen bir sistemi, seçilen bir dilde, koda dökmeye verilen addır.

→ **Başka bir örnek : Dosya Bölme**

SplitIt, File Splitter, vb programlar, dosyaları bölerek daha küçük boyutta, birden fazla parçaya ayırırlar. Böylece dosyalar, diskete veya cd ye sığacak boyuta getirilebilir. Bu şekilde, dosyaları bölen bir program yazalım.

Gerçek dünyada dosyaları bölme

→Bir defteri (yani bir dosyayı) bölmek, daha küçük parçalara ayırmak, defteri yırtmayla mümkün olabilir. Öğrenciler (üniversite de – çok defter taşımamak için) çoğu zaman defterlerini, seperator (ayraç) bir sayfa kullanarak daha küçük bölümlere bölerler. Böylece bir kısmına bir ders ile ilgili notlarını, diğer kısmına diğer bir dersle ilgili notlarını yazabilirler.

→Yine başka bir benzetme olarak, bir defter farklı kişilere verilmek üzere paylaştırılmak için ayrılabilir.

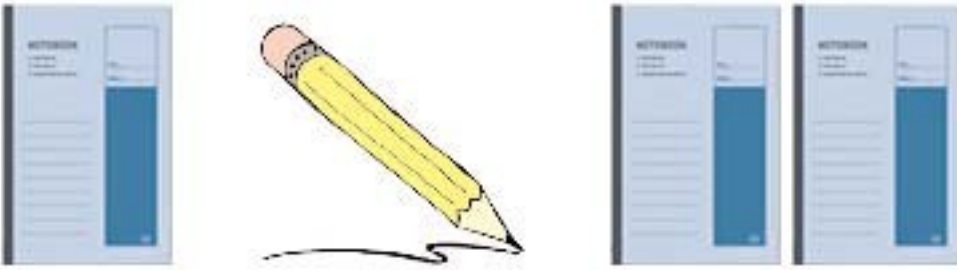
Bir defteri üçe böleceğimizi düşünelim ve böldükten sonra 3 farklı kişiye vereceğimizi düşünelim. Bunun için benim aklıma 3 yöntem geliyor.

Birinci yöntem, defteri parçalara ayırmak, gerekli ortalardan kopararak bölmek.



Figür 1.4.5 B : Bir defterin makas aracılığıyla kesilerek bölünmesi

İkinci yöntem de ise, 2 tane yeni defter alabiliriz ve orjinal defterin 2. ve 3. bölümü bu yeni iki deftere yazabiliriz. Daha sonra, bu iki yeni defteri, 2. ve 3. kişiye vererek, 1. kişide de orjinal defteri bırakabiliriz.



Figür 1.4.5 C : Yeni iki defter alınarak, ilk defterdeki bölümlerin bu defterlere yazılması

Üçüncü yöntem de ise, defterin, 2. ve 3. bölümlerini, fotokopi çekerek, 2. ve 3. kişiye verebiliriz.



Figür 1.4.5 D : Defterin gerekli bölümlerinin fotokopi çektilererek çoğaltılması

Burada dikkat etmemiz gereken bir durum söz konusudur. Nasıl karıncalar, buldukları yiyecekleri, yuvalarına tek-tek taşıyorsa, bizde, bu işlemlerin küçük parçalar halinde seri bir şekilde yapıldığını düşünmeliyiz. Örneğin, yazı yazarken, harf-harf yazılır. Fotokopi işlemi, kaynaktaki noktaların TEK TEK taranıp, hedef kağıda TEK TEK kopyalanması ile mümkün olur. Bilgisayarda da işlemler buna benzer gerçekleşir.

★ **Bilinmesi gereken bilgisayar bilgisi**

Bilgisayarda veri teşkil eden en küçük veri birimine **byte** diyoruz (aslında bundan daha küçüğü de bulunur fakat genel programlamada en küçük veri byte'tır).

Byte Bilgisayarda, metinlerin her bir karakteri 1 er byte a denk gelir. Yani, bir karakteri belirtmek için 1 **byte** gerekir (bazı farklı durumlar söz konusudur, daha sonra farklılıkları göreceğiz). Yani “deneme” metni 6 byte alan kaplar, “test” metni ise 4 byte alan kaplar

Figür 1.4.5 E : Bir anket formundan alınmış kesit

Yukarıda gördüğünüz formda, ad, soyad ve doğum tarihi bilgilerini girmemiz için boş kutular bırakılmıştır. Bildiğimiz üzere, her bir kutuya, bir karakter (harf veya sayı) gelebilir. İşte nasıl bir kutuya bir karakter denk geliyorsa, bir karakterin boyutu da bir byte'tır. Byte'ları, yukarıdaki formdaki gibi, kutular olarak düşünebiliriz.

Ek bilgi, Byte'ın katları :

- Byte
- Kilo Byte (1024 byte)
- Mega Byte (1024 kilo byte)
- Giga Byte (1024 mega byte)
- Tera Byte (1024 giga byte)
- Exa Byte (1024 tera byte)

O halde, (bu durum için) bilgisayardaki en küçük veri tipi karakterler diyebiliriz. Bu durumda, yapılacak işlemlerin, byte-byte yani karakter - karakter yapıldığını göz önünde tutmalıyız.

Bu belirtilen 3 yöntemi inceleyelim. Tersten başlayarak, üçüncü yönteme bakalım; ‘Üçüncü yöntem de, defterin, 2. ve 3. bölümlerini, fotokopi çekerek, 2. ve 3. kişiye verebiliriz.’ Bu yöntemin avantajları, hızlı olmasıdır, dezavantajları ise, sonucun bize ayrı kağıtlar olarak verilmesi ve fotokopiye ödeyeceğimiz ücrettir.

İkinci yöntemi incelersek; ‘İkinci yöntem de ise, 2 tane yeni defter alabiliriz ve orjinal defterin 2. ve 3. bölümü bu yeni iki deftere yazabiliriz. Daha sonra, bu iki yeni defteri, 2. ve 3. kişiye vererek, 1. kişide de orjinal defteri bırakabiliriz.’ Bu yöntemin avantajı, sadece 2. ve 3. kişiye verilecek olan yeni defterlerin düzenli olmasıdır. Yani çıktıyı, 3. yöntemdeki gibi kağıtlar halinde değil, toplu olarak bir defter halinde alabilirsiniz. Dezavantajları ise, 2 defterin ücreti ve yazılması süresidir.

Birinci yöntemi incelersek; ‘Birinci yöntem, defteri parçalara ayırmak, gerekli ortallardan kopararak bölmek’. Bu yöntemin avantajı hızlı ve ücretsiz olmasıdır. Dezavantajı ise, çıktının düzgün bir formatta olmamasıdır. Yani çıktı olarak verdiğimiz üç parça içinde, bir tanesinin ön kapağı, bir tanesinin arka kapağı, bir tanesinin de ön ve arka kapağı olmayacaktır. Ayrıca bunu profesyonel bir iş ortamında yaptığımızda, pekte düzgün bir yöntem olarak kullanılmayacaktır.

Yöntem	Avantaj	Dezavantaj
Yöntem 1	Hızlı ve Ücretsiz	Çıktı düzgün formatta değil
Yöntem 2	Defterler düzenli	Ödenecek ücret ve yavaş yazılması
Yöntem 3	Hızlı	Fotokopi ücreti, düzensiz çıktı

Figür 1.4.5 F : Yöntemlerin avantajları ve dezavantajları

Bu gösterdiğimiz avantaj ve dezavantajlar, gerçektekilerdir. Şimdi, bu avantaj ve dezavantajların bilgisayar ortamında avantaj veya dezavantaj olup olmadığını bulalım. Bilindiği gibi, bilgisayarda, dosyalar, nesneler oluşturmak, kopyalamak tamamen ücretsizdir, çünkü oluşturulan her nesne, yapaydır.

Böylelikle, avantajların ve dezavantajların; ücretli ve ücretsiz olması önemli değildir. O halde, tablodan bu özellikleri çıkarıyoruz.

Yöntem	Avantaj	Dezavantaj
Yöntem 1	Hızlı	Çıktı düzgün formatta değil
Yöntem 2	Defterler düzenli	Yavaş yazılması
Yöntem 3	Hızlı	Düzensiz çıktı

Figür 1.4.5 G : Yöntemlerin avantajları ve dezavantajları - (ücret etkeninin çıkartılmış hali)

Aynı şekilde, bilgisayarda işlemler gerçeğe göre çok daha hızlı gerçekleşmektedir. Yani bizim bir sayfayı harf-harf (elle) yazığımız sürede, bilgisayar 25-30 kitabı yazabilir. O zaman diyebiliriz ki, bilgisayarda hızlı-yavaş faktöründe bulunmamaktadır. Daha doğrusu, bilgisayar insana göre çok daha hızlı çalıştığı için, hız faktörünün önemi kalmamaktadır.

Yöntem	Avantaj	Dezavantaj
Yöntem 1		Çıktı düzgün formatta değil
Yöntem 2	Defterler düzenli	
Yöntem 3		Düzensiz çıktı

Figür 1.4.5 H : Yöntemlerin avantajları ve dezavantajları - (ücret etkeninin ve hız etkeninin çıkartılmış hali)

Yukarıdaki tablodan en avantajlı yöntemin 2. yöntem olduğunu görürüz. Sadece ikinci yöntemin bir avantajı bulunmaktadır, üstüne üstlük, diğer yöntemlerin dezavantajları bulunmaktadır.

Bilgisayar dünyasında dosyaları bölme

Yine gerçek dünyaya benzer olarak, dosyaları oluşturan sayfalar, farklı (daha küçük boyutta ki) dosyalara aktarılabilir, bu işlem sonucunda dosya bölünmüş olunur.

Yukarda gösterilen örnekler, çok basit örneklerdir, gerçek hayatta genelde daha karmaşık ve büyük sistemler bulunur. İşte bu yüzden, sistemler (decompose) dekompose edilmeli yani basite indirgenmelidir. Bunun için anlamının bir bölümü olan '**analiz**' kullanılmalıdır.

Bu evre sistemin tasarımını ve planlanmasını içerir. Bu evrede; algoritmasını kurduğumuz sistemi programsal olarak ifade ederken kullanacağımız yapıyı tasarlarız.

Planlama evresi, programın yazılacağı dile göre değişmemektedir. Sadece, işin planından sonra yapacak olduğunuz daha ayrıntılı teknik dökümanda bazı değişiklikler yapmanız gerekmektedir. Çünkü kullanılacak sistemleri ve teknikleri dilin özellikleri belirlemektedir. Algoritma ile planlama arasındaki fark; algoritmanın evrensel bir dil olmasıdır. Algoritma her programlama dili için aynıdır ancak plan ve tasarım dil ve teknolojiye göre değişebilir. Bu evre programın teknik kısmıdır ve Programlama'ya giriş bölümünde anlatılmaktadır.

Proje yönetimi konusunda ayrıntılı bilgi alabileceğiniz siteler, ve kitaplar hakkında bilgi veren sitelerden bazıları:

<http://www.projectmanagementbooks.com/> (proje yönetimi kitaplarını tanıtan bir site)

<http://www.method123.com/free-project-management-book.php> (ücretsiz kitap)

Ayrıca, bu serinin ikinci kitabında, proje yönetimiyle ilgili ayrıntılı bilgi bulunmaktadır.

Bir projenin dökümanları 3 evrede incelenir.

1 – Proposal (Öneri – Teklif):

Yapılacak olan projenin, isminin, açıklamasının kapsamının (özet olarak), hangi sistemlerde geliştirilebileceğinin, hangi bilimlerle alakalı olduğunun belirtildiği 1-3 (bu sayı projeden projeye değişmektedir. 1-3 sayfa miktarı, küçük-orta tabanlı bir proje için belirtilmiş bir sayıdır) sayfalık dökümandır.

Bununla birlikte, bir teklifte, gereken donanım, yazılım, ekipman, ekip elemanları, gereken maddi kaynak, tahmini bütçe belirtilir. Ayrıca, projenin uygulanacağı kapsam belirtilir (platform, projeni nerede kullanılacağı). Aşağıda örnekte bir teklif şablonu gösterilmiştir.

[illegible]

Figür 1.4.5 I : Teklif sayfa yapısı örneği

Teklif herkese sunulabilir. Teknik bilgiye sahip olan veya olmayan kişilere; işletme yöneticilerine, belirtili konu üzerinde uzman olmayan mühendislere.... sunulabilir.

Teklif, ön araştırma yapıldıktan sonra sunulabilir. Ön araştırma ile platform – gereksinimler ve projenin genel kapsamı belirlenir. Bir teklif, ön araştırma yapılmadan verilirse, belirlenecek olan gereksinimler ve platformlar eksik veya yanlış olacaktır. Zaten, bir projeyi başlatanda tekliftir. Teklif sunumu sonucunda proje kabul edilirse, ayrıntılı araştırma ve teknik rapor hazırlanır.

2 – Proje planı ve ilerleme raporu

Bu döküman, projenin türünü ve ayrıntılı açıklamasını, projenin geliştirileceği metodolojiyi (bilimsel metotları), bilimsel metotların açıklamalarını, amaçlarını – hedeflerini, platform ve gereksinimlerinin ayrıntılarını, araştırma ve geliştirme evrelerinin ayrıntılarını içermektedir.

Plan dökümanının uzunluğu, 10 sayfa ile N (projenin büyüklüğü ile genişlemektedir) sayfa arasında değişmektedir. Bir plan dökümanında, daha sonra göreceğimiz;

- WBS
- Flow chart
- Gantt chart
- Activity Network Diagram

bulunur.

Plan, yapılacak olan proje konusunda teknik bilgiye sahip olan kişilere gösterilmelidir. Ancak bir planın bazı bölümlerini, teknik bilgiye sahip olmayan kişilerde rahatlıkla anlayabilir. Örneğin, yukarıdaki 4 tip diagram, bir işin finansörü, araştırmacısı, ekipman

sağlayıcısı için gereklidir. Sonuçta plan yine, teklifte olduğu gibi herkese sunulabilir. Teklifin sunulduğu kişiler, sadece kendilerini ilgilendiren bölümü önemser ve okurlar.

Plandan önce, bir araştırma yapılmalıdır. Bu araştırma, normal araştırmanın %50'sini veya %100'ünü oluşturur. Yani, yapacak olduğunuz araştırmanın ya yarısını yada tamamını, planı hazırlamadan yaparsınız. Daha sonra planı hazırlarsınız. Planı hazırladıktan sonraki evre, teknik rapordur. Bu evrede, hem teknik raporu yazıp hemde geriye kalan %50'lik araştırma bölümünü gerçekleştirebilirsiniz. Ancak bazı durumlarda, bir sistmei kurmadan bütün sistemin araştırılması yani araştırmanın tamamının planın öncesinde yapılması gerekebilir. Bu durum projeden projeye değişmektedir.

3 – Teknik rapor

Yapılan araştırmadan sonra, projenin BÜTÜN teknik ayrıntılarının yazılacağı, dökümandır. Projenin kalbi olarak adlandırılabilir. Sayfa sayısı konusunda bir belirtme yapmak yanlış olacaktır. Bazı projeler sadece 5-10-20 sayfa sürerken, 15-20 bin sayfa belki de daha fazla süren teknik raporlar bulunur.

Teknik rapor hazırlamak başlı başına ayrı bir konudur. Bu konuda detaylı bilgi almanız için, proje yönetim kitapları almanızı veya internetten araştırma yaparak bu konuda bilgi sahibi olmanızı tavsiye ederim.

Teknik rapor sadece ve sadece teknik bilgiye sahip olan kişiye sunulabilir. Zaten, kullanılacak olan terimler ve anlatımlarda da, okuyucunun profesyonel olduğu esas alınır.

Örnek bir teknik raporun (yazılım geliştirme) içindekiler bölümünü aşağıda görebilirsiniz

1. Giriş & Tanıtım
2. Alt yapı – arka plan - araştırma
3. Gereksinimler ve tasarım
4. Geliştirme evresi (implementasyon)
5. Test
6. Proje yönetimi
7. Sonuç
8. Referanslar
9. Index

[Ara verme bölümü]

1.4.6 Kod yazımı

Bu evrede, algoritması kurulmuş ve planlanmış bir sistem, seçilmiş olan programlama dilinde ifade edilir. Tabiki kodlama hemen hemen her dilde farklıdır çünkü dilin özellikleri, teknikleri ve kuralları dilden dile değişmektedir.

★ Geliştirme evresi

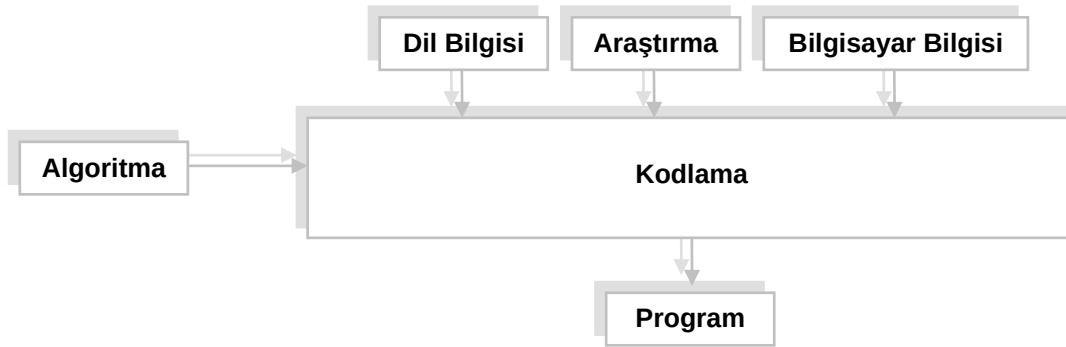
Bir programın algoritmasını düşünebiliyorsak (öyle ki bu zor bir proses değil), programını yazmak mümkün olacaktır. Aksi taktirde, bir programın yazılması mümkün

değildir. Ancak, basit ve standart programları, internet sitelerindeki örneklerle bakarak, forumlara sorarak, kopyalayarak, birilerinden yardım alarak yazılabilir. Tabi ki bu tip programlar da sistem oturmadığı için problemler oluşur ve tam olarak çalışmaz ve anlaşılmaz.

Geliştirme evresinde, aklımızda kurduğumuz algoritmayı nasıl (yazacağımız) dile çevireceğimizi bilmek gerekmektedir. Bu proses için, biraz bilgi, biraz deneyim, biraz bilgisayar bilgisi, birazda internet araştırması gerekmektedir.

Algoritma, Dil bilgisi, Bilgisayar bilgisi, Araştırma (kitap/internet) , Kodlama

Yukardaki konularda bilgi sahibi isek / veya işlemleri gerçekleştirebiliyorsak, program yazmak mümkündür. Ancak her zaman, programlamanın başlangıç noktası, algoritmadır. Aşağıdaki diagram, bir programın gelişme evresinin nasıl olduğunu açıkça gösterir



Figür 1.4.6 A : Programlama şeması

★ Algoritma

Algoritma, mantığa dayalı bir sistemdir. Algoritma kurabilmek için, gerçek hayattan benzer bir olay bularak, algoritması kurulacak olan durumla ilişkilendirmek gerekmektedir. Bunun için pratik düşünmek, deneyim ve sistematik çalışmak gerekmektedir.

★ Dil bilgisi

Dil bilgisi, ezbere ve deneyime dayalı bilgidir. Hedef seçilen dilin, syntax'ını, terimlerini, kurallarını, koşullarını ve prosedürlerini öğrenmek gerekmektedir. Tabi bundan önce, Programlamaya giriş bölümü okunmalıdır.

★ Bilgisayar bilgisi

Bilgisayar bilgisi; kitaptan alınacak bilgi, internetten yapılan araştırmalar sonucu elde edilecek bilgi, bilgisayar dergilerinden elde edilen bilgi, sorular sonucunda elde edilen bilgilerden oluşur. Bunun için deneyim ve araştırma şarttır. Kitabı okuma süresince, (hemen her konu içerisinde) bilgisayar ile ilgili genel bilgi verilmektedir. Ancak bununla birlikte, araştırmacı olmak, programlama yeteneğini geliştirir ve süresini kısaltır.

★ Araştırma

Program geliştirebilmek için araştırmak çok önemlidir. Araştırma kaynakları olarak; başta internet olmak üzere, kitaplar, bilgisayar dergileri v.b şeklinde sıralanabilir. Önemli olan, her dili, bütün ayrıntılarıyla bilmek değil, bir dil için, bir durumu anlayabilmektir.

Kimse, programlama alanında her bilgiyi bilemez, ancak birçok konuyu anlayabiliyorsa, o zaman usta bir programcıdır. Yıllar evvel, daha çocukken, babama (makine mühendisi) her makinenin yapısını bilip bilmediğini sormuştum. Babamda bana, *'her makinenin yapısını bilmeme gerek yok, önemli olan, makinenin içini açtığı anda, fonksiyonları gerçekleştiren bileşenler arasındaki ilişkiyi kurmak, ve ne işlev gerçekleştirdiklerini anlamaktır'* demişti.

O yüzden, ezberci bir yöntemle değil, algılayarak ve uygulayarak program geliştirme, profesyonel programlamanın temelini oluşturur. İnternette binlerce programlama ve algoritma sitesi (forum veya portal olarak) bulunmaktadır. Ayrıca yüzlerce arama motoru, belirttiğiniz kritere göre size istediğiniz sonucu görüntüleme kapasitesine sahiptir.

Bir programlama dili ve sistemi (java, delphi, vb, c#, c++, asp...) sadece algoritması kurulmuş bir sistemi geliştirmeye yarar. Bu konu ile ilgili bir formül yazalım.

Algoritma kurabilme

Bu yeti, sistemlerin, parçalarını (bileşenlerini) tanımlayabilmeye yarar.

Bu yetinin iyi olması ile, daha çok algoritma kurabiliriz. Böylece;

1 – Daha çok program, programlanabilir

2 – Programlar daha farklı yöntemlerle programlanabilir

★ Bir dile iyi hakim olma

Bu yeti, bir dilin hangi işlemleri, hangi yollarla yapabileceğinin bilinmesine yarar. Böylece, o dilin, hangi kapsamdaki programların geliştirilip geliştirilemeyeceğini, öğrenebiliriz. Ayrıca, bir dile iyi hakim olma, geliştirme süresini azaltır, daha kısa ve daha kolay yöntemlerin kullanılmasını sağlar.

İnternette araştırma yaparak, daha önceden yapılmış (sizin yapıyor olduğunuz projeye) benzer projeler bulmak, kodlama algoritmaları bulmak mümkündür. Ancak ben, algoritma için araştırma yapmayı doğru bulmuyorum. Daha öncedende bahsettiğim gibi algoritmayı kendimiz kurup, sonra bunu seçili dilde nasıl ifade edeceğimizi bulmak için araştırılma kullanılabilir. Bu olayı bir örnekle anlatayım.

-Bir inşaat mühendisisiniz ve bir binanın çizimlerini yapıyorsunuz. Daha sonra işçilere planı veriyorsunuz ve işçiler o plana göre tek-tekk tuğlaları dizerek işi yapıyor.

-Bir inşaat mühendisisiniz ve bir binanın çizimlerini yapıyorsunuz. Daha sonra işçilere planı veriyorsunuz ve işçiler yong gibi büyük kalıplı tuğlalar kullanarak işi yapıyor



Figür 1.4.6 B: Ytong (solda), Tuğla (sağda)

Büyük kalıplı tuğlalar ile iş çok daha kısa sürer. Eğer kodu tek tek, satır satır yazarsanız kod uzun sürebilir, ancak belli kısımlarında başkalarından yararlanarak yazarsanız bu süre azalır. Dikkat edilmesi gereken nokta, sistemin algoritmasını kendinizin kurması gerektiğidir!!!

Kısacası, algoritma kurabiliyorsak, ve kurduğumuz algoritmaları birleştirerek program haline getirebiliyorsak, bunu bir dille ifade etmek için o dili bilmemize gerek bile yoktur. Önemli olan daima algoritmadır. Dil için gerekli bilgiyi, arama motorlarından veya kitaplardan kolaylıkla öğrenebiliriz.

Yukarıda bahsettiğimiz gibi, bir dili iyi bilmek, ona hakim olmak, hem daha profesyonel programlama yapmamızı sağlar, hem daha hızlı kod yazmamızı, daha kısa sürede programları tamamlamayı ve daha iyi yöntemlerle program geliştirmemizi sağlar.

Kitaplar içinde araştırma (bir programlama dili ile alakalı bilgi veren kitaplar), hızlı olmadığından sıkıcıdır. Her ne kadar indeksi ve içeriği olsada, eğer siz (ör.) ‘dosya nasıl kopyalanır’ sorusunun cevabını merak ediyorsanız, ne indekste nede içerikte böyle bir madde bulamazsınız. Bu işlem, FileCopy veya Copy komutunun içeriği altında geçebilir.

Sayfa sayısı çok olan kitaplarda işler daha da karışır. İnternet bu yüzden artı bir puan alır. Çünkü, eğer ingilizcenizde var ise internette bulamayacağınız bilgi yoktur. Öyle ki, hızla gelişen teknoloji sayesinde, artık Türkçe binlerce döküman bulunmaktadır.

Kitaplar yerine e-book yani elektronik kitapları tavsiye ederim. Çünkü içinde arama yapabiliyorsunuz ve size bütün sonuçları listeliyor. Bu şekilde, internetteki arama hızına ulaşabilirsiniz.

<http://scholar.google.com> adresinden arama yaparsanız elde edeceğiniz sonuçların hepsi teknik sonuç yani teknik bilgi içeren sonuç olacaktır. Yani standart arama motorlarında aramaktansa, sadece teknik bilgilerin olduğu bir arama motoru kullanmak daha düzgün sonuçlar getirecektir. Ayrıca,

<http://directory.google.com/Top/Computers/Algorithms/>

<http://directory.google.com/Top/Computers/Programming/>

http://dir.yahoo.com/Computers_and_Internet/Programming_and_Development/

gibi, binlerce programlama ve algoritma sitesinin listelerinin olduđu gruplarda arama yapabilirsiniz.

İnternet üzerindeki open source kodları bulmak için, <http://www.google.com/codesearch> adresindeki arama motorundan yararlanmanızı tavsiye ederim.

Programlama tabanlı dergilerde de bazen çok ilginç ipuçları, daha önceden düşünmediğimiz ancak işimize yarayabilecek bilgiler bulunabilir. Bunun için ayda 4-5 YTL vererek bir dergi sahibi olmak ve okumak önemlidir.

★ Kodlama

Kodlama yeteneğinin gelişmesi için, dile hakim olmak önemlidir. Bunun haricinde, iyi araştırma ve program geliştirme yani deneyim çok önemlidir.

Örneğin, ben üniversitenin ilk yılında Java dilini hiç bilmiyordum hatta hiç görmemiştim. Üniversite 1’de Programlamaya Giriş dersi alıyordum ve ödev olarak, java da basit bir program hazırlanması gerekiyordu. İlk kez Java başına oturup, ödevi 15 dakikada tamamlayarak (öyleki bu ödev için 2 saatlik labaratuvar dersi bulunuyordu), 80-90 gibi yüksek bir not almıştım.

İkinci ödevi, derse giderken yolda yaptığım için syntax hatalarından (yazım hataları - ileride açıklanacaktır) dolayı, 50 gibi bir not almıştım. Hatta ödevleri kontrol eden kişi bana, sıkıntı çekiyorsam ona danışabileceğimi söylemişti. Son ödevden 100 alınca O’da şaşırđı. 100 almak için sadece 15 dakika ayırmıştım.

Benzer bir olaya, bir PHP forum kurarken rastladım. O zamanlar PHP bilmeyen biri olarak, php forumda bir noktayı değiştirmemiz gerekiyordu, arkadaşımınla bilgisayar başına geçip, yapmamız gerekeni kısa sürede yaptık ve sorunu çözdük.

Benim başımdan buna benzer bir çok olay geçti ve daha sonra algoritmanın, programlamada ne kadar önemli bir rol oynadığını anladım. Şimdi hemen hemen hangi dilin başına otursam oturayım, çözmem uzun süre almaz, çünkü, bir programın ne amaca yaradığını bulduktan sonra, içeride neyin olup bittiğini çıkarabiliyorum.

Üniversitede bir hocamız, “Markete gittiniz, 6 tane patates cipsi aldınız ve kasaya geldiniz, hesapta 2.4 sterlin tuttu. Burada farklı bir durum söz konusu mudur?” şeklinde bir soru sormuştu. Bizde herhangi bir farklı durum olmadığını söyledik. Daha sonra, “Markete gittiniz, 6 tane patates cipsi aldınız ve kasaya geldiniz, hesapta 102 sterlin tuttu. Burada farklı bir durum söz konusu mudur?” şeklinde soruya zannedersem hepimiz aynı cevabı vereceğiz. 6 patates cipsi nasıl olurda 102 sterlin tutabilir ? (1 sterlin 2.5 YTL yani 102 sterlin yaklaşık 255 YTL)... Burada hocamızın anlatmak istediğı, patates cipsinin fiyatını bilmesek bile, ne kadar olabileceğini tahmin ederiz. Aynı şekilde bir programın içinde ne yazıldığını, satır satır bilmesekte, (bu kitabı okuduktan sonra) ne gibi sistemler – ifadeler olacağını tahmin edebileceğiz. Zaten kitabın amacı da bir programın içinde gerçekleşen işlemleri, o programın kodlarını görmeden anlayabilmektir

Deneyim, zekayı artırır, daha önceden çözülmüş problemleri, çözmek için ek olarak düşünme gerektirmez. Böylece, programlamadaki algoritması kurulan sistemdeki işlemleri tasarlamak çok daha kısa sürecektir. Bununla birlikte, bilgi dağarcığının da gelişmesi gerekmektedir.

Kodlarken, sadece verilen işlemi bilgisayara yaptırmaya odaklanmamalıyız. Bununla birlikte en az sistem kaynağı harcama, en kısa sürede yapma, yapılan işlemleri bir daha yapmama ve daha profesyonel yazmaya dikkat etmeliyiz. İleriki bölümlerde kodlama yaparken dikkat edilmesi gereken noktalar belirtilmiştir.

Daha öncede söylediğim gibi, ‘herşeyi kimse bilemez, ama bildiğimizi kullanabiliyor, ve o bilgiden bir ürün üretebiliyorsak, o zaman bizden üstün kimse yoktur.’ Ayrıca, ‘kullanılmayan bilgi gereksiz bilgidir, eğer gerekecekse, zaten biliyoruzdur veya önümüze çıkacaktır. Biliyorsak gerektiğinde kullanırız, bilmiyorsak ve karşımıza çıkıyorsa, o bilgiyi, o an öğrenebiliriz.’

Örneğin, siz, veya başka bir kişi, dünya üzerinde olan 2000’den fazla programlama dilini ezberlese, bu bilgi, sizin veya o kişinin ne işine yarar? Böyle bir işlemi yapmak sadece zaman kaybıdır. Siz eğer 4-5 dili, 10-15 mimariyi biliyor, çok iyi algoritma kurabiliyor ve iyi analiz yapabiliyorsanız, zaten ‘2200 programlama dilinden’ %75 ini çözebilir veya kullanabilirsiniz demektir.

Programın çıktı olarak verilmesi

Basit olarak; tasarımı ve kodlaması bitmiş program, kullanıcıya sunulur.

Bunun haricinde, programın çıktı olarak sunulma evresinde, software engineering – yazılım mühendisliğinin, yazılım geliştirme ile ilgili birçok ilkesi ve kuralı bulunmaktadır. Örneğin yazılımın testinin yapılması. Ancak bu ve benzeri işlemler konumuzu etkilememektedir. Bunların bir kısmı programlamanın, bir kısmı da yazılım mühendisliğinin kapsamı içerisinde. İleriki bölümlerde bu konu ile ilgili ayrıntılı bilgi bulunmaktadır.

Şimdi bu dediklerimizi size ispatlayalım. Hiç bilmediğimiz bir dili ele alsak, onun çalışma prosedürünü öğrendikten sonra yapmanız gerekenler;

Google’a veya başka bir arama motoruna girip,
“file open java code read write how to” veya
“java basit dosya işlemleri” yazmanız

Not: Buradaki Java bir dilin ismidir.

Sonuçlar arasındaki sayfalardan birinde aşağıdaki kod bulunuyor :

```
import java.io.* ; // işlemleri yapacağımız paketimiz
public class FProcess {
    public static void main(String args[]) {
        try {
            File file = new File("MyFile.txt") ; // yeni dosya
            olustur.
            FileWriter fwriter = new FileWriter(file) ;
            BufferedWriter bw = new BufferedWriter(fwriter) ;
```

```

        bw.write("Test");
        bw.close() ; //dosya yazma işlemi bittikten sonra dosya
kapatılır
    }
    catch(IOException ioexc) {}
}

```

Sitelerde genelde işlemlerin kodu ve hatta açıklaması da bulunmaktadır. Aslında open source’a karşı olmama rağmen böyle bir konuda, sitelerden, kodlardan, yazılardan yararlanılmasını tavsiye ederim. Bundan sonra, yapmanız gereken, kodu projenize uyarlamak. Tabi bu işlem biraz deneyimle ilgilidir.

Kendinizi bir mühendis gibi düşünün, siz, ustalara – teknikerlere neyi nasıl yapacaklarını iyi açıkladıktan sonra, o kişinin hangi adımları izlediği elini ne kadar kullandığı nasıl kaldırdığı... gibi gereksiz ayrıntıların önemi yoktur.

Zamanla, yukarıdaki kodun ne olduğunu çok iyi bir şekilde kavrayacaksınız. Örneğin; daha sonra okuyacağınız bölümlerde “class” ın ne olduğunu öğrendikten “prosedür” ün ne olduğunu öğrendikten, “döngü” nün ne olduğunu öğrendikten sonra, bu kodu anlamak sizin için çocuk oyuncağı olacaktır. Tabi aynı durum diğer diller içinde geçerlidir. Fakat şuan sakın anlamaya çalışmayın! Bu bilmediğiniz bir dili sözlüksüz öğrenmek gibidir.

Örneğin; C#.Net

```

using System.IO;
public class FProcess{
    public static void X(){
        try{
            StreamWriter writer = new StreamWriter("MyFile.txt");
            bw.WrieLine("File created using StreamWriter class.");
            bw.Close();
        }
        catch (Exception ex){}
    }
}

```

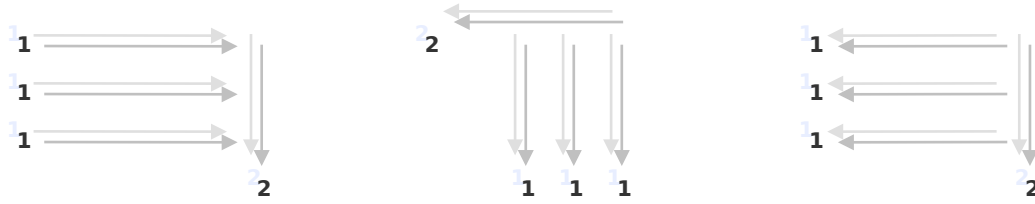
Ne kadar benzer olduğu açıkça görünüyor. Tabi diller içleri arasında bu kadar benzerliğe sahip olmayabilirler. Ancak önemli olan işleyiştir. İleriki bölümleri bitirdiğinizde bu işleyişlere öğrenebileceksiniz.

[Ara verme bölümü]

1.4.7 Programlama akışı

Hayatta, zaman, nasıl **önceden** – **sonraya** doğru akıp gidiyorsa, programlamada kendi hayatı içinde (çalışıyor olduğu sürece) programın başından, işlemleri gerçekleştirmeye başlamakta ve sonuna doğru devam etmektedir.

Kullanmakta olduğumuz latin alfabesi, soldan sağa doğru sonrasında ise, yukarıdan aşağıya doğru yazılan bir alfabedir. Dünyada 1 milyardan fazla kişinin konuştuğu Çince, yukarıdan aşağıya doğru yazılır sonrasında da, sağ üst köşeden başlayarak yazılır. Arap alfabesi ise, sağdan başlayıp sola doğru sonrasında ise aşağıya doğru yazılan bir alfabedir.



Figür 2.1 A : Yazım yönleri

İlk figürde latin, ikincisinde, çin, sonuncusunda arap alfabesinin yazılış yönleri gösterilmiştir. Hepimizin tahmin edeceği gibi, programlama dilleri için de, kullanılacak olan akış, uluslararası dil kabul edilen **ingilizcenin** kullandığı alfabe olan latin alfabesininki gibi olacaktır.

Programlama da, yazılan kodlar, yukarıdan aşağıya doğru işlenir. Birçok satırda her bir satır bir işlem anlamına gelmektedir ve kodlar, satır satır işlenmektedir.

İşlem 1
İşlem 2
İşlem 3
İşlem 4
.....

Yukarıdaki satırda yazılı ifadeleri birer kod satırı olarak düşünelim. Bu işlemler satır satır sırayla gerçekleştirilecektir.

Bazı dillerde, gerçekleştirilecek her ifade sonuna bir noktalı virgül işareti konulması gerekmektedir. Yani her bir işlem sonunda noktalı virgül konulur, bilgisayar bu işlemleri gerçekleştirirken her bir noktalı virgüle kadar olan kısmı dikkate alır. Bazı dillerde ise sadece bir alt işlemlerin herbirini birer satıra yazmak yeterlidir (yukarıdaki örnekteki gibi).

Bazı sistemlerde , yazılan ifadeler gerçekleştirilmeye başlanır, ve işlemin bitmesi beklenir. Yani, “işlem 1” işlemi çalıştırılır, işlemin tamamlanması beklenir ve ancak işlem bittikten sonra diğer işlemlere geçilebilir. Aynı şekilde bu süreç diğer işlemler içinde aynıdır. Bazı sistemlerde ise, çağrılan işlem, gerçekleştirilmeye başlanır ve hemen sonrasında, diğer işleme geçilir. Yani işlemin sonuçlanması beklenmez. Genelde, programlamada ilk tanımladığımız sistem geçerlidir. Bu konu ile ayrıntılı bilgileri daha sonra öğreneceğiz.

Alıştırmalar

- Analiz nedir?
- Bir sistem nasıl analiz edilir?
- Operatör nedir?
- Lowest Limitation nedir?
- Lowest Limitation'a nasıl karar verilir?
- Bir sistem nasıl anlaşılır?
- Kod hamallığı nedir?
- Byte nedir?
- Proposal nedir?
- Kod yazımı hangi evrelerden oluşur?

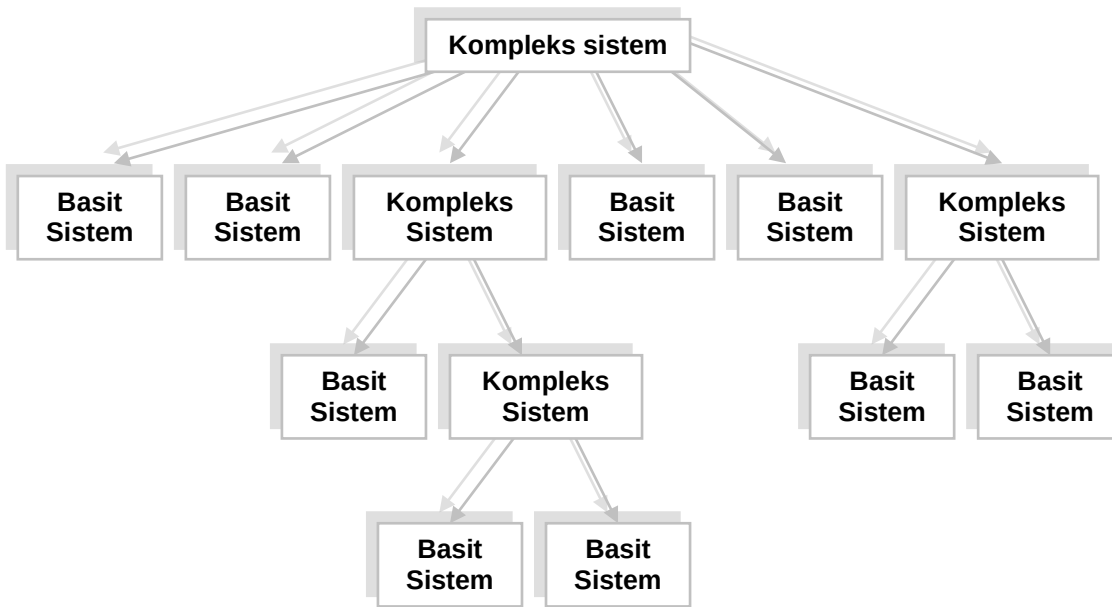
1.5 Planlama

1.5.1 Geniş ve kapsamlı sistemler

Daha önce analizin tanımını vermiştik, ona benzer başka bir tanım ise “kompleks bir materyeli, bileşenlerine ayırıp inceleyerek anlama süreci”dir. Aslında analizin bu tanımında hemen hemen aynı anlama gelmektedir.

Büyük ve geniş sistemlerde, (daha önceki örneklerde verildiği gibi) sadece dosya parçalama, dosya sıkıştırma gibi, basit işlemler yerine karışık ve kompleks işlemler de (veritabanları – yönetim programları – erp...) bulunmaktadır.

Önemli nokta: Büyük ve karmaşık sistemler, birden fazla, küçük ve basit işlemlerden oluşur. Bu yüzden, büyük sistemler, analiz edilerek, daha küçük sistemlere (büyük sistemlerin bileşenlerine) ayrılır ve her bir basit işlem için ayrı algoritmalar kurulur. Daha sonra bu ayrı algoritmalar birbirleriyle birleştirilerek büyük sistemin temel algoritmasını oluşturur.



Figür 1.5.1 A : Basit Sistem - Kompleks sistem

Sonuç olarak “Neler Programlanabilir” : Algoritması tam olarak kurulabilen, her sistem, programlanabilir, programsal olarak ifade edilebilir. Şuandaki teknolojide programlanabilecek en zor sistem, Yapay Zekadır. Eğer zeka’nın algoritması tam olarak kurulabilse, yapay zeka da programlanabilir ve keşfedilmiş olunur.

Bir sistemi, alt sistemlere bölme olayı hakkında gerekli bilgileri, Lowest limitation bölümünde vermiştik. Bu işi elekten geçen taneler gibi düşünebiliriz. İşlemleri, belirli bir seviyeye gelinceye kadar bölmeliyiz (yani eleğin deliklerinden geçebilecek kadar küçük hale gelinceye kadar).

Düşünüyorum, o halde varım, anlayabiliyorum, o halde programlayabilirim.

1.5.2 WBS

Analiz ve planlamaya girmeden önce, gündelik hayatta sık sık kullanılan ve çok önemli bir görevi olan WBS (Work Breakdown Structure) İş dökümü yapısı hakkında bilgi vermemiz gerekmektedir. Bir işin bölümü için, çok basit olarak, bir iş yerinde, belirli bir grup çalışana verilen işin, iş bölümü ile paylaşılmasının gerektiğini düşünebiliriz.

Birkaç konu öncesinde anlatılmış olan analiz, veya bir önceki bölümde gösterilmiş olan planlama konularında, bir işin daha küçük parçalara bölündüğünü söylemiştik. İşte parçalara bölünen bir iş, WBS üzerinde gösterilir.

Bu konu üzerinde, günümüze kadar, karşımıza binlerce örnek çıkmıştır. Bir işin detaylara bölünmesi, paylaştırılması için hazırlanan her diagram veya bir liste (a işlemi, b işlemi, c işlemi) bir WBS olabilir.

Örneğin bir alış veriş listesini bile basit bir WBS olarak düşünebiliriz.

- meyve suyu al
- cips al
- dondurma al

→ Başka bir örnek olarak, geçen ay satış yapılan müşterilerden, XY ürününü alanlara ulaşp, ürünlerinin hatalı olduğunu ve yenileriyle değiştirilmesi gerektiğini bildirmemiz ve ardından yeni bir XY ürününü müşteriye götürüp teslim etmemiz gerektiğini varsayalım. Yapılacak işlerin sırasını hemen bir liste halinde çıkarabiliriz.

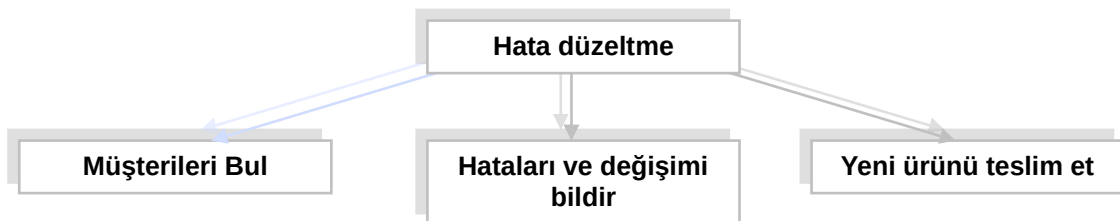
Kalın yazılan kelimeler birer eylem bildirmektedir. Her eylem bir iş ifade etmektedir.

Geçen ay satış yapılan müşterilerden XY ürünü alanları bul! (Çünkü elimizde bu liste henüz yok)

Hepsine telefonla ulaşp, ürünlerinin hatalı olduğunu ve değiştirilmesi gerektiğini bildir

Yeni ürünü al, müşteriye teslim et.

Şimdi bu işlemleri bir iş bölümü kalıbı diagramı olarak görüntüleyelim.



Figür 1.5.2 A : WBS örneği

Bu hata düzeltme işlemini bir ekibin aynı anda yaptığını ve ekipteki çalışan sayısının 3 olduğunu varsayalım. O zaman, bir çalışan müşterileri bulur, bir diğeri arayp değişimin gerektiğini bildirir, üçüncüsü de yeni ürünleri teslim edip eskilerini alır, diyebiliriz. İşte bu iş bölümüdür ve yukardaki diagram bir WBS'dir (bu işin WBS'sidir).

Görüldüğü üzere, bir iş bölünürken, görevin içerdiği emirdeki her bir eylem için bir işlem oluşturulur. Tabiki bu oluşturulan işlem, alt işlemlere bölünebilir.

Örneğin, ilk işlem :

- Geçen ay satış yapılan müşterileri bul
- Daha sonra bu ürünler içinden, XY ürünü alanları bul

Üçüncü işlem de :

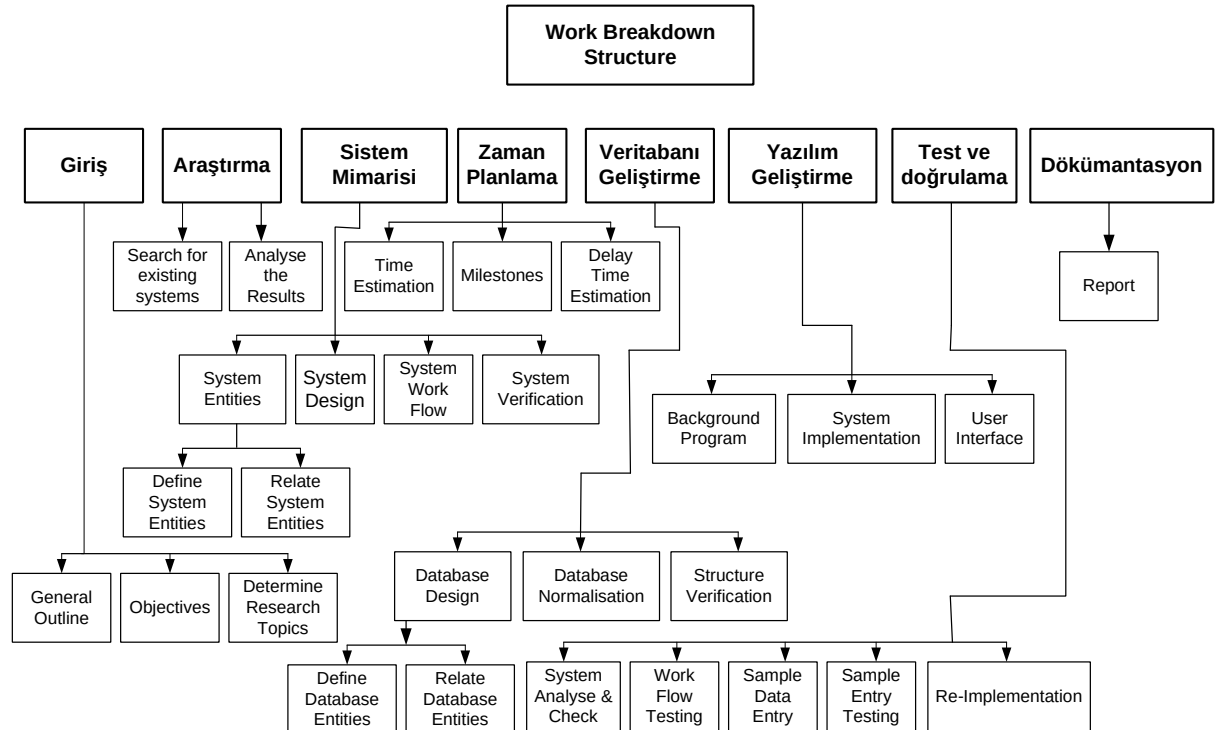
- Yeni (çalışıyor olan) ürünü müşteriye teslim et
- Çalışmıyor olan ürünü teslim al

şeklinde bölünebilir.

WBS’de zaman – iş ilişkisi bulunmamaktadır. Yani iki veya daha fazla işlem aynı anda yapılabilmektedir ve sizin bunu WBS üstünde göstermenize gerek bulunmamaktadır. Yapılacak olan işlemlerin sırası AND dediğimiz diagram üzerinde görüntülenmektedir. AND bir sonraki konudur ve ayrıntılı bilgi bu konu içinde verilmiştir.

WBS konusunda, anlatılacak çok fazla ayrıntı bulunmamaktadır. Bu bölüme kadar, Analiz, Sistemler, Lowest limitation konularında anlatılmış olanları diagramsal olarak ifade etmek için WBS kullanırız.

Aşağıdaki örnekte, üniversite son sınıfta bitirme projem için yaptığım projenin WBS’si görüntülenmektedir. Hazırladığım proje bir tür yazılım projesiydi.



Figür 1.5.2 B : WBS örneği

Yukarıdaki wbs’de, ilk evre, giriştir. Bu ifade, sistemin ne yapacağını, hangi teknik ve sistemlerin belirlendiği evredir.

Sonraki evre, araştırma evresidir. Üçüncü evre Sistem Mimarisi, dördüncü evre zaman planlanmasıdır. Yazılım (programlama) projeleri genelde, bu yapıya uygun olarak düzenlenmektedir.

WBS, aslında daha önceden bahsettiğimiz, ANALİZ işleminin, diagramsal olarak ifade edilmesidir. İşlemleri, mümkün olan düzeyde, ve aynı basit seviyeye getirene kadar bölerek bir WBS elde edilebilir.

‘Aynı basit seviye’, WBS’nin dallarının aynı seviyede olması anlamına gelmemektedir. Önemli olan, işlemleri, parçalama kriterinizin aynı olması ve aynı kabul seviyesini buluncaya kadar bölmek, yani analiz ederek, o işlemi oluşturan daha basit, alt işlemlere ayırmaktadır.

Bir WBS oluşturma işleminde, 4 tip parçalama – analiz kriteri kullanabiliriz. Önemli olan nokta ise, bu kriterlerden sadece birini değil, birkaçını veya hepsini kullanabiliriz.

- 1 – Operatöre göre bölme**
- 2 – Süreye göre bölme**
- 3 – İşlem hacmine göre bölme**
- 4 – Konuya göre bölme**

★ **Operatöre göre bölme:** Eğer bir projede, işlemleri gerçekleştirecek operatörler farklı ise, bu işlem daha küçük parçalara, OPERATÖRE GÖRE bölünebilir. Daha açıkcası, bütün olarak gördüğümüz bir işlem birden fazla operatör tarafından gerçekleştiriliyorsa, bu işlem, operatörlere göre bölünebilir.

Örneğin, bir yazılımın işlemlerini OPERATÖRE GÖRE belirleyelim. İlk önce, ‘yazılım oluşturma’ işlemini parçalayarak başlıyoruz.

Yazılım oluşturma işleminde çalışacak departmanları (veya çalışacak farklı ekipleri – bölümleri – takımları) göz önüne alalım.

Araştırma
Geliştirme
Test

Yukardaki listeye göre, ‘yazılım oluşturma’ işlemini, 3 parçaya böldük ve bu bölme işleminde, ‘Operatör’ kriterini kullandık. Ancak benzer durumlar için bile olsa her zaman aynı sonuçları alamayabiliriz. Çünkü operatörler ve projenin içeriği değişebilir.

Örneğin, bu ekiplere ek olarak, bir arşiv ekibi olabilir, ve bu arşiv ekibinin görevinin, yazılım hakkındaki her bilgiyi rapor etmek olduğunu varsayalım. Bu durumda, listelenen üç işleme birde raporlama veya döküman oluşturma işlemi eklemek gerekecektir.

Başka bir örnek vermek gerekirse, şirketlerde, projenin ne kadar başarı sağlayacağını, ne kadar satılabileceğini, ne kadar kazanç getirebileceğini tahmin eden, hesaplayan, istatistiklere yorumlayan bir ‘piyasa araştırma’ ekibi olabilir. Böylece, bu ekiple birlikte, işlemlerin başına, ‘Genel Bakış’ işlemi eklenir.

→ Başka bir örnek olarak; bir marketi düşünelim. Marketin, telefonla gelen siparişleri evlere ulaştırdığını düşünelim. Bu durumda, bir kişi, siparişleri not alıyor, bir kişi siparişlerin içeriğini raflardan – reyonlardan topluyor ve bir kişide bu siparişleri sahiplerine ulaştırıyor olsun.

Burada, sipariş ulařtırma iřlemini 3'e ayırdık

Sipariř alma
Sipariř ieriđini hazırlama
Sipariř teslim

Her biri farklı kiřiler yani operatörler tarafından yapıldıđı iin burada operatöre göre bölme kriterini kullandık.

★ **Süreye göre bölme:** Süreye göre bölme kriteri, daha ok, alt iřlemleri ok fazla olan iřlemleri bölme de kullanılır.

Örneđin, yukardaki diagramda, 'Software Development – Yazılım geliřtirme' evresinin süresi oldukça uzundur. Tabi ki WBS de iřlemlerin süreleri gösterilmez, bu tip bir gösterim iin Gantt Chart kullanılır. Biz bu iřlemin, uzun süreceđini biliyoruz, sadece diagramda göstermiyoruz. Projenin yaklaşık %50-%60 lık süresini tek başına iřgal edecek olan bu iřlem, proje deki en uzun iřlemdir. O yüzden bu iřlemi, ya ok alt iřlemlere, yada sadece belirli süre gruplarına göre bölmemiz gerekecek.

WBS leri farklı düzeyde kiřilere sunabiliriz. Örneđin, yukardaki WBS bir yöneticiye sunulacak bir WBS'dir. Dikkat ettiyseniz, ok ayrıntı içermemektedir. Varsayalım bu WBS'yi yöneticiye gösterdiniz ve proje kabul oldu, daha sonra her bir ekibe, neler yapacađını anlatan bir döküman ile birlikte, sadece onlara özel bir WBS göndermeniz ve o WBS üstünde sadece onların yapacakları iřlemlerin bulunması gerekmektedir. Tabiki, belirtili ekibe göndereceđiniz WBS, standart (yöneticiye sunulan) WBS'den daha ayrıntılı ve daha detaylı olmalıdır. Aynı řekilde, daha alt opeatörler iin daha detaylı ve farklı WBS'ler hazırlanabilir.

İřte bu yüzden, uzun iřlemleri, süreye göre bölmeliyiz. Sadece yazılım geliřtirmenin ayrıntılarının bulunduđu bir WBS'de süreye göre deđil diđer kriterlere göre bölme iřlemi yapmalıyız.

Bir iřlemi, farklı bir yöntem olarak, bir zaman dilimine göre paralayabiliriz. Örneđin, projenin bir iřleminin (yazılım geliřtirme) süresinin 3 ay olduđunu varsayalım. Eđer süre birimini 'ay' olarak alırsak, 'yazılım geliřtirme' iřlemini 3 e ayırabiliriz.

'Yazılım geliřtirme' iřleminin süresinin 4 ay olduđunu varsayalım ve ilk iřlemin iki, diđer iki iřlemin birer ay süreceđini varsayalım. Yine de bu iřlemi üçe bölebiliriz. Önemli olan iřlem sürelerinin, birim olarak aldıđımız 'ay' ın katları řeklinde olmasıdır. Tek dikkat etmemiz gereken husus, sürelerin birbirilerine mümkün olduđunca yakın olması gerektiđidir.

★ **İřlem hacmine göre bölme;** ok geniř olan evreler, daha az sayıda iřlem yapan evrelere bölünmesiyle gerekleřir. Örnek olarak veritabanına veri giriřini verebiliriz. Bu iřten sorumlu bir kiřiyi (operatörü) düşünelim. Kiři, öncelikle, mali kayıtları, daha sonra, matematiksel kayıtları, daha sonra müşteri kayıtlarını veritabanına girmeyi hedeflesin. Burada iřlemlerin kısa sürdüđünü ancak, farklı birok türde peřpeře iřlemin olduđunu varsayalım.

Böyle bir durumda, bu evreyi; Mali verileri girme, Matematiksel verileri girme ve müşteri verilerini girme řeklinde alt evrelere bölebiliriz.

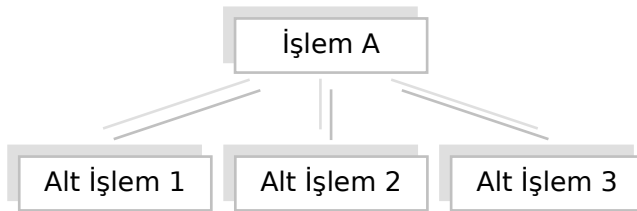
Süreye göre ile işlem hacmine göre bölme arasındaki fark ise, süreye göre bölünen işlemler daha uzun, işlem hacmine göre bölmede ise daha kısa işlemler bulunmaktadır.

★ **Konuya göre bölme:** Yine yukardaki örneği alalım. Veritabanına bilgi girişi esnasında, bazı fonksiyonlar kullanarak, işlemlerimizi kolaylaştırabiliriz. Örneğin bir mali işlemde, faiz hesaplamalarını yapan bir fonksiyon girmeniz gerektiğini düşünelim. Bu işlem veritabanına veri girişi ile alakalıdır ancak, konu olarak kayıt girişi değil, fonksiyon tanımı olarak ifade edilir.

Aynı evre içinde birden fazla kavram – kuram bulunuyor ise, yani yapılacak işlemler, farklı konular – kategoriler altında toplanabiliyor ise, bu evre konuya göre bölünebilir.

★ Süre belirleme

Bir işlemin, ne kadar süreceğini belirleyebilmek için, o işin WBS'ını çıkartarak alt işlemlerinin sürelerini toplarız. Alt işlemlerin süresini belirlemek için tabiki deneyim gerekmektedir. Ancak, yinede, süre belirleme de zorlanıyorsa, o zaman, bu işlemi yeterince alt işlemlere bölmemişiz demektir. Unutmayalım ki, bir işlemin süresi, o işlemin alt işlemlerinin sürelerinin toplamıdır.



Figür 1.5.2 C : WBS örneği (2)

Yukarıdaki “İşlem A” işleminin süresi, doğal olarak alt işlemlerin sürelerinin toplamı kadardır yani ;

$$\text{İşlem A Süresi} = \text{Alt işlem 1 süresi} + \text{Alt işlem 2 süresi} + \text{Alt işlem 3 süresi}$$

şeklinde ifade edilebilir. O halde, eğer bir işlemin süresini saptayamıyorsak, alt işlemlerinin sürelerini saptayıp toplamını alırsak ve bir üst işlemin süresini hesaplamış oluruz.

Örneğin, arama motoru yazacağımızı ve bu arama motoru için ne kadar süre araştırma yapmamız gerektiğini düşünelim. Şuan bu süreyi bu işin uzmanları dahi bir anda hesaplayamayacaktır.

Ancak bu işlemi daha alt bileşenlerine, onları da daha alt bileşenlerine böldüğümüzde süre tahmininde bulunma daha kolay hale gelecektir.

NOT: Aşağıdaki teknik bilgileri bilmenize veya anlamınıza gerek bulunmamaktadır. Sadece, kompleks işlemlerin süresi hesaplanamazken, basit ve kısa işlemlerin sürelerinin hesaplanabileceği gösterilmek için kullanılmış olan bir örnektir.

Araştırma

- Varolan arama motorlarının sistemlerinin araştırılması

- o Indexing algoritmalarının araştırılması
 - Kelime grupları (isim, fiil, sıfat..)
 - Synonyms (Aynı anlama gelen kelimeler – eş anlamlılar)
- o Information Retrieval (bilgi getirme) sisteminin araştırılması
- o Crawler'ın (link toplayıcılarının) araştırılması
- o Page Rank (sayfa sıralama) algoritmalarının araştırılması
- o Lexical analyses (Dil analizinin yapılması) araştırılması
- Yeni sistemlerin araştırılması
- Varolan ve yeni sistemlerin karşılaştırılması – başarıları
- Gereken donanımın araştırılması
- İşin maliyetinin araştırılması

Yukarıdaki gibi bir hiyerarşi çıkardığınızda, 3. seviyede olan “kelime grupları” veya “synonyms” işlemlerinin süreleri çok kolay bir şekilde hesaplanabilecektir. İşte bu şekilde, süresini hesaplamakta zorlandığımız işlemleri bölerek, hesaplamak kolaylaşacaktır.

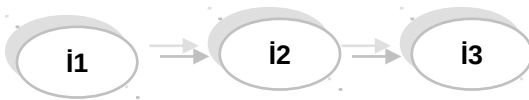
Ayrıca, eğer bir işlemin süresini internetten veya daha önce yapılmış olan araştırmalardan bulabiliyorsak zaten o süreyi yazabiliriz ve o işlemi daha alt işlemlere bölmeye gerek kalmaz.

1.5.3 AND

AND (Activity Network diagram), yani ağ aktivite diagramı; bir WBS'de belirtilen işlemlerin arasındaki **sırasal** ilişkiyi göstermek amacıyla kullanılan bir tür diagramdır.

AND , işlem sırasını, yani hangi işlemden sonra hangi işlemin geleceğini bildirir. Yukarıdaki örnekte, bir ay önce XY ürünü alan müşterilerin listesi bulunmadan, onlara ürünlerinin hatalı olduğu bildirilemez, ve bildirilemediği için değiştirme yapılamaz. İlk adım olarak, bir önceki ay, XY ürünü alan müşterileri bulmak gerekmektedir. Daha sonra, bu müşterilere bildirmek ve en sonunda, ürünlerini değiştirmek gerekmektedir.

İşte bu yüzden,



Figür 1.5.3 A: AND Örneği
i1: İşlem 1, i2: İşlem 2, i3: İşlem 3

Bu örnekte, her işlem bir evrede gerçekleşir. Bu tip durumlarda, dikkat edilmesi gereken husus, bir evrenin tamamlanmadan bir diğerine geçilemeyeceğidir. WBS'de işlem sırası gözükmez, sadece hangi işlemlerin, hangi alt işlemlerden oluştuğu görüntülenir.

Yukarıdaki diagramda, işlem 3'ün yapılması için işlem 2'nin yapılması ve işlem 2'nin yapılması için işlem 1'in yapılması gerektiği gösterilmiştir. Bu işlemi 3 farklı kişinin yapacağını varsayalım.

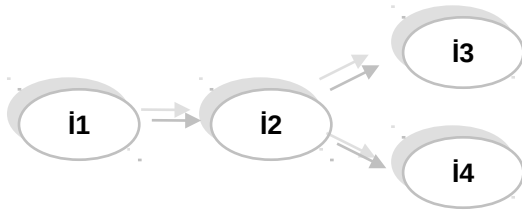
Öncelikle İşlem1, sonrasında İşlem2, en son olarak İşlem3'ün gerçekleşeceğini biliyoruz.

			
	İşlem 1'i yapacak kişi	İşlem 2'yi yapacak kişi	İşlem 3'ü yapacak kişi
İşlem 1 yapılıyor	İşlem 1'i yapıyor	İşlem 1'i yapan kişinin işi bitirmesini bekliyor	İşlem 1'i yapan kişinin işi bitirmesini bekliyor ki, İşlem 2'yi yapacak kişi işe başlasın ve bitirsin
İşlem 2 yapılıyor	Yapılacak işlem yok	İşlem 2'i yapıyor	İşlem 2'i yapan kişinin işi bitirmesini bekliyor
İşlem 3 yapılıyor	Yapılacak işlem yok	Yapılacak işlem yok	İşlem 3'ü yapıyor

Figür 1.5.3 B: Bir iş zaman çizelgesi

Bazı durumlarda, iki veya daha fazla işlem aynı evrede gerçekleşebilir. Örneğin; bir ürünün üretilip dağıtılma evresini ele alalım. Bu süreçte, ilk önce bir ürün üretiliyor, daha sonra stoğa yerleştiriliyor, ve stoktan yurt içi firmalara ve yurt dışı firmalara dağıtım yapılıyor.

Üretimi yap
Stoğa yerleştir
Yurt içi ve yurt dışı firmalara dağıtım yap



Figür 1.5.3 C : AND Örneği

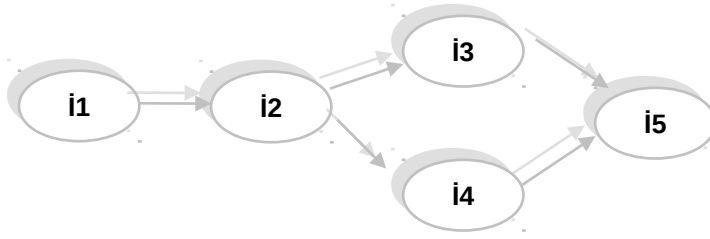
i1: Üretimi yap, i2: Stoğa yerleştir, i3: Yurt dışı firmalara dağıtım yap, i4: Yurt içi firmalara dağıtım yap

Yukarıda kullanılan örneklerde, “Üretimi yap”, “Stoğa yerleştir” .. gibi ifadeler kullanılmıştır. Tabiki böyle ifadeleri çalıştıracak bir dil yoktur. Yani bu şekilde cümleler halinde emirler veripte bu cümleleri (Emirleri) anlayarak o işlevi gerçekleştirecek bir sistem (programlama dili) yoktur. Bu örneklerde ifade edilmek istenen, o satırlarda, yapılacak olan işlemin açıklamasıdır.

Üretilen ürün, stoğa yerleştirildikten sonra, diğer işlemde (yurt içi firmalara dağıtım) bağımsız olarak “Yurt dışı” firmalara dağıtılabiliyor veya diğer işlemde (yurt dışı firmalara dağıtım) bağımsız olarak “Yurt içi” firmalara dağıtılabiliyor.

Yani, i3 ün gerçekleşmesi için i4 ün veya i4 ün gerçekleşmesi için i3 ün yapılmasına gerek yoktur.

Başka bir örnek olarak; bir otomobilin gidiyor olduğunu varsayalım. Daha sonra, bu otomobilin kaza yaptığını ve bundan sonra olası iki işlemten birinin (tamir edilme, yeni parçalar alınma) yapılıp, otomobilin sürmeye devam edildiğini varsayalım.

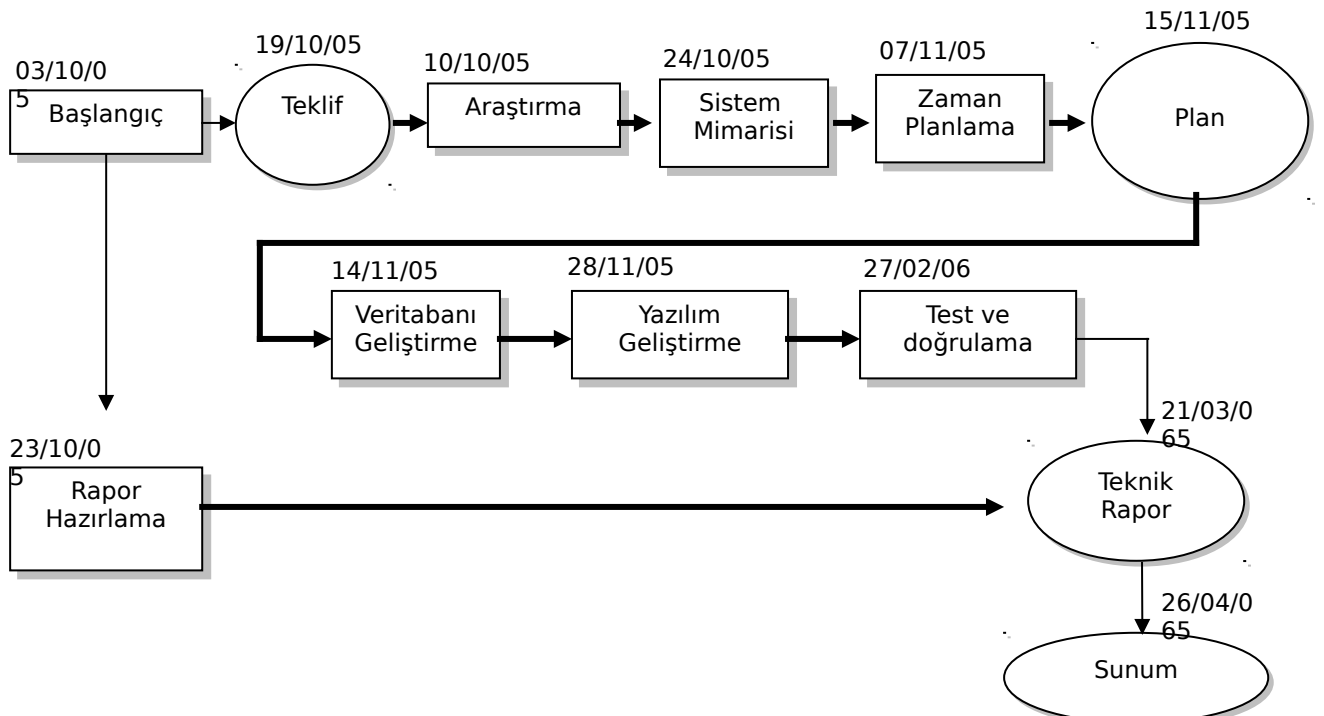


Figür 1.5.3 D : AND örneği

**i1 : Otomobil gidiyor, i2: Otomobil kaza yaptı, i3: Otomobil tamir edildi
i4 : Otomobilin parçası değiştirildi, i5: Otomobil yola devam etti.**

Görüldüğü üzere, i2 gerçekleşmeden, i3 veya i4 gerçekleşemez. Ancak i5'in gerçekleşmesi için hem i3 hem i4'ün gerçekleşmesine gerek yoktur. Sadece biri gerçekleşebilir ve i3'ten veya i4'ten, i5'e geçilir. AND bu şekilde işler ve bir olayın işlemlerini bu şekilde ifade eder.

Yukardaki WBS'nin(1.5.1 B) AND'si aşağıdaki diagramda gösterilmiştir.



Figür 1.5.3 E : Kapsamlı bir AND örneği

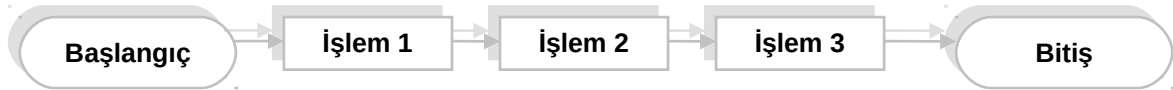
AND’de,

- elipsler : Rapor veya dökümanları belirtmek
- dikdörtgenler : İşlemleri belirtmek için kullanılır.

Dikkat ederseniz, **Başlangıç** işleminden iki farklı ok çıkmakta, biri, teklif dökümanı çıktısı vermekte, diğer ise, proje geliştiriliyor oldukça sürececek olan rapor yazımı işlemine bağlanmaktadır. Yani, bir yandan, (yukarıdaki) araştırma – geliştirme – test ... gibi işlemleri yaparken, aynı anda, raporun yazımı yapılacaktır.

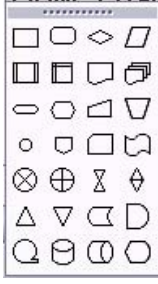
1.5.4 Flow Chart

Flowchart (iş akış diagramı) bir işi diagramsal olarak ifade etmek için kullanılır. Bir sorunun çözülmesi, bir sistemin çalışma prensibinin belirlenmesi için flow chart kullanılabilir. Flow chart’ın en büyük özelliği, bütün işlemlerin görülebilir olmasıdır. Bu durum, işlemlerin birbiriyle olan ilişkisinin anlaşılmasını kolaylaştırır. Bu konuyla alakalı internette birçok yazı veya kitap bulunabilir. Temel olarak flow chart, bir akış üstüne kuruludur (flow : akış). Aslında flow chart yapı olarak AND’ye benzer, çünkü flowchart ta hangi işlemin hangi işlemden sonra olacağı belirtilir yani işlem dilizişi durumu söz konusudur.








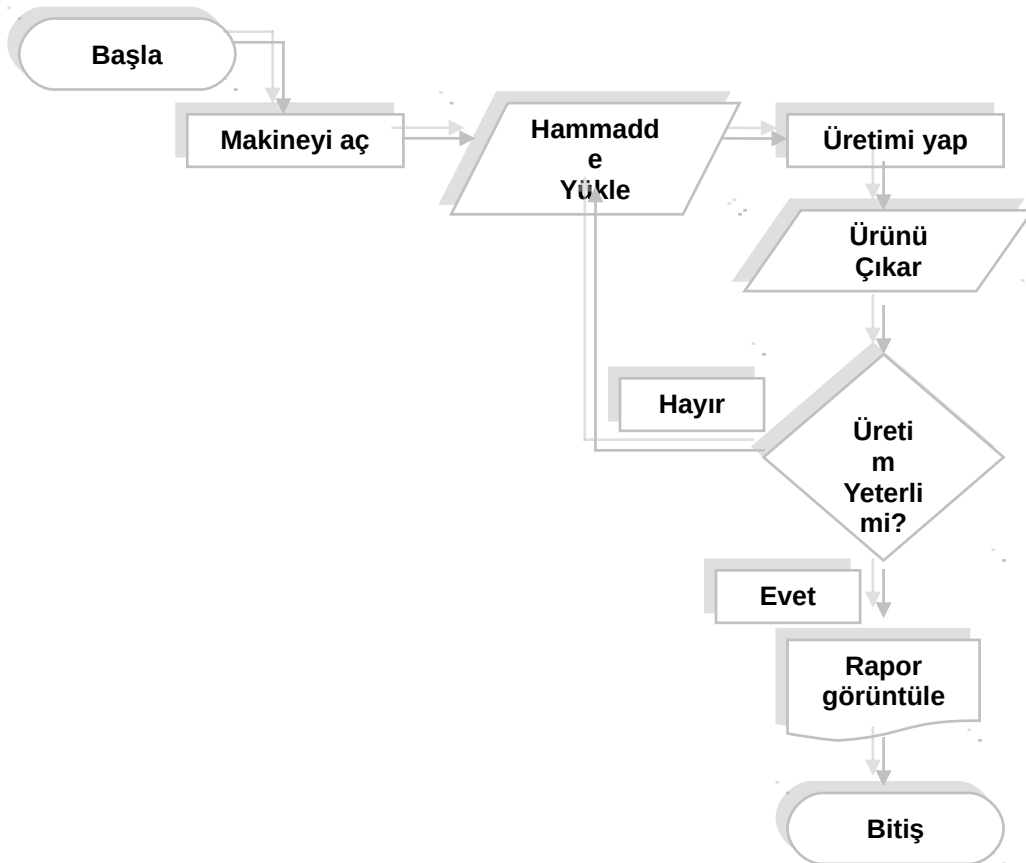
Figür 1.5.4 A : Flow chart

İşlemler, kutular içine yazılarak, başlangıç ve bitiş belirlenir, bununla birlikte işlemler bir sonrakine akarlar. Yukarıda, başlangıcın akabinde, sırayla, 1., 2., ve 3. işlem yapılır ve sistem çalışma modundan çıkar.



Figür 1.5.4 B : Flow chart elemanları

-  İşlem belirtir. Bu şekli çizdikten sonra, içine yapılacak olan işlem yazılır.
-  Başlangıç / Bitiş belirtmek için kullanılır.
-  Karar verme bölümü. Bir karar verip o karar göre hangi işlemi çağıracağını belirleyen işlem
-  Giriş / Çıkış. Bir materyalin giriş çıkışını ifade etmek için kullanılır.
-  Rapor görüntüleme için kullanılır.



Figür 1.5.4 C : Flow chart örneği

Yukarıdaki diagramda, ilk olarak makine çalıştırılır. Daha sonra ham madde girişi, ardından üretim yapılır ve ürün çıkar. Eğer yeterli ürün üretilmişse, rapor verilir ve işlem biter. Aksi takdirde, yine ham madde yüklenmesi yapılır ve üretim gerçekleştirilir.

Flow chart; genelde, sıra, karar, başlangıç ve bitiş işlemlerini belirlemede kullanılır.

1.5.5 PSEUDO ('sūdo) KOD

Pseudo kod, standartını bizim belirleyebileceğimiz, bir işlemi metinsel olarak ifade eden bir tür dildir. Ancak bir programlama dili değildir ve Pseudo dilinde yazılmış bir kodu çalıştıracak bir program yoktur. Yani pseudo kod bir tür talimat listesidir.

1.1 bölümünde; öncelikle NE yazacağımızı, daha sonra yazdıklarımızı NASIL programlama diline çevireceğimizi belirtmiştim. İşte yapacaklarımızın listesini, metinsel bir şekilde ifade etmek için pseudo kod denilen bu belirtme dilini kullanırız. Bir konuşma dilinde (ingilizce, Türkçe...) genelde 4 tip cümle bulunur:

Belirtme cümlesi	: Bir olayı veya yargıyı belirtir (<i>Ahmet eve geldi</i>)
Soru Cümlesi	: Bir bilgiyi öğrenmek için kullanılır (<i>Eve geldin mi?</i>)
Şart Cümlesi	: Bir şart ve sonuç belirtir (<i>Eğer yemeyeceksen ben yiyebilirim</i>)
Emir Cümlesi	: Bir işin emredilmesini için kullanılır (<i>Buraya gel</i>)

Daha önceki kısımlarda, amacımızın, bilgisayara, istediklerimizi söyleyecek emirleri verecek kodları yazmak olduğunu söylemiştim. Yani bilgisayar köle, biz efendi idik. Böyle bir durumda, bilgisayara bir işi yapması için emir veririz ve emir verirkende doğal olarak **emir cümlesi** kullanırız.

Bilgisayarada, bir konuşma dilinde kullandığımıza benzer emir cümleleriyle emir veririz. İşte bir işlemi gerçekleştirmek için, yapılacak alt işlemleri emir cümlesi şeklinde yazdığımız ifadeye pseudo kod denir. Birinci amacımız, bir sistemi analiz ederek, bilgisayarın o işlemleri gerçekleştirmesi için gereken kodları yazmaktır. İşte bu kodların, bir bilgisayar dilince çevrilmeden bir önceki haline pseudo kod denir.

Yukarıda yazmış olduğumuz işlemler pseudo kodu formatında ifade edilebilir.

```
Üretimi yap
Stoğa yerleştir
Yurt içi ve yurt dışı firmalara dağıtım yap
```

Pseudo kod, mantık kodudur, yapılacak işlemin içeriğini doğal bir dille ifade etme yoludur. Daha sonra, belirttiğim gibi, bu kod istenilen dile çevrilir ve yazılır.

Bu dilde, kod yukardan aşağıya doğru akar. İlk satırdaki işlem ilk, sonraki satırlardaki işlemler sonra gerçekleşir.

Bir işlem Flow Diagram ile ifade edilebileceği gibi pseudo kod ile de ifade edilebilir. Aslında yazdığımız her bir satır işlem bir pseudo kod satırıdır veya bir flow chart işlem kutusudur diyebiliriz.

Aslında yazdığımız herhangi bir ifade, pseudo kod olabilir ve bu kod evrenseldir. Yani, bir işlemi, bir programlama dili ile ifade ettiğinizde, o dili bilmeyen kişiler, o işlemi anlamakta zorlanabilir ancak eğer bir işlemi pseudo kod ile yazmışsanız, bu ifade, diğer insanlar tarafından daha kolay bir şekilde anlaşılacaktır.

Örneğin, bir dosya içinde kayıt arama yapan program yaptığımızı düşünelim:

```
Belirtilen bir dosyayı aç
Dosya içinde aranan metin var ise, bunu kullanıcıya bildir
Dosyayı kapat
```

Gördüğünüz gibi; bu işlemler, peşpeşe yapılır: belirtilen dosya açılır, dosya içinde aranan metin var ise kullanıcıya bildirilir, dosya kapatılır. 1.3 bölümünde verdiğimiz Winzip örneğini ele alırsak;

```
Belirtili dosyayı aç
Dosya içindeki bütün verilerin listesini çıkar
Dosya içindeki bütün verilerin listesinden en çok kullanılan 10
tanmesini seç (bir dosya içinde en çok kullanılan kelimeleri
seçmek gibi)
Bu seçilen 10 tane en çok kullanılan veriye birer kod ver
(kod1, kod2, kod3...)
Dosya içindeki en çok kullanılan veriler yerine, kodlarını yaz
(böylece boyut azalacaktır)
Dosyayı kapat
```

İşte yazacağımız sıkıştırma programı yukarıdaki koddan ibarettir. Gerisi, sadece çocuk oyuncağıdır. Örneğin, son bir kaç paragrafta geçmiş olan kelimeler içinde en çok kullanılanların tablosu aşağıda görüntülenmiştir.

bir	17
pseudo	6
dosya	5
çok	8
içinde	8
işlem	8
ile	4

Figür 1.5.5 A: Belirli bir metinde geçen kelimeler ve geçen kelimelerin kaç kere geçtikleri

Pseudo kod içinde belirtilen işlemlerde, aynı WBS'de olduğu gibi alt işlemlere bölünebilir. Böyle durumlarda başka kodlar yazılabilir.

Bu konu ile ilgili örnekler 1.8 bölümünde bulunmaktadır.

Alıştırma

- Kompleks sistemler nelerden oluşur?
- WBS nedir?
- İşlemleri için kaç tür bölme çeşidi vardır?

- AND nedir?
- Flow chart nedir?
- Flow chart elemanları nelerdir?
- Pseudo kod nedir, ne işe yarar?

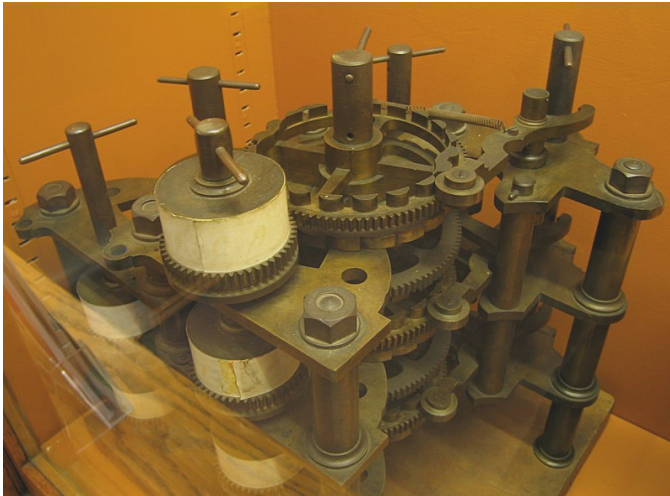
1.6 Programlama kısa tarihi ve yapısı

1830’larda başlayan bilgisayar ve programlama, sırayla;

- teknolojinin gelişmesiyle
- bilgisayarların güçlenmesi
- kompleks sistemlerin artmasıyla
- teferruatlı ve gelişmiş programlama dillerinin artmasıyla

ilerlemiştir. Yukarıdaki gelişmeler birbirlerine bağlı olarak ilerlemiş ve bugünkü halini almıştır.

İlk programlanabilir bilgisayar teorisini, Charles Babbage (ingiliz matematikçi – filozof – makine mühendisi) ortaya atmıştır. Babbage, bu konuda, yaptığı buluş olan: fark makinesini keşfetti. Bu makine, bir elektriksel değerler serisini otomatik olarak hesaplayabilme yetisine sahip olmayı öngörüyordu. Makineye geliştirmek için; sonlu farklar yönteminden yararlanan Babbage, çarpma ve bölme işlemlerinden yararlanmaksızın hesaplama yaptı [e14].



Figür 1.6 A : Fark Makinesi

Daha sonra, Ada Lovelace, Babbage’nin makinesini kullanarak, onu programladı (tabiki bir programlama dili ile değil, sadece donanımsal ifade olarak). Lovelace, dünyanın ilk programcısı olarak bilinir. Bilgisayar için, “It can do whatever we know how to order it”, yani, “bilgisayar, ona nasıl emredeceğimizi bildiğimiz herşeyi yapabilir” ifadesini kullandı.

Daha sonraki yıllarda, ünlü ingiliz matematikçi George Boole, Boolean cebirini geliştirdi. Bilgisayarın anlayabileceği iki ifade olan 1 ve 0, yani elektrik var, veya yok ifadelerinin temelini oluşturdu. Bugün birçok programlama dilinde kullanılan veri türlerinden biri olan “boolean” adı, George Boole’dan esinlenerek verilmiştir.

1.7 Efektif Programlama

Kitabın bu bölümünde, programlamanın kolay olduğunu ve hemen hemen herkes tarafından yapılabileceğini kavradıktan sonra, programlamanın değilde, iyi – yani efektif programlamanın önemli olduğunu söyleyebiliriz. Çünkü, eğer ki herkes program yazabiliyorsa, bu durum, program piyasasının değerini düşürür, ve bu programlar içinde en iyi olanı seçilir. Bu ilişki aynı; altın – demir ilişkisine benzemektedir. İkiside metal olmasına karşın aralarındaki değer farkı çok fazladır. Bunun en büyük nedeni, Altın elementinin doğada az bulunmasıdır. İşte bizim yapacağımız programlarda, altın gibi değerli olmalı, efektif olmalı ki, süre ve masraftan zarara uğratmasın.

Bir programın veya sistemin “iyi” olup olmadığını anlamak için, öncelikle o programın “iyi” olup olmadığını gösteren – ispatlayan kriterleri yani özellikleri bilmeliyiz.

Bir programın önemli özellikleri;

- Kullanılabilir olması (kullanılabilirlik)
- İsteklere cevap verebilmesi (tam istenilenlere göre tasarlanmış olması)
- Hata içermemesi (doğruluk)
- Kolay değiştirilebilmesi veya eklentilerin kolay yapılabilmesi (güncellenebilme)
- Güvenli olması (güven)
- Zamanında bitmesi (süre)
- Hızlı çalışabilmesidir (performans).

Tasarlanmakta olan birçok yazılım için en büyük problemler;

- geç tamamlanıyor olmaları
- kullanımlarının zor olması
- hatalar içermeleri
- daha sonra düzenlenememeleri

şeklinde sıralanabilir.

Birçok program;

- zamanı dışında tamamlanır
- kullanımı zordur
- hatalıdır
- değiştirilemez
- yanlış tasarlanmıştır

Dikkat ederseniz, bir program için önemli olan kriterler, o programı tasarlayan için en büyük problemlerdir. Yukarıdaki kriterleri biraz daha ayrıntılı açıklayayım.

Süre:

Akıp geçen zaman, aslında para ile ölçülemeyen hayatımızdanda parçalar alıp götürür. Bir projenin kısa sürede tamamlanabilmesi, hem zamanımızı harcamadığımızdan, hem de, müşterimiz beklemediğinden (ve vakit harcamadığından) çok önemlidir.

Bir projenin süresinden sonra tamamlanması, o proje için yaşanabilecek en büyük problemdir diyebiliriz. Müşteriler için, **SÖZ** çok önemlidir. Zamanında yetiştirilmeyen proje hem size, hem de müşteriye kayıp yaşatır. Tabi müşteriniz bu kayıp riskini almaz, bu riski sizin üzerine yükler.

Varsayalım ki bir yazılım firmanız bulunuyor ve bir müşteriniz sizden kendilerine bir muhasebe programı yapmanızı istedi (bespoke yazılım – firmaya özel yazılım). Siz, projesini – planını yapıp, bu işin 3 ay süreceğini (3 ay sonra tamamlanacağını) teklifinizde sundunuz. Teklifte, ödemenin, proje tesliminde (tamamlandıktan sonra) yapılacağını belirttiniz. Teklif kabul edildi ve daha sonra sizin şirketiniz projeyi geliştirmeye başladı. Ancak, bir tür nedenden ötürü, projeyi bu belirtili süre olan, 3 ay süre içinde bitiremediniz.

Bu durumda, yapmış olduğunuz anlaşma bozulmuş olacaktır. Yani daha sonra projeyi bitirseniz bile, müşterinin projeyi alacağı garanti değildir. Normalde baktığınızda, siz, üç ay boyunca, çalışanlarınıza boşuna maaş ödemiş hatta boşuna sigorta, boşuna masraf ödemiş olursunuz. Ancak, müşteri firmanın beklentisini, boşa çıkardığınız için, birde üzerine tazminat ödersiniz. Tazminatta, proje bedelinin %30’dur diyelim. Böylece, hem kendi masraflarınızı hemde tazminatı ödeyerek, çok büyük zarara girmiş olursunuz.

Tazminat ödersiniz çünkü, müşteriniz, size güvenmiştir ve o süreçte belki kendi kullanıyor olduğu programın kullanım süresi bitecektir ve akabinde, şirketinizin hazırladığı projeyi kullanmaya başlamayı planlıyorlar olabilirler ve bu durum gerçekleşmediğinde (projeyi tamamlayamadığınızda), müşterileriniz, başka bir yerden proje almak zorunda kalabilirler, bu sırada anlaşmalar yapılır, müşteriniz işlemleri zamanında yetiştiremeyebilir.... Bu durumun üzerine birde sözünüzde duramadığınız için, müşterinizi kaybettiğinizi hatta, bunun diğer müşterileriniz tarafından duyulduğunda kaybedeceklerinizi de eklerseniz, inanılmaz bir kötü durumla karşılaştığınızı söylemek en uygun ifade olur.

Peki eğer bespoke değilse, generic çalışıyorsanız? Varsayalım ki, bir Player (görüntü ve ses dosyalarını oynatan – çalan bir program, örnek Media Player, Winamp, ...) program tasarlıyorsunuz. Bu programı tasarlarken, web sitenize, 2 ay sonra bitireceğinizi yazıyorsunuz. Bu tip programları kullanan milyonlarca kullanıcı, web sitenizden bu haberi görüyor, ve kendilerini ona göre hazırlıyorlar. Bir müşterinin kaybedilmesinde en büyük etken, hayal kırıklığıdır.

Eğer, 2 ay içinde bu genel kullanıma sunacağınız projeyi sunmaz iseniz; birincisi, sizden daha sonra, aynı tip proje yapmaya başlayan, ve siz ilk başladığınız için önce olsanız bile, size yetişebilecekleri için, rakip sayınız artacaktır. Bununla birlikte, firmanızın imajı, milyonlar önünde azalacaktır ve kullanıcılar: “bu projeyi yapabilecek kadar profesyonel değiller” düşüncesine kapılacaktır. Ayrıca, bundan sonra yine bir proje üretecek olsanız ve bir süre belirtseniz, kullanıcılarınız yada müşterileriniz bu süreye inanmayacaklardır. Bu durum prestijinizi doğal olarak sarsacaktır.

Katkısından dolayı aziz dostum, İşletme Uzmanı, Önder BAKAL’a teşekkür ederim

Müşteri bizle olan ilk diyalogunda olaya kesinlikle ticari bakacaktır. Kişi yada kişiler bilinçaltında bizi deneyecek ve eğer beklediğini bulamazsa ilk darbeyi alacaktır. Müşteri alacağı ürün yada hizmette elbette maksimum faydayı arayacaktır. Ancak müşterinin başına gelebilecek en büyük talihsizlik hayal kırıklığı olsa gerek. Çünkü

özne, yani ana tema aranan özelliklerde değilse o cümle eksik kalacak belkide amacından sapmış olacaktır.

Globalleşen dünya artık, zaman, kimsenin kimseye muhtaç olmadan hayatını sürdürmeye doğru yönelmiştir. Öyleyse hiçbir müşteri bir kişiyi yada kurumu tekel olarak görmeyecektir ve görmemelidir de. Unutmayınız ki bilgiye ulaşabildiğiniz ölçüde varsınız hayat arenasında. Biz, arz edenlere düşen görev, bu hususta müşteriye beklediğinden öte var olabilmek ve beklediğinden önce ulaşabilmektir. Eğer müşteri aradığını bulamazsa onun adına hiç üzülmezin sizden daha iyisini daha kalitelisini ve daha ucuzunu her an bulabilir..

Peki, süreyi belirlerken nelere dikkat etmeliyiz? Bir işin süresinin belirlenmesinde WBS kullanmanız gerektiğini ve eğer bir işlemin süresini tam olarak belirleyemiyorsanız, o işlemi onu oluşturan daha alt işlemlere bölmeniz gerektiğini ifade etmiştim. Bir işin süresinin belirlenmesinde iki yaklaşım vardır diyebiliriz. İlk olarak, yazılım mühendisliğinin belirlediği standartlar bulunmaktadır. Bu standartlar, bir araştırmanın ne kadar süre içinde yapılması gerektiğini, bir kodun ne kadar sürede yazılması gerektiğini, bir planın ne kadar süre içinde yapılması gerektiğini açıkça belirtmektedir. Bu işlem aslında programlamacıyla alakalı değildir. Programlama yapan kişi yani kodcu sadece verilen işi yapmaktadır. Bunun için size, yazılım mühendisliği kitapları incelemenizi veya internetten, “time planning”, “time costing” konularını araştırmanızı veya “Algorithmic Cost modelling”, “expert judgement”, “estimation by analogy”, “Parkinson’s Law”, “pricing to win” konularını anlatan makaleleri veya kitapları okumanızı veya bu kitabı yazarken yararlandığım “Software Engineering, 6th Edition, Ian SOMMERVILLE” kitabını okumanızı tavsiye ederim. Başka bir seçenek olarakta, bu kitap serisinin ikincisini okuyabilirsiniz.

İkinci yaklaşım hem kendinizin, hemde çalıştığınız ekibin kod yazma / tasarlama / planlama hızını kendinizin ölçmesidir. Ancak tasarlıyor olduğunuz projenin içinde yapılacak olan işlemlerin hepsi aynı değildir. Örneğin bir projede, kullanıcı arabiriminin tasarlanması ile birlikte, kod yazımı veritabanı tasarlama süresi gibi farklı süreçte tasarlanabilen işlemler bulunmaktadır. İşte bu yüzden program geliştirme hızınızı veya başkalarının program geliştirme hızını ölçerken, bu ölçümleri belirli gruplara göre bölmelisiniz ve bu grupların ayrı ayrı istatistiğini tutmalısınız. Örneğin;

- Standart form tasarımı (veri tabanı uygulamaları için)
- Kompleks form tasarımı (media player gibi – winamp gibi)
- Bileşen tasarımı
- Standart kod yazımı (kod olayları ve bağlantılar için)
- Sistem hierarşisinin kurulması
- Algoritma yazımı
- Veritabanı işlemleri
- Sınıfların tasarlanması
- Diğer kodlar
- Veri tabanı tasarımı
- Veri tabanı veri girişi
- Kontroller & Testler

- Araştırma
- Planlama

gibi gruplara ayırabilirsiniz. Tabiki unutmamız gereken, bu işlemlerin bir planlama (yani nasıl yapılacağını belirttiğimiz süre) süresi birde gerçekleştirme süresi olduğudur. Bu tip hesaplamalarla birlikte yapmanız gereken;

- değerlerin maksimum olanı
- değerlerin minimum olanı
- sapma oranları
- ortalama (3 farklı şekilde ortalama bulunmaktadır)

değerlerini de hesaplamaktır. Bu değerlerin hesaplanabilmesi için, İstatistik konusunda araştırma yapmanızı (standard deviation, varriance, mean, histogram.... konularını) tavsiye ederim. İyi bir yazılımcı olmak için bahsedilen bütün bu konularda bilgili olmanız gerekmektedir. Kitabın konusuyla direkt olarak alakalı olmadığından bu konuda sadece önerilerde bulunuyorum. Ancak yazılım mühendisliği ile alakalı bütün konuları bir kitaba sıkıştırmanın imkansızdır.

Bu konu haricinde dikkat edilmesi gereken en önemli nokta, genel süreye, test süresini ve test süresinden sonra projedeki hataların tekrar yazılma süresini de ekledikten ve diğer süreleri ekledikten sonra, bu süreyi %5-%15 arası arttırmamızdır. Eğer bu süreyi çok arttırsak müşteri kayıp ederiz, çünkü müşteri bizim belirttiğimiz süreden daha az sürede yapabilecek başka bir firma bulabilir. Eğer, erken bitirir ve erken sunarsak, o zaman, müşteri, bizim, müşteriden daha fazla ücret almak için süreyi uzattığımızı düşünecektir. Eğer süreyi normalden az tutarsanız, zaten olabilecek en büyük problem buradan kaynaklanacaktır. Çünkü belirttiğiniz sürede yetiştiremezseniz tazminat ödersiniz, tazminat ödememek için hızlı kod yazdığınızda ise birçok hata oluşacaktır. O yüzden, zaman planlama çok önemlidir. Ayrıca, çok az sürede yapacağınızı söylemek, bir müşteriyi tedirgin edecektir. Sizin, gerekli özeni göstermeden projeyi bitirmeyi planladığınızı (zamanı az söylediğiniz için) düşünebilirler.

Bunun haricinde, zaman planlamaya; evlenecek olan, izne çıkacak olan, hamile olan, askere gidecek olan çalışanlarınız da etki etmektedir. Projeye başladığınızı ve akabinde, 2 kişinin yıllık izne çıkmak istediğini, veya birinin evlendiğini veya birinin çocuk sahibi olduğunu düşünelim. Böyle bir durumda, kaybedeceğiniz vakti önceden hesaplamaz iseniz, daha sonra çok büyük problemler yaşayabilirsiniz. Vakit kayıplarına ek olarak resmi tatilleri de eklemeniz gerekmektedir.

Zaman, en değerli kavramdır. Birçok Türk firması sahibi, ne yazık ki, “ben çalışıyorum, kazanıyorum, kazandığım bana yetiyor, o zaman şirket iyi çalışıyor” mantığına esir olmuştur ve zamandan yapılacak kâr önemli değil gibi gözükmektedir. Unutmayalım ki, **yeterince kâr edememekte, zarardır**. Bu nedenle zaman kriterine çok önem vermeliyiz.

Kullanılabilirlik:

Kullanılabilirlik, süre kadar olmasada çok önemli bir problemdir. Müşterileriniz, kullanamadıkları veya anlayamadıkları ürünlere ücret ödemeyecektir yani satın almayacaktır. O yüzden, kullanılabilirlik; programın özelliklerinden veya işlevlerinden daha önemlidir.

Kilitli bir kutu içinde hazineniz olduğunu düşünün. Eğer onu açıp kullanamaz iseniz bir işe yaramayacaktır. Aynı şekilde, istediğiniz kadar mükemmel bir program tasarlayın. Eğer o program kullanılamıyorsa, veya özellikleri anlaşılamiyorsa, o kadar uğraşı boşuna vermişsiniz demektir.

Bir roman yazarı, karakterleri ve hikayeyi kafasında oluşturur, ve karakterlerin bütün özelliklerini, hikayenin gidişini belirleyeceği için, romanın “yönetmeni” ‘dir. Halbuki, bir okuyucu, kitabı eline alsa ve roman karakterleri tanıtılmamış olsa, direkt olarak konuya girilmiş ve olaylar başlamış olsa, doğal olarak kitabı anlayamayacaktır (anlamakta zorlanacaktır). Yani nasıl yazarlar karakterleri; tanıtıyorsa, bizde yazdığımız programları tanıtmalıyız ve programlarımızı kullanabilir bir halde tasarlamalıyız. Aslında kullanılabilirlik, bizim için çok kolay görünür fakat en zor evredir. Çünkü, programı tasarlarken, “bu özelliği, ben kolay bir şekilde anlayabiliyorum, o zaman kullanıcı da anlayabilecektir” düşüncesine kapıldığımız her an, programda kullanılabilirlik açısından bir eksik yapıyoruz demektir.

Program yazmaya başladıktan sonra bilgisayar profesyoneli konumuna geleceksiniz. Belki de şuan öylesiniz. Ancak profesyonellerin, amatörler gibi düşünmesi çok zordur. Profesyonel olduğunuzda, diğer kullanıcılarında, profesyonel olduğunu; en azından amatör olmadıklarını ister istemez düşüneceksiniz ve o şekilde, yani kullanıcıların da size karmaşık gelmeyen durumları kolayca anlayabileceğini düşüneceksiniz. Bu bütün profesyonellerin başına gelen bir durumdur. İşte bu nedenle kitap yazanlarda, bir konuyu iyi anlatmadan sadece örnekle geçirirler. Yani okuyucunun hemen o olayı kavradığını düşünürler. Fakat amatörler, herşeyi, profesyoneller kadar bilseler zaten amatör olmazdılar.

Bilimadamları, tezlerini doğrulamak için, denekler kullanırlar. Örneğin, beynin çalışma yapısı ile ilgili bir konuyu araştıran bir bilimadamını düşünelim. Bilimadamı bu konuyu araştırır, bazı fikirler ve tezler ortaya koyar. İşte bu yüzden, geliştirdiği tezleri kendisine asla uygulayamaz. Çünkü bir soru için olası cevapları (kendi tezine göre) biliyordur ve vereceği cevaplar anlamlı olmayacaktır, sadece kendi tezini destekleyecektir. Ancak bir bilimsel tezin ispatlanması için birçok farklı deneğin kullanılması gerekmekte ve bu deneklerin konuyu bilimsel olarak araştırmamış olmaları gerekmektedir.

Belki bazılarına saçma gelecektir ancak, **programcılar ile kullanıcılar düşmandır** diyebiliriz. Çünkü programcılar, çok açıklama yapmadan, çok kolaylık sunmadan, standart ifadeleri açıklayarak programı yazmak isterler. Kullanıcılar ise, en ince ayrıntının bile anlatılmasını, en basit işlemin gösterilmesini, en basit işlemin daha da kolay ifade edilmesini isterler. Programcılarda bu istekleri yapmak istemezler.

Kullanılabilirliğin de bir maliyeti bulunmaktadır. Programı yazıp verdiğiniz müşterideki kullanıcılar, eğer kullanılabilirlik yeterli değilse, anlayamadıkları için, firmanızı sürekli arayacaklardır ve sürekli teknik destek isteyeceklerdir. Hatta, bazen, teknik servis elemanınızın müşterinizin firmasına gidip, sorun oluşturan işlemleri, bizzat göstermesi gerekebilir. İşte burada teknik servis elemanlarına verilen ücretler, teknik destek vermek için müşteri firmaya gidilmesi esnasındaki ulaşım masrafları, kullanılabilirliğin maliyetleri arasındadır.

Kullanılabilirlik faktörünü iyi hale getirebilmek için, daha önceden hazırlanmış olan benzer programları, kullanıcılara göstererek, hangilerinin daha kolay ve anlaşılır olduğunu belirleyebilir ve sizde, bu plana göre kullanılabilirliği kolay bir proje geliştirebilirsiniz. Eğer kullanıcılar daha önceden yapmış olan bir programın kullanılabilirliğini beğenmez ise, o

zaman sizin onlardan daha kolay ve daha anlaşılır bir proje tasarlamamız gerekmektedir. Bu tasarımı yaparken yine daha önceki programlardan yararlanabilirsiniz. Kullanıcıya, hangi bölümlerin anlaşılmadığını ve daha kolay nasıl olacağını sorarak sonuca ulaşabilirsiniz.

Tabiki bu konu üstünde birçok fikir geliştirilebilir. Örneğin; programın bir bölümünü tasarladıktan sonra, programı, müşteri şirkete gidip, kullanıcılara sunabilirsiniz. Alacağınız yorumlar ve istekler doğrultusunda, programı biraz değiştirip, daha sonraki bölümlerde bu aldığınız yorumlara göre tasarlayabilirsiniz.

Başka bir yöntem olarak, bir kullanıcıyı programın geliştirilme süresinin belirli bir evresinde (örneğin form tasarlamada), müşteri firmadan, belirli bir süreliğine yanınıza alabilirsiniz. Siz yaptıkça, kullanıcıya sunarsınız ve daha kolay nasıl anlayabileceğinizi gösterirsiniz. Yani kullanıcıyı bir denek gibi kullanabilirsiniz.

Tabi birde, insan psikolojisini iyi bilmek gerekiyor. Bu konuda bir uzmandan destek almanızda fayda var. Belki sizde, benim gibi yüzbinlerce internet sitesine girmişsinizdir. Birçoğunda, sadece belirli bölümleri okursunuz ve diğerlerini önemsemezsiniz. Bu konuda, internet sitelerinin optimizasyonu (yani daha iyi işlev gerçekleştirilebilmesiyle) ile ilgili, Brad Callen'ın, bir çok makalesi bulunmaktadır. Özellikle insanların bir web sitesine girdiklerine hangi bölümlere baktığını, hangi bölümlerin ve yazıların ilgilerini çektiğini elirttiği bir görüntüsü bulunmaktadır. Bu konuda öğrendiklerinizi, form tasarlamada da kullanabilirsiniz. Ayrıca görsel iletişim konusunda araştırma yapmanızı tavsiye ederim.

Güvenilirlik:

Efektif programlamaya etki eden bir diğer faktörde, güvenilirliktir. Güvenilirlik, bir firmanın bilgilerinin, şifrelerinin, işlemlerinin ... kendi izin verdiği kullanıcılar tarafından kendi belirlediği izinlerde kullanılması anlamına gelmektedir. Dışarıdan bakıldığında kolay gibi görünen bu etken, malesef EN BÜYÜK problemlerden biridir ancak genel olarak, kullanılabilirlik kadar önemli değildir. Bill GATES'in dediği gibi gelecekte önemli kriterlerden biri güvenliktir. Çünkü her tür programda güvenilirlik son derece önemli değildir.

Yukarıda ifade etmek istediğim, güvenlik kriterinin, her programda aynı önemde olmadığıdır. Bir banka yönetim programında, güvenlik en üst derecedeyken, kişisel bir ajanda da güvenlik daha az önemlidir.

İki tür güvenilirlik problemi bulunmaktadır. İlk tür güvenlik problemi, verilerin, karışık hale gelmesi – silinmesi – gerektiğinde değiştirilmemesi, yani gereken bir işlemin yapılmaması veya yanlış bir işlemin yapılması sonucu bir verinin yok olması durumudur. Varsayalım ki, müşterileriniz, tasarladığınız programa fatura bilgilerini giriyorlar. Eğer girildiği düşünülen bir fatura, veritabanına eklenmemiş olursa, daha sonra o faturayı veritabanında bulamayacakları için problemler oluşabilecektir. Bu duruma programın güvenilir olması denir.

Başka bir örnek vereyim: Müşterinize, yaptığınız yeni programı kurmadan önce, müşterinizin kullandığı önceki programdaki verileri, yeni programın veritabanına AKTARmanız gerekiyor. Aktarma, çok büyük bir problemdir. Çoğu zaman tabloların yapısı, kolonlar uymadığı için bazı kayıtlar tam olarak aktarılmaz, bazıları karışır... Böyle bir durumda, eski programın kullandığı veritabanındaki bütün verilerin tam ve eksiksiz bir şekilde yeni veritabanına aktarmanız gerekmektedir, müşterileriniz size burada “güven”

duymalıdır ve bu güveni siz bütün ayrıntıları inceleyerek ve yanlış yapmayarak sağlamalısınız.

İkinci tür güvenlik problemi ise; sizden(programcıdan) değil başkalarından kaynaklanabilecek problemlerdir. Yani, üçüncü bir kişinin, izinsiz olarak, müşterimizin verilerine ulaşması; görüntülemesi – değiştirmesi – silmesi - eklemesi durumudur.

Son 5-10 yıl içinde popüler hale gelen hacking – hacker sözcüklerini birçoğumuz duymuştur. Duymamışlar için açıklayayım. Hacking, bir başkasına ait bilgisayar – programa – veritabanına girerek, bir firmanın, kuruluşun veya kişinin kullanıcı bilgilerini çalarak, verileri değiştiren – silen – okuyan kişilerdir.

Müşterileriniz, tasarlamış olduğunuz programa bir anlamda, “şirketlerini” emanet etmiş olurlar. O yüzden kuracağımız sistem sağlam ve güvenilir olmalıdır. Güvenilir olmalıdır ki, hackerlar, sisteme girip verileri çalamasın.

Hackerlar, yazdığınız programın, kullanıcıların veya sistemin açıklarını bularak sisteme girerler. Kullanıcıların açıklarını engellemek en kolayıdır. Kullanıcıların hangi noktalara dikkat etmesi gerektiğini belirtmek, şifrelerini kolay – tahmin edilebilir değilde, karmaşık seçmelerini belirtmek ve bazı genel güvenlik önlemlerini almak yeterli olacaktır.

Programın açıklarını kapatmak bizim için en zor evredir. Birçok filmde, yazarların sözlerinde, düşmanımızla savaşırken, düşmanımızın açıklarını (zayıf noktalarını) bulmanın ve bu açıklardan faydalanmanın önemi belirtilir. Ancak, hackerlar zaten bizim açıklarımızı bulup, yazdığımız programları hack ediyorlar. Ayrıca, hackerların kullandığı sabit bir sistem, yöntem bulunmuyor. İşte bu yüzden, düşmanımızın açığını bulmaktansa, düşmanımızı tanımak ve düşmanımızın yapabileceklerini öğrenmek en iyisi olacaktır. Yani hackerlar siz yöntem geliştirdikçe, yöntem geliştirirler.

Bu durum, “**insan yapısı**” ve “**yaratıcı yapısı**” örneğine benzemektedir. Yaptığımız projeler, “insan yapısı” olacaktır ve insanlar hata yaptıklarından, potansiyel olarak, insanların yaptıkları hatalı olacaktır. Halbuki insanlar, “yaratıcı yapısı”dır. Yani hackerlar, sabit olan bir programı kırmak (ele geçirmek) için birçok yöntem geliştirebilirler.

Eskiden, otomobil hırsızlarının, herhangi bir ekipman taşımadığını zannederdim. Daha sonra, çalınmasın diye, sigortaları (elektrik sigortaları) çıkartılmış bir otomobili çaldıklarını öğrendim. Demek ki, hırsızlarda, otomobil çalmaya hazırlıklı geliyorlar.

İşte böyle olduğu için, yukarıda bahsettiğim gibi, düşmanımızı tanımamız gerekir. Bir doktor, ilaç geliştireceği zaman, önce, hastalığa yol açan mikrobu çok iyi tanıması gerekir. Tanınmalıdır ki, o mikrobun, açıklarını veya negatif yönlerini öğrenebilsin, nasıl – hangi bölgeye saldırdığını anlayabilsin ve buna karşı önlem alabilsin. İşte aynen bir doktorun yaptığı gibi bizde, düşmanımızın yani hackerların ne yaptığını sistemleri nasıl ele geçirdiğini öğrenmeliyiz ve buna göre çözümler üretmeliyiz.

O yüzden, zamanında, bende, hacking sitelerine, forumlarına girmiştim, hacking tekniklerini araştırmıştım. Böylece, hackerların kullandıkları yöntemleri, yani sistemleri nasıl ele geçirdiklerini, öğrendim ve bu yöntemleri engellemek için gerekli önlemleri gerçekleştiren programları yazacak bilgiye sahip oldum. Size de, aynı yöntemi öneririm. Bunun için,

forumlara, web sitelerine, kitaplara bakarak gerekli bilgiye sahip olup, hackingi elimizden geldiği engelleyebilirsiniz.

Değiştirilebilir olması:

Dünya değişiyor. Dünya değiştikçe, ihtiyaçlar, metodlar, yöntemler, kurallar... da değişecektir. Bu değişiklikler oldukça, yazılım kullanıcılarının istekleri değişecektir ve yapmış olduğunu projeleri daha sonra değiştirmek – yenilemek – düzenlemek – eklemeler yapmak zorunda kalacaksınız.

Hiçbir zaman, bir projenin, en iyisini yaparak piyasaya sunmaya kalkmayın. Tasarladığınız programın hatasız olmasına dikkat edin ancak aklınıza her gelen özelliği, programa eklemeyin. Yoksa bu özelliklerin sonu gelmez ve asla tasarladığınız programı bitiremezsiniz. Daha sonraki sürümleri düşünün, bu özelliklerin bazılarını daha sonraki sürümlerde kullanıcıya sunabilirsiniz.

Bunun haricinde, şüphesiz ki programlar, hatasız tamamlanamayacaktır ve bu hataları düzeltmek için programın daha sonradan kolay değiştirilebilir ve anlaşılabilir olması gerekmektedir. Aslında bu madde, ikinci madde olarak gördüğümüz kullanılabilirliğin, programcılar için olanıdır diyebiliriz.

Varsayalım ki bir yazılım firmasında çalışmaktayız. Eğer yazdığımız programın kod kısmını sadece bizim anlayacağımız şekilde tasarlarsak, bizden sonraki diğer programcılar programı anlamakta, değişiklik yapmakta zorlanacaktır.

Ayrıca, müşteriniz, sürekli değişiklikler, eklentiler isteyebilir. Yapılacak olan her değişikliği müşteri için olduğu gibi sizin içinde bir maliyeti olacaktır. Böyle bir durumda yapmanız gereken maliyeti düşürmek, yani, değişikliklerin daha kolay yapılmasını sağlamaktır.

Bir projenin değiştirilebilir olması, o projenin, parça parça – bölüm bölüm tasarlanması ile mümkün olur. Bu konu yazılım mühendisliğinin önemli konularından biridir. Bu konu ile ilgili ayrıntılı bilgiler, bir sonraki serimizde anlatılacaktır.

Hata içermemesi:

Şüphesiz ki, bir ürünün bozuk – defolu – hatalı olması, karşılaşılabilecek en büyük problemlerden biridir. Bu konu ile ilgili fazla ayrıntı vermenin gerekli olduğunu düşünmüyorum. Hepimiz bir ürün aldığında ve o ürün çalışmadığında – yada işlevini yerine getirmediğinde, moral bozukluğu yaşamışızdır. Hataların azaltılması ile ilgili gerekli bilgiler sekizinci bölümde verilmiştir.

İsteklere cevap verebilmesi:

Kullanılacak olan program, yapacak olduğu işi tam olarak, eksiksiz yapabilecek şekilde tasarlanmalıdır. Örneğin bir bilgisayar, gerekli programları çalıştırmıyor ise, o bilgisayar görevini yapamıyor anlamına gelir. Kullanıcının programdan istedikleri, kurulacak olan sistemin ayrıntılarıdır. Bu kriter, diğer kriterler kadar önemlidir. İstenilen işlemi yapamayan bir programın anlamı olmayacaktır.

Bir cep telefonundan, kişilerin telefon numaralarını kaydetmesi (rehber) gerekmektedir. Rehberi olmayan bir cep telefonunu düşünün. Böyle bir cep telefonu kimse tarafından tercih edilmeyecektir. Bu konu ile ilgili ayrntı vermenin gerekli olduğunu düşünmüyorum. Hepimiz, bir ürün aldığında o ürünün, gerekli işlemleri yerine getirmesini umar ve ister. Bu nedenle, müşteriler, ürünü almadan, o ürünün teknik özelliklerini araştırır.

Diğer kriterler

Bu kriterler haricinde, yeni teknolojileri yeni sistemleri takip etmek ve bu sistemlerin özelliklerini bilmek, tasarlayacağınız sistemin için en uygun olanını seçmek çok büyük önem taşımaktadır. Başka bir kriter, kişinin, çok iyi bir programlayıcı olması gerekmektedir. Aslında burada önemli olan, gerçek hayatta olmakta olan olayların bilgisayar ortamına aktarılmasıdır ve bu işlemi gerçekleştirecek kişinin, şuan okuyor olduğunuz veya benzeri bir kitap okuyarak, bir sistemin nasıl programlandığını çok iyi bilmesi gerekmektedir.

Modüler programlama

Modüler programlama, uluslararası kabul görmüş, mühendislik tasarım stratejisidir. Öyle ki, bu stratejide, büyük sistemler tek bir parça yerine, birbirinden ayrı ancak bağlantılı daha küçük ünitelerden meydana gelmektedir.

Bu konu hakkında daha önceki kısımlarda (kompleks sistem – basit sistem, analiz..) bilgi vermiştik. Şuan ise sadece bütün bu verilen bilgileri tek bir çatı, modüler programlama altında topluyoruz.

Modüler kodlama yapmak;

- **Geliştirme**
Proje planlı bir şekilde bölündüğü için ayrı modüller birbirinden bağımsız ekipler tarafından geliştirilebilir.
- **Yeniden kullanılabilir kod**
Proje planlı bir şekilde tasarlandığı için, proje içindeki bazı kodlar, kütüphaneler ve sınıflar benzer işlemleri gerçekleştirecek olan programlarda da kullanılabilir.
- **Bakım**
Sistemin tamamen değiştirilmesine veya parçalanmasına gerek kalmadan hataların olduğu bölümler rahatlıkla değiştirilebilir.
- **Uzantı oluşturabilme**
Sisteme, yeniliklerin katılması, sistem baz alınarak başka projelerin geliştirilmesi konusunda modüler kodlama yapmak çok büyük kolaylık sağlayacaktır.
- **Doğruluk**
Sistem, birbirine bağlanmadan test edildikten sonra, birbiriyle ilişkilendirildikten sonra tekrar test edildiği için modüler olmayan bir sistem göre daha doğrudur ve daha sağlamdır.

... avantajlarını sağlamaktadır.

Yukarıda bahsi geçen konular hakkında ayrıntılı bilgiler bu kitabın bir sonraki sürümünde bulunacaktır. Ancak söylediğim gibi isterseniz hem internetten araştırma yapabilir hemde başka kaynaklardan yararlanabilirsiniz.

1.8 Örnek Programlama

İlk bölümden beri verdiğimiz örneklerin haricinde, artık yavaş yavaş nasıl programlama yapabileceğimizi görmek, ve program yazabilecek yetimiz olduğu gerçeğine daha yakın olabilmek için bazı ek örneklerin gösterilmesi gerekmektedir. Aşağıdaki örnekler içinde hem güncel hayatta karşı karşıya kalabileceğiniz hemde diğerlerinden farklı, ilginç program örnekleri bulunmaktadır.

1.8.1 Örnek: El ile yazılmış gibi yazı yazan bir program

Size, kitabı yazarken yapıyor olduğum ve aklıma gelen, basit ama hoş olabilecek bir program örneğini göstermek ile başlayacağım. E-posta, E-Ticaret, E-İş, E-Randevu....., Dünya ülkelerinin hızla e'leştiği günümüzde, artık neredeyse her 'şey' bilgisayara kayıt ediliyor. Teknolojinin birçok faydası olmasına karşın, hayatımızdan bazı değerli ve önemli varlıkları – duyguları alıp gidiyor. Ancak bazı şeyler, orjinal tutulmalı, örneğin günlükler gibi... Bu proje, bilgisayarda yazdığımız bir metni, sanki sizin elinizden yazılmış gibi yazılı hale getiriyor.

Örneğin; “test deneme metni” şeklinde bir metin girdiğimizde, kendi yazımızla (aşağıdaki örnekte bir öğrencinin yazı stili kullanılmıştır) yazılmış bir resim oluşturup bu resmi görüntüleyecektir.



Figür 1.8.1 A: Bilgisayar tarafından bir kişinin el yazısı ile yazılmış bir metin örneği

Peki nasıl oluyorda, bilgisayarda yazdığımız bir metin, kendi yazımız gibi yazılıyor ve yukardaki şekilde bir resim ortaya çıkıyor?

Yine ilk bölümden bir hatırlatma yapalım. Bilgisayar olmasaydı, yukarıdaki işlemi nasıl yapardık? Yani varsayalım ki bir arkadaşımız var, ve bu arkadaşımız bir kızdan hoşlanıyor, fakat ona açılmıyor. Bizde, ona yardımcı olmak adına, arkadaşımızın elinden yazılmış gibi bir mektup yazıp bu (arkadaşımızın beğendiği) kıza göndermeyi planlıyoruz. Bu durumda benim aklıma iki yöntem geliyor (Dikkat: eğer bahsi geçen bayan, olumsuz cevap verirse, bu durumdan yaptığımız program suçlu olmayacaktır!) :

- Arkadaşımızın yazısını taklit ederek bir mektup yazmak. Bu yöntem her zaman başarıya ulaşmayacaktır. Çünkü, taklit olduğu kolay bir şekilde anlaşılabilir. Ayrıca, bir yazıyı, bir kişinin kendi yazısına benzetmek çok zordur. Arkadaşımız gibi yazıyorken bir anda dalıp kendi yazımız ile yazabiliriz ve bu durumda yazdıklarımızın anlamsız kılacaktır.
- Daha orjinal bir yöntem olarak, arkadaşımızın daha önceden yazdığı metinlerin olduğu sayfaların fotokopisini çekip, tek tek gerekli harfleri keserek, bir yazı haline getirebiliriz. Sonrada bu yazıların üstünden (gazlı kalemle – kalın olursa çok daha iyi sonuç verecektir) geçip iyice gerçeğe yakın hale getirebiliriz.

Bir keresinde televizyonda bir dizide (çok eskilerde), bir şahıstan fidye istemek için el yazısı belli olmasın diye gazetelerden ve kitaplardan kesilmiş harflerin kullanıldığını görmüştüm. İşte bizde aynı yöntemi kullanabiliriz.



Figür 1.8.1 B: Çeşitli gazetelerden kesilmiş harflerle bir metnin oluşturulması

Bu işlemi gerçekleştirmek için ilk olarak, arkadaşımızın harfleri nasıl yazdığını bilmeliyiz yani kullanacağımız her bir harfi, arkadaşımızın nasıl yazdığını bilmeliyiz. Örneğin, yukarıdaki örnekteki (test deneme metni) ifadeyi yazabilmek için, arkadaşımızın yazdığı harfleri (aşağıda), bulmalı ve yazılı kağıttan kesmeliyiz.

mnstdei

Figür 1.8.1 C: Bir kişinin el yazısından çıkarılan bazı harfler

Daha sonrasında ise, bu harfleri gerektiği kadar çoğaltıp, yan yana koyarak metni elde etmeliyiz. Bu işlemi bilgisayarda gerçekleştirmek için, ilk olarak, arkadaşımızın yazdığı metinlerin olduğu bir veya birkaç sayfa buluyoruz. Sonrasında tarayıcı kullanarak bu sayfaları (resim olarak) bilgisayara aktarıyoruz. Bundan sonraki işlem ise, aktarılmış sayfalarda bulunan harflerden, sırayla kullanacağımız harfleri tek tek çekip, farklı dosyalar halinde kayıt etmeliyiz. Yani, (örneğin) bir “a” bulup bu resmi bir resim dosyası olarak (a.gif, a.png, a.jpg, a.bmp) şeklinde kayıt etmeliyiz. Bu işlemi, her bir harf, rakam ve işaret için yaparak, herhangi bir metin yazabilmek için hazır hâle geliriz.

Sonrasında ise, yazacağımız program, girilen metni, tek tek harflere ayırır, her bir harf için, gerekli resim dosyasını yükleyip, ana resmimizin (kağıdımızın) üzerine yapıştırır.

Bu uygulamada dikkat edeceğimiz nokta koordinatlı iyi ayarlamaktır. Unutmayalım ki bir daktilodaki gibi, her karakter yazdıkça, sayfa bir miktar sola doğru kayar. Aynı şekilde, satır bittiğinde, bir sonraki satır için, ruloyu biraz daha aşağıya kaydırmamız gerekir. Bu şekilde, yatay düzlemdeki koordinata x, dikey düzlemdeki koordinata y isimlerini verelim. Unutmayalım ki, sayfa, her bir karakterde, yaklaşık olarak 1-3 cm sağa kayacak, her bir satırda yaklaşık olarak 4-6 cm aşağı kayacak, ve yine her satırda, satır başına geçecektir!



Figür 1.8.1 D: Daktilonun işleyiş şekli

El yazısı ile yaz :

```
Girilen metindeki her bir satır için
Her bir satır içindeki her bir karakter için
Her karakter için, o anki karakteri yaz
Her karakterden sonra kağıdı, 2 (örneğin) cm sağa kaydır
Her satırdan sonra, son yazılan noktayı, sıfırla, (satır başına
al!)
Her satırdan sonra, kağıdı 5 cm aşağı kaydır
```

Yukarıdaki kodun nasıl programlama dilince çevrileceği sonraki bölümlerde anlatılacaktır.

Böyle bir programı yazabilmek için temelde yapabilmemiz gereken sadece yukarıdaki ifadeyi yazmaktır. Tabiki daha önceden de belirttiğim gibi yukarıdaki kodu çalıştırabilecek bir dil yoktur. Bundan sonra yapmanız gereken, yukarıdaki metinlerin, kullanacağınız programlama dilinde, nasıl ifade edildiğini bulmaktır.

1.8.2 İnteraktif dosya yönetimi

Bu program, birden fazla kişinin bir proje üzerinde, birden fazla farklı yerde çalıştığı bir firmanın dosyalarını yönetmek için tasarlanabilir. Örneğin; bir inşaat firması, aynı anda, şantiyede, ana ofiste ve fabrikada çalışma yapıyor. Öyleki; ofistekiler, proje ile ilgili çizimleri yapıyor, şantiyedekiler, işi gerçekleştiriyor ve yapılanları not edip bildiriyor ayrıca, gerekli olan malzemelerin listesini çıkarıyor, fabrikadakiler, gerekli üretimleri yapıyor, yine üretimlerle ilgili gerekli malzemelerin listesini çıkarıyor, gelen çizimlerde bazı değişiklikler yapabiliyor.....

Böylesine karışık bir işlemde, onlarca dosya aynı anda kullanılacaktır. Varsayalım ki, bir malzeme listesi dosyası, birden fazla mühendisin bilgisayarında bulunuyor. Şantiyeden bir kişi, bir mühendisi arayıp, bir malzemenin gerekli olduğunu bildiriyor ve mühendis kendi bilgisayarındaki dosyaya bu malzemeyi ekliyor. Sonra, şantiyeden başka bir kişi, başka bir mühendisi arayıp, bir malzemenin arttığını ve sonraki sipariş için gerekli olmadığını söylüyor, sonrasında aranan mühendis kendi bilgisayarındaki dosyayı açıyor ve listeden, belirtilen malzemeyi çıkarıyor. Varsayalım ki bu işlemler başka mühendisler tarafından da gerçekleşiyor. Sonrasında, gerekli malzemeleri tedarik etmek istediğimizde de, önümüze birçok farklı dosya çıkıyor!!! İşte böyle bir durumu engellemek için bahsedilen programı tasarlayalım.

Sunucu : Sunucu (İng. Server), internet üzerinde sürekli olarak çalışan ve www.sayfa.com şeklinde açılan web sitelerinin gerekli içerik dosyalarının tutulduğu gelişmiş bilgisayarlardır. Böylece, siz ne zaman nereden www.sayfa.com yazarsanız yazın, istenilen siteye ulaşabilirsiniz.

Böyle bir problemi çözmek için benim aklıma gelen, bütün dosyaların herkesin bilgisayarında tutulacağı ancak, herhangi bir kişinin bilgisayarındaki dosyada değişiklik olduğu anda bu değişikliğin otomatik olarak diğerlerine de bildirilmesi gerektiğidir. Ayrıca eğer bütün dosyaların, her değiştirilmeden sonra birer kopyasını internet üzerinde bir sunucu da yedeğini alırsak, yanlış bir değişikliktede geri alma olanağımız olacaktır. Sonuç olarak istediğimiz bütün gerekli dosyaların hepsinin, herkesin bilgisayarında olması gerektiğidir.

Not: Şantiye, fabrika ve ofisin farklı şehirlerde hatta farklı ülkelerde olabileceğini unutmayalım. Dikkat etmemiz gereken dosya transferlerini internet üzerinden yapmamız gerektirir.

Programımızın yapacağı görevleri listeleyelim (bir nevi pseudo kod diyebiliriz) :

İşlem 1. Bir kişi bilgisayarında dosya oluşturduğunda (yeni bir dosya kayıt ettiğinde), bu dosyanın aynısı o anda, diğer bütün şirket çalışanlarının bilgisayarına otomatik olarak kopyalanacaktır. Böylece diğer çalışanlarda hangi dosyaların kullanıldığını görebilecektir.

İşlem 2. Dosyalardan birinde herhangi bir değişiklik olduğu anda bu değiştirilmiş dosya, öncelikle internet'e (transfer edebilmek için) aktarılacak sonrasında ise, internetten, diğer bütün kullanıcıların bilgisayarlarına aktarılacaktır.

Sistemi analiz ettik, yani sistemin NEler yapacağını belirledik. Şimdi bu işlemlerin NASIL yapılacağını yazalım.

İşlem 1:

Eğer bir dosya, sunucuda yoksa, ve o kişinin bilgisayarında varsa o zaman o dosya yenidir demektir. Bir dosyanın oluşturulup oluşturulmadığını anlamak için belirli aralıklarla belgelerin içinde bulunduğu klasörleri, kontrol etmeliyiz. Sürekli olarak, bilgisayar açık kaldığı sürecek, belirli aralıklarla (örneğin: 5 saniyede bir..) kontrol etmeliyiz. Aynı şekilde, eğer sunucuda olupta, bilgisayarda olmayan bir dosya var ise bu dosya bilgisayara indirilmelidir.

Dosya Organizasyonu (yeni oluşturulan dosyaların indirilmesi ve yüklenmesi) :

Sürekli kontrol et (işlem sürekli olarak gerçekleştirilmek zorundadır)
Her bir dosya için; eğer sunucuda (internette) olmayan ve bilgisayarda olan bir dosya var ise bu dosyayı internete yükle (upload).

Sürekli kontrol et
Her bir dosya için; eğer sunucuda (internette) olupta bilgisayarda olmayan bir dosya var ise bu dosyayı internetten, bilgisayara indir (download).

İşlem 2:

Bir kullanıcı dosyasında değişiklik yaparsa, bu dosya sunucuda olsa dahi, yeni, yani değiştirilmiş hali yinede sunucuya yüklenmelidir.

Eğer sunucuda bir dosya varsa, ve bu dosya kullanıcının bilgisayarında olsa dahi, sunucudaki dosyada değişiklik yapılmışsa, bu dosya kullanıcının bilgisayarına indirilir.

Dosya Organizasyonu (değiştirilen dosyaların tekrar yüklenmesi ve indirilmesi) :

Sürekli kontrol et

Her bir dosya için; eğer sunucuda olan dosyanın son deęiřtirme tarihi bilgisayardakinden daha sonra ise, yani sunucudaki dosya, deęiřtirilmiřse, bu dosyayı bilgisayara indir

Sürekli kontrol et

Her bir dosya için, eğer bilgisayardaki dosya deęiřtirilmiřse, bilgisayardaki dosyayı, sunucuya yükle

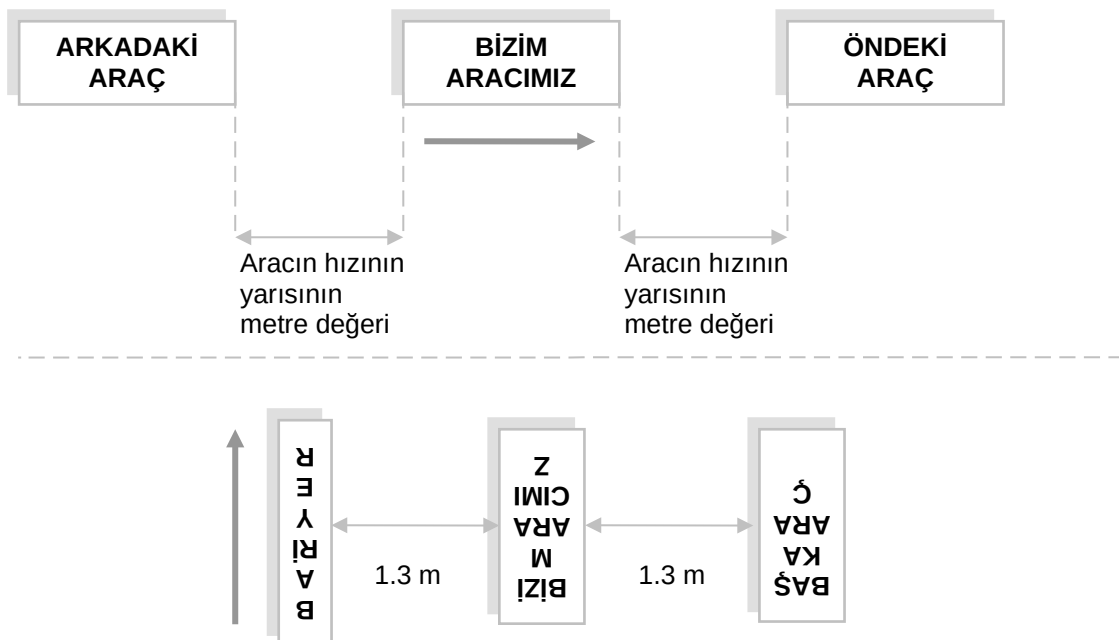
1.8.3 Otomobil uyarı sistemi

Geliřen teknoloji ile birlikte, trafik kazalarını veya olası kaza durumlarını önceden belirten, tehlikeli bir hakeret yaptığınızda veya tehlikeli olabilecek bir durum olduğunda sizi uyaran sistemleri düşünelim.

Varsayalım ki bir otomobil firmasında çalışıyoruz ve otomobiller hareket halindeyken, herhangi bir yönde bulunan araç veya bariyerlerle arasında olan mesafe belirli bir miktarın altına inerse, buna göre, sürücüyü uyaran bir sistem tasarlıyoruz.

Bu konuyla ilgili teknik bilgi olarak;

- Sürücü, bu uyarı sistemini aktive etmek için bir butona basar
- (Ehliyet alanlar bilir) Aracın, önündeki araç ve arkasındaki araç ile arasındaki mesafe, kilometre olarak belirtilen hızın, yarısının metre cinsindenki hali olmalıdır. (örnek, 80 KM hızla gidiyorsak, hem önümüzdeki ile hemde arkamızdaki araç ile aramızdaki fark 40 metre olmalıdır)
- (Sol veya sağdaki) bariyerlerle veya başka bir otomobille arasındaki yatay fark 1.3 metreden az olmamalıdır.
- Bütün işlemler m(metre) olarak yapılmalıdır.
- Yukarıda belirlenen kurallar geçildiğinde sarı alarm verilir. Ancak eğer gerçekte ölçülen deęerler yukarıda belirlenen deęerleri %10 geçerse, yani otomobil ile başka otomobiller veya bariyerler arasındaki yatay fark 1.3 m'nin %10 eksięi olan olan 1.17 m'yi geçerse, KIRMIZI alarm verilir. Aynı şekilde, eğer ön ve arkadaki araç ile olan mesafe hızın yarısının metre cinsindenki deęerinin %10 azındanda daha aza inerse (örneğin, 80 KM hızla giderken, mesafe, 40 metrenin %10 eksięi olan 36 metreye düşerse..) kırmızı alarm verilir



Figür 1.8.3 A: Otomobil uyarı sistemi - kontrol şeması

İşlem 1:

Sürekli kontrol et
0 anki hızı ölç, ikiye böl
Eğer öndeki araç ile aradaki fark hesaplanan değerden küçükse, sarı alarm ver
Ancak eğer, öndeki araç ile aradaki fark, hesaplanan değerinden %10 kadar daha küçükse, kırmızı alarm ver
(Aynı durumu arkadaki araç için yazacağız)

İşlem 2:

Sürekli kontrol et
0 anki sağdaki ve soldaki mesafeleri ölç
Eğer ölçülen mesafelerden biri, 1.3 metreden az ise, sarı alarm ver
Eğer ölçülen mesafelerden biri, 1.17 metreden az ise, kırmızı alarm ver

Daha benzeri birçok uygulamayı, ileriki bölümlerde göreceğiz. Burada önemli olan, yapılacak işlemi ifade edebilecek metinleri yazabilmek.

Birinci bölümde kullanılmış olan terimlerin ingilizceleri

akıl: mind	nesne: object
algoritma: algorithm	operatör: operator
analiz: analyse	öneri - tekli: proposal
anlama: understanding	planlama: planning
araştırma: research	problem: problem
başlangıç: start up	program: program
bilgi: information	programlama: programming
byte: byte	proje: project
çıktı: output	rapor: report
çözüm: solution	sistem: system
deneyim: experience	süre: time
dil: language	şema: schema
diyagram: diagram	tanımlayıcı: identifier
dosya: file	teknik: technical
döküman: document	tümdengelim: deduction
durum: state - condition - case - situation	tümevarım: induction
en düşük limit: lowest limitation	veri: data
fonksiyon: function	yapay: virtual - artificial
geliştirme: development	yazılım: software

gerçek: real girdi: input ilerleme: progress iş: work işlem: operation - process karışık: complex kodlama: coding mantık: logic mantıksal: logical matematik: maths - mathematics	zeka: intelligence
--	--------------------

SONUÇ

İlkokulda hemen hemen herkesin duyduğu bir söz vardır “Matematikte problemi anladıysan, sorunun yarısını çözmüşsün demektir.”. Küçükken sözler önemsiz ve anlamsız gelir ancak büyüğünce onları anlarız, küçüklere anlatmaya çalışsakta onlar malesef ifade ettiğimizi kendileri daha sonra anlarlar.

Bir projeyi geliştirmek, onun işlemlerini belirlemek için, projeyi tasarlamak gerekir. Yukarıda AND, WBS, Flow Chart, Pseudo kod konularına giriş yaptım. Bu ifade araçlarına ek olarak Gannt Chart ve UML konusunda da bilgilerin bulunduğu bölümler bulunmaktadır.

Ancak bizim şimdiye kadar çıkarmamız gereken, programlama için önce çalışma prensibini, nasıl çalışacağını bulmak gerekir. Nasıl bir makine mühendisi, bir makineyi açtığında, onun nasıl çalışabileceğini hangi işlemleri nasıl gerçekleştirdiğini anlayabiliyorsa, bizimde, işleyişte olan bir sistemi önce kavramamız gerekmektedir.

Sakin bazı bölümleri kaçırdım anlamadım, şuan program yazamıyorum ... gibi düşüncelere kapılmayın çünkü şimdiye kadar ki bölüm sadece girişti. Bundan sonraki bölümlerde, önceki bölümlerde anlattıklarımı nasıl planlı ve programlı olarak hayata geçireceksiniz, dış dünyada olan çalışan bir sistemin programını nasıl yazacaksınız ... benzeri konuları anlatacağım.

Programlamaya ilk başladığımda, bilgisayarına bir programlama dili olan Visual Basic kurmuştum ve bir pencere açtıktan sonra, pencere üstüne bir buton yerleştirmiştım. Butona çift tıkladıktan sonra, önüme bir listenin geleceğı, ve o listeden yapmak istediğim işlemi seçeceğimi zannediyordum. Örneğın, dosya aç, dosya kapa, mp3 çal, dosya böl..... Sonra karşıma, kod ekranı çıkınca şaşırdım. O şaşkınlıktan dolayı bu anımı hiç unutmuyorum. İşte burada belirtmek istediğim, tabiki emir cümlelerimizi anlayacak bir programlama dilinin olmadığıdır. Ancak, o satırlarda yazılması gereken emirleri ifade edebiliyorsak gerisi sadece CODE WORK, yani kodlamadır.

İlk başta, bir köle robotunuzun olduğunu düşünün. Ancak normal olarak şıandaki robotlar gibi düşünme kabiliyetinin olmadığını, o yüzden, bütün emirleri sizin vermeniz gerektiğini varsayın. Sizin için emir vermek kolay olacaktır. İşte programlama bu kadar kolay bir işlemdir.

2

Programlama Sistemi

İçerik

- 2 Programlama ilkeleri
 - 2.1 Programlama akışı
 - 2.2 Neleri programlarınız
 - 2.3 Matematiksel ifadeler - Programsal ifadeler
 - 2.3.1 Değişkenler
 - 2.3.2 Veri Türleri
 - 2.3.3 Değişkenlerin tanımlanması
 - 2.3.4 Matematiksel İşlemler & Operatörler
 - 2.3.5 Array
 - 2.3.6 Parantezler
 - 2.3.7 Toplam sembolü & Çarpım sembolü
 - 2.3.8 Döngü
 - 2.3.9 Matematik mantığı & Önergeler
 - 2.3.10 Koşul ifadeleri
 - 2.3.11 Fonksiyonlar
 - 2.3.12 Diğer döngü tipleri
 - 2.3.13 Sabit değerler
 - 2.4 Programın yüzleri
 - 2.5 Program & Durum

Amaçlar

- İkinci bölüm,
 - temel programlama ilkelerini,
 - değişkenleri, matematiksel ifadeleri, koşullu ifadelerini,

- parantezleri, döngüleri, toplam sembolünü, çarpım sembolünü, fonksiyonları, önermeleri anlatmaktadır.

Anahtar sözcükler

matematik, değişkenü parantez, toplam sembolü, çarpım sembolü, döngü, mantık, önermeler, koşul ifadeleri, fonksyionlar

2.1 Neleri programlarız?

Hatırlatma 1 Unutmayalımki, ilk bölümde bahsettiğimiz gibi, eğer bir sistemin – bir işlemin pseudo kodunu yazabiliyorsak, yani hangi işlemleri gerçekleştireceğini yazabiliyorsak, bundan sonra yapmamız gereken tek şey, bu metin ile ifade edilmiş pseudo kodlarını program kodlarına çevirmektir. *(Lütfen, Hatırlatma 1, HT1 ifadelerini kullandığımda bu paragrafta yazılı olanları hatırlayınız!)*

Kitabın açıklamasında, kitabın 2 amacının olduğunu ve bu amaçların: bir program yazmak için

**(1. amacımız) hangi metinleri nasıl yazacağımızı öğrenmeliyiz
sonrasında ise bu (2. amacımız) metinleri nasıl bilgisayar dillerine
çevireceğimizi öğrenmeliyiz**

olduğunu belirtmiştim. Birinci bölümde, 1.amacımızın nasıl gerçekleştirilebileceğini anlattım. Şimdi ise sıra, 2. amacımızı öğrenmeye geldi.

Programlama dillerinin, hatta işletim sistemlerinin (windows XP, windows 2000...) en önemli özelliği bazı işlemleri sizin yerinize yapmalarıdır. Yani herkes tarafından ortak kullanılan veya sıklıkla kullanılan işlemleri, programlama dilleri bizim için hazırlayarak yazmaktadırlar. Aksi takdirde, bilgisayarın ve programlamanın icat olmasından bu yana, bilişim konusunda bir gelişme söz konusu olamazdı. Her insan, kendisinden daha önceki insanların keşfetmiş olduğu icatları ve teknikleri kullanır, bulunan teknikleri tekrar keşfetmek için zaman harcamaz ve böylece bilim konusunda ilerler, aksi takdirde her insan en baştan başlar ve öldüğünde sonrakilere hiç bir faydası olmazdı. Bu durum bilimde de aynıdır. Bilimadamları, daha önceki tezleri, teorileri, çalışmaları da önemseyerek ve onlar üzerinden fikirler üreterek yeni sistemler icat etmişlerdir.

Bir mühendisi de işin, basit fakat uzun süre alan kısmından uzak tutan, işi tarif edip; işin, çalışanlar tarafından anlaşılmasıdır. Aksi takdirde, bütün işi bir mühendisin yapmasını bekleyemeyiz. Mühendisler işin, planını – stratejisini tasarlarlar ve işçilere yapmak üzere anlatırlar. İşte programlama da bu şekilde, planını yaptığımız işin, programlama diline anlatılmasıyla gerçekleşmektedir.

Her dil, yazıldığı bir üst dilden daha fazla işlem gerçekleştirir ve programcılara daha çok kısaltma imkanı sağlar. Bu sistemi, bir üst düzey yetkilinin, orta düzey yetkiliye bir görev vermesi, sonra bu görevinde alt düzey bir yetkiliye verilmesi olarak görebiliriz. En basit şekilde, sizin yazıyor olduğunuz 5 satırlık kod, aslında binlerce satır tutabilir.

Bu şekilde baktığımızda, programlama dilinde olan bir işlemi bir daha programlamamıza gerek bulunmamaktadır. Birinci bölümde, lowest limitationdan bahsetmiştik. Bir işlemi, o işlemin bir operatör tarafından yapılabilecek seviye gelinceye kadar bölünmesi gerektiğini söylemiştik. Biz, sadece programlama dilinde olmayan ve bizim için gerekli olan işlemleri programlamalıyız. Gerisi zaten programlama dili tarafından otomatik olarak yapılmaktadır ve bizim **Amerika'yı baştan keşfetmemize gerek yoktur**. Bizde, bir programı yazarken, programı yazıyor olduğumuz programlama dilinin yapabileceği işlemleri lowest limitation yani alt limit olarak almalıyız. Çünkü, programlama dilinin yaptığı bir işi bizim düşünmemize, planlamamıza, tasarlamamıza gerek bulunmamaktadır. Bu kodlar

usta programcılar tarafından hazırlanmıştır ve artık o dilin bir parçasıdır. Usta kodcular tarafından yazılıyor olsa bile hatalar içerebilmektedir. Ancak hatalar içerse bile, bizim bu kodlara müdahale şansımız zaten yoktur. O yüzden, dili tasarlayan programcıların yazdığı kodlara güvenmeliyiz.

Örneğin, pseudo kodda yazdığımız;

```
Pseudo İşlem 1
Pseudo İşlem 2
Pseudo İşlem 3
```

işlemlerini düşünelim (bu işlemlerin, -örneğin- dosyayı aç, dosyayı sıkıştır, dosyayı kapat gibi bir programlama algoritmasının pseudo kodu olduğunu unutmayalım). Sonra bu işlemleri programlayacağımız programlama dilinin, yapabileceği işlemleri, yani anlayabileceği ifadeleri düşünelim. Varsayalım ki; dosya açma işlemi için “İşlem A”, dosya kapama işlemi için “İşlem D” şeklinde birer kod kullanıyoruz.

```
İşlem A
Pseudo İşlem 2
İşlem D
```

Programlama dili tarafından anlaşılabilmesini bildiğimiz “Pseudo İşlem 1”, “İşlem A” olarak, “Pseudo İşlem 3”te “İşlem D” olarak ifade edilmiş ve kodlanmıştır. Ancak, Pseudo İşlem 2’yi tek bir işlem ile gerçekleştiren bir kod bulunmadığından bu işlemi yapacak olan alt işlemlere böleriz

```
İşlem A
    İşlem B
    İşlem C
İşlem D
```

Bu durum, ilk bölümde vermiş olduğumuz Kompleks Sistem -Basit sistem örneğine benzer. Buna ek olarak bizde bazı prosedürler yazarak, daha kompleks sistemleri basitleştirebiliriz. Zaten programlamanın temeli, basitleştirmekten ibarettir. Çünkü, bilgisayar, insan beynine göre çok aptaldır. Bu yüzden, bir işlemi, olabildiğince basitleştirerek, bilgisayarın anlayabileceği seviyeye getirmek gerekir.

Başka bir örnek olarak; bir otomobili kullanırken, vites değiştirmek için, debriyaj pedalına basarsınız. Debriyaj bu sırada, krank mili ile, tekerlekler arasındaki bağlantıyı kesmektedir. Ancak, debriyaj pedalına bastıktan sonra, debriyajın neler yaptığının önemi **bulunmamaktadır**. Siz, sadece, debriyaja basarsınız ve gerisini önemsemezsiniz.

Başka bir örnek olarak; çamaşırları, çamaşır makinesine attıktan sonra, makine, içindeki çamaşırları, kaç kere sağa döndürmüş, kaç kere sola döndürmüş, kaç litre su doldurmuş.... önemsenmez. Yani siz ayarı yaptıktan çalışma düğmesine bastıktan sonra geri kalanların (yapılanların) önemi yoktur.

Sonuç olarak diyebiliriz ki, bir işi tek başına, yardım almadan, bizden sadece çalıştırma direktifi aldıktan sonra yapabilen bir operatör varsa, biz sadece çalışma emrini veririz, gerisini önemsemeyiz.

Yazıyor olduğumuz kod en sonunda elektriksel sinyallere dönüşür, tabi ki bu arada birçok evreden geçecektir. Ancak bizim için önemli olan, programın çalıştıktan sonra ne yaptığı değil bizim ne yaptığımızdır.

Bir insana çözebileceği ve anlayabileceği sorular sorulur veya anlayabilecekleri anlatılır. Örneğin bu kitaba, bilmediğiniz bir bölümle başlasaydım, anlamamız çok zor olurdu. Bir programlama dilinde anlayacakları sınırlıdır, ve o belirtili sınırdan işlem yapabilir.

Programlama, programlayacak olduğumuz sistemin, programlama dilinin anlayacağı seviyelere kadar bölünmesini de içermektedir. İlk bölümde verdiğim sıkıştırma ve bölme programlarını hatırlarsak, dış dünyadaki bir nesne olan defteri, bilgisayar içinde olan bir nesneye benzetmiştik. Dış dünyadan bir nesneyi bilgisayar içinde tanımlarken sadece bazı özellikleri, olayları, metodları tanımlamalıyız. BAZI diyorum çünkü, dış dünyada bir nesnenin yüzlerce özelliği, metodu olabilir ancak eğer yaptığımız işle ilgili değilse, bu özellikleri tanımlamamıza gerek bulunmamaktadır.

Örneğin, sıkıştırma örneğinde, defterin sayfalarının çizgili veya kareli olması defterin bir özelliğidir. Ancak, konumuzda yeri olmadığı, daha doğrusu işleme etkisi olmadığı için bu özellik önemsenmez. Buradan çıkaracağımız sonuç olarak, programlama da sadece gerekli olan, duruma veya işleyişe etkisi olan kavramları işin içine katmaktır.

Programlama dillerinin içerisinde, belirli kodlar zaten yazılmıştır yani sizin kullanımınız için hazırlanmıştır.

Kütüphane Programlama dilleri bilgisayarınıza kurulduklarında yanlarında, kütüphane (**dll uzantılı – dynamic link library – dinamik bağlantı kütüphanesi**) dosyalarıyla birlikte gelirler. Bu kütüphane dosyalarının içinde programlama dili tarafından kullanılan veya programlama dili tarafından kullanıcıya komut olarak sunulan işlemler bulunmaktadır.

Bu olayı bir örnekle açıklayayım. Örneğin; bir otomobil üretme fabrikasını düşünelim. Bu fabrika yurt dışından gelen otomobil parçalarını kullanarak, birleştirip otomobil üretiyor olsun. Dikkat ederseniz, hazır olarak gelen otomobil parçalarının nasıl çalıştığını bilmeye, onları kontrol etmeye, onların içini anlamaya çalışmaya gerek bulunmamaktadır. Biz sadece parçaları birleştirme ve gerekli diğer işlemleri yapmakla yükümlü oluruz.

Programlamada da işlemler aynı şekilde işler. Bir programlama dilinin gerçekleştirdiği işlemde nelere yapıldığını bilmemize gerek bulunmamaktadır. Tabi ki çok daha iyi bilgi sahibi olmak ve mükemmel programcı seviyesine ulaşmak için bu bilgileri bilmek faydalı olacaktır. Ancak giriş ve orta seviye için bu bilgilerin önemi bulunmamaktadır.

Programlamayı bildiğinizi ve benim bunu sadece keşfedeceğimi daha önce belirtmiştim. Programın mantığını tasarladıktan sonra, kodunu yazabilmek için, programlama dilinin bize ne gibi kısaltmalar sağladığını bilmek gerekir. Çünkü bu kısaltmalar olmadan programlama yapmamız neredeyse imkansızdır (çok uzun süre alacaktır). O zaman temel programlama dilleri ne gibi ifadelerden ne gibi kısaltmalardan oluşuyor onları görelim.

Peki ya nasıl programlarız?

Ziya Paşa'nın ünlü bir sözü vardır: “Lafla uslanmayanın hakkı, tekdir, tekdir ile uslanmayanın hakkı dayaktır”. Bu sözden anlayabileceğimiz, bir kişiye anlayacağı şekilde davranmamız gerektiğidir. Aynı şekilde, bilgisayarla da anlayacağı dilde konuşmamız gerekmektedir. Tabi buradan, programlayamadığımız bilgisayara vuracağımız – onu pencereden aşağıya atacağımız anlamı çıkarmamalıdır. Bilgisayarları programlamak için onların dilinden konuşmalıyız yani bir programlama dili kullanmalıyız. Lady Lovelace'in “It can do whatever we know how to order it”, yani, “bilgisayar, ona nasıl emredeceğimizi bildiğimiz herşeyi yapabilir” ifadesini hatırlayalım.

Bilgisayarla konuşabilmek için yüzlerce programlama dili ve sistem bulunmaktadır. Dünya üzerinde de onlarca konuşma dili bulunmaktadır. Bu kadar farklı konuşma dili olmasına rağmen, bütün bu diller birbirlerine çevrilebiliyorsa (örneğin, İngilizce'den, Almanca'ya) hepsinde kullanılan temel yapı ortaktır. Yani bütün dillerde, bir cümle yapısı vardır, cümle yapısıyla birlikte, özne, tümleç, yüklem, zarf, fiil, sıfat, zamir.... gibi kelime türleri de bulunmaktadır. Dikkat etmemiz gereken nokta, bir (konuşma) dille ifade ettiğimiz olayı başka bir dillede ifade edebilmemizdir. Aynı şekilde bir programlama dilinde ifade ettiğimiz olayı – emri başka dillerde de ifade edebiliyoruz. Burada önemli olansa, kitabın başından beri tekrarladığım sistemin mantığıdır.

[Ara verme bölümü]

Alıştırmalar

- Programlama dillerinin görevi nedir?
- Neleri programlarız?
- Kütüphane nedir? Ne işe yarar?

2.2 Matematiksel ifadeler - Programsal ifadeler

Bu bölüme kadar, teknik konulara giriş yapmamıştım. Sadece bazı teorik bilgileri vermiştim. Bu bölüm ise, artık programlamanın terimlerinin anlatılmasının başladığı bölümdür.

Programlama ifadeleriyle birlikte matematiksel ifadeleri de gözden geçirelim. En kolay öğrenme metodu “**benzetme**” dir. Size baştan, döngü, değişken, koşul gibi kavramları öğretmeye çalışırsam (ki bazı büyük dersaneler, bazı kitap, bazı üniversitelerdeki hocalar bu şekilde uygular) konuyu anlamakta zorlanırsınız. Ancak, beynimiz için sihirli bir kelime olan “**gibi**” (yada **aynı**) sözcüğünü kullanarak, öğrenilmek istenen bir terimi daha önceden bildiğiniz bir terime benzeterek öğretme, çok daha kolay olacaktır.

2.2.1 Değişkenler:



Figür 2.2.1 A : Posta kutusu

Yukarıda, genelde Amerika ve Avrupa ülkelerinde kullanılan bir posta kutusu görünmektedir.

Bu posta kutusunu inceleyelim.

★ Posta kutusunun ne gibi özellikleri vardır:

Adres : Posta kutusunun bulunduğu evin adresidir. Bu adres ile, hangi posta kutusunun ifade edildiği, belirtilebilir. Adres özelliği sayesinde, bir mektup başka bir posta kutusuna bırakılmaz, yani adres özelliği TEK’tir. Aynı adreste iki posta kutusu olamaz (aynı adreste iki posta kutusunun olması saçma olacaktır).

Tür ve Boyut : Posta kutusuna yerleştirilebilecek nesnelerin belirli bir türü ve boyutu vardır. Örneğin bu kutuya sadece ve sadece postalanacak nesneler yerleştirilir, posta kutusuna bir ayakkabı yerleştirmek saçma olur. Ayrıca, postalanacak bile olsa, posta kutusuna, büyük bir koli yerleştiremeyiz çünkü kapasitesinden büyüktür.

Yani posta kutusunun belirli bir türü ve limiti bulunmaktadır. Ancak bu belirtilen türde ve uygun boyutta nesneleri posta kutusuna yerleştirebiliriz.

İçinde sahip oldukları : Posta kutusuna yerleştirilen mektupları veya küçük paketleri belirtmek için, Sahip oldukları nesneleri belirten bir özellik kullanılmalıdır. Yani bir posta kutusu içinde olan nesneler bulunmaktadır.

Süreklilik alanı : Bir posta kutusuna başkasının postası bırakılmaz, posta kutuları sadece bahçesinde bulundukları ev için geçerlidir.

Süreklilik süresi : Bir posta kutusu, ev sahibi taşınana kadar yerinde kalır, daha sonra, sökülüp yeni taşınılan yere monte edilir. Yani varlıklar süresi, ev sahibinin o bölgede kalma süresiyle aynıdır.

★ **Posta kutusuna ne gibi işlemler uygulanabilir:**

Mektup bırakma: Kişi, bir mektup yazıp, gönderilmek üzere posta kutusuna bırakabilir (Daha sonra postacı gönderilecek olan bu mektubu ve diğer kişilerin mektuplarını toplar).

Mektup alma (okuma): Kişi kendisine gelen (postacı tarafından posta kutusuna bırakılan) mektupları alıp okuyabilir.

Mektup yırtma : Kişi, bir mektubu göndermekten vazgeçebilir, veya kendisine (istemediği kişiden) gelen bir mektubu yırtabilir (çöpe atma).

Bir posta kutusu alınıp, başka bir yere yerleştirilebilir :

Kişi, taşınma esnasında (1) posta kutusunu söker, daha sonra (2) yeni evinin bahçesine yerleştirir.

Temizleme: Uzun süredir kontrol edilmeyen bir posta kutusu içinde biriken mektuplar toplanıp kutudan (ayıklanmak üzere) çıkartılabilir.

Bu farklı girişten sonra asıl konuya geçelim. Ancak lütfen az önce posta kutusu ile ilgili anlattıklarımı unutmayınız.

Değişken Değişken, bilgisayar ve matematik biliminde, sembolik bir ifade veya bir niceliği (miktar) ifade etmek için kullanılan semboldür. Matematikte, değişken, sık sık bilinmeyen bir niceliğin (potansiyel değişiminin) tanımlanması için; bilgisayar biliminde ise, niceliğin depolanabileceği bir yer, alan ifade eder. Değişkenler, sıklıkla bilinen ve sabit olan değerlerle mukayese edilir. [e7]

Yukarıdaki tanıma biraz daha ayrıntılı açıklayayım. Akışkan, yapışkan... gibi kelimelerdeki “-kan, -ken (orjinal ek : -gan, -gen)” ekleri, bu kelimelerin ifade ettiği işlevlerin, sürekli olarak gerçekleşebileceği, anlamına gelmektedir. Yani bir nesnenin, bir işlevi sürekli olarak gerçekleştirdiğini düşünün, böyle bir durumda, fiilden sonra “-ken, -kan” ekini kullanırız.

Örneğin, “akışkan” kelimesi genelde sıvı ve gaz maddeler için kullanılır: “su” akışkanı sürekli olarak akabildiği için, “akış” eyleminin sonuna “-kan” eki getirilmiştir ve bu eylemin sürekli olarak gerçekleşebileceği belirtilmiştir. Aynı şekilde, “kay**gan**” kelimesinde de, eklenen “-gan” eki, (örnek olarak) bir yüzeyin sürekli olarak kayabileceğini belirtmiştir.

Aynı şekilde, değişken kelimesinde geçmekte olan “-ken” eki, bu ifadenin, sürekli olarak değişebileceğini anlatmaktadır.

Matematik

Değişkenler, matematikte (bilinmeyen değerleri tutmak amacıyla) de kullanılır.

Örneğin;

$$x = 3 - 7$$

eşitliğinde, x , "-4" değeri yerine geçen bir ifadedir. Daha açık ifade ile, değişken; değeri, eşitlik sonunda çözülebilecek ifadeye verilen addır.

$$x = 3 - 5 + 18 / 3 \text{ ise } x = 4$$

Böyle bir ifadende anlaşılacağı gibi, değişkenler; bir değer yerine geçen sembollerdir. Değişkenlerin belirli limitleri vardır. Bu limitler değişkeni tanımlarken yapılır.

Örneğin;

$x = \{x | x \in \mathbb{Z}^+, x < 3\}$ Bu ifadenin anlamı, x öyleki; pozitif tam sayılar kümesine dahil ve 3'ten küçük bir değerdir.

$y = \{y | y \in \mathbb{N}, y < 2\}$ Bu ifadenin anlamı, y öyleki; doğal sayılar kümesine dahil ve 2 den küçük bir değerdir.

Bildiğiniz üzere, sayıların ait olabileceği farklı kümeler bulunmaktadır. Doğal sayılar, Tam sayılar, Rasyonel sayılar, Reel sayılar, Kompleks sayılar.... Belirli bir kümeye ait tanımlanan bir değişken, ancak o kümenin elemanlarından birini (veya alt kümenin elemanlarından birini) değer olarak alabilmektedir. Yani tam sayılarda belirlenen bir değişken, rasyonel bir değeri alamaz.

Örnek : $X \in \mathbb{N}$: Yani X değişkeni bir doğal sayıdır (0 dan başlayıp pozitif sonsuza kadar giden sayı doğrusu)

Yukardaki tanımlamaya göre

$X = 3$ veya $X = 4$ veya $X = 67835$ veya $X = 2364535$ ifadeleri doğru ancak

$X = -56$ veya $X = 0.563$ veya $X = \sqrt{3}$ veya $X = 9.8$ ifadeleri yanlış olur. Çünkü, X değerine atanan değer, belirtilen küme dışındadır.

Böylece, bir değişkenin alabileceği değerleri kapsayan bir küme olduğu, özelliğine, sahip olduğunu söyleyebiliriz.

Bilgisayar

Değişkenler, içlerinde değer tutan bir tür kaptır diyebiliriz. Bir kutuyu (yani değişkeni) düşünelim. Bu kutu içine bir nesne (değer) yerleştirebiliriz. Yukarıda gösterilmiş olan posta kutusu örneğini, posta kutusunun özelliklerini ve posta kutusuna uygulanabilecek işlemleri aklınıza getiriniz ve sonraki paragrafları bu şekilde okuyunuz. Bir değişken, hafızadaki (memory – ram) belirli bölge olarak tanımlanabilir.

Bilgisayar dünyasında da, değişkenler için belirli değer kümeleri yada, belirli limitler ve izinler bulunmaktadır. Bir değişken, limitlerinin ve izinlerinin, onayladığı her değeri alabilir.

Ayrı bir özellik olarak, bir değişkenin, 'ismi' yada 'tanımlayıcısı' özelliği bulunmaktadır. Bir önceki sayfada, $X \in N$ ifadesinde, bir değişkeni tanımlarken, isim olarak X karakterini kullandık. Matematikte, değişkenlerin isimleri için genelde, harfler (yani karakterler) kullanılır, ancak programlama da her türlü, her uzunlukta isim (kurallara uyduğu sürece) kullanılabilir.

Programsal anlamda değişken, üzerine atanmış olan değeri, aktif olduğu sürece tutar(süreklilik süresi), bu süre içinde, yeniden üstüne yazılma ve okunma işlemlerinin de gerçekleştirilmesine olanak tanır. Başka bir deyişle değişken; bir tür değeri, belirli bir süre hafızada, belirli bir veya birden fazla işlemde kullanmak üzere tutmaya yarayan, bir tür tutamaçtır.

Şimdiye kadar, bir değişkenin 2 özelliğini işledik. İsim (Belirteç) ve Tür (Küme) Birde, belirttiği değişkene atanan, yani o değişkenin o an içinde tutuyor olduğu değer özelliği bulunmaktadır.

$X \in N$ örneğini yeniden incelersek, X değişken ismi, N ait olduğu kümeyi belirtir. $X = 3$ ifadesinde, X adlı değişkene 3 değerini atamış bulunmaktayız.

Bundan sonra eğer X'e başka bir değer atamazsak, X'in değeri hep 3 olarak kalacaktır.

Örneğin,

$X, Y \in N$, $X = 3$, $Y = X$, ifadeleri sonucunda, X'e 3 değerini atamış, Y'ye ise, X'in değerini atamış oluyoruz. Dolayısıyla, Y'nin değeri, X'in değerine eşit oluyor ve 3 olarak ifade ediliyor. Yani örnekteki ifadeler sonucunda, X'in ve Y'nin değerleri '3' olur.

Değişkenlerin, 'varsayılan değer' özelliği bulunmaktadır. Bu değer çoğu zaman 0'dır. Örneğin, bir değişken tanımlandığında ($X \in N$ ifadesi ile) X in değeri (atanmadığı için) her zaman 0 olarak ifade edilir, başlangıç değeri (varsayılan değeri) 0'dır. Matematikte, işlemler sayılar ile yapıldığı için, çoğunlukla varsayılan değer 0 olur.

Ancak programlamada, matematikten farklı olarak, farklı tiplerde (sayılardan farklı : örneğin yazısal tip) değişkenler olabileceğinden, farklı varsayılan değerler olabilmektedir. Her kümenin bir tane varsayılan değeri vardır (mutlaka vardır ve bir tanedir). Yani aynı kümeye dahil olan bütün değişkenlerin varsayılan değeri aynıdır.

Örneğin, yazısal bir türün (kümenin) varsayılan değeri, "" yani boşluktur. Bununla birlikte, parasal bir türün (kümenin) varsayılan değeri 0.00 'dır.

Değişkenlerin, geçerlilik süresi ve geçerlilik alanı özellikleri bulunmaktadır. Örneğin, $X=3$ gibi bir ifade, o problem (matematiksel problem) çözülene kadar geçerli (geçerlilik süresi) olacaktır ve X değişkeni, diğer bir problemde (matematiksel problem) kullanılmayacaktır (geçerlilik alanı). Kullanılsa bile, değeri, bir önceki problemden kalan değeri değil, varsayılan değeri olur. (Yani değişken yeniden tanımlanmış olunur.) Matematikte problem için belirtilen değişkenler sadece o problemler için geçerlidir. Başka bir problemde, aynı isimde ve aynı kümeye dahil olan bir değişken tanımlandığında, değerini, bir önceki problemden almaz, çünkü değişken her tanımlandığında varsayılan değer ile başlar ve diğer problemlerle ilişkisi bulunmaz.

Ancak bazı durumlarda, bir değişken için belirlenen değer, bütün matematiksel problemler veya bazı matematiksel problemler için kullanılabilir. Örneğin bir dizi problem çözüyor olduğumuz varsayalım ve bu problemlerde kullanılacak π (Pi sayısı her ne kadar sabit sayı olarakta bilinse, bir değişkendir) değeri için $22/7$ verildiğini düşünelim. Belirtili problemler içinde, π değeri hep $22/7$ olacaktır. Eğer bir π değeri belirtilmese, varsayılan değer olarak 3.14 (bütün matematiksel işlemler için) kullanılacaktır.

Fizik problemlerinde, konulara göre bir çok sabit alınan ancak duruma göre değişebilen değer vardır.

Örneğin : $E_p = m.g.h$ (Potansiyel enerji = kütle . yerçekimi ivmesi . yükseklik) g değişkenin değeri, temel olarak 9.8 kg.m/s^2 alınır ancak bu değişkenin, dünyanın farklı bölgelerinde farklı değerler alır. Örneğin bu problem için bu değer 10 kg.m/s^2 alınabilir veya Ay için 1.63 kg.m/s^2 olarak ifade edilebilir. Görüldüğü üzere, bir değişkenin birden fazla problem için veya bütün matematiksel (veya fiziksel) işlemler için kullanılabilir.

★ Değişkenlerin özellikleri

Sonuç olarak bir değişkenin,

İsmi : Tanımlayıcısı

Değer Türü

Değeri

Geçerlilik süresi (ve geçerlilik alanı)

Varsayılan değeri özellikleri bulunmaktadır. Hem matematikteki, hem bilgisayardaki değişkenler bu özelliklere sahiptir.

★ Değişkenlere uygulanabilecek işlemler

Değer okuma : Bir değişkenin değeri okunabilir

Değer yazma : Bir değişkene değer yazılabilir

Değer sıfırlama : Bir değişkenin değeri varsayılan değere atanabilir

Değişken oluşturulabilir : Bir değişken tanımlanabilir. (Yeni bir değişken oluşturulabilir)

Değişken yok edilebilir : Bir değişken, görevi bittiğinde, hafızadan silinebilir

Görüldüğü üzere, değişkenlerin özellikleri de, değişkenlere uygulanabilecek işlemler de bir posta kutusuna uygulanabilen işlemler ve özellikleriyle neredeyse aynıdır.

Değişken, programlar içinde, belirli bir anlık bilgileri kayıt etmek için kullanılır. Bu kaydı, işaret bırakmaya benzetebiliriz; bir ormanda yolumuzu (**Hansel ile Gratel'in, ormanda kaybolmamak için işaret bırakması gibi**) kaybetmemek için işaret bırakır ve diğer adımlarımızı bu bıraktığımız işaretlere göre atarız.

Örneğin, Ctrl+C tuşlarını veya (sağ tuş menüsünden) Kopyala komutunu kullanarak hafızaya bir metni veya resmi yazıyorsunuz (hafızaya atıyorsunuz-**kopyalama**), sonra Ctrl+V tuşlarını veya (sağ tuş menüsünden) Yapıştır komutunu kullanarak, o alana, hafızada kayıtlı bulunan metni veya resmi ekliyorsunuz. İşte, hafızada tutulan (ta ki bir daha ki bilgisayar kapanana kadar), metin veya resim, bir değişken olan Kopyalama Alanı'nın değeridir.

Şimdi anlattıklarımı daha iyi kavramanız için bir örnek daha vereyim. Yukarıda bahsettiğim posta kutusunu düşünelim. Dikkat edersek, bir posta kutusu, bir veriyi (postayı) belirli bir süre tutar (saklar). Daha sonra içindekiler okunabilir, silinebilir (yırtıp atılabilir). O halde, bir posta kutusu bir değişken gibi hareket eder.

Bilgisayar dünyasında, sistem aynı şekilde işler. Bilgisayar dünyasında bazı standart veri türleri vardır ve biz tanımladığımız her veriyi bu türlerden bir veya birkaç tanesine uydurmalıyız ki o veriyi tanımlayabilelim.

Veri türü bir değişkenin alabileceği izin verilen verilerin tümü ve bir değişkenin verisine izin verilen uygulanabilecek işlemlerin tümüdür.

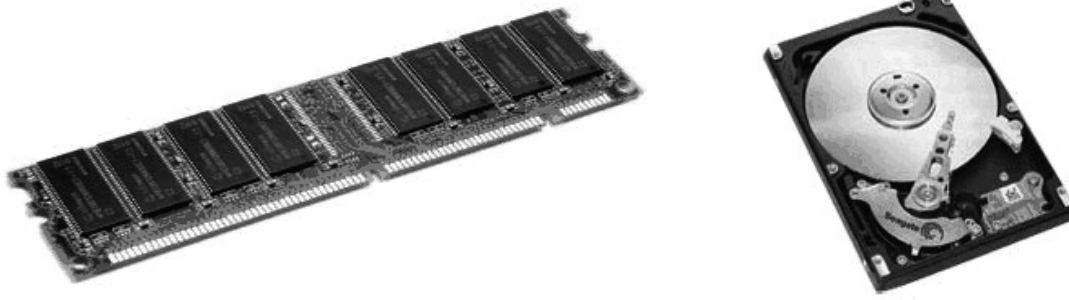
Veri Türü	Açıklama
Tam sayı türleri	Sadece tam sayı değeri alabilen türlerdir
Ondalık sayı türleri	Hem tam sayı hemde ondalık sayı tutabilen türlerdir
İkili veri türü	Temelde 1/0 olarak, Evet/Hayır, Olumlu/Olumsuz, Var/Yok, İnce/Kalın.... gibi sadece iki değer tutabilen veri türüdür
Metin veri türü	Sayı yerine metin de tutabilen veri türüdür. Örneğin; $x=3$ yerine metinsel olarak ifade edilen y değişkenine "deneme" adlı metin değerini de atayabiliriz. $y = \text{"metin"}$ Burda dikkat edilmesi gereken nokta, metinsel verilerin, (dile göre değişir) tek veya çift tırnak arasında yazılması zorunluluğudur

Figür 2.2.1 A : Genel veri türleri

Daha açık bir ifade ile; bir değişkenin alabileceği değerleri kapsayan bir tür kümedir diyebiliriz. Örneğin, tam sayı türlerinden tanımlanmış bir değişken, sadece tam sayı değerlerini alabilmektedir yani veri tipi, tam sayılar kümesidir. Bir değişkenin verisine uygulanabilecek işlemleri daha sonraki bölümlerde göreceğiz.

Tabiki bu veri türlerinin haricinde başka veri türleri de bulunmaktadır ancak bu türler daha sonraki bölümlerin konularıdır.

★ Değişkenler, nerede tutulur?

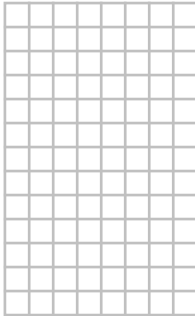


Figür 2.2.1 B : Hafıza (solda), Harddisk (sağda)

Değişkenler hafızada (Memory denilen donanım) tutulur.

Burada bilmeniz gereken donanımla alakalı olan bir bölüm bulunmaktadır. Bilgisayar donanım parçaları olan Hafıza (Memory) ve Sabit disk (Harddisk) veriyi saklamaya yararlar. Aralarındaki fark nedir? Sabit disk içinde bir çeşit cdler olan (yukarıdaki figürde görünüyor) bir aygıttır. Bu cd üzerinde milyonlarca mıknatıs vardır ve mıknatıs konumuna göre verileri kaydeder. Avantajı, elektriksiz ortamda da verileri saklar, yani bilgisayarı kapattığınızda verileriniz silinmez, ayrıca uzun ömürlüdür ve ucuzdur. Dezavantaj olarak, çok yavaş çalışır.

Hafıza ise, verileri, elektriksel değerler olarak tutmaya yarar. Bir dizi (milyonlarca flip-flop (veriyi elektriksel olarak 1 veya 0 şeklinde tutan bir devre elemanı)'tan oluşur ve ancak elektrik enerjisi verildiğinde çalışır. Dezavantaj olarak elektrik kesildiğinde hafıza üzerindeki bütün veriler silinir. Avantajları da, verinin elektriksel olarak (aktif) tutulması ve çok hızlı (harddiske göre çok hızlı) işlenebilmesidir.



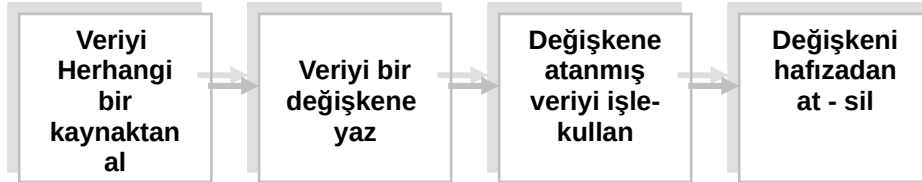
Figür 2.2.1 C : Hafıza

Hafızayı bir tür grid (çizelge – ızgara)'e benzetebiliriz. Her bir hücre (yukarıdaki sembolde bulunan her bir küçük kutu) bir bitlik (yani 1 veya 0) değer alabilir. Hafıza, verileri hücre hücre işlemez, satır satır işler. Yani hafıza üstünde işlenebilecek en küçük veri byte'tır ve her satır, (bazen) bir byte'a denk gelmektedir. Genelde işlemciler 16-32-64 bitlik satırlarla çalışırlar. Bu konuya ilişkin ayrıntılı bilgiler, dördüncü bölümde bulunmaktadır.

Bilgisayara bir programı çalıştırma veya bir dosyayı açma emri verdiğinizde, bilgisayar o program veya dosyayı harddiskte bulur, onu okur, hafızaya yazar, ve ancak

bilgisayar o program veya dosyayı hafızaya yazdıktan sonra kullanabilirsiniz. Değişkenin hafızada nereye yazılacağını biz direkt olarak belirleyemeyiz ancak yazıldıktan sonra hafızada hangi adreste kayıtlı olduğunu öğrenebiliriz.

Dikkat ederseniz, programların açılması, uzun sürebilir ancak kapanması daha kısa sürer. Aynı şekilde, dosya açma kayıt gibi işlemler uzun sürebilirken, kopyalama - yapıştırma veya dosya üzerinde değişiklik yapma işlemleri daha kısa sürer. İşte bu fark; sabit disk ile, hafızanın hız farkından kaynaklanır. Hafızaya alınan bir programın değişkenleri doğal yine olarak hafızada tutulur ve işlemleri hafızada durduğu sürece yapılabilir.



Figür 2.2.1 D : Verilerin işlenmesi

Yukardaki şekilde veri işlemenin nasıl gerçekleştiği gösterilmiştir. Bu olayı, şu şekilde ifade edeyim: Varsayalım ki bir kütüphanede çalışıyorsunuz ve göreviniz, bütün kitapların içlerinde geçen konuların listesini çıkarmak. Rafta duran kitabın bizlere faydası yoktur. Yani gidip o kitabı o raftan kullanılmak üzere çıkarmadığımızda sadece depolanmış bir bilgidir diyebiliriz. Ancak dikkat edelimki, binlerce kitap içinde, bir kişiye bir veya iki tanesi (belirli bir araştırma için) gerekmektedir (hepsi değil!!!).

Eğer bütün kitapları aşağıya döküp bunlar içinden istediklerinizi yapmaya çalışırsanız, bütün masalar - yerler dolmuş olur ve diğer insanların kullanım alanlarını da işgal etmiş olursunuz. Halbuki, sırayla, bir elinize alsanız, o kitabın konularını yazsanız ve daha sonra yerine koysanız, bu işlemi diğer kitaplar içinde yapsanız hem karışıklık olmaz, hemde çok daha az yer kullanarak bu işlemi bitirmiş olursunuz.

Sonuç olarak:

Kütüphane	Bilgisayar
Kütüphanelerde milyonlarca bilgi bulunmaktadır.	Bilgisayarın okuyabileceği (programların okuyabileceği) milyonlarca bilgi bulunmaktadır.
İlgili rafa gidip o kitabı alıp okumazsak, bize hiçbir faydası yoktur.	Eğer o dosyayı, bilgiyi tutan veritabanını açıp bilgiyi program içinden okumaz isek, bize hiçbir faydası yoktur. İşte, bilgileri işlemek üzerine HAFIZA'ya aldığımızda (bilgiler hafızadan başka bir yerde işlenemezler!) onları hafızada tutmamıza yarayan tutamaçlar değişkenlerdir.
Kütüphanedeki bütün kitapları açıp tek tek okumak yerine, ilgili olanını açıp okuruz	Bilgisayardaki her dosyayı açmak yerine ilgili olanı açarız
Varsayalımki 10 farklı konu üzerinde araştırma yapıyoruz. Kütüphanede de her bir konu için 5 farklı kitap bulunuyor.	Bilgisayar dünyasında bu olay aynı şekilde gerçekleşmektedir. Örneğin, program içinde kullanacağımız verileri dosyalardan veritabanından veya belirli

Eğer bu konularla ilgili, bütün kitapları alırsak fazla çalışma alanı (masa gibi) kaplamış oluruz, ayrıca bu yükü taşıyamayız. Onun için, kitapları tek tek alıp, okuyarak, istediğimiz bilgiye ulaştıktan (veya ulaşamadıktan) sonra kitabı aldığımız yere bırakmaklı ve daha sonra sıradaki kitaba aynı işlemleri uygulamalıyız.

kaynaklardan okuyacağımızı düşünün. Eğer bütün kaynakların hepsini aynı anda açarsak, bu hafızada çok alan kullanılmasına neden olacaktır.

Figür 2.2.1 E : Kütüphanelerin bilgisayar dünyasında gösterimi

Program içinde, aktif olarak kullanılacak olan bütün veriler için değişkenler kullanılabilir. Örneğin, X isimli kullanıcı programı açtı ve programı kullanıyor. O an aktif olarak programı kullanan kullanıcının ismini öğrenmek için, daha önceden (program açıldığında) kullanıcının ismini bir değişkene değer olarak yazmalıyız ki, daha sonra bu bilgi sorulduğunda, bu değişkenin değerinden öğrenebilelim.

Başka bir örnek olarak, bir çarpma işlemini düşünelim.

				A	B	C
			x	D	E	F
+						
	H	J	K	L	M	N

Figür 2.2.1 F : Çarpma işlemi örneği

Yukarıdaki çarpma işleminde, ilk olarak sağ hanelerden başladığımızda C ile F'yi çarpıp, eldeliği bir kenara yazarız. İşte o kenara yazılan eldelik sayısı bir değişken olarak tanımlanabilir. Başka bir örnek olarak; bir arkadaşımız bizi arıyor, ancak o an telefonu açamıyoruz. Daha sonra cep telefonumuzun, cevapsız aramalar bölümüne bakarak, kimin aradığını görüyoruz. İşte bu süreç içinde (yani bir cep telefonumuza bakıncaya kadar) “cevapsız aranan numara” bir değişken olarak kayıt edilir.

Hafızamızda tuttuğumuz her tür bilgiyi birer değişken olarak düşünebiliriz. Bunun haricinde, program içinde farklı yerlerde, son yapılan işlemi, son durumu, son sayıyı, son değeri tutan değişkenler de kullanılabilir. Eğer bir değişken uzun süreli hafızada tutulması gerekiyorsa, bu bilgiyi, sabitdisk'e de kayıt edebiliriz. Ancak sonra onu işlemek için yine kullanmak üzere hafızaya aktarmamız gerekecektir. Bu konuyla alakalı ayrıntılı bilgiler ileriki bölümlerde mevcuttur.

Hatırlatma_1'i hatırlayalım, amacımız, gerçek dünyada metinsel olarak ifade ettiğimiz bir işlemi bilgisayar içine programsal kod olarak aktarabilmek. Gerçek dünyada, bir değeri, geçici olarak veya belirli bir süre hafızada tutmayı ifade ettiğimiz (“eldelik”, “hafızada tut”, “bunu kenara yaz”, “not et”, “hesapla .. orada yazılı kalsın ..”,) her ifade için, programlamada, “var” ifadesini kullanırız.

3 kere 7 = 21, **elde** var 2

Bu faiz oranını **hesapla**

Hızı ikiye **böl** (böldükten sonra o değerin bir yere yazılı olması gerekiyor)

.....

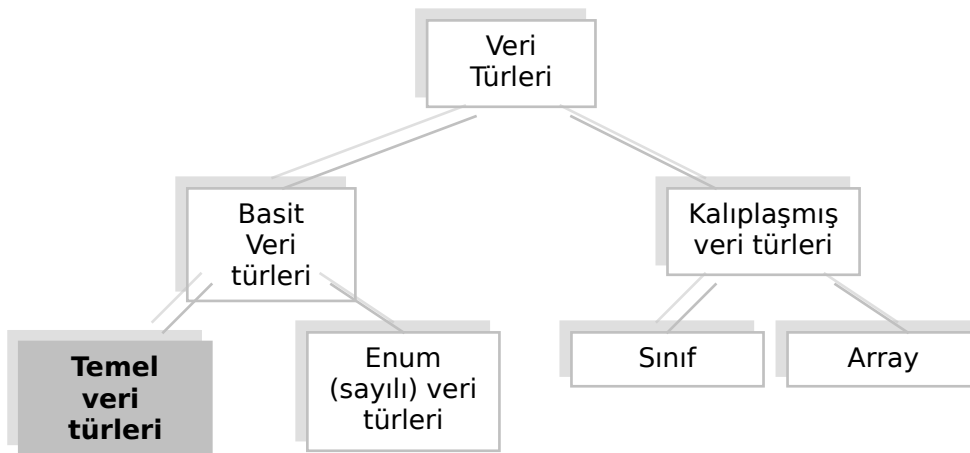
Çeviri 1 : Değişkenler

2.2.2 Veri türleri

Gerçek dünyada veya bilgisayar dünyasında işleyecek olduğumuz verileri, belirli kalıplarla ifade etmemiz gerekmektedir. Bu kalıplar; rakamlar, sayılar, tarihler, ifadeler ve karakterlerdir. Nasıl matematikte, fizikte veya diğer bilimlerde, skalar ölçüler(3 metre, 5 cm, 2 kg, 43C....) yani bir miktar ve bir birimden oluşan ölçüler kullanılıyorsa, bilgisayarda da her verinin bir türü bulunmaktadır.

Veri türlerini kek kalıbı gibi düşünebiliriz. İçine koyacağımız kek o kalıbın şeklini alır. Aynı şekilde, elimizde birkaç farklı kek kalıbı olduğunu ve verileri bu kalıplardan en uygununa koymamız gerektiğini düşünelim.

Genel olarak, programlama dillerinde çok fazla sayıda, temel veri türü bulunmaz. Çünkü, oluşturulan sınıflarla, istenilen özel veri türü elde edilebilir. Daha açıkcası, bir sınıf oluştururken, sınıfını oluşturduğumuz tür'ün, özelliklerini değişkenler olarak belirtiriz. Bu değişkenlerinde, belirli veri türleri bulunmaktadır.



Figür 2.2.2 A : Veri Türleri

Veri tipleri yukarıdaki gibi ayrılmaktadır. İlk olarak yukarıdaki figürde verilen temel veri türlerini inceleyeceğiz.

İsim	Boyut	Açıklama
------	-------	----------

	(Byte)	
Byte / Tinyint	1	Bu veri türü; 0 ile 255 arasında (yani 0 ile 2 üzeri 8 eksi 1 (2^8-1)) tam sayı değeri alabilen bir veri türüdür. [0-255]
Short / Smallint	2	Bu veri türü; -32.768 ($2^{\text{üssü } 15}$) ile +32.767 ($2^{\text{üssü } 15}$, eksi 1) arasında tam sayı değerleri alabilen veri türüdür. Aslında, 2 Byte = 16 bit olduğundan bu türün alabileceği değer $2^{\text{üssü } 16}$ 'dır. Ancak, negatif sayıların da tanımlanmasına olanak verildiği için yarısı negatif, yarısını pozitif sayılar kümesine ayrılır. Pozitif bölümdeki, “eksi 1” ifadesi, 0 değerini de içerisine almasından kaynaklanmaktadır (Pozitif ve negatif sayıların tanımlanması için, “one’s complement ve two’s complement” konusunu daha ileride göreceğiz).
Int / Integer	4	4 (byte) x 8 (byte başına bit) = 32 bit Bu veri türü; $-(2^{\text{üssü } 31})$ ile $+(2^{\text{üssü } 31})-1$ arasındaki tam sayı değerlerini alabilen veri türüdür.
Long / Big integer	8	8 (byte) x 8 (byte başına bit) = 64 bit Bu veri türü; $-(2^{\text{üssü } 63})$ ile $+(2^{\text{üssü } 63})-1$ arasındaki tam sayı değerlerini alabilen veri türüdür.
Single / Real / Float / Double	4-16	Bu veri türlerinin özellikleri, dile göre değişmektedir. Ondalık sayıları hafızada tutmak amacıyla kullanılmaktadır. Dilin özelliklerine göre limitleri ve virgülden sonraki desteklediği hane değişmektedir.
String / Char / Varchar	-	Metinsel veri türlerini belirtmek için kullanılır. Alabileceği maksimum karakter (uzunluk), kullanılacak olan dile göre değişmektedir.
Boolean / Bit	2	Sadece 1 ve 0 değeri alabilen veri türüdür
Date / Datetime	4-8	Tarih verisi tutmak için kullanılır. Boyutu belirtilen dile göre değişmektedir

Figür 2.2.2 B : Veri Türleri (2)

Sayısal tipler

Byte, bilgisayardaki en küçük ikinci veri türüdür. 8 Bit [sadece bir veya sıfır değerindeki bir sayı]'ın yanyana dizilmesinden meydana gelmektedir. Byte tipinde tanımlanmış bir değişken, 0 ile 255 arasında bir tam sayı değeri alabilir. 0 ile 255 arasında olan değer, 2 lik sayı düzlemine çevrilir.

Örneğin, 17 sayısı, 2 lik düzlemde, $(00010001)_2$ şeklinde gösterilir. İkilik düzlemdeki bu sayısı çözümlersek;

$$\begin{array}{cccccccc}
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & (00010001)_2 \\
 0 \times 2^7 & 1 \times 2^6 & 1 \times 2^5 & 1 \times 2^4 & 1 \times 2^3 & 1 \times 2^2 & 1 \times 2^1 & 1 \times 2^0 & 17
 \end{array}$$

Figür 2.2.2 C : 2'lik düzlemde gösterim

şeklinde bir tablo elde ederiz.

Matematikteki sayı tabanlarını gözden geçirirsek, bu konuyu daha iyi kavrarız ve hatırlarız. Bilgisayardaki veriler, sadece, “elektiriksel veri var(1) veya yok(0)” şeklinde olduğu için, işlemlerde kullanılacak sayı tabanı 2 değeri alabilen 2'lik sayı tabanıdır.

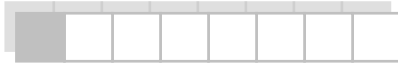
Daha sonra, yukarıdaki şekilde hafızaya yazılır. Aynı şekilde, diğer sayısal tiplerde hafızaya aynı şekilde kendi boyutları ölçüsünde yazılırlar. Sayısal tipler, bilgisayar tarafından en hızlı işlenen verilerdir. Çünkü, boyutları sabittir ve işlemciler, sayısal değerleri daha hızlı okurlar. En hızlı çalışan veri tipi TAM SAYI – integer’dır. Integer – Short – Long gibi değişken türlerin boyutları (alabilecekleri sayıların kapsamı), kullanılan dile, hatta, kullanılan işlemciye göre değişebilmektedir.

Sayısal veri türlerinde, işlemler çok hızlı gerçekleşir. Ayrıca bu veri tiplerinin birbirlerine dönüştürülmesi de çok hızlı gerçekleşmektedir. Sayısal veri türleri hafızaya yazılırken, sadece pozitif yazılabilirler (çünkü negatif elektrik – pozitif elektrik diye bir tanım aslında yoktur) . Peki negatif sayılar nasıl tanımlanıyor? Negatif sayıların hafızaya tanımlanması için temel 4 yöntem bulunmaktadır.

Sign and Magnitude (işaret ve değer)

Bu yöntemlerden ilki; Sign And Magnitude (İşaret ve Değer)’dir. İşaret “+” veya “-” olarak ifade edilir, değer ise sayısal değerdir.

Bu yöntemde, bir sayıyı ifade ederken kullanılacak olan İLK bit, o sayının negatif veya pozitif olduğunu sonraki bitlerde sayının değerini belirtir



Figür 2.2.2 D : 8 Bit gösterimi ve sayıların pozitif ve negatif olarak ifade edilmesi

Yukarıdaki figürde, ilk bit (gri) eğer 0 ise, devam eden bitlerde ifade edilen sayının pozitif olduğu, eğer 1 ise, devam eden bitlerde ifade edilen sayının negatif olduğu anlaşılır.

Örneğin, “3” değerini ifade etmek için

00000011 ifadesini kullanırız.
 $2^1 \times 1 + 2^0 \times 1 = 3$

“-3” değerini ifade etmek için ise
10000011 ifadesini kullanırız.

Dikkat ettiyseniz, integer veri tipleri 4 byte lık alan kaplamasından dolayı 4byte x 8 bit = 32 bit’e sahip olmaları gerekmektedir. Ancak, açıklamasında;

Bu veri türü; $-(2^{\text{üssü } 31})$ ile $+(2^{\text{üssü } 31})-1$ arasındaki tam sayı değerlerini alabilen veri türüdür.

Üst olarak, 32 bit yerine 31 bit değeri kullanılmıştır. İşte bu aradaki 1 bitlik kayıp, o değerin negatif mi pozitif mi olduğunu belirtmek için kullanılmıştır. Tabi böyle bir durumda, “0” değeri için bir negatif birde pozitif gösterim olacaktır. Halbuki “0” ne pozitifdir nede negatif.. Ancak bilgisayar dünyası ile matematik dünyası arasında, bazı durumlar böyle farklılıklar olabilir.

Ones’ complement, Two’s complement, Excess N

Yukarıda ilk yöntemi anlattığım için, bu yöntemlerin ayrıntılarını anlatmaya gerek bulunmamaktadır. Bu yöntemler birbirlerine çok benzerler. Ones' complement, ilk bit'i 0 olduğunda, sayının pozitif olduğunu belirtir. Eğer ilk bit 1 ise, gerçek değer, belirtilen değer, bir bütünden(yani 7 bitlik alınabilecek en büyük değer olan 127) farkı kadardır ve negatiftir.

Daha açık bir ifade ile,

00000001 ifadesi +1 değerini alırken
10000000 ifadesi -127 değerini alır

İkinci ifadede, ilk bittten sonraki kısmın değeri 0'dır (0000000). 7 Bit'in alabileceği en büyük değer 127'dir [(1111111)₂]. 127'den de 0'ı çıkardığımızda (bir bütün olan 127'den farkını aldığımızda) geriye, 127 kalır. Başına “-“ işareti koyarak negatif yaparız ve -127'yi elde ederiz. Aşağıdaki tablo ones' complement'in değerlerini göstermektedir. Ones' complement, 0/127 ve -0/-127 arasındaki değerleri alabilmektedir. (Tabi ki -0 ile +0 arasında bir fark bulunmamaktadır) Böyle bir durumda hem + hemde – sıfır değeri olduğu için bir adet değer kaybı bulunmaktadır.

8 Bit Ones' Complement

Değer	Ones' complement
0	0
1	1
...	...
1111101	125
1111110	126
1111111	127
10000000	-127
10000001	-126
10000010	-125
...	...
11111110	-1
11111111	-0

Figür 2.2.2 E : Ones' Complement

Two's complement de, benzer bir şekilde, çalışmaktadır ancak Two's complement'in değerleri 0/127 ve -1/-128 dir. Yani -0 ifadesi bulunmamaktadır.

8 Bit Two's Complement

Değer	Two's complement
0	0
1	1
...	...

1111110	126
1111111	127
10000000	-128
10000001	-127
10000010	-126
...	...
11111110	-2
11111111	-1

Figür 2.2.2 F : Two's complement

Birde değeri atanmasında farklı olan Excess-N formatı bulunmaktadır.

Değer	Excess-127
0	-127
1	-126
...	...
11111111	0
10000000	1
...	...
11111111	128

Figür 2.2.2 G : Excess-127

Hepsini birlikte tasarlar isek;

4-bit Integer Gösterimleri

Ondalık	İşaretsiz	Sign and Magnitude	Ones' Complement	Two's Complement	Excess-7
8	1000	-	-	-	1111
7	111	111	111	111	1110
6	110	110	110	110	1101
5	101	101	101	101	1100
4	100	100	100	100	1011
3	11	11	11	11	1010
2	10	10	10	10	1001
1	1	1	1	1	1000
(+)0	0	0	0	0	111
(-)0	-	1000	1111	-	-
-1	-	1001	1110	1111	110
-2	-	1010	1101	1110	101
-3	-	1011	1100	1101	100
-4	-	1100	1011	1100	11
-5	-	1101	1010	1011	10
-6	-	1110	1001	1010	1
-7	-	1111	1000	1001	0
-8	-	-	-	1000	-

Figür 2.2.2 H : 4-Bit integer gösterimi

Metinsel türler (String / Char)

Bilgisayar üzerinde sadece sayısal tipler değil metinsel veri tipleri de kullanılmaktadır. Char, character'in yani karakterin kısaltılmış halidir ve sadece bir adet harf – rakam – işaret (.,!/&%+^'...) ifade etmek için kullanılır. Bir karakter bir byte alan kaplamaktadır ve byte veri tipinde olduğu gibi rakam 0-255 arası rakam tutmak yerine, AZ-az-09.... gibi karakterlerin karakter kodlarını hafızada tutmak için kullanılan veri türüdür. Char sadece bir karakter içerebilir. Bir karakterin karakter kodunu hafızada tutar. Her karaktere karşılık gelen bir kod numarası bulunmaktadır. Yani bir metin içindeki her bir karakteri belirtmek için 1 byte'lık veri kullanılır.

	32		3	51		F	70		Y	89		I	108
!	33		4	52		G	71		Z	90		m	109
"	34		5	53		H	72		[91		n	110
#	35		6	54		I	73		\	92		o	111
\$	36		7	55		J	74]	93		p	112
%	37		8	56		K	75		^	94		q	113
&	38		9	57		L	76			95		r	114
'	39		:	58		M	77		`	96		s	115
(40		;	59		N	78		a	97		t	116
)	41		<	60		O	79		b	98		u	117
*	42		=	61		P	80		c	99		v	118
+	43		>	62		Q	81		d	100		w	119
,	44		?	63		R	82		e	101		x	120
-	45		@	64		S	83		f	102		y	121
.	46		A	65		T	84		g	103		z	122
/	47		B	66		U	85		h	104		{	123
0	48		C	67		V	86		i	105			124
1	49		D	68		W	87		j	106		}	125
2	50		E	69		X	88		k	107		~	126

Figür 2.2.2 I : Karakter kodu tablosu

Yukarıdaki tabloda, ilk kolonlarda (soldaki) karakter, bir sağında ise o karakterin kodu listelenmiştir. Örneğin “X” karakterinin karakter kodu 88’dir ve “e” karakterinin karakter kodu 101’dir.

String(ingilizce: Dizgi. dizgi. karakter dizisi. sicim, ip;) karakter dizisi - ip demektir. Metinsel verileri hafızada tutmak için kullanılmaktadır. Yani, bir ipe dizilmiş (boncukları) karakterleri ifade etmektedir. Metinlerin uzunluğu değişebileceğinden, boyutu belli değildir. Sadece üzerine yazılan veriye göre belirlenir. Bir dizi, karakter düşünün, bu karakterler bir sıra ile birbirine bağlanıyor ve bu sıra sonuçta bir metinsel veriyi oluşturuyor.

Figür 2.2.2 J : Bir metnin gösterimi

String tipinde bir değişkene bir metin ataması yaptığımızda (örneğin "kitap") önce, bu kelime içindeki bütün karakterler, tek tek karakter kodlarına çevrilir. (k=107, i=106, t=116, a=112) Daha sonra, bu karakterler, byte vektörüne (B) girer, byte vektörünün alabileceği aralıkta (0-255) olduğu için, byte-byte hafızaya yazılır.

Ancak, bu sistem eskiden kullanılmaktaydı. Şuan ise, bir karakteri belirtirken genelde,

2 byte kullanılmaktadır. İlk byte, kullanılmış olan alfabeyi, ikincisi ise, karakter kodunu ifade etmektedir (ileriki bölümlerde bu konu ile ilgili bilgiler verilecektir). Böylece, sadece a-z, A-Z arası harfler değil, arapça harfleri, yunanca harfleri, rusça harfleri, çince harfleri.... gibi diğer alfabelerin karakterleri de ifade edilebilmektedir.

Dikkat ederseniz, temel veri türleri tablosunda, her değişken türünün bir boyutu bulunmaktadır çünkü sabittir. Ancak string'in boyutu önceden belirli değildir. Girilen verinin uzunluğuna göre belirlenmektedir.

Eskiden, bir metnin sonuna, "0" kodlu iptal karakteri konulurdu. Böylece o metnin bitiş noktası belirtilmiş olunurdu. Ancak sonrasında string türdeki veriler hafızaya yazılırken, ilk olarak, o metnin uzunluğu (integer olarak) hafızaya yazılır. Sonrasında ise, her bir karakter için bir veya (dile ve sisteme göre) iki byte kullanılarak yazılır.

Tarih ve saat veri türleri

Bu veri türlerinin özellikleri ve boyutları kullanılan programlama diline göre değişmektedir. Tarih belirlemede, birçok dil bir noktayı baz olarak alır. Yani bir tarihi başlangıç olarak seçer (örneğin: 21 Eylül 1921), sonra girilen tarih değeri ile bu seçilmiş tarih arasındaki gün değerini integer olarak hafızada tutar. Daha açık olarak, 19 Haziran 1973 tarihini belirtmek istediğinizde programlama dili bu tarihten, dil tarafından kabul edilmiş başlangıç değer olan 21 Eylül 1921'i çıkarır ve sonuç olarak 18930 değerini üretir ve hafızaya bu sayıyı yazar.

Benzer durum zaman değişken tipi içinde geçerlidir. Örneğin 00:00 baz alınır ve üstüne eklenecek dakika integer cinsinden yazılır. Mesela, 12:34 saatini belirtmek için, 754 değeri (754 dakika eder) hafızaya yazılır.

Bool/boolean/bit türündeki veri tipi

Bu veri tipi sadece true(doğru) ve false(yanlış) değerleri alabilen bir veri türüdür. Hafızada 2 byte yer kaplar. Aslında sadece 1 veya 0 değeri alacağı düşünüldüğünden, neden 1 bit değilde (1 bit, ya sıfır yada bir değerini alabilir) 2 byte olduğunu merak etmişsinizdir. Boolean veri türü birçok dilde 2 byte lık bir short olarak ifade edilebilir. Eğer, değer sıfır ise, FALSE, eğer sıfırdan farklı bir değer ise TRUE değeri alır. Yani bir boolean değişkene true ve false değerleri haricinde sayı değeride atayabilirsiniz ve eğer bu atadığınız sayı 0 ise, false, 0'dan farklı ise true değerini alır.

Değer belirtme:

Değişkenlere atanacak veya işlem yapılacak değerleri belirli formatlarda belirtmemiz zorunludur. Örneğin, metinsel bir ifadeyi tanımlıyorsak, bu değeri tırnak içinde yazmalıyız.

“Deneme”

Bazı dillerde metinsel veri belirtmek için tek tırnak (') kullanılır. Yani değer, iki tane tek tırnak arasına yazılır.

Eğer herhangi bir sayısal tür belirtiyorsa, o zaman direkt olarak sayıyı yazmalıyız.

Eğer bir karakter verisi belirtmek istiyorsak, bazı dillerde tek tırnak arasında yazılmalıdır bazı dillerde ise çift tırnak ve bir “c” harfi arasına yazılmalıdır

‘d’
“d”c

Eğer bir değişkene, true veya false değeri atamak istiyorsak bu kelimeleri aynen yazarız

True
False

Konuya devam etmeden önce hex (onaltılık sayı tabanı) hakkındaki bilgiye göz atalım

Hex veri formatı ile ilgili olarak verilmesi gereken bir ayrıntı bulunuyor. Bahsettiğim gibi Hex, 16’lık sayı tabanında tanımlanan bir veri formatıdır. Ondalık sayı tabanında bildiğimiz gibi 10 rakam bulunur. 16’lık tabanda ek olarak 6 tane daha işaretin bulunması ve bu işaretlerin sırayla, 10,11,12,13,14 ve 15 değerlerine karşılık gelmesi gerekmektedir.

Daha açık bir ifade ile;

0-9 arası sayılar 0-9 rakamlarıyla ifade edilir
16’lık düzlemde 10 değeri A olarak ifade edilir
16’lık düzlemde 11 değeri B olarak ifade edilir
16’lık düzlemde 12 değeri C olarak ifade edilir
16’lık düzlemde 13 değeri D olarak ifade edilir
16’lık düzlemde 14 değeri E olarak ifade edilir
16’lık düzlemde 15 değeri F olarak ifade edilir

Örnek olarak (A)₁₆ ondalık düzlemde 10’a denk gelmektedir.

Onaltılık yani hexadecimal bir değer belirtmek istiyorsak, bazı dillerde başına &H ön ekini getiririz, bazı dillerde de 0x ön ekini getiririz.

&H9 değeri 9 olur
&H11 değeri 17 olur
&H10 değeri 10 olur

0x11 değeri 17 olur
0x12 değeri 18 olur

Eğer bir virgüllü sayı (single / double / float gibi türlerde) değeri belirtmek istiyorsak; ilk önce sayısı yazıp sonrasında, kullanacağımız türe göre ekini getirmeliyiz (single ise : s, double ise : d, float ise : f).

3.4s	single olur
3.4d	double olur
3.4f	float olur

Belirtilen bu değerlerin hepsi aslında sabit değerlerdir. Yani kodcu tarafından yazılmış değiştirilemez (**değişken** olmadıkları için) değerlerdir.

2.2.3 Değişkenlerin tanımlanması

Bir değişkenin tanımlanmasını, bir uçuş için rezervasyon yapmaya benzetebiliriz. Öncelikle, rezervasyon yaptırırız, bu rezervasyon için yer ayrılır. Ancak ayrılan bu rezervasyonun ücreti ödenmez ise, geçerliliği bulunmayacaktır. Ayrıca, belirli bir süre sonra iptal edilecektir. Ek olarak, uçuş tamamlandıktan sonra da biletin önemi bulunmayacaktır.

Bir değişkeni kullanmak için öncelikle o değişkeni tanımlamamız gerekmektedir. Değişkenleri tanımlamak için, diller farklı ifadeler kullanmaktadır yani değişken tanımlama dile göre değişmektedir.

Bazı dillerde, değişken tanımlama, değişkenin türünün isminin ardından, değişkenin isminin yazılması ile sağlanır. Bazı dillerde ise, belirli anahtar sözcükler, sonrasında değişken ismi, ve son olarakta değişkenin türü yazılır. (aşağıdaki ifadelerde geçen değişken1, değişken ismini ifade etmektedir)

```
int değişken1;  
dim değişken1 as integer  
var: değişken1 as integer;  
var: değişken1: int;  
declare @değişken1 int;  
$değişken1;  
değişken1 : INTEGER;  
INTEGER :: değişken1, değişken2, değişken3  
var değişken1;  
[çeşitli dillerde değişken tanımlama ifadeleri]
```

İlk satırdaki ifade de, öncelikle veri tipi olan “int” ifadesi yazılmış akabinde, değişkenin ismi (değişken1) yazılmıştır. İkinci satırdaki “dim” ifadesi bir değişken belirtmek için kullanılan bir tür anahtar sözcüktür sonrasındaki kelime değişkenin ismidir. “as” ifadesi yine bir anahtar sözcüktür ve en son olarakta değişkenin türü belirtilmiştir. Bu veya benzeri ifadelerden birini (dile göre) kullanarak bir değişken tanımlarız ve o değişken artık hazır hale gelir.

Bir değişkeni tanımladığımızda, sadece o isim ve türde bir değişken olduğunu belirtiriz. Aksi takdirde, o değişkeni kullanamayız. Az önceki örnekte belirttiğim gibi, bir uçuş için rezervasyonumuzun olmasının pek bir anlamı yoktur. Eğer ücreti ödersek, rezervasyon doğrulanır ve bilet geçerli olur.

Aynı şekilde, bir değişken belirtildikten sonra eğer o değişkene bir değer atanmaz ise, kullanılamaz, anlamı olmaz.

Sayısal değişken tanımlarken dikkat etmemiz gereken nokta, o değişkenin uyacağı en uygun ve en küçük kümeyi seçmektedir. Örneğin bir program içinde, bir kişinin yaşını hafızada tutan bir değişken belirtmek istiyorsak, byte değişkeni kullanırız. Böyle bir durumda, tabiki, integer, smallint gibi değişkenlerde kullanabiliriz. Ancak, bu tip değişkenler daha çok yer kaplayacaktır (byte olarak). İşte bizde, sadece gerekli olduğu kadar büyük bir küme seçimi

yapmalıyız. Biliyorki bir kişinin yaşı, byte veri tipinin üst limiti olan 255'i asla geçemeyecektir. Yani 255 yaşından büyük bir kişi olamayacaktır. İşte bu nedenle, 255 değeri dolayısıyla byte veri tipi bizim için uygundur. Yaklaşık 1000-2000 kitabın bulunduğu ve kitap sayısının 15 binden fazla olmayacağı bir kitapçıyı düşünelim. Bu kitapçı kitaplarına birer seri numara verirken, smallint veri tipiyle tanımlanmış bir değişken kullanılmalıdır. Eğer byte kullanırsa, byte'ın üst limiti 255 olduğundan program hata verecektir. Eğer integer kullansa, boşuna 2 byte alan kaybetmiş olacaktır. Çünkü, kitap sayısı 15 bini geçmemektedir ve smallint'in üst limiti (32.767) bu değerden (15 binden) çok üstündür. Aynı şekilde varsayalımki, yüzde (%) olarak bir değeri hafızada tutacağız. Örneğin, bu ayki satışların, toplam alınan mala oranını hafızada tutmak için float değişkeni kullanabiliriz ve örnek olarak %54 değerini belirtmek için 0.54 veya 54 kullanabiliriz. Ancak dikkat edersek olabilecek değerler 0(%0) ile 100(%100) arasındadır. Bu nedenle böyle bir durumda yine byte değişkenini kullanabiliriz.

Nothing / Null (gerçek dünyada “yok”)

Değeri atanmamış ve sadece tanımlanmış değerlere, otomatik olarak NOTHING veya NULL değeri atanır. Nothing – Null (bazı dillerde nothing – bazı dillerde null), yok, boş, hiçbirşey anlamına gelmektedir. Atanmış olan değişkenin kullanılması için bir anlam bulunmadığını belirtir.

Tanımladığımız değişkenler NULL (nothing) değerini alır ve o değişkenlerin değerlerini okuduğumuzda bir anlam ifade etmez. Aynı matematikteki boş küme (\emptyset) veya sonsuz (∞) gibi, belirtilir ancak anlamı yoktur.

Empty (gerçek dünyada “boş”)

Bazı diller ise, bir TEMEL TİPTE değişken tanımlandığında, o değişkene, otomatik olarak belirtili türün, varsayılan değerini atar. Bu yapılan atamadaki varsayılan değere empty (yani boş) (veya başlangıç değeri) denir.

Her BASİT türün, bir boş değeri vardır. Örneğin, değeri atanmamış ancak tanımlanmış bir sayısal değişkenin, varsayılan değeri her zaman sıfırdır.

Dim i as integer
int i;

Visual Basic dilinde değişken tanımlama
C dilinde değişken tanımlama

Bu tanımlamadan sonra, i değişkeninin değeri, sıfır olur. Aynı durum diğer tam sayı tipleri içinde geçerlidir.

Dim i as long
Dim i as byte
Dim i as numeric
Dim i as short

Ondalıklı sayıların başlangıç değeri 0.0 (sıfır nokta sıfır) dır.

Dim i as double

Burada, i değişkeninin değeri 0.0 olur.

Not: Yukarıdaki empty, null, nothing anahtar sözcükleri dillere göre

değişebilmektedir.

“” (Çift tırnak)

Metin türünde tanımlanmış olan bir değişkenin varsayılan değeri “boş metin” dir. Boş metin, çift tırnak ile ifade edilir. Daha önceki bölümlerde, değerlerin (sayısal, ondalık, metinsel) nasıl ifade edilidiğini görmüştük. Metinsel değerler de tırnak içinde ifade ediliyordu. Peki olmayan bir metni nasıl olurda tırnak içinde ifade ederiz. Tabiki tırnak içine herhangi bir metin girmeyerek.

Bu bölümün başında, değişkenlerin süreklilik alanı olduğunu söylemiştim. Bir değişkenin süreklilik alanını o değişkeni tanımlarken belirtiriz. Ancak bu konu hakkında ayrıntılı bilgiler ileriki bölümlerde verilecektir.

[Ara verme bölümü]

2.2.4 Matematiksel İşlemler & Operatörler

İlk bölümde, programlamanın temelini matematik olduğunu söylemiştim. Bu yüzden diyebiliriz ki programlar içinde matematiksel ifadeler bulunacaktır. Programlar içinde matematiksel ifadeler bulunacaksa; o zaman matematiksel işlemler yapılacaktır ve bu matematiksel işlemler matematiksel operatörler tarafından gerçekleştirilecektir.

Programlama dilleri çok sayıda matematiksel operatör yani matematiksel işlem desteklemektedirler. Bunların bir kısmını bu bölümde diğer kısmını da ileriki bölümlerde göreceğiz. Matematiksel operatörler aslında belirli gruplara dahildir ve belirli özellikleri bulunmaktadır ancak bu bölümde bu konular üzerinde durmayacağız.

Değer atama (Eşittir Operatörü “=“)

Tanımlamış olduğumuz değişkenlere değer atamak için, “assignment” yani atama operatörü kullanılır. Operatörün tanımını daha önceki bölümlerde vermiştim. Atama operatörü ise matematikten bildiğimiz eşittir ile aynı görevi üstlenmektedir. Yani aynı matematikte bir değişkene değer atarken ki gibi kullanılır.

Örneğin;

$x \in \mathbb{Z}^+$ şeklinde tanımlanmış bir değişkene değer atamak için, değişkenin ismini yazdıktan sonra eşittir işareti kullanırız, sonrasında da değişkene atamak istediğimiz değeri yazarız.

$$x = 3$$

şeklinde bir değişkene (x) değer (3) atanabilir.

Biraz daha ayrıntılı açıklayayım. "Deneme" adında bir değişkenimizin olduğunu düşünelim ve bu değişkene 3 değerini atamak istediğimizi düşünelim. Aynı matematikteki gibi;

$$\text{Deneme} = 3$$

şeklinde yazmamız yeterli olacaktır. Burada kullanılan eşittir "=" atama operatörüdür ve genelde birçok dilde bu şekilde kullanılır. Bazı dillerde "Deneme := 3" (iki nokta üst üste - eşittir) şeklinde bazı dillerde ise "Set Deneme = 3" şeklinde kullanılır.

Bir atama operatörü kullanıldığında, değişkenin o anki değerini değiştiririz (üstüne yazarız) ve eski değerini yok etmiş oluruz. Tabi burada ek bir durum söz konusudur. Nasıl bir değişkene değer atayabiliyorsak, bir değişkene başka bir değişkenin değerini de atayabiliriz.

Varsayalım ki elimizde iki değişken olsun: x ve y. x değişkenine bir değer atıyoruz.

x=4

Dikkat etmemiz gereken, x değişkenine bir daha başka bir değer atayınca kadar hep bu değeri (yani 4'ü) tutacağıdır. Yani bundan sonra x'e başka değer ataması yapmaz isek x her zaman 4 olacaktır.

Y değişkenine de, X değişkeninin değerini atıyoruz. Bu işlem için Y=X yazıyoruz.

Y=X

Bu ifadeden sonra, Y'nin değeri de 4 oluyor. Yani sonuç olarak bu yaptığımız iki ifade sonucunda hem x hemde y değişkenlerinin değeri 4 oluyor.

X=4

Y=X

(x=4) (y=4)

Örneğin;

Tanımla: Deneme

Tanımla: Deneme2

Deneme=2

Deneme2=Deneme

Böyle bir atamada, değişkenin değeri ilk değişkene atanır (eşitliğin sol tarafındaki). Yani Deneme2 adlı değişkenin değeri 2 olur. Bunun dışında atama esnasında farklı durumlar söz konusu olabilir, bu konuyla ilgili ayrıntılı bilgi, pointer işlemleri bölümünde anlatılmaktadır.

Dört işlem operatörleri

Dört işlem operatörleri tahmin ettiğiniz gibi Toplama İşlemi (+) Operatörü, Çıkarma İşlemi (-) Operatörü, Çarpma İşlemi (*) Operatörü, Bölme İşlemi (/) Operatörüdür ve bu operatörlerin kullanımı matematiktekiyle aynıdır.

Yani bir matematiksel işlem gerçekleştirmek istiyorsak, bu dört işlem operatörlerinden istediğimizin başına ve sonuna (solu ve sağına) işlem yapmak istediğimiz sayıları yazıyoruz.

3 + 5

12 / 2
7 * 3
12 - 23 gibi...

Bir matematiksel işlem yapıp bu işlemin sonucunu bir değişkene atayalım.

$$X = 3 + 5$$

Yukarıdaki ifadede ki “+” işareti operatördür, “3” ile “5” ise işlenendir, yani işlem den etkilenendir.

Yukarıdaki işlemin sonucunun hepimiz 8 olduğunu biliyoruz. Peki bu bilgiyi neden tekrarladık ? (ilkokul öğrencileri bile çok basit bir şekilde sonucun 8 olduğunu bilebilir) Buradaki amacım 3 ile 5’i toplamak değildi. Bir atama operatörü (bazı dillerde =, bazı dillerde :=) gördüğümüzde yapmamız gereken; ilk önce, eşittirin sol tarafını tamamen unutmaktır. Daha sonra, eşitliğin sağ tarafına bakıp sağ taraftaki işlemi gerçekleştirirsiniz. En son olarak, (bu sefer eşitliğin sağ tarafını kapatırsınız) elde edilen değeri sol taraftaki değişkene atayabilirsiniz.

$$\begin{aligned} X &= 3 + 5 \\ (\text{sol tarafı görmeyin}) &= 3+5 \\ (\text{sol tarafı görmeyin}) &= 8 \\ X &= (\text{sağ tarafı görmeyin}) \\ X &= 8 \end{aligned}$$

%99.99’unuz, yine bir değişikliğin olmadığını düşünüyordur. Bende ilk başta böyle düşünmüştüm. Her ne kadar programlama ile matematik çok benzese de, bazı noktalarda ayrılır. Örneğin matematiksel bir işlem yazalım.

$$\begin{aligned} X &= \{X \mid X \in \mathbb{Z}^+\} \\ X &= 3 \\ X &= X + 1 \end{aligned}$$

Bu ifade, matematikte, matematiksel cinayet olarak adlandırılır. Bir sayı, kendisinden bir fazlasıyla eşit olamaz. Ancak bu ifade programlama da doğru çalışabilen bir ifadedir.

Bir eşitlik gördüğümüzde, ve bu eşitlik yukarıdaki gibi değer atama eşitliği ise, o zaman eşitliğin sol tarafını kapatacağız. Eşitliğin sağ tarafındaki işlemi yaptıktan sonra, sol taraftaki değişkene atayacağız.

$$\begin{aligned} X &= X + 1 \\ &= X + 1 \text{ (sol tarafı kapattık - görmezden geldik)} \\ &= 3 + 1 \text{ (X'in en son atanan değerini X'in yerine yazdık)} \\ &= 4 \text{ (işlemi tamamladık)} \end{aligned}$$

Şimdi sol taraftaki X değişkenine sağ tarafta hesaplanan 4 değerinin atamasını yaparız. Böyle bir işlem sonucunda, X, 4 değerini alır.

Başka bir örnek;

$$Y = 1$$
$$Y = Y + 5$$

Bu işlem sonucunda, Y, 6 değerini alır.

Bir eşitlik ifadesinde, sol tarafta sadece ve sadece bir değişken bulunabilir. Çünkü bir değere, değer atanamaz. Ancak bir değişkene değer atanabilir veya başka bir değişkenin değeri atanabilir. Yani;

$$X = 3$$

şeklinde X değişkenine 3 değeri atanabilir veya

$$Y=4$$
$$X=Y$$

şeklinde, X değişkenine, Y değişkeninin değeri (4) atanabilir.

Sadece matematiksel değerler yerine, metinsel değerler de atayabilmekteyiz. Örneğin;

$$X = \text{"test"}$$
$$Y = 3.4$$

İfadeler (Expression – Statement)

Türkçeye çevrilen kitaplarda, internet sitelerinde, hem expression hemde statement sözcükleri için **ifade** kelimesi kullanılır. Nitekim, anlamları birbirlerine çok yakındır.

Expression

1. Anlatım, anlatım, **deyim**. i.
2. deyim, tabir.
3. (yüzdeki) **ifade**.
4. ifade, anlatım, dışavurum.

Statement

1. Deyim, Söylem, Tümce. deyim. hesap raporu.
2. **ifade**; demeç, beyanat.
3. anlatım, deyim.

[e15].

Hâl böyle olunca bu iki terim birbiriyle her zaman karışmaktadır. Eğer ben birine **tabir** birine **ifade** dersem, bir internet sitesindeki makalede, başka bir kitapta, veya birisiyle konuşurken bu kelimeyi onlar farklı yorumlayabilir (yani statement ile expression karışabilir). İşte bu nedenle, bu kelimelerin orjinal olarak bırakıyorum ve ingilizce halleriyle kullanıyorum. Siz istediğiniz gibi çevirebilirsiniz.

$$X = Y + 1$$

Yukarıda gösterdiğim örnek için; $Y + 1$ 'in bir expression, $X = Y + 1$ eşitliğinde bir statement olduğunu söyleyebiliriz. Expression'lar tek başlarına bir anlam ifade etmezler, ancak bir değerleri bulunur. Örneğin eğer Y değişkeninin değeri 2 olsaydı, $Y + 1$ expression'ın değeri 3 olacaktı. Ancak bir expression'u programlama dilinde direkt olarak kullanamazsınız. Bu yaptığınız işlemin sonucunu bir yere yazmalısınız yani bir değişkene... O yüzden, programlama dilleri, statement'larla çalışmaktadır. Statement'ların değeri ve türü

yoktur. Ancak belirttiğim gibi expression'lar bir değere ve hatta o değerin türüne (bu örnekte bir integer veya tam sayı kökünden gelen bir başka tür: byte, small int..) sahiptirler. Bir expression'ın çalışması, o anki durumu veya değişkenlerini değerini değiştirmez, ancak bir statement'ın çalıştırılması değişkenin değerini veya o anki durumu değiştirebilir.

Değişkenlerin Ayarlanması (başlangıçtaki ayarlarının yapılması)

Bazı dillerde, temel veri türünde belirlenen değişkenlerin başlangıç değerlerinin atanması zorunludur. Yukarıda, bir değişken belirlediğimizde (bazı programlama dillerinde) otomatik olarak bir empty yani boş değerini aldığını söylemiştim. Bazı dillerde de, mutlaka bir değişkene başlangıç değeri vermeniz gerekmektedir. Örneğin; tanımlanacak olan bir integer'ın (tamsayı) başlangıç değerinin 0 olacağını hepimiz tahmin ederiz ancak yine de, bazı diller bu değer girilmesini zorunlu kılar.

```
int sayı = 0
Dim sayı as Integer = 0
```

.. şeklinde ifadelerle tanımladığımız değişkene bir başlangıç değeri vermiş oluruz. Aynı şekilde, bu durum diğer veri türleri içinde geçerlidir.

```
Byte b = 0
String test = ""
Float f = 0.0
```

Bir değişkene başlangıç değeri olarak illa ki o değişkenin boş değerini (bu örnekteki gibi 0) vermek zorunda değiliz. Bir değişkeni istediğimiz (o değişkenin izin verdiği aralıklarda) değerler başlatabiliriz.

```
int i = 234
float k = 4.3424
String deneme = "test"
```

Casting

Matematikten hatırladığımız üzere, bir sayı kümesi bir başka sayı kümesini kapsayabilmektedir. Örneğin, doğal sayılar, tam sayılar kümesi tarafından kapsanmaktadır. Yani, tam sayılar kümesi, hem doğal sayıların sahip olduğu bütün sayıları, hemde, daha fazla ek (negatif sayılar kümesi) sayıları kapsamaktadır.



Daha önceki bölümlerde, veri türlerinin, kümeler gibi olduğunu söylemiştim. Bir kümeye ait bir eleman başka bir kümeye geçebiliyorsa (belirli durumlarda), o halde, bir veri türüne ait elemanda başka bir veri türüne geçebilmektedir. İşte bu olaya, casting denir.

Veri türleri arasında sayı değerleri için tasarlanmış olanlar arasında veri türü değiştirilebilir. Örneğin, integer türünde belirttiğimiz bir sayıyı, byte'a çevirebiliriz.

```
int i = 34  
byte k;  
k = i;
```

Yukarıdaki ifadede, yeni tanımlanmış olan k adlı byte türündeki değişkene değer olarak bir integer türündeki değişkenin değeri (yani 34) atanıyor. Aynı şekilde, byte olarak tanımladığımız bir değişken, integer türünde tanımladığımız bir değişkene de atanabilir.

```
byte k = 29;  
int i ;  
i = k;
```

Dönüştürme işlemi sadece integer ile byte arasında değil diğer türler arasında da olabilir, ancak string hariç... Çünkü bir metin sayısal bir ifadeye dönüştürülemez. Bunun haricinde, bir float (ondalık sayı) bir tamsayıya (integer, short, byte) dönüştürülebilir.

```
Float x = 3.4;  
int i;  
i = x;
```

Yukarıdaki ifade de, i'nin değeri 3 olur. Bir ondalık sayı, tam sayıya çevrilirken, tam sayıda ondalık kısım desteklenmediği için, doğal olarak virgülden sonrası önemsenmez. Bazı diller, bu dönüşüm sırasında yuvarlama yapar bazıları da sadece tam kısmı kullanırlar.

Dönüştürme işleminde dikkat edilmesi gereken, hataların oluşmamasıdır. Çünkü eğer bir değeri başka bir türe dönüştürüyorsanız ve bu dönüştürdüğünüz tür, o değeri desteklemiyorsa, hata oluşur. Örneğin, bildiğimiz üzere byte veri türü, 0-255 arası tam sayı değerlerini desteklemektedir. Bir integer'ı byte'a çevirirken, 255'ten büyük bir sayı kullanırsak doğal olarak limit dışında bir veri olduğu için hata oluşacaktır.

```
int i=258;  
byte k;  
k = i ;
```

Yukarıdaki ifadede, son satırda hata oluşacaktır. İşte bu gibi durumlarda dikkatli olmamız gerekmektedir.

Birçok dilde, dönüştürme işlemi yaparken bir arabulucu kullanmamız gerekmektedir. Yani, bir değeri, başka türe dönüştürürken, "bu değeri x türüne dönüştür" gibi bir ifade belirtmemiz gerekir.

```
int i = 343;  
short k;  
k = (short) i;
```

Yukarıdaki ifadede, son satırda, i değişkeni önüne yazılı olan parantez içindeki short, takip ettiği değeri veya değişkenin değerini, parantez içinde yazılı olan türe değiştir anlamına gelmektedir.

```
byte c = 43;  
int m;  
m = (int) c;
```

Yine aynı şekilde, yukarıdaki ifadede de, c değişkeninin değeri, byte'tan, int'e çevrilmiştir. Bir değerin veya değişkenin önüne parantez içinde hangi türün ismini yazarsak, değer o türe dönüşür. Bazı dillerde ise farklı anahtar sözcükler kullanılır. Bu konu hakkında ayrıntılı bilgi dil syntax'larında mevcuttur.

İki tip dönüştürme söz konusudur: Upcasting ve downcasting. Eğer bir değer, bir üst kümeye çıkıyorsa (yani daha geniş bir kümeye, örneğin byte'tan, integer'a) upcasting olarak, bir alt kademeye iniyorsa (yani daha az geniş bir kümeye, örneğin integer'dan, byte'a) downcasting olarak adlandırılır.

2.2.5 Array

Kalıplaşmış veri türleri arasında array veri türü bulunmaktadır. Array, dizi demektir ve anlaşılacağı gibi bir dizi değişken anlamına gelmektedir. Nasıl matematikde diziler bulunuyorsa, programlamada da bir dizi AYNI TÜRDE değişken tanımlamak, array'ler ile mümkündür. Array'ler, aynı türde, aynı isimde, belirli bir sayıda, bir dizi değişken belirtmek için kullanılırlar.

Array'leri daha iyi kavramak için, bir stadyumda çalışan bir görevliyi düşünelim. Görevli, çalışıyor olduğu stadyumda, futbol maçı yapacak olan bütün takımların 11'lerin listesini (yani sahada oyun oynayacak olan futbolcuları) bir kağıda yazdığını düşünelim.

1. Coşkun
2. Mehmet
3. Mustafa
4. Ali
5. Ergün
6. Suat
7. Fuat
8. Halil
9. Bedir
10. Murat
11. Kadir

Görevlinin, bu listelerden boş yüzlerce hazırladığını ve maçtan önce doldurduğunu düşünelim. Dikkat edersek, listenin içinde olabilecek eleman (yani futbolcu sayısı) her zaman 11'dir, ve listeye yazılacak bu 11 veri, birer isim olduğu ve isimlerde ancak metin yani string olarak ifade edildiği için, 11 tane string değişken kullanmamız gerekmektedir. İşte böyle bir durumda, 11 tane aynı türden peşpeşe değişken belirtmekten, bir dizi belirtip, bu diziyi kullanırız.

Bir değişkeni tanımlarken ismini ve türünü belirtiriz, bir array tanımlarken ise, isim ve tür bilgilerini belirttikten sonra, ek olarak sadece, o dizinin boyutunu belirtiriz.

Örneğin;

```
int a;  
ifadesi integer türünde “a” isminde bir değişken belirtirken,  
  
int a[3];
```

ifadesi integer türünde “a” isminde 3 değişken belirtir. Bu belirtilen 3 değişken, a[0], a[1], a[2] şeklindedir. Burada dikkat edilmesi gereken nokta: hemen hemen her programlama dilinde indeksler (yani bir dizinin kaçınıcı elemanı olduğunu bildiren sayı) her zaman 0’dan başlar.

İlk ifadede a=3, şeklinde değer atama yapabilirken, dizi olarak tanımlanmış bir değişkene a[0] = 3, a[1]=6, a[2] = 23 şeklinde veri ataması yapılır.

Başka bir örnek olarak;

```
string futbolcu[11];  
futbolcu[0] = “Coşkun”  
futbolcu[1] = “Mehmet”  
futbolcu[2] = “Mustafa”  
futbolcu[3] = “Ali”  
futbolcu[4] = “Ergün”  
futbolcu[5] = “Suat”  
futbolcu[6] = “Fuat”  
futbolcu[7] = “Halil”  
futbolcu[8] = “Bedir”  
futbolcu[9] = “Murat”  
futbolcu[10] = “Kadir”
```

Sonuç olarak bir dizi aynı türde, aynı isimde değişkene array yani dizi denir. Bazı dillerde array belirlemek a(3) şeklinde yapılmaktadır yani köşeli parantez yerine normal parantez kullanılmaktadır.

2.2.6 Parantezler

Matematikte 3 çeşit parantez bulunmaktadır: Tırnak, Köşeli, Normal parantez. Bu parantezler içindeki işlemler, parantez önceliğine göre yapılır. İlk olarak, Normal parantez, daha sonra köşeli, daha sonrada tırnak parantezin içi yapılır. Ayrıca, bir ifade de, ilk önce, en içte bulunan işlemlerden başlanılarak, işlemler gerçekleştirilir ve bu sıraya göre yapılır. Örneğin;

$$\{2 * [(3+5) * 7]\}$$

ifadesinde, ilk önce, en iç parantez olan normal parantezin içi işlenir (). Bu parantezin içinin değeri 3+5 =8 ‘dir. Daha sonra, bir dış parantezde bulunan ifade olan 7 ile çarpılır ve 56 elde edilir. En son olarak, 2 ile çarpılır ve 112 elde edilir.

Programlamada bu üç parantezin sadece bir tanesi, normal parantez matematik işlemleri için kullanılır. Diğer parantezlerin kullanıldığı yerlerde yine normal parantez kullanılır. Yukardaki örneği program içinde yazdığımızda, ifade;

$$(2 * ((3+5) * 7))$$

şeklini alacaktır. Yine aynı şekilde, en içten başlayıp dışa doğru hesaplama yaparız. Diğer parantezler başka işlemler için kullanılır ancak birçok programlama dilinde normal parantezler matematiksel işlemler için kullanılmaktadır. Diğer parantezlerin ne gibi işlemlerde kullanıldığı ileriki bölümlerde işlenecektir.

[Ara verme bölümü]

2.2.7 Toplam sembolü / Çarpım sembolü:

Birçoğunuz, matematikten bildiğimiz toplam / çarpım sembolünün konumuzla ne alakası olduğunu merak ediyordur. Matematikteki, toplam sembolü yada sigma, ve Çarpım sembolünü hatırlatayım. Toplam sembolü, 1'den (veya belirtili başka bir sayıdan) başlayarak belirtilen sayıya kadar olan sayıları toplayıp sonucunu veren işlemdir. Örneğin, belirtilen sayının 6 olduğunu düşünersek; 1 den 6 ya kadar olan sayıların toplamını bulur.

$$1 + 2 + 3 + 4 + 5 + 6 = 21$$

Aynı şekilde, belirtilen sayı eğer 7 olsa idi, 1'den 7'ye, 9 olsa idi, 1'den 9'a kadar olan sayıların toplamını bulup bize sonuç olarak verirdi. Aynı şekilde, çarpım sembolü, de, 1'den belirtilen sayıya kadar olan sayıları çarpar. Yukarıdaki örnekle devam edelim, 1'den 6'ya kadar olan sayıların çarpımını bulalım. (Bu işlemin sonucu 6!'dir – 6 faktöryel)

$$1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$$

Peki bu ifadeleri nasıl tanımlıyoruz, yani bu ifadelerin sembolleri nelerdir...

$$\sum_{i=1}^5 i$$

Yukarıdaki ifade, toplam sembolü ifadesidir. 1'den, 5'e kadar olan sayıların toplamına eşittir.

- Yukarıdaki ifade, i değişkenine, başlangıç değeri olan "1" i atar (atama yapar) (altta yazılı olan değer).
- Daha sonra, bu değeri bir havuza ekler.
- Sonraki adım olarak, i'nin değeri bir artırılır ve i değişkenin değeri "2" olur.
- Aynı şekilde bu değerde, havuza eklenir.
- Bu işlem, peşpeşe i'nin değeri, üstte yazılı "üst limit" (bu örnekte 5'tir) e ulaşınca kadar artar ve havuza eklenir.
- Sonuçta havuzdaki sayılar toplanır : $1 + 2 + 3 + 4 + 5 = 15$ değeri elde edilir.

Eğer ifade,

$$\sum_{i=1}^5 2i$$

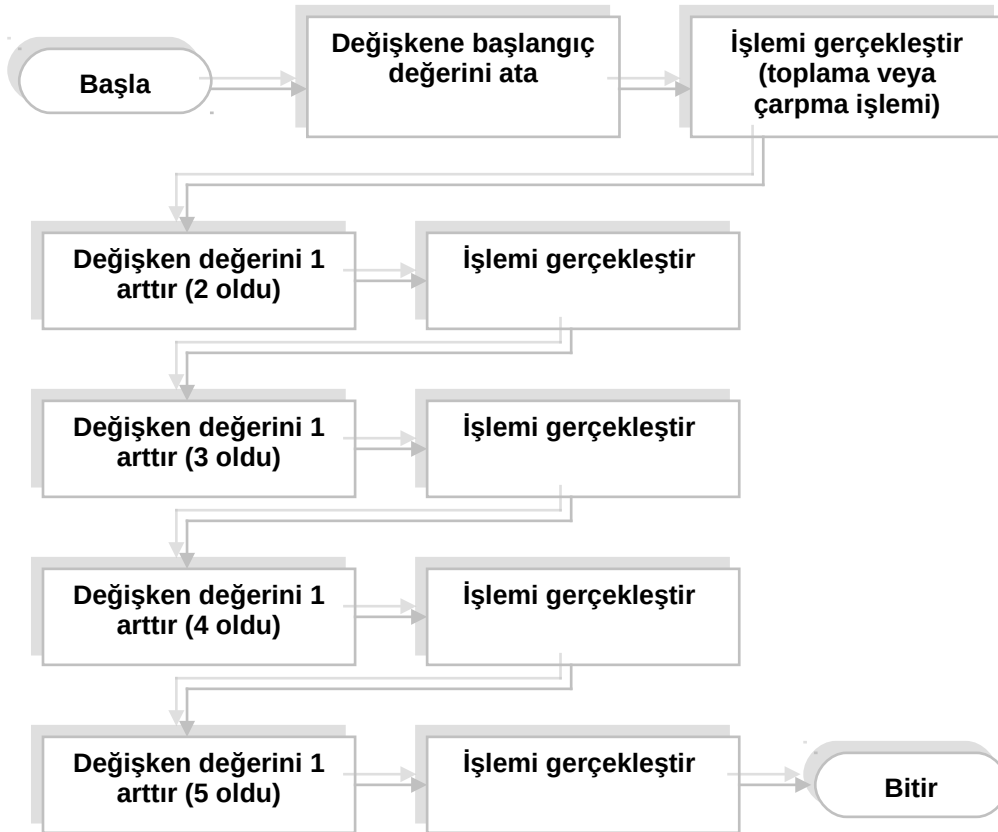
olsa idi, o zaman, i'nin değışen her bir değeri için, değerin 2 katını havuza ekleyecektik. Burada önemli olan yapılan işlem değil, işlem sırasında, i'nin bir değerden başlayıp, değerin bir bir artması ve bu sırada, almış olduđu değlerle, işlemi yapması, bu sürecide üst limite ulaşınca kadar tekrar etmesidir. Yani yapılan işlemin toplama olması önemli değil, i'nin değerin bir bir değışmesidir ve bu değışiklikten sonra i'nin o anki değeriyle işlemin yapılmasıdır.

Aynı durum, Çarpım sembolü içinde geçerlidir.

$$\prod_{i=1}^5 i$$

Burada, yine i başlangıç değeri olarak "1" atanır, ve bir bir artarak, üst limit olan 5'e ulaşınca kadar devam eder. Bu sırada, elde edilen bütün i değeri çarpılır ve sonuç olarak $1 \times 2 \times 3 \times 4 \times 5 = 120$ elde edilir.

Bu işlemi akış diagramı olarak ifade edelim



Figür 2.2.6 A Toplam sembolünün akış diagramı

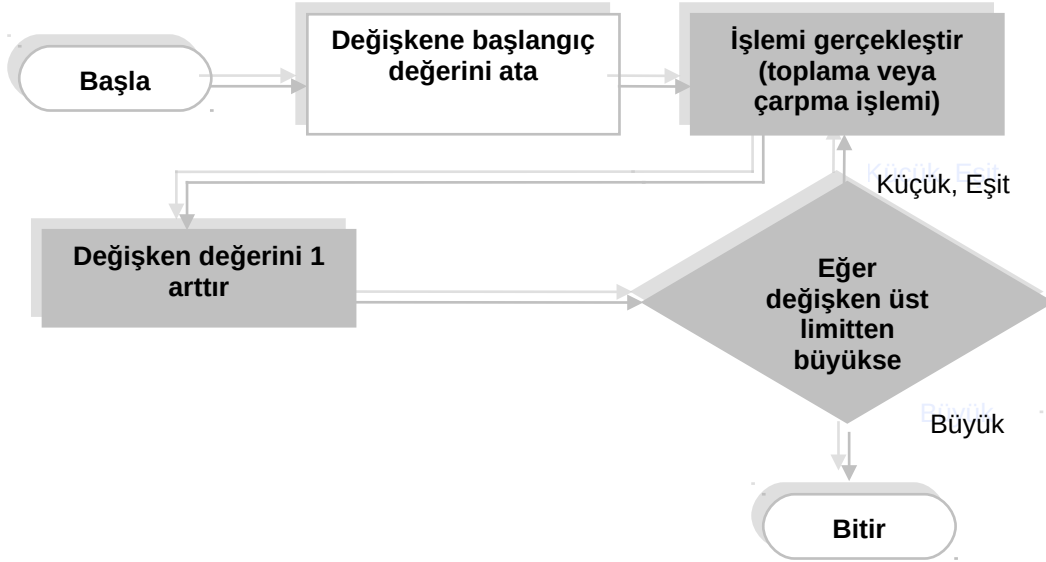
Yukarıda gördüğünüz gibi, benzer işlemi 5 kere peş peşe yaptıktan sonra, aynı toplam sembolü veya çarpım sembolünde olduğu gibi kısaltıcı işlemler kullanabiliriz. Yani

$$1+2+3+4+5 = 21$$

ifadesini kullanmaktansa

$$\sum_{i=1}^5 i = 21$$

ifadesini kullanmak daha mantıklı olackatır.



Figür 2.2.6 B Toplam sembolünün akış diagramı

Yukardaki eşkenar dörtgene ilk ulaştığımızda değişkenin değeri 2 olacaktır ve üst limitten küçük olduğu için bir kez daha "işlemi gerçekleştir" kutusuna dönecektir. Bu süreç 3 kere daha devam edecektir ve bundan sonraki 4. sünde, değişken değeri 6 olacaktır ve bu değer üst limitten büyük olduğu için işlem bitirilecektir. Yukarıdaki işlem, belirtilen değere göre, 5 kere değil N kere devam eder ve sonuca ulaşır.

Gri renkteki diagram nesneleri, birden fazla kere çağrılabilir. Yukarıdaki diagramda aslında tek tek yapıyor olduğumuz bir seri işlemi, otomatikleştirdir. Toplu bir silahı düşünelim. Dikkat edersek, her tetiğe bastığımızda, hem horoz, mermilerden hizada olana vurur ve ateş ettirir, hemde, topu bir döndürür (yani bir sonrakine geçer). Aynı şekilde toplam sembolünde, hem toplama işlemini **gerçekleştirir** hemde o andaki değişkenin değerini bir **arttırır**.



Figür 2.2.6 C Toplu bir silah

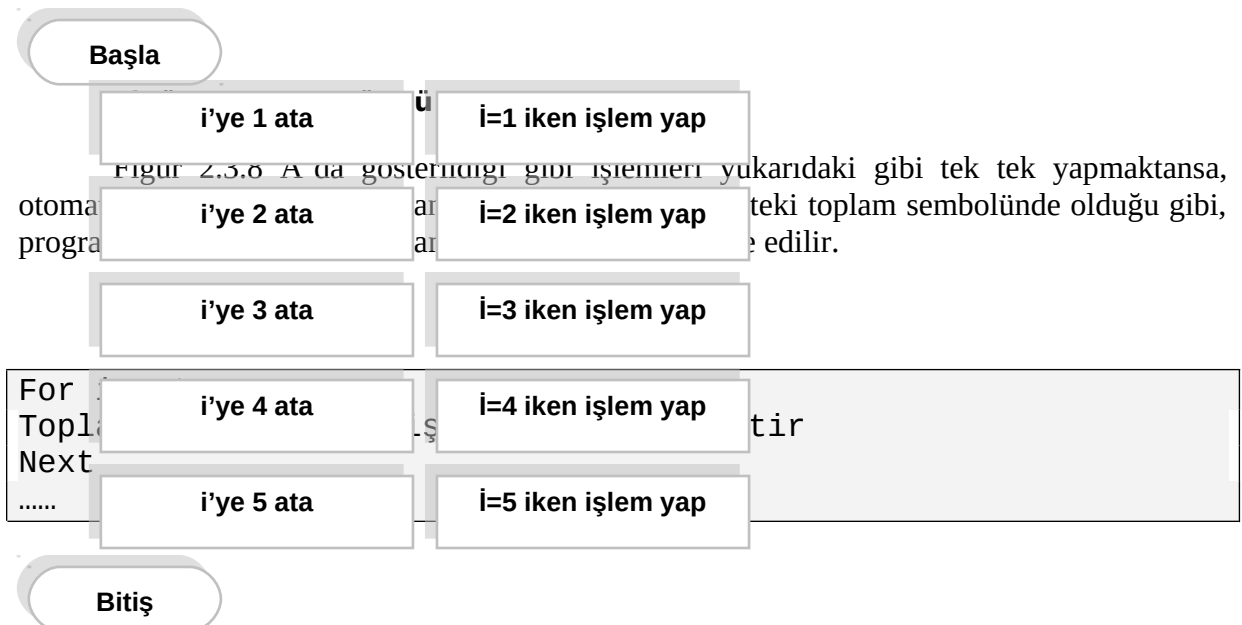
İlk bölümde, fonksiyonların bir kaset çalar (yani teyp) gibi çalıştığını içine hangi kaseti yerleştirirseniz o kasetin içindeki müziği çaldığını söylemiştim. İşte aslında toplam sembolü veya çarpım sembolü de, aynı fonksiyonlar gibi, **girilen değerlere göre**, işlemi, belirtilen sayı kere tekrar edecektir.

2.2.8 Döngü

İkinci bölümün başında, programlama dillerinin aslında kodculara kolaylık sağlamak için tasarlandığını belirtmiştim. Bu açıdan baktığımızda, programlamada, birçok kısaltma bulunması olağandır.

Ben bu ifadeyi ilk duyduğumda, gerçek hayatta daha önce karşılaşmadığım için (denk gelmediğim için) ne olduğunu anlayamadım. Hatta bu kelime ile ilgili bildiğim tek ifadenin "kısır döngü" deyimi olduğunu hatırlıyorum. Biri karşınıza çıkıp, programlama da döngü vardır derse sizde benim gibi şaşırabilir veya anlamayabilirsiniz. Çünkü yaklaşım oldukça saçmadır.

Az önce gösterilmiş olan, toplam ve çarpım sembollerini hatırlayalım. Değişkene bir başlangıç değeri verip, onu birer arttırarak, işlemi yapıp; daha sonra üst limite ulaşınca kadar tekrarlıyor.



$$\sum_{i=1}^5$$

Yukarıdaki ifadenin matematiksel karşılığı $\sum_{i=1}^5$ şeklinde olacaktır. Yukarıdaki ifadenin bir sigma veya çarpım sembolünden farkı yoktur. “**For**” kelime anlamı olarak “**için**” demektir, “**to**” burada “**e kadar**” anlamına gelmekte, “**next**” ise sonraki demektir. Yani “**i**” değişkeni **için**, 1 den 5 **e kadar** anlamına gelmektedir. Figür 2.2.3 A’daki, akış diagramı, bu işlem için aynen geçerlidir.

For ve Next bloğu arasındaki işlem veya işlemler, bu döngü sürdükçe devam eder. İlk başta “i” değişkenine “1” değeri atanır. Yani; bilgisayara “i eşittir 1 için” denilir. Daha sonra “i” değeri “1” olmak üzere blok arasındaki işlem (her ne işlem yapılması isteniyor ise, her ne işlem yapılması yazılmış ise) gerçekleşir. Program “Next” satırına ulaştınca, “i” nin değerini bir artırır, For satırına geri döner, eğer, “i” üst limitin dışında ise, işlem bir daha tekrar edilmez ve “Next” satırından sonraki satıra atlanır, bu satırdan devam edilir. (bloklar ile ilgili ayrıntılı bilgi 5. bölümde verilmiştir)

Blok arasındaki bölüme, toplama veya çarpma işlemi yerine herhangi bir işlem yazabiliriz.

DEBUG Aşağıdaki örneğe geçmeden önce; hata tespitinde veya kodcuya mesaj göndermekte kullanabileceğimiz bir araçtan bahsedeyim. Program içinde herhangi bir mesajı dışarı, kodcuya göstermek için Debug aracını kullanırız. Program içinden dışarı herhangi bir veriyi göstermek için kullanılan debug farklı dillerde farklı şekillerde ifade edilebilir. (Debug sadece, programlarken, kodcu tarafından kullanılan bir araçtır)

```
cout << “mesaj”
debug.writeline(“mesaj”)
system.out.println(“mesaj”)
printf(“mesaj”)
```

şeklinde farklı dillerde farklı olarak ifade edilir.

Not: Programlarda, kullanıcıya mesaj görüntülemek için MsgBox(mesaj kutusu) kullanılır. Aşağıdaki, mesaj kutusuna bir örnektir. Böyle bir mesaj kutusunu görüntülemek için sadece Mesaj göster komutunu kullanıp, göstereceği mesajı yazmanız yeterli olacaktır.



Figür 2.2.8 B : Mesaj kutusu

Not: Aynı şekilde, programlarda, kodcuya mesaj görüntülemek için başka bir yöntemde bulunmaktadır. Bu işlemin nasıl yapıldığını bilmemize veya anlamamıza

gerek bulunmamaktadır. Örneğin, programdan “Test Mesajı” görüntülemesini istediğimizde, aşağıdaki gibi bir çıktı verir.



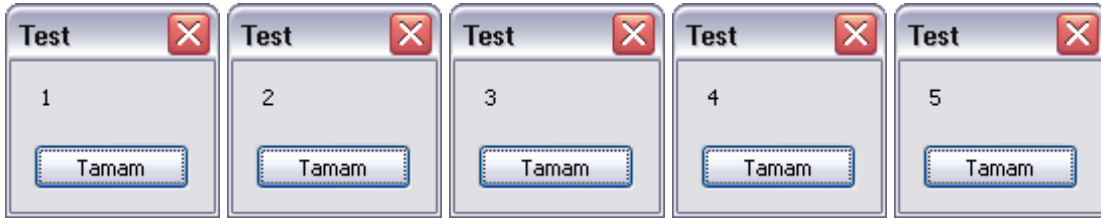
Figür 2.2.8 C : Çıktı örneği

Bu ifadelerin anlamları yoktur, istediğiniz mesajı görüntüleyebilirsiniz. Burada sadece size örnek göstermek için test mesajı metnini kullandım.

Örneğin;

```
For i=1 to 5
Şuandaki 'i' nin değerini görüntüle      (yukarda bahsedildiği
gibi mesaj kutusu ile)
Next
.....
```

şeklinde bir örnek, bize 1,2,3,4,5 çıktısını görüntüleyecektir.



Figür 2.2.8 D : Bir döngü ile mesaj gösterimi

Döngü işlemini, biz kendimiz yazsa idik (yani programdaki for ifadesini kullanmak yerine)

Değişkene başlangıç değerini ver

İşlemi yap

Değişken değerini arttır

Eğer değişken değeri, üst limitten yüksekse bitir, değilse, "İşlemi yap (2. satıra)" satırına geri dön

şeklinde olacaktı.

Peki döngüleri nerede kullanırız? Döngü kullanmadan bir program yazmak neredeyse imkansızdır. Bu güne kadar yüzlerce proje geliştirdim ancak döngüsüz 3-4 tane proje yazabildim. Bir firmasının ve müşteriler içinde borcunu ödemeyenleri bulmanız gerekiyor. O zaman her bir müşteri için, 1. müşteriden başlayarak, sonuncu müşteriye kadar devam edecek bir döngü yazmanız ve içine, i ninci müşterinin borcunu ödeyip ödemediğini kontrol etmeniz gerekir. Bu işlemi tek tek her bir müşteri için yapmaktansa, bir döngü kullanarak hepsi için otomatik olarak yapmak daha kolaydır.

Örneğin;

```
For i = 1 to Müşteri Sayısı
```

```
i'ninci müşteri borcunu ödemiş mi?  
Next
```

Döngüler seriler için de kullanılır. Örneğin;

Müşteriler (1) = 1. müşteri yani Ahmet Özsoy
Müşteriler (2) = 2. müşteri yani Canan Yeşilçiçek
Müşteriler (3) = 3. müşteri yani Adnan Güvenoğlu

```
For i = 1 to 3  
İsmi yaz : Müşteriler ( i )  
Next
```

Yukarıdaki ifade, sırayla bütün müşterilerin ismini yazacaktır.

Varsayalım ki elimizde bir matematiksel seri olsun. (Daha önceden işlediğimiz array konusunu hatırlayabilirsiniz)

Seri = {5, 4, 6, 17, 98, 0, 11}
Seri (1) bize serideki ilk nesneyi verir: 5
Seri (2) bize serideki ikinci nesneyi verir : 4
Seri (N) bize serideki N. Nesneyi verir

```
For i = 1 to 7  
Sayı görüntüle : Seri ( i )  
Next
```

Program, burada da, seri içindeki bütün rakamları görüntüleyecektir.

Döngülerde, dikkat edersek, yapılacak olan işlemin aynı olduğunu sadece işlemin bir parametresinin değiştiğini görürüz. Yani aynı işlemi bir seri içindeki elemanlara tek tek uygulamaktansa, döngü sayesinde tek bir kere de, bütün elemanlara uygulayabiliriz. “HER BİRİ” için metni kullandığımız ifadeleri, otomatikleştirmek, döngülerle mümkündür.

Aynı döngü ifadesi, birçok dilde aşağıdaki şekilde ifade edilir.

```
For (i = 1; i<8; i++){  
işlemler  
}
```

ÇEVİRME	<p>Hatırlatma_1'i hatırlayalım, amacımız, gerçek dünyada metinsel olarak ifade ettiğimiz bir işlemi bilgisayar içine programsal kod olarak aktarabilmek. Gerçek dünyada, "her biri <nesne>", "hepsi <nesne>", "her biri <nesneler>", "bütün <nesneler>", "Tümü <nesne>" şeklinde kullandığımız ifadeler için, programlama dünyasında "for" yada "döngü" ifadesini kullanırız.</p> <p>Her bir elemanın maaşını hesapla Hepsini (her işçiyi) çağır Her biri kaydolsun Bütün müşterilerin faturalarını görmek istiyorum Kayıtların tümünün dökümünü almak istiyorum</p>
----------------	---

Çeviri 2 : Döngü

[Ara verme bölümü]

2.2.9 Matematik Mantığı ve Önergeler

Matematikteki konu olan mantığı gördüğümüzde, birçoğumuz, "bu konuyu nerede kullanıcaz, ne işimize yarayabilir" demişizdir. Ancak bilgisayar dünyası matematikle o kadar içli dışlı ki, en alakasız görünen konular bile bilgisayarı anlamınıza yardımcı olur.

A ise B, B ise C, B veya C gibi ifadeleri hepimiz hatırlarız. Tabi bunlardan önce, ve daha önemlisi önermeleri... Aşağıdaki cümleler önermelere örnektir:

3 asal sayı değildir.
Ahmet 121 kilo değildir.
Bir yılda 5673 gün vardır.
Bugün hava yağmurludur.
71 asal sayıdır.
Bora 53 yaşındadır.
Yeşil bir renktir.
Otomobiller uçar.

Öneri, yargı belirten bir çeşit ifadedir ve bir önerinin değeri ya "doğru" dur yada "yanlış" tır. Birde, iki veya daha fazla önerinin birleştirilmesi durumu söz konusudur. Örneğin;

Ahmet doktor **ise**, 30 yaşındadır
Bugün hava yağmurlu **ise**, Emel şemsiye almıştır.
Elimdeki top mavi **ise**, borsa düşmüştür.

Dikkat ederseniz, ifadeler, bir koşul bir de sonuçlandırma bölümlerinden oluşmaktadır. Eğer, koşul doğru ise, sonrasında takip eden ifade doğru olmak zorundadır. Bu sistem programlamada da benzer şekilde işlemektedir.

Örneğin;

Eğer saat 7:00 **ise**, alarmı çal.
Eğer kullanıcı doğru şifreyi girdiyse, programı çalıştır.

Eğer sıcaklık 30 derece **değilse**, ısıtıcıyı çalıştır.

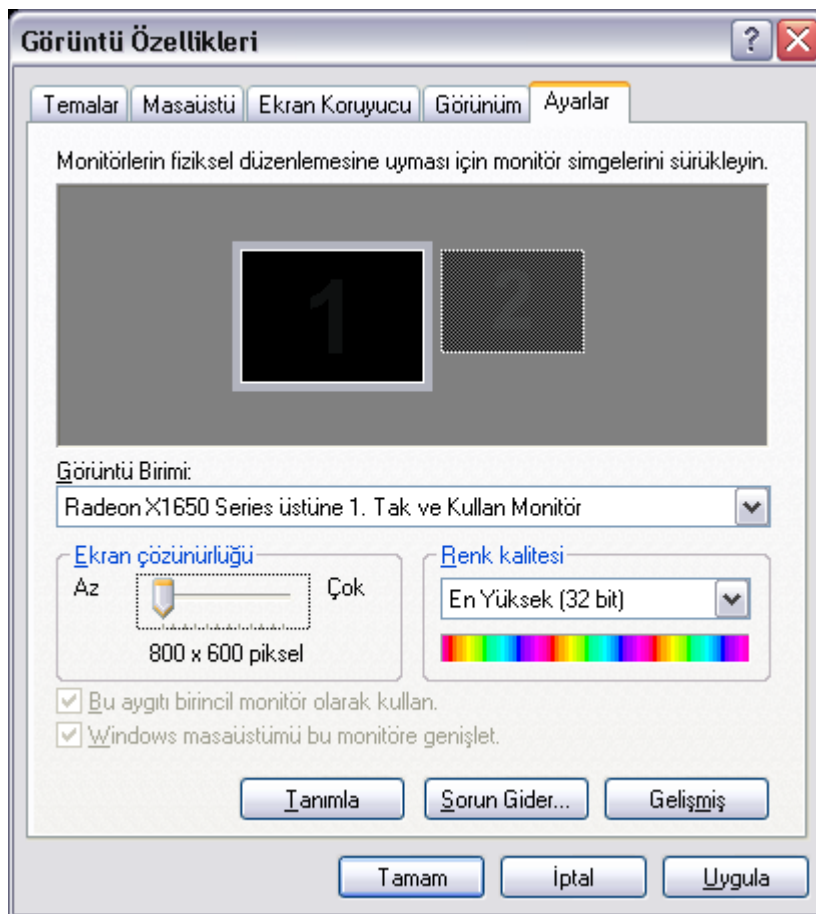
Dikkat ettiysek bütün ifadeler;

[Eğer] [koşul] [ise] [sonuçlandırma]

şeklinde. Yani eğer bir koşulun değeri doğru sonuç ise, [sonuçlandırmada] yazılı işlemin yapılma emri verilmektedir.

2.2.10 Koşul ifadeleri

Peki programlama da koşullu ifadeler nasıl çalışıyor. Aynı yukarıda anlatıldığı gibi... Örneğin; Bilgisayarımızın ekran çözünürlüğünü değiştirdiğimiz yeri hatırlayalım.



Figür 2.2.10 A: Görüntü özellikleri

Yazdığımız bir program içinde bir “koşul” kullanmak istiyoruz. Bu koşulu kullanmak için aşağıdaki adımları tek tek gerçekleştiriyoruz.

1-) Yapmak istediğimiz: Ekran genişliği 800 pixelden küçükse program çalışmasın istiyoruz. Şimdi bunu yavaş yavaş programlamaya dönüştürüyoruz.

2-) Yani diye biliriz ki Ekran Genişliği 800’den küçük olmayacak.

3-) Peki bunu yazısal veya matematiksel olarak nasıl ifade edebiliriz. İstediğimiz ekran genişliği;

EkranGenişlik < 800 OLMAMALI

yada

EkranGenişlik >= 800 OLMALI

Gördüğünüz gibi buraya kadar programlama bilmemize gerek bulunmamaktadır.

(EkranGenişlik >= 800)

Eğer (EkranGenişlik >= 800) ise [programın açılmasına izin ver]

Şuan koşul hazır. [programın açılmasına izin ver] bu bölüm şuan bizi hiç ilgilendirmiyor. Bu bölüme daha sonra başka bir komut yazarak programın açılmasını sağlayacağız. Bir "if" yani koşul (if Türkçe “eğer” anlamına gelmektedir) ifadesinde iki bölüm bulunur. Koşul ve koşulun sağlanması durumundaki sonuçta yapılması gereken işlem.

Koşul	İşlem
Eğer yağmur yağıyorsa	Şemsiyeni al
Eğer otomobilin lastiği patladıysa	Lastiği değiştir
Eğer yemeğin bittiyse	Diğer yemeği ye

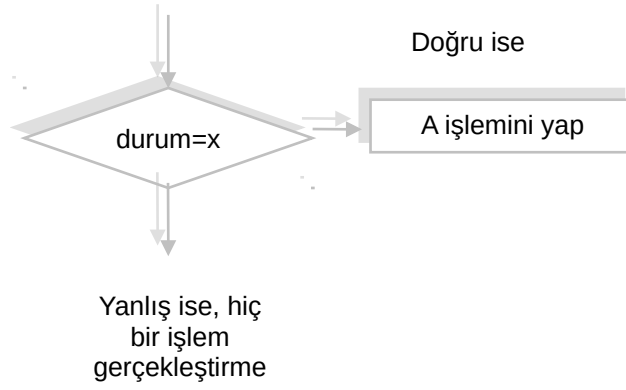
Figür 2.2.10 B: Koşuk - işlem tablosu

İlk bölümde, programların, bir durum için o an karar verebilme yetenekleri olmadıklarını, ve bu nedenle, onları, daha sonra olabilecek her durum için onları; “x” durumunda “a” işlemini, “y” durumunda “b” işlemini yap şeklinde tembihlediğimizi söylemiştik.

Yani, programın karşılaşacağı her durum için bir IF kullanabileceğimizi söylemek doğru olacaktır. Her bir X durumunda, A işlemini yap ifadesini

Eğer durum=x ise, A işlemini yap

şeklinde ifade edilebilmektedir.



Figür 2.2.10 C: Eğer koşulu

Koşul ifadeleri, bilgisayar programlarında sıklıkla kullanılır. 2.5 bölümünde, bu konu ile ilgili ayrıntılı bilgiler verilmiştir. Aşağıda bazı dillerde geçen koşul ifadeleri görünmektedir.

```

If A = 4 Then
    İşlem
End If

if (A=4) İşlem

if (A=4) {
    İşlem
}
  
```

ÇEVİRME	<p>Yine hatırlatma_1'e uyarlama yapacak olursak; gerçek hayattaki “eğer”, “.. öyleyse”, “.. öyle değilse”, “ise”, “o zaman” ifadelerini bilgisayardaki “if”, “then” ve daha sonra göreceğimiz “else” ve “elseif” ifadeli olarak tanımlayabiliriz.</p> <p>Eğer saat 6 ise alarmı çal Kontrol bittiyse bilgisayarı kapat Ahmet geldiyse ve parası varsa gidelim Durum öyleyse, başka bir çözüm bul!</p>
----------------	---

Çeviri 3: Koşul

[Ara verme bölümü]

2.2.11 Fonksiyonlar

Bir sonraki bölüm, matematik fonksiyonları. A ve B iki küme olmak üzere, A'dan B'ye tek yönlü bağıntıya gönderme ya da fonksiyon denir. Genellikle küçük harflerle gösterilirler (f,g,h gibi). A'dan B'ye tanımlanan bir gönderme $f : A \rightarrow B$ şeklinde

gösterilir. En basit anlamda, matematiksel bir gönderme, bir öğeyi sadece ve sadece tek bir öğeye gönderen bağıntıdır.[e8]

Fonksiyonlardan ilk bölümde bahsetmiştim. Bildiğimiz gibi matematikte birçok fonksiyon bulunmaktadır. Örneğin; sin, cos, tan, exp, sign, Bunlarla birlikte, kişi, kendi bir fonksiyon da tanımlayabilir.

Örnek olarak:

$$f(x) = x+2 \text{ ise } f(3) = ?$$

şeklinde bir sorunun çözülmesi için, fonksiyon eşitliğinde, x gördüğümüz yere, x'in değeri olarak belirtilen "3" sayısını yazarsınız ve sonucun $3+2 = 5$ olduğunu bulursunuz.

Buradaki önemli nokta, x adlı değişkene hangi değer girilirse, fonksiyonun tanımındaki (sağ taraf) x adlı değişkene de aynı değer yazılacaktır.

Unutmayalım ki burada, f fonksiyon ismi, x, o fonksiyona girdi olarak verilen bir değişkendir. Eğer bir değişken fonksiyona girdi olarak veriliyorsa (buradaki x), bu değişkenlere "argüman" denir.

Peki, programlamada fonksiyonlar nasıl oluyor? Hemen hemen aynı şekilde... Function Türkçe de fonksiyon demektir.

```
Function f(x)
Fonksiyonun içerisindeki işlemler
End Function
```

Fonksiyonun içerisine gerekli işlemleri yazalım.

```
Function f(x)
f = x + 3
End Function
```

Dikkat edersek, 1. ve 3. satırı çıkardığımızda kalan ifade, matematikte yazılan ifade ile neredeyse aynıdır ve anlaşılacağı gibi, girilen x değerinin 3 fazlasını döndürür. Yukarıdaki kodda, kalın olarak belirtilmiş bölümleri birleştirirsek, zaten matematiksel bir fonksiyon elde ederiz ($f(x)=x+3$). Burada ilk satırda belirtilen x, bir değişkendir ancak, fonksiyonları ve prosedürleri çağırırken kullanılan değişkenlere o fonksiyon veya prosedürün argümanları denmektedir.

Başka bir örnek verelim;

```
Function g(x, y)
g = x * y
End Function
```

Bu fonksiyon ise, girilen x ve y argümanlarını çarpıp (* çarpma işlemi sembolü) sonucu döndürmektedir.

Başka bir örnek verelim. Bir mutlak değer bulma fonksiyonu yazmamız gerektiğini düşünelim. Mutlak değer, matematikten de hatırladığımız gibi, girilen bir sayı eğer pozitif ise, girilen sayıyı, eğer girilen sayı negatif ise, girilen sayının pozitif halini döndüren fonksiyondur.

```
function Mutlak(x)
end function
```

Dikkat ettiyseniz yapılması gereken işi ifade ederken, eğer kullandık, hemde iki kere.
Eğer sayı pozitifse, pozitif sonuç döndür.
Eğer sayı negatifse, o sayıyı pozitif yap ve sonucu döndür.

Matematikte negatif bir sayıyı pozitif yapmak için “-1” ile çarpılır. Daha önceden öğrendiğimiz koşullu ifadeleri burada kullanalım.

```
function Mutlak(x)

if x > 0 then
mutlak = x
end if

if x<0 then
mutlak = x * -1
end if

end function
```

Gördüğümüz gibi ilk ifadede, sayıyı (x) aynı şekilde döndürdük (geri verdik). Ancak eğer $x < 0$ ise, yani x negatifse, o zaman sayıyı -1 ile çarpıp pozitif yaptık ve o şekilde döndürdük. Ancak burada unuttuğumuz bir durum söz konusudur. Eğer, x’in değeri 0 olur ise, ne negatif, ne de pozitif olacaktır. İşte bu yüzden bir başka koşul ifadesi daha eklemek zorundayız.

```
function Mutlak(x)
if x > 0 then
mutlak = x
end if
if x<0 then

mutlak = x * -1
end if
if x = 0 then
mutlak = 0
end if
end function
```

Son ifade, x’in sıfıra eşit olma durumunda, sıfır döndürüleceğini belirtir. Fonksiyonlar, sadece sayılarla değil, metinlerle, daha sonra göreceğimiz sınıflarla da işlemler yaparlar. Sayılarla örnekleri sadece, matematiğe yakın olduğunu ifade etmek için gösterdik.

Fonksiyonlar, programlarda yüzlerce farklı durumda kullanılabilirler. Fonksiyonlar, bir işlemi bizim için yapıp sonucunu döndüren prosedürlerdir. Matematik, fizik, kimya,... veya herhangi bir bilimde, kullanılan bütün formüller, fonksiyondur diyebiliriz. Çünkü, formüllerde, bir veya birkaç değeri girdi olarak alıp, bir sonucu bize döndürürler (verirler).

Son bir örnekle bu bölümü tamamlayalım. Faiz problemlerinden hatırlarsanız, faiz hesaplamanın, $f = A \cdot n \cdot t / 100$ şeklinde bir formülü vardır. Burada, A anapara, n faiz oranı, t (yıl) faiz zamanıdır. Sonuçta bize, f yani faiz miktarı dönecektir. Bu ifadeyi fonksiyon haline getirelim.

```
Function faiz ( anapara, oran, zaman)
faiz = anapara * oran * zaman / 100
end function
```

Gördüğünüz gibi, matematiksel formüller bu şekilde programsal fonksiyonlara çevrilir.

ÇEVİRME	<p>Hatırlatma_1'e uyarılama yapacak olursak; gerçek hayattaki bir veya birden fazla değeri kullanarak, başka bir değer veya sonuç elde etmek için yaptığımız işlemlere, yani aslında gerçek hayattaki fonksiyonlar, programlamada da fonksiyon (function) olarak ifade edilir.</p> <p>Bir bankaya gittiniz, kredi almayı düşünüyorsunuz, maaşınızı ve gelirlerinizi hesaplayıp, size verilebilecek kredi miktarı belirleniyor. İşte burada bir fonksiyon (basit bir matematiksel işlem) kullanılır. Programlamadaki fonksiyonlar, sadece sayısal işlemler yapmak zorunda değildir.</p> <p>Fonksiyonlar, bir veya birden fazla değeri girdi olarak alıp, onlarla çeşitli işlemler yapıp, sonra bir sonuç çıkaran işlem topluluklarıdır. Örneğin, bir çorbaya; mercimek, tuz, salça, su eklersiniz, sonuç olarakta mercimek çorbası çıkar.</p>
----------------	--

Çeviri 4: Fonksiyon

2.2.12 Diğer Döngü Tipleri

Programlamada, işlerimizi kolaylaştıran en önemli etkenlerden biri, bir kere tanımladığımız bir işlemi binlerce, milyonlarca,.. sonsuz kere hiç bir ekstra emek sarf etmeden kullanabilmemizdir.

Günlük hayatımızda, iş yerinde – evde – okulda, hergün benzer işler yaparız. Örneğin iş yerinde, müdürümüzün bize, kontrol etmemiz için verdiği onlarca dosyanın olduğunu ve bu dosyaların hepsini incelememiz gerektiğini düşünelim ve hatta dosyaları kontrol etmeden eve gidemeyeceğimizi varsayalım.

Yaptığımız işlem;

- dosyayı al / aç
- dosyayı kontrol et
- dosyayı kapat / yerine yerleştir

şeklinde alt işlemlerden oluşacaktır. Dikkat ederseniz, bu işlem bir dosya için değil bütün dosyalar için, bütün dosyalar bitinceye kadar uygulanacaktır.

Yani aslında yapılan işlem;

- dosyayı al / aç
- dosyayı kontrol et
- dosyayı kapat / yerine yerleştir
- yine başa dön

şeklinde olacaktır. Ancak, dikkat ettiysek, her bir işlem yaptığımızda, içinde bulunduğumuz ortamda (ofiste) değişiklikler olacaktır. Örneğin, kontrol edilecek dosya sayısı azalacak, kontrol edilen dosya sayısı artacaktır.

Bununla birlikte, bu işlemleri sonsuza kadar gerçekleştirmeyeceğiz. Kontrol edilecek dosya kalmadığında, işimiz bitecek ve eve gidebileceğiz. İşte bu yüzden, “yine başa dön” ifadesine, (daha önceki bölümlerde belirttiğimiz) bir koşul eklemeliyiz ve ancak bu koşul doğru ise, başa dönmeli ve uygulayacak olduğumuz işlemi, bir sonraki dosya için yapmalıyız.

Böylece işlem;

- dosyayı al / aç
- dosyayı kontrol et
- dosyayı kapat / yerine yerleştir
- yine başa dön (eğer dosya kalmışsa)

“Eğer dosya kalmışsa” koşulu doğru olduğu sürece, işlem devam edecektir. Zaten kontrol edilecek bir dosya kalmadıysa, bir iş neden yapalım ki? Sizin için 4. ifadenin çok önemi yoktur. Çünkü, kontrol edilecek dosya kalmayınca işi bırakır ve evimize gideriz. Ancak daha önceden de belirttiğim gibi, bilgisayarlar sizin gibi karar verme yetisine sahip değildirler. Bu yüzden bilgisayarlara her durumda ne gibi işlemler yapması gerektiğini belirtmeliyiz.

Yukarıdaki 4 işlem, bilgisayar dünyasında, benzer bir şekilde ifade edilir.

Dosyayı al

Dosyayı kontrol et

Dosyayı kapat

Eğer dosya sayısı 0’dan büyükse(yani başka dosya kalmışsa) başa git

Günlük hayatta kullandığımız bazı ifadeleri ve o ifadeler içinde geçen bazı özel kelimelerin örneklerine bakalım:

- Ateşin düşünceye **dek** su iç / Ben söyleyinceye **dek** burdan ayrılma
- Yangın sönmedikçe soğutmayı sürdürün / Depo dolmadıkça benzin doldurun
- **Sürekli** ip atla. / **Hep** kontrol et

Yukarıdaki ifadelerde, peş peşe yapılan işlemlerin olduğunu söyleyebiliriz. İlk örnekte “su iç” eylemi, bir koşul olan “ateşin sönmesi” durumu gerçekleşinceye kadar devamlı olarak uygulanacaktır.

İkinci örnekte, bir koşul sağlanmadıkça, devam etmesi gereken bir eylem söz konusudur. Son örnekte ise, bir koşul belirtmeksizin aynı işlemin yapılması istenmiştir.

Yukarıdaki ifadeleri 3 kategori altında toplayabiliriz

- Bir koşul sağlanıncaya kadar (... oluncaya dek...)
- Bir koşul sağlandıkça (...oldukça... /olduğu sürece)
- Sürekli olarak (hep / her zaman)

Bu işlemler programlama dünyasında aynı şekilde ifade edilirler. Ayrıntılı bilgiler, 5. bölümde verilecektir.

ÇEVİRME	<p>Hatırlatma_1'e uyarlama yapacak olursak; gerçek hayatta belirli bir koşul sağlanıncaya kadar (.. oluncaya dek, oluncaya kadar), sürekli (sürekli, her zaman, hep) veya bir koşul istenilen değerde olduğu sürece(... oldukça, olduğu sürece) yapılmasının sürdürülmesini istediğimiz durumlar olabilir.</p> <p>Sürekli kontrol et (bir makine veya program açık olduğu sürece kontrol eder, veya bir güvenlik görevlisi görevdeyken sürekli kontrol eder..)</p> <p>Elindeki ürünler bitinceye kadar eve dönmek yok.</p> <p>Param oldukça sana yardım ederim.</p>
---------	--

Çeviri 5: Diğer döngü tipleri

[Ara verme bölümü]

2.2.13 Sabit değerler

Değişkenlerden sonra birde değişmeyen değerleri tutan const (sabit)'ları göreceğiz. Const sabit değer anlamına gelmektedir. Matematikten hatırlayacağımız sabit sayı Pİ değerini düşünelim. Değeri hep sabit, hep aynıdır ve değiştirilemez.

$$Pİ = 3.14$$

Aynı şekilde programlamada da sabit değerleri belirtmek için const ifadesi bulunmaktadır.

Const Pİ = 3.14 şeklinde kullanılır yani ilk olarak const ifadesi daha sonra değişkenin ismi (aslında değişmeyen ismi) akabinde, değer yazılır.

ÇEVİRME	<p>Hatırlatma_1'e uyarlama yapacak olursak; gerçek hayatta belirli süreç içinde değeri hep sabit olan değişkenler için, yani sabit değerler için, programlamada, “const”, “final”, “define” gibi terimlerden birini kullanarak ifade ederiz.</p> <p>Fizik veya kimyadan hatırladığımız fonksiyon sabitleri vardır (g = yer çekimi ivmesi, Avagadro sayısı = $6,02 \times 10^{23}$....). Anayasada, değişmeyecek ilk beş madde vardır.</p> <p>İşte bu gibi durumlardaki değerleri ifade etmek için, programlamadaki sabit değerleri kullanırız.</p> <p>Pi = 3.14 g = 9.8 m/s² eV = 1.602176x10⁻¹⁹ J MADDE 1. – Türkiye Devleti bir Cumhuriyettir. (değişmez) Senin bu ayki çalışma saatlerin 08:00 – 17:00 (sabit)</p>
----------------	--

Çeviri 6: Sabit değerler

Alıştırmalar

- Değişken nedir? Ne işe yarar?
- Değişkenlerin ne gibi özellikleri vardır?
- Değişkenlere ne gibi işlemler uygulanabilir?
- Değişkenler nerede tutulur?
- Değişkenler nasıl tanımlanır?
- Veri türleri nelerdir?
- Temel veri türleri nelerdir?
- Array nedir?
- Değişkenlere değer atama nasıl yapılır?
- Atama operatörü nedir? Nasıl kullanılır?
- Arrayler nasıl belirtilir?
- Arraylere veri ataması nasıl yapılır?
- Programlamada kaç çeşit parantez vardır?
- Döngü nedir? Nasıl çalışır?
- Koşul ifadeleri nedir? Ne işe yararlar?
- Fonksiyonlar nasıl tanımlanır?
- Fonksiyonlar ne için kullanılır?
- Kaç tane döngü tipi vardır?

2.3 Programın yüzleri

Kaynak kod, yada open source olarak geçen tabiri hepimiz son yıllarda (özellikle 2004 sonrasında) sıklıkla duymuşuzdur. Peki nedir bu kaynak kod?

Programların iki yüzü vardır: bir ön, birde arka yüzü. Şuanda kullanmakta olduğunuz, word, excel, paint, winamp gibi programlarda gördüğünüz programların ön yüzüdür. Yani siz programın çalışmış, işlev gören kısmını görmekte ve kullanmaktasınız. Halbuki, birde o programın tasarlanması aşaması var, bu kısma da o programın arka yüzü denir.

Bu durumu şöyle bir örnekle açıklayalım: Varsayalım ki bir makineyi tasarlıyorsunuz (örnek, kolileme makinesi). Siz kartonları, yapıştırıcıyı ve koli içine konulacak ürünleri makineye veriyorsunuz ve makine size (içinde – siz görmeden) koliyi oluşturuyor içine ürünleri yerleştiriyor ve çıktı olarak veriyor.

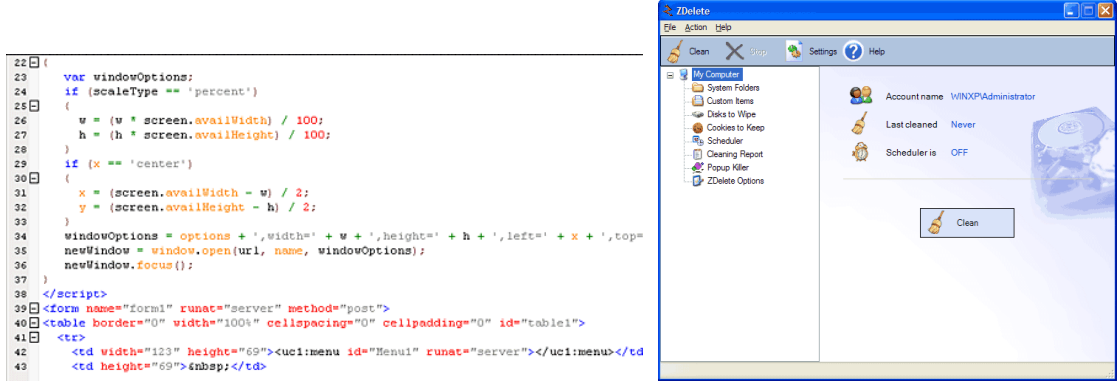
Sizin burada gördüğünüz makinenin ön yüzüdür. Ancak önemli olan makinenin arka yüzüdür. Arka yüzü ise, makinenin içeride hangi işlemleri nasıl gerçekleştirdiği nasıl yaptığıdır.

Aynı şekilde bir cep telefonu düşünün. Artık inanılmaz derecede küçük boyutlara kadar küçültülebilen telefonlar ile, sesli iletişim – yazılı iletişim, mesaj kayıt, okuma, sesli mesaj, sesli numara bulma, fotoğraf ve görüntü çekme işlemleri yapılabilmektedir. Bu kadar çok işlem yapan cep telefonlarının içindeki chip'ler bu işlemlerin yapılmasını mümkün kılar. Bizim yine görüyor ve kullanıyor olduğumuz, cep telefonunun ön yüzüdür. Arka yüzü ise, telefon içerisindeki chip'lerin neler yaptığını içeren bölümdür.

Buradaki önemkli kısım; chip'e bakarak, içinde ne yazdığını, bir makineye baktığımızda içinde ne gibi işlemler gerçekleştiğini anlamamız zor olacaktır. Ancak arka yüzü olan bir sistemden o sistemin ön yüzü çok kolay bir şekilde oluşturulabilecektir. Yani arka yüz, bir sistemin planlanma – geliştirilme evresidir, ön yüzde ise sadece, planı ve programı belli olan sistemden birden fazla üretilir ve müşterilere sunulur.

Aynı durum programlar içinde geçerlidir. Word, excel, winamp, winzip gibi programları kullanırken, içerisinde ne gibi kodlar olduğunu bilmiyoruz. Halbuki, bu programların arka yüzü, kodlar tarafından meydana gelmektedir.

Bir program kodlanarak yazılır ve sonunda, program kodlarına direkt olarak ulaşamayacağınız program arabirimi ortaya çıkar.



Figür 2.3 A: Kod ve program

Bir ürünün altında yatan emek, arka yüzde gizlidir ve ön yüze bakılarak arka yüzdeki emeği ve işlemleri anlamak için çok profesyonel olmak gerekir. İşte programların, arka yüzüne, “KAYNAK KOD” denilir.

Eğer elimizde bir programın kaynak kodu var ise, program içinde istediğimiz değişiklikleri – eklemeleri yapıp kendi programımızı çıkarabiliriz veya kaynak koda bakarak başka programlar geliştirebiliriz. Kaynak kod, yani açık kod, açık olduğu için ücretsizdir. Bu da, kaynak kodun diğerlerine göre pozitif bir yanıdır.

Bizim yazacağımız kodlarda normal olarak, o programın kaynak kodları olacaktır. Eğer yazdığımız programların kodlarını internette yayınlarsak o kodlar açık kaynak kod olacaktır. Tahminimce çoğu kişi, yazdığı – emek verdiği kodların HEPSİNİN internette yayınlanmasına karşıdır benim gibi. Ancak yazdığımız kodların bir kısmını veya bazı bölümlerini yayınlayıp insanlara yardımcı olmak, ayrıca, kullandığımız sistemleri kullanmak bizim ve geliştiriciler için faydalı olabilir.

[Ara verme bölümü]

Alıştırmalar

- Kaynak kod nedir?
- Programın arka yüzü ve ön yüzü hakkında bilgi veriniz

2.4 Program / Durum

Bilgisayarların karar verme yeteneği olmadığını söylemiştik. Bu nedenle, dış dünyada oluşabilecek gerekli her durum tanımlanmalı ve bu tanımlanan durumlar için programın hangi işlemleri gerçekleştireceği belirtilmelidir.

Peki durum nedir?

Durum Durum, oluşturulan sanal ortamdaki, ortam değişkenlerinin bazılarının belirtilen değerlerde olmasına verilen addır. Yani, programlama yapmak için oluşturduğumuz boş, sanal ortamdaki özelliklerin, değerlerinin belli limitlerde, belli seviyelerde olmasına **durum** denir.

Varsayalım ki bir program yazıyoruz. Yazdığımız program; bir sensörden, bir aracın harita üstünde nerede olduğunu gösteriyor ve aracın durumu ile ilgili 3 farklı sonuç verebiliyor olsun: Orta hızda gidiyor, hızlı gidiyor, duruyor. Araç, firmamıza ait ve firmamızda çalışan bir eleman tarafından kullanılmakta olsun. Yazdığımız program bu aracın gerekenden fazla hızlı gitmesi durumunda, bu sürücüyü uyarıyor.

Bu işlemin nasıl yapıldığı, sensörlerin nasıl çalıştığını hiç önemsemiyoruz. Burada önemli olan, hangi durumlarda hangi işlemin yapılacağıdır.

Durum	İşlem
Orta hızda gidiyor	-
Hızlı gidiyor	Aracın şoförünü arayıp, hızlı gittiğini haber ver
Duruyor	Eğer 10 dk'dan fazla duruyor ise, telefonla arayıp molayı bitirmesini söyle

İşte programlarda genelde, state-based yani durum tabanlı çalışırlar. Bir durum değiştiğinde, o duruma ilişkin belirtilmiş olan işlevi gerçekleştirirler. Dikkat ederseniz, Orta hızda gidiyor durumuna herhangi bir işlem yazmadık. Çünkü bu durumda yapılması gereken bir işlem bulunmamaktadır. Bu nedenle, programa bu durumu belirtmemize gerek bulunmamaktadır.

Durum	İşlem
Hızlı gidiyor	Aracın şoförünü arayıp, hızlı gittiğini haber ver
Duruyor	Eğer 10 dk'dan fazla duruyor ise, telefonla arayıp molayı bitirmesini söyle

Ancak bazen de, belirtmediğimiz durumlar gerçekleşebilir. Örneğin, varsayalımki, sensörün algılayabileceği bir durum daha söz konusu olsun: Yavaş gidiyor. Ancak eğer biz bu durumda, ne olacağını belirtmez isek, hiç bir işlem gerçekleşmeyecektir.

Program – durum ilişkisinde, önemli olan, programın karşılaşacağı bütün durumları bilmek – tespit etmek, daha sonra bu durumlarda yapılacak olan işlemleri belirlemektir.

Örneğin, Garri Kasparov ismini çoğumuz duymuşuzdur. Dünyanın satranç konusunda yenilmez ismi olan Kasparov'u yenmek için IBM tarafından bir bilgisayar ve program geliştirildi. "Deep Blue, IBM tarafından geliştirilen, satranç oynayabilen bir bilgisayardır. 1997'de dünya şampiyonu Garry Kasparov'u yenmiştir."

Bu programın nasıl yazılabileceğini düşünelim. Binlerce farklı durum için, yani satranç taşlarının, binlerce farklı varyasyonda sıralanışı veya bir kurala göre dizilişini düşünelim, daha sonra bu durumlarda yapılacak olan işlemleri düşünelim. İşte bu işlemler, programcılar tarafından programa girilerek, mükemmel bir program haline getirilmiştir ve ancak bu şekilde insan beyni yenilebilinmiştir. Aslında buradan herkeste, "bilgisayar insan beynini yendi" düşüncesi oluşabilir ancak, unutmayalım ki, o programı tasarlayanlarda insandır.

[Ara verme bölümü]

Alıştırmalar

- Program - durum hakkında bilgi veriniz.

İkinci bölümde kullanılmış olan terimlerin ingilizceleri

akış : flow	sabitdisk : harddisk
kütüphane : library	blok : block
dinamik : dynamic	dizi : array
değişken : variable	ikilik : binary
adres : address	işaret : sign
tür : type	magnitude : büyüklük (değeri - skalar)
boyut : size	tanımlama : declare
tanımlayıcı : identifier	operatör : operator
değer : value	ifade : statement & expression
varsayılan : default	başlangıç değerlerinin atanması : initialize
veri : data	döngü : loop
hafıza : memoy	kaynak kod : source code

SONUÇ

Bu bölümde, geçen bölümde öğrenmiş olduğumuz nasıl programlama yapacağımızın, dillerle nasıl ifade edildiğini öğrendik. Programlama dilleri, bir operatördür. Yazdığınız programı bir işin mühendisi, yazdığınız kodları anlayan ve bu işlevleri gerçekleştiren programlama dilini de, tanımlanan işi yapan işçiler olarak görebilirsiniz. İşçi olmadan mühendisin bir anlamı olmayacaktır, aynı şekilde, mühendis olmadan işçi iş yapamayacaktır. Bu ilişki, beyin ile el arasındaki ilişki gibidir. Beyninizle düşündüğünüzü, elinizle tasarlayıp gerçekleştiremezseniz bir anlamı olmayacaktır. Aynı şekilde, düşünemediğinizde de herhangi bir proje geliştirme olanağınız olmayacaktır.

Programlama dilleri içinde söylelebilecek en iyi sözün “one system built on another” yani bir sistem başka bir sistemin üstüne kurulmuştur, o sistemde başka bir sistemin üstüne, o sistemde başka bir sistemin üstüne (Matrix)..... kurulmuştur. Yani pianoyu baştan keşfetmek yerine sadece gerekli olan işlemleri programlama diline tanımlamalıyız.

Zannederseniz birçoğunuz, bugüne kadar okulda öğrendiğiniz ve bir kısmının işe yaramadığını düşündüğünüz matematiksel bilgilerin öneminin şimdi farkına varmışsınızdır. Bende matematiğin bu kadar güçlü olduğunu daha sonradan öğrendim. Matematiğin özelliği doğada bulunmayan tek bilim olmasıdır.

Gerçek hayata baktığınızda, bir konuşma dilinde, yüz binlerce sözcük, binlerce cümle kalıbı bulunmaktadır. Programlama dillerinde ise, en fazla 1-2 bin anahtar sözcük, kullanılan diğer özel kalıpları da katarsak 5 bin anahtar sözcük ve 200-400 arası ifade kalıbı bulunmaktadır. Türkçe gibi, inanılmaz derecede çok ekle donatılmış ve zor bili dili rahatlıkla konuşabiliyorsak, bir programlama dilini konuşmamız o kadar da zor olmasa gerek.

[Ara verme bölümü]

3 Nesne Programlama

İçerik

- 3 Nesne Programlama
- 3.1 Beyin ve Sınıf - Varlık
- 3.2 Beyinde Programlama
- 3.3 Inheritance - Sub Class - Super class
- 3.4 Nesne - Sınıf

Amaçlar

Üçüncü bölüm,

- beynin çalışma yapısını, bu çalışma yapısına göre programlamanın nasıl mümkün olabileceğini,
- sınıf ve nesne kavramını,
- inheritance - sub - super class ifadelerini anlatmaktadır.

Anahtar sözcükler

beyin, class, object, sınıf, nesne, inheritance, super class, sub class

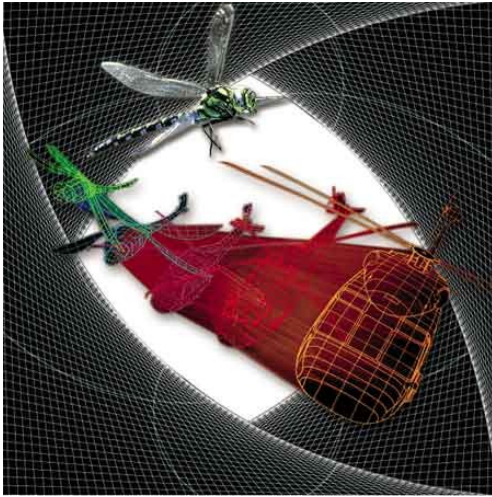
3.1 Beyin ve Sınıf / Varlık

Dış dünyada olan bir sistemi, bilgisayar içine nasıl taşıyoruz? Bu sorunun cevabını ilk bölümde vermiştik. Bu bölüm ilk başta konumuzla alakasız gibi görünebilir, ancak sonra hepimiz, ulaşmaya çalıştığımız amacı farkedecek ve işlerin nasıl daha kolaylaşacağını göreceksiniz.

KURAL A : İnsanoğlu bilim üzerine elde ettiği hemen hemen bütün başarıların sırrı olarak, insanı ve doğayı gösterir.

Örneğin gözün çalışma yapısı kopyalanır, kamera elde edilir yapay zeka ile resim tanıma ve işleme üstünde kullanılır, insan eli – insan kolu şekli kopyalanır, robot kolu yapılır, sayılangozun kendi kabuğunu onaran sıvısının formülü kopyalanır güzellik ve bakım kremi yapılır, örümceğin ağının yapısı kopyalanır dünyanın en sağlam maddesi yapılır, güneşin içindeki sonsuz enerji kaynağı: füzyon kopyalanarak dünya da yapılmaya çalışılır

Tasarım, teknolojik çalışmaların en önemli unsurlarındandır ve bu aşamada pek çok ayrıntının bir arada düşünülmesi gerekir. Hemen hemen bütün teknolojik ürünlerin tasarlanmasında ve geliştirilmesinde doğadaki varlıklar örnek alınır. Örneğin helikopter üretiminde karşılaşılan manevra sorunu, bilim adamları tarafından oldukça dikkat çekici bir yöntemle aşılmıştır: Yusufçuğun(helikopter böceği) gövde ve kanat tasarımı temel alınarak
[e16]



Figür 3.1 A: Helikopter böceği

KURAL B : Daha açık ifade etmek gerekirse, önümüzde, Yaratıcının bizlere sunduğu binlerce örnek bulunmaktadır. Bizim ise yapmamız gereken o örnekleri KOPYAlamak olacaktır. Bilimde elde edilen başarıların %95'i bu yolla mümkün olmuştur ve bundan sonraki süreçte de sorunlar, yine doğadan ve insandan yararlanarak çözülecektir. Bilim, iki konu üzerinde çalışır. Birincisi, doğa – insan – canlılardır, ikincisi, bu sistemlere benzetilerek geliştirilecek olan buluşlardır. İki konuda da, temel yine önümüzdeki örneklerdir.

Bilim

Bilimdeki nesne

Doğa veya insandaki sistem

Makine Müh.	Piston	İnsan kol ve bacakları
Makine Müh.	Pompa	Kalp
Bilgisayar - Elektronik Müh.	Kamera	Göz
Bilgisayar – Elektronik Müh.	Mikrofon	Kulak
Makine Müh.	Uçak	Kuşlar
Yapay Zeka Müh.	Yapay sinir ağları	Beyin – Nöron yapısı
Kimya Müh.	Boya	Çiçek ve hayvan renkleri
Biyoloji	Koyun Klonlama	Doğadaki üreme
.....		

Sonuçta, **bilimin kaynağı**, zaten varolan **doğadır**.

Felsefenin en önemli konularından biri olan, “Bilgi doğuştan mı öğrenilmiştir?”

- Yani her tür bilgiyi biliyoruzdur, daha sonra onları bildiğimizi farkederiz.
- Sonradan mı öğrenilir? Beynimizde hiç olmayan bilgileri yaşamımız ilerledikçe öğreniriz,

konusunun cevabı belki hiç bir zaman bulunamayacaktır. Ancak, “buluşlar doğuştan mı?, yoksa sonradan mı” sorusunun cevabı tabiki “**doğuşandır**” olacaktır. Yani, insanlar doğada zaten olan olayları farkeder - tespit eder.

KURAL C : Bir sistemi bilgisayara anlatmak, bilgisayarın o konuyu öğrenmesi demektir. Aynı şekilde, bir konuyu öğrencilere anlatmaktaki amaç, o konunun öğrenciler tarafından öğrenilmesi anlamına gelmektedir.

KURAL D : Evren üzerinde türler(sınıflar) ve varlıklar bulunur.

Tür	(Felsefe) Kendi içinde bir birim olan ve üzerinde cins kavramının bulunduğu mantıksal kavram: "Parça bütünü, cins türün yerine geçti mi daralma olur. Hayvan canlı varlık karşısında türdür, aslan karşısında cinstir." [e23]
Sınıf	(Mantık) Belli ortak belirtileri olan tek tek nesneler öbeği. (Biyoloji) Takımlardan oluşan birlik, dalların alt bölümü: "Memeliler, kuşlar, balıklar, omurgalılar dalının birer sınıfıdır." [e24]
Varlık	Varlık, felsefenin temel kavramlarından birisidir. Varolan, ya da varolduğu söylenen şey, varlık kavramının içeriğini oluşturur. [e24]

**** Sınıf ve tür terimleri aynı anlamda kullanılmıştır.**

İnsanlar kendi aralarında iletişim kurabilmek için, beyinlerindeki bilgileri başka kişilere bir yöntem ile aktarmalıdır. Böylelikle, anlatmak istedikleri türleri ve sınıfları karşılarındakilere ifade etmiş olurlar. İnsanlar iletişim işlemini ifade etmek için genelde sesli iletişim kullanırlar. Ancak bunun haricinde, sesli iletişimden çok daha güçlü olan görsel iletişimde bulunur.

Yukarıdaki tanımlara dikkat ederseniz, **tür** kelimesinin birbirinden farklı kategorilerde olan varlıkları nitelendirmek için kullanılan bir ifade olduğunu anlayabiliriz. İnsan bir türdür, hayvanlar, bitkiler, bakteriler, cansız varlıklar, eşyalar, kavramlar, düşünceler ... hepsi birer türdür. Doğal dillerde (konuşma dillerinde) “isim” şeklinde ifade ettiğimiz her terim, bir “tür” tür.

Örnek:

“Sinemanın koltukları eskimiş”

Sinema : isim – Yani bir tür

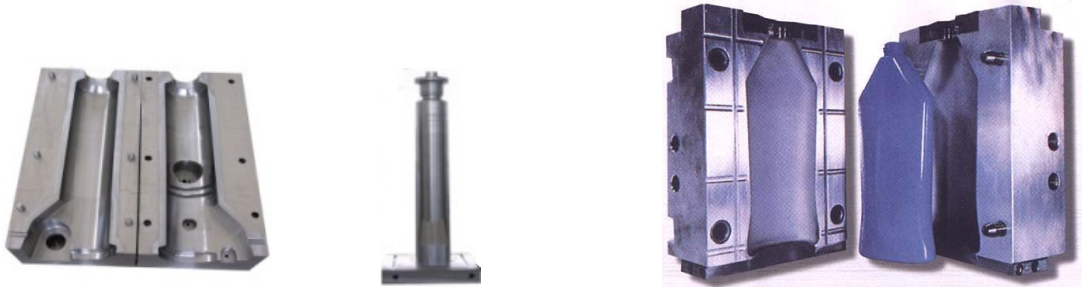
Koltuk : isim – Yani bir tür

Varlıklar ise, o nesne gruplarına (yani türlere) ait olan gerçeklerdir.

Örneğin, köpek, bir türdür. Karabaş adlı bir köpek ise, “köpek” sınıfına ait bir varlıktır. Başka bir örnek olarak, insan bir türdür. Albert Einstein ise, “insan” sınıfına ait bir varlıktır. (Programlama şimdi de matematikten çıkıp Türkçe dil bilimi ile ilgili hale geldi)

Türler, gerçek bir varlık ifade etmezler. Örneğin, insan denildiğinde, aklımıza bir insan (arkadaşımız Ahmet, öğretmenimiz Fuat...) gelmeyecektir, **insan sınıfı** gelecektir. Halbuki, Albert Einstein denildiğinde, direkt olarak, o **VARLIK** yani Albert Einstein ifade edilmiş olunacaktır.

Daha açık bir ifade ile, bir **türü**, **kalıp** olarak görebiliriz. Varlıkları da o kalıp kullanılarak oluşturulan **gerçekler** olarak görebiliriz. Örneğin, aşağıdaki, resimde, solda bir kalıp, ortada ise o kalıp kullanılarak üretilmiş ürün gösterilmektedir. Aynı şekilde, yine sağdaki resimde bir çamaşır deterjanı kabı ve o kabı üretmek için tasarlanmış kalıbı görüyorsunuz.



Figür 3.1 B: Bir kalıp ve o kalıptan üretilen ürün örneği

Bir kalıp tasarladıktan sonra, o kalıbı kullanarak, milyonlarca (aşınma ve yıpranma olmadığını düşünürsek) aynı ürün üretebiliriz.



Figür 3.1 C: Bir kalıp ve o kalıptan üretilen ürünler örneği

İşte bu örnekte de görüldüğü gibi, kalıpları: sınıflar yani türler, kalıplar kullanılarak üretilenleri, varlıklar olarak ifade edebiliriz. Sınıf (veya tür) aynı tip varlıklar için bir kalıptır. Bir kere tasarlanır, ancak o sınıf türünde milyonlarca varlık üretilebilir. O sınıf tasarlanmadan o sınıfa ait bir ürün yani varlık üretilemez! Ancak tek başına bir sınıfta bir anlamı yoktur.

Fabrikanızda bir kalıp tasalardığınızı düşünün. Eğer bu kalıbı kullanarak ürün üretmez, kalıbın bir anlamı olmayacaktır ve boş yere oluşturulmuş olacaktır.

Tür (veya sınıf) , NE/KİM.. sorusunun, Varlık ise HANGİ/KİMİN... sorusunun cevabıdır. Örneğin;

- Oradaki **ne**?
- Oradaki bir **araba**.
- **Hangi** araba?
- **34 DR 392** plakalı araba.

Bir varlık, “bir” tanedir ve evrende yaratılmış olanlardan biridir, **reeldir**. Sınıflar ise insanlar tarafından oluşturulur. Gerçekte, dünya, gezegen, otomobil diye nesneler yoktur, yani bu tanımlar gerçek varlıklara karşılık **gelmemektedir**. Sınıflar, varlıkları tanımlamak için birer ortak addır diyebiliriz.

Türler tanımlama için kullanılır. Varlıkların isimlerinin veya tanımlayıcı özelliklerinin (örneğin yukarıdaki plaka gibi, insanların ismi veya TC kimlik numaraları gibi) kullanılma amacı ise, binlerce aynı sınıfa ait varlık arasında, belirttiğiniz olanını seçmektir. Yani aynı ingilizcedeki gibi: “a car” ifadesi, herhangi “bir araba”yı ifade ederken, “the car” ifadesi, gösterilen veya bahsi geçen “bir araba”yı belirtmektedir.

Yukarıdaki örnekteki gibi, istediğimiz bir nesneyi ifade etmek için onun sadece türünü değil, o varlığı diğerlerinden ayıran özelliğini ve bu özelliğin değerini de belirtmeliyiz. Örneğin; TC Kimlik Nosu 432XXXXXX olan kişi şeklinde yaptığımız tanımlama ile sadece bir kişiyi belirtmiş oluruz.

Beyin

Beynimizde 100 milyar'dan fazla nöron bulunur. Bu nöronlar farklı türlerde yine milyonlarca bilgiyi hafızamızda tutarlar. Yani hafızamızı bu verileri tutan bir VERİ HAVUZU olarak görebiliriz. Peki, bir işlem anında, milyonlarca verinin olduğu bir havuzdan, bu verileri nasıl elde ediyoruz?

Bin sayfadan oluşan bir kitap düşünelim. Bu kitap içinde bir bilgi bulmak istediğimizde bin sayfayı baştan aşağıya okumayız! Kitabın arka bölümünde bulunan terimlerin hangi sayfalarda geçtiğini bulur ve o sayfaları okuruz ve istediğimiz sonuca ulaşırız.

Beyin içinde de işler aynı şekilde işliyor. Beyin de, nöronlara veya bilgilere bir tür anahtar sözcük “isim” atıyor ve daha sonra sadece bu ismin geçtiği yerleri işliyor. **İsim** yapısını basit düşünmemek gerekir. Zira, “eylem” de bir harekete, işe verilen isimdir, “sıfat” ta bir özelliğe verilen isimdir

Sonuç olarak diyebiliriz ki, beynimizde türler ve varlıklar bulunmaktadır. Bu türler dünyada isim diye ifade ettiğimiz her kavram olabilir.

KURAL E : Öğrenmenin, bilimdeki tanımı: “Things learn when they change their behaviour in a way that makes them perform better in future[e25]” yani “Sistemler

(insan – yapay zeka), bir sonraki seferde daha iyi işlem gerçekleştirmek için değişiklik gerçekleştirdiğinde öğrenirler” şeklindedir. Yani, bir sistem öğrendiğinde, hatası azalır veya yok olur, böylece bir sonraki (karşılaştığı aynı) durumda, daha iyi sonuçlar elde edilebilecek seçimler yapılır.

Ben bu tanımı biraz daha farklı yorumluyorum. “Learning is defining unknown information in terms which are known. That is, unknown terms could be only defined with the known information” yani “öğrenme, bilinmeyen bir bilgiyi, bilinen bilgiler halinde ifade etmektir, yani bilinmeyen terimler sadece bilinen terimlerle ifade edilebilmektedir”. Yani, bilmediğiniz bir konuyu, tanımladığınızda, zaten o konu ile ilgili ayrıntılara vâkıf olmuş olursunuz ve bu nedenle hata yapmazsınız (veya daha az yaparsınız).

Bu konuya biraz daha açıklık getireyim. Bir nesneyi veya konsepti tanımlarken onu mutlaka bir tür’e atamak zorundayız. Örneğin, “otomobil” terimini tanımlarken; *“Otomobil kavramının ilk ortaya çıktığı zamanı göz önüne alırsak at kullanılmadan, itmeden veya çekmeden kendiliğinden hareket edebilen öz itmeli taşıt anlamına gelmektedir”* [e26] ifadesini kullanıyoruz. Dikkat edersek, en son 3 kelimedede, otomobilin bir **TAŞIT** olduğu yani taşıt türüne ait olduğu belirtiliyor. Buradan anlıyoruz ki, otomobil bir taşıttır. Daha öncesinde kullanılan kelimeler ise, çoğunlukla sıfattır yani bu TAŞIT’ın özelliklerini belirtir (kendiliğinden hareket edebilen, öz itmeli...).

Şimdi kuralı yazalım: Bir tanımlama yaparken;

- 1 – Bilinmeyen tanımın türünü buluruz
- 2 – Bu tanımın özelliklerini ve diğer ayrıntılarını tespit ederiz
- 3 – Tanımı hafızamıza kayıt ederiz.

KURAL F : Dikkat ederseniz, (varsayıyoruz ki) bilmediğimiz otomobil kelimesini tanımlarken, kullandığımız bütün kelimeleri biliyoruz!!! Aksi takdirde, bu terimi – tanımı öğrenemezdik!

KURAL G : Buradan çıkarabileceğimiz sonuç olarak öğrenebileceklerimiz, öğrendiklerimizle, ve öğrendiklerimizi kullanarak anlayabileceklerimizle sınırlıdır.

Eğer otomobili tanımlama da gerekli olan kelimelerden biri, örneğin “taşıt” kelimesini bilmiyor olsaydık, otomobil terimini öğrenemezdik. Böyle bir durumda, önce “taşıt” teriminin anlamını öğrenmek, akabinde, otomobil teriminin anlamını öğrenmek gerekmektedir.

Taşıt: Farklı konumlar arasında kişi veya eşya nakliyesine yarayan **araçtır**. Dikkat edersek, taşıt bir **araçtır**. Taşıt’ın türü “**araç**” tır. Yukarıda nasıl Otomobil bir Taşıttır dediysek, taşıtta, bir araçtır.

Araç’ın tanımını incelersek:

Araç: İnsan hayatında yapılan işleri kolaylaştırmak üzere tasarlanmış nesnedir.

Peki ya nesnenin tanımı nedir?

Nesne: Uzayda boşluk kaplayan her varlığa nesne (veya madde) denir.

Peki ya varlığın tanımı nedir?

Varlık: Varlık, felsefenin temel kavramlarından birisidir. Varolan, ya da varolduğu söylenen şey, varlık kavramının içeriğini oluşturur. [e27]

KURAL H : Yine buradan aracın bir nesne olduğunu anlayabiliriz. Bu işlemi devam ettirsek, en üst türe yani **VARLIK'a** ulaşırız. Bir sınıfın üst türünü bilmeden alt tür tanımlayamayız

Temel Tür : Varlık

Alt tür : Nesne

Alt tür : Araç

Alt tür : Taşıt

Alt tür : Otomobil

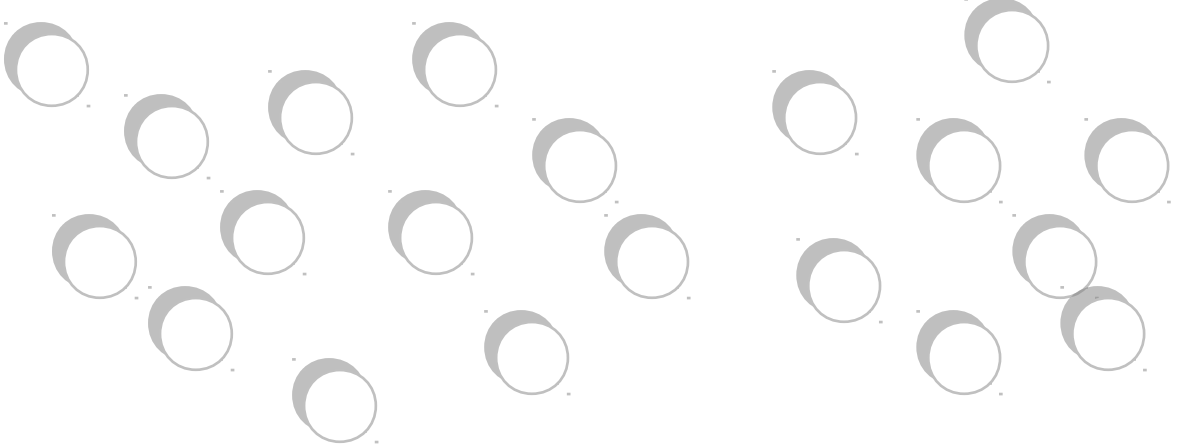
Yukarıda gördüğünüz terimlerin en üstünde, varlık bulunuyor, daha sonra onların alt türleri bulunmaktadır. Bu yapı bir tür zincire benzemektedir. Zincirin ilk halkası daima varlıktır, son halkası ise, tanımladığınız sınıftır.

Biraz beynin çalışma yapısından bahsedelim. Yeni bir terim öğrendiğinizde, beyin, öncelikle, beyninizde bulunan diğer nöronlarla ilişkisi bulunmayan bir nöron grubunu aktif hale getirir. Daha sonra bu grubu adlandırır ve yeni öğrenilen bu terimin ismini yazar. Bundan sonraki evrede, yeni terim; beyinde daha önceden kayıtlı olan nöron gruplarıyla ilişkilendirilir. Eğer ilişkilendirileceği nöron grubu yok ise, o zaman bu terimi öğrenememiş oluruz.

Bir örnek verelim. Yeni bir terim olan “zoad” kelimesini duyduğumuzu düşünelim. Eğer bu kelimenin ne anlama geldiğini bilmiyorsak, bizim için hiçbir anlam ifade etmeyecektir. Daha sonra aynı kelimeyi duyduğumuzda, **sadece** bu kelimeyi daha önceden söyleyen kişiyi ve söylendiği ortamı hatırlayabiliriz. İşte, bir terimin diğer nöron gruplarıyla ilişkilendirilmemiş haline en güzel örnek bu durumdur.

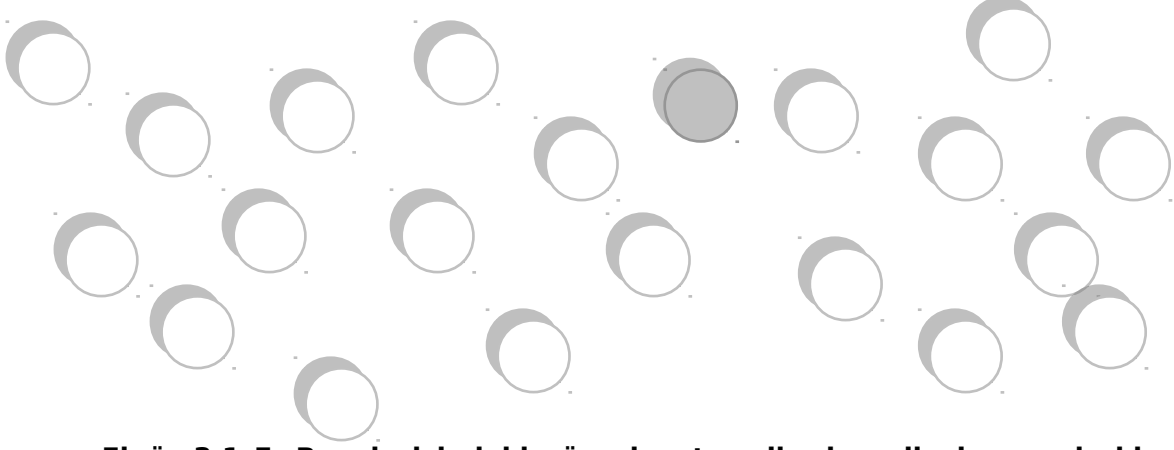
Bu işlem, yeni terimin, özelliklerinin (sıfat), türünün(isim), eylemlerinin(fiil) belirlenmesiyle devam eder. Bu işlemler sonucunda, yeni terim, artık öğrenilmiş bir terim olur.

Şimdi bu anlattıklarımı grafiksel olarak görelim. Aşağıdaki kutunun beynimizin içi olduğunu düşünelim ve içinde milyonlarca nöron grubu olduğunu düşünelim.



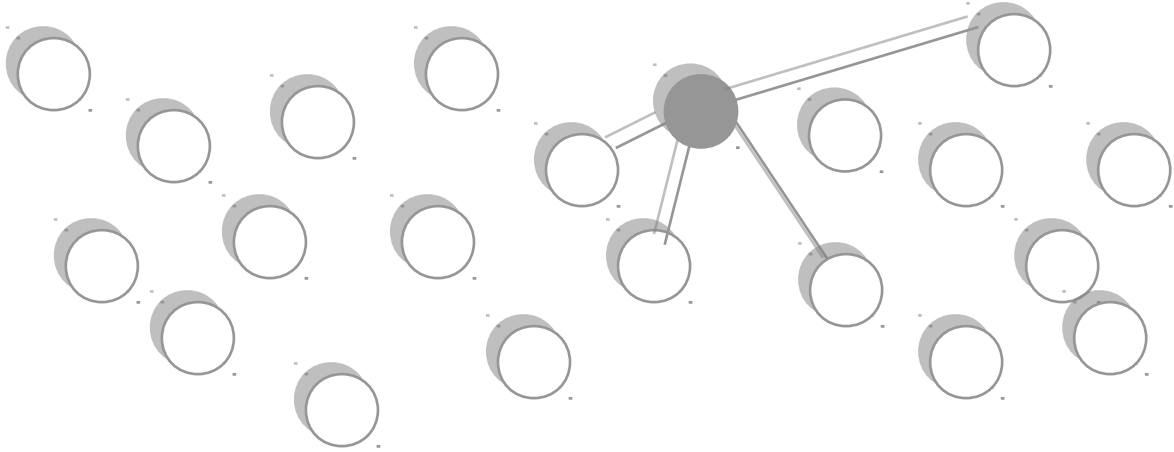
Figür 3.1 D: Beynin içindeki nöronları temsil eden elipsler

Daha sonra yeni terimi ekleyelim (gri renkte olan)



Figür 3.1 E: Beynin içindeki nöronları temsil eden elipsler, yeni eklenen nöron

1. Evre yeni nöron grubunun eklenmesiydi. 2. Evre nöron grubunun diğer nöron gruplarıyla bağlanmasıydı.



Figür 3.1 F: Beynin içindeki nöronları temsil eden elipsler, nöron bağlantıları

3. Evrede, eğer yeni terimle ilgili olarak bilmediğimiz bir başka terim bulunuyorsa, bilmeliyiz ki öncelikle bu bilinmeyen terimi öğrenmemiz gerekmektedir. Ancak bunun akabinde, yeni öğrenilecek olan terimi öğrenebiliriz.

Aynı şekilde, otomobil kelimesini öğrenirken, öncelikle boş bir nöron grubu aktif hale getirilir sonrasında ise, otomobil, taşıttan türediği için taşıt nöron grubu ile bir ilişki kurulur ve sonrasında ise özellikleri eklenir, yani bir otomobilin, bir taşıta göre eklentileri ve farklılıkları belirtilir.

Bu konuyu çok ayrıntılı olarak anlatmak istemiyorum. İşin bilimsel kısmı önemli değildir. Bu örnekleri yani beynin çalışma yapısını sadece olayların nasıl gerçekleştiğini anlatmak amacıyla gösterdim. Ayrıntılı bilgi için kitabın web sitesini inceleyebilirsiniz.

Alıştırmalar

- İnsanoğlunun bilimdeki başarısının sırrı nedir?
- İcatlar nasıl gerçekleştirilir?
- Tür nedir? Örnek veriniz
- Sınıf nedir? Örnek veriniz
- Öğrenmenin tanımını yapınız
- Bir nesne nasıl sınıflandırılır?
- Yeni bir bilgi öğrenilirken ne gibi işlemler gerçekleşir?

3.2 Beyinde programlama

KURAL A'da, insan oğlunun insanı ve doğayı taklit ettiğini belirtmişim. Aynı işlem programlama yapısı içinde geçerlidir. Yani, programlama sistemi, beynin çalışma yapısına çok yakın bir şekilde çalışır. Zaten programlamadaki “nesnelerin programlanması”, beyinden esinlenerek tasarlanmıştır.

KURAL D'de, dünyada türlerin yani sınıfların bulunduğunu söylemişim. Aynı ifade programlama içinde geçerlidir ve programlama da, sınıflardan ve o sınıflardan türetilen nesnelerden oluşmaktadır.

Önceki bölümlerde, değişkenlerin belirli veri türleri olduğunu belirtmişim. Programlama da kullanılan değişken türleri, en fazla 20-30 (temel veri tipi) adettir. Halbuki, gerçek dünyada özellikleri birbirlerinden farklı milyonlarca TÜR bulunmaktadır. İsim olarak ifade ettiğimiz her sözcüğün bir tür olduğunu söylemiştik. Peki nasıl oluyorda, gerçek dünyadaki milyonlarca türler, sadece 20 tip veri türü ile ifade edilebiliyor? İşte bu sorunun cevabını yazının devamında bulabilirsiniz (sadece soruyu aklınızda tutmanız için bu ifadeyi kullandım).

Tür yada **Sınıf**, bilgisayar dünyasında "**Class**" olarak ifade edilir. Aynı şekilde, gerçek dünyadaki **varlıklar** da, bilgisayarda "**Object**" olarak yani nesne olarak ifade edilir.

ADT Class bir "**Abstract Data Type**" dır **yani özet veri türüdür**.

Abstract data type yani **özet veri türü**, bir veya birden fazla özellik kullanarak, bir sınıfı tanımlayan bir yapıdır.

Evrende, milyonlarca farklı sınıf bulunduğunu yineliyorum. Peki bütün bu sınıflar kök sınıf olan varlıktan yola çıktıysa, neden birbirlerinden farklılar ve bu farklılığı ne sağlıyor? Yani neden bir sınıf bir başkasından farklıdır? Sınıflar arası farkın nedeni nedir ve bu farklılığı sağlayan nedir?

Bir sınıf sadece isime sahip değildir. İsim bütün sınıflar için ortak özelliktir ancak, sınıfların isimden başka özellikleri de bulunmaktadır.

- **Sınıfları nitelendiren özellikler (insan sınıfının, kilo özelliği, boy özelliği)**
- **Sınıflara uygulanabilecek olan işlemler (makine sınıfının, çalıştır işlemi)**
- **Sınıfların durum değişikliklerini bildiren olaylar (otomobil sınıfının kaza yaptı olayı)**

bulunmaktadır. Bu belirtilen üç tip eleman, diğer sınıflar ile arasındaki farkı oluşturur ve sınıflar arasındaki fark sınıfların özelliklerinden, metotlarından ve olaylarından kaynaklanmaktadır. Bir sınıfın **ismi**, bir **üst sınıfı**, **özellikleri** – **metotları** ve **olayları** asla başka bir sınıfla aynı şekilde olamaz.

Nasıl insanların DNA'ları birbirlerinden farklı ise ve hiçbir şekilde iki farklı insanın DNA'sı aynı olamıyorsa, sınıflarında bu beş temel öğesi asla ve asla tıpatıp benzer olmayacaktır. Metotları aynı, özellikleri aynı olsa bile, isimleri mutlaka farklı olacaktır.

Bölüm 3.1 ve Bölüm 3.2'nin buraya kadar olan bölümünde, sınıflar ve varlıklar hakkında bilgi verdim. Şimdi ise, bu verdiğim bilgileri pratiğe dökceğiz.

TANIMLAMA Bir varlıkları tanımlarken, ilk olarak; o nesnenin hangi sınıfa yani hangi türe ait olduğunu belirtiriz.

(iki kişi arasındaki konuşma olarak düşünelim)

- Ahmet, burada bir şey var.
- Nedir o?
- Bu bir çeşit eski **tekne**

Yukarıdaki örnekte gördüğümüz gibi, ilk olarak bir varlığın, türü yani sınıfı belirlenir ve belirtilir.

BELİRTME Daha sonra, bu nesnenin, görünen ve diğerlerinden farklı olduğunu düşündüğümüz özelliklerini, tek tek, belirtiriz.

- Peki nasıl bir tekne?
- **Rengi kırmızı, eski**
- Modeli belli mi?
- **Belli değil**
- Hangi motor tipinde?
- **Eski model KW serisi bir motoru var, hız göstergesinde en yüksek değer 70'i gösteriyor**

Yukarıdaki örnekte de dikkat edersek, NASIL – HANGİ ... gibi soruların cevaplarını aldık. Yani ilk olarak varlığın sınıfının tanımlanmasından sonra bu varlığın özellikleri belirtilir. İlk olarak türü belirtirken bir isim kullandık sonrasında, o varlığın özelliklerini ve bu özelliklerin değerleri (örneğin yukarıdaki özelliklerden biri RENK, değeri ise KIRMIZI) belirtirken sıfat kullandık.

Varsayalım ki, bir arkadaşınızın hiç bilmediği bir varlığı ona anlatmaya çalışıyorsunuz. Ne gibi ifadeler kullanırdınız? Ben bunu belirlemek için birçok deney yaptım ve bunlardan birini sizinle paylaşayayım.

Aşağıdaki yazışmada geçen, **yazar**, beni, **denekte**, o an için test ettiğim kişiyi temsil ediyor. [...] deneğin ismini ifade etmek için kullanılmıştır, önemli değildir. Lütfen aşağıdaki deneyi okumadan geçmeyin.

YAZAR:

[...], seni denek olarak kullanabilir miyim?

DENEK:

Olur kullanabilirsin.

YAZAR:

Programlama için 1-2 soru soracağım sende cevaplayacaksın.

DENEK:

Tamam.

YAZAR:

[...], **netrino** diye bir varlık bulunuyor bunun ne olduğunu billyormusun?

DENEK:

Hayır

YAZAR:

Peki anlatmaya çalışayım.

Akse adlı bir gezegende bulunan bir madde. Dünyada bulunmuyor.

YAZAR:

Şuan tam olarak ne olduğunu anladın mı?

Yoksa başka bilgi istermisin?

DENEK:

Evet anladım ama sorularım var.

DENEK:

Bir madde.

Katı, sıvı, gaz ?

YAZAR:

Katı bir madde.

YAZAR:

Başka sorun bulunuyor mu?

DENEK:

Taş gibi mi ?

YAZAR:

Değil

DENEK:

Tahta gibi mi peki?

YAZAR:

Değil

DENEK:

Katı bir madde diyelim

YAZAR:

Peki, bu maddeyi görsen - tutsan tanır mısın?

DENEK:

Tanıyamam.

YAZAR:

Neden?

DENEK:

Çünkü, şeklini ebatını rengini, fiziki özelliklerini bilmiyorum

YAZAR:

Peki ben sana anlatsam, sonrasında görsen tanır mısın?

DENEK:

Tanırım

YAZAR:

Tamam, bükülebilen bir madde, kolay kopmuyor, buna rağmen sert, parlak sarı bir renkte, sudan daha ağır (özkütle olarak)

YAZAR:

Şimdi sen bu kelimeleri kullanmadan bana, sana anlattığım maddeyi tanımlarmısın

Yada kullanmamaya çalış, yani tamamen aynı şekilde olmasın

Yani şöyle düşün, ben YAZAR değil başkasıyım sen bana bunu anlat.

DENEK:

Tamam

DENEK:

Dünyada olmayan bir tür madde

DENEK:

Taştan daha yumuşak, sarı parlak renkte altın gibi mesela

DENEK:

Gevşeyebilen (gevşemeden kasıt bükülebilen maddeler gevşer)

YAZAR:

Anladım

YAZAR:

Peki neye benzer daha çok?

DENEK:

Hafif kategorisinde diyebiliriz

DENEK:

Teli anımsatıyor bana yada ince levha

YAZAR:

Çok teşekkürler, deneyi bitti

Dikkat ederseniz, ilk olarak, (yazar) ben, deneğe, tanımlama yaparken, anlatmak istediğim varlığın öncelikle bir tür madde olduğunu söyledim, sonrasında bu maddenin özelliklerini belirttim. Aynı şekilde, denekte, bu varlığı bana anlatırken ilk olarak bu varlığın sınıfını ve sonrasında özelliklerini belirtti.

İlk kez gördüğünüz bir otomobili, telefonda arkadaşınıza anlatırken ne gibi ifadeler kullanırsınız?

- Mavi renkte, yerden oldukça yüksek, 5 silindirli, bir otomobil.

Programlamada da sistem aynı şekilde işlemektedir. Öncelikle, bir sınıf tanımlarız ve sonrasında bu sınıfın özelliklerini belirtiriz.

İlk iki bölümde, sistemlerin analiz edilerek daha küçük parçalara ayrıldığını söylemiştim. Aynı şekilde sınıflarda veya o sınıflara ait varlıklarda, sahip olduğu özellikler bakımından analiz edilerek daha küçük parçalara ayrılırlar. Çoğu zaman bir varlığı tek bir özellik ve onun değeri ile belirtemeyiz. Bir varlığın bir dizi özelliği bulunmaktadır. Örneğin bir kişiyi belirtmek için, ismini, soyismini, baba-adını ve bazı diğer gerekli bilgileri kullanırız.

Bir sınıf, birden fazla değişkenden oluşur. Örnek olarak, "kişi" şeklinde bir sınıf tanıtaçağımızda, "kişinin ismi" (metin veri türünde), "kişinin yaşı" (tamsayı veri türünde), "kişinin kilosu" (ondalık veri türünde) değişkenlerini tanımlayabiliriz.

İşte bu gibi durumlarda, bir veya birden fazla özelliğe sahip olabilen sınıfları tanımlamak için, standart değişken türleri yeterli olmayabilir, ve biz bu gibi durumlarda "sınıf" ları kullanırız.

"İnsan" sınıfını düşünelim. İnsan tek bir değerle ifade edilebilecek bir nesne – yada canlı değildir. Bir insanın birçok özelliği ve bu özelliklerin çok farklı değerleri olabilir. Az önce belirttiğim gibi bir insanın kilosu o sınıfın (insan sınıfının) bir özelliğidir.

Bir sınıf tanımlamak, programsal bir ifadeden oluşur her dilde farklı ifade edilir ancak, ilk bölümde belirttiğim gibi dillerin farklılığının önemi yoktur.

Class: Otomobil

Yukarıda anlattığım gibi, bu terimin ilk kez duyulduğunu ve beynin bu terimi öğrenmeye çalıştığını düşünelim. İkinci işlem, bu sınıfın türünü atamadır. Otomobilin türü "taşıttır". (Class, sınıf anlamına gelir, Type ise Tür)

```
Class: Otomobil Type: Taşıt
```

Pseudo kodunun standartlarını kendimizin belirlediğini ve bu ifadelerin başka şekilde olabileceğini hatırlatayım. Örnek:

```
Sınıf = Otomobil / Üst Tür = Taşıt
```

Önemli olan isimler ve şekiller değil, bu durumda bir sınıfın tanımlanmış olmasıdır. Dış dünyadaki sistemler; sınıflar üzerine kurulu olduğu için programlamada da sistemler sınıflar üzerine kuruludur. Bir sınıfı bilgisayar dünyasına tanımlama için ilk iki adımı tamamladık. Bundan sonrada, sıra, özelliklerin, olayların ve metodların tanımlanmasındadır.

Özellikler, bir isim ve bir değerden oluşan ve dil yapısında **sıfat** olarak nitelendirdiğimiz terimlerdir

Bir otomobilin ne özellikleri bulunur?

Hızı (kaç km) : (örneğin) 220 Km/h

Motoru Hacmi (cc) : (örneğin) 2000 cc

Markası: (örneğin) Audi

Modeli: (örneğin) A6

Ağırlığı: (örneğin) 980 KG

v.s.

Bu belirlediğimiz özellikleri sınıf tanımımızın içine yazalım

```
Sınıf = Otomobil / Üst Tür = Taşıt
Özellik : Hız
Özellik : Motor Hacmi
Özellik : Marka
Özellik : Model
Özellik : Ağırlık
```

Bu bölümde dikkat etmemiz gereken, hazırlıyor olduğumuz sınıfı nasıl bir kullanıcı veya amaç için hazırladığımızdır. Yani hangi amaca hizmet edeceğidir. Örnek olarak, bir otomobil galerisinde kullanılmak üzere bir program yazdığımızı düşünelim. Bir galeride, müşteriye satışta, otomobilin ağırlığının önemi yoktur (o şekilde varsayalım). Ancak otomobili sınıfımıza "**fiyat**" özelliğini de eklemeliyiz ki, satış işleminde fiyatla işlem yapabilelim.

```
Sınıf = Otomobil / Üst Tür = Taşıt
Özellik : Hız
Özellik : Motor Hacmi
Özellik : Marka
Özellik : Model
Özellik : Fiyat
```

Aynı sınıf için birden fazla, farklı tanım olabilir. Peki nasıl oluyorda, aynı türü ifade eden sınıflar birden fazla farklı şekilde tanımlanabiliyor? Daha önceden belirttiğimiz gibi, bir programı tasarlarken, hedef kullanıcı kitlesi çok önemlidir.

Belirtilen bu sınıfın 2 farklı tanımını yapalım

Otomobil sınıfının 1. tanımı

Otomobil (üreticileri) tasarımcıları tarafından kullanılacak olan bir program için tasarlanan sınıf

Otomobil sınıfının 2. tanımı

Otomobil satıcıları (galeriler) tarafından kullanılacak bir program için tasarlanan sınıf

Sınıf Otomobil / Üst Tür = Taşıt

Özellik: Model (aracın modelinin ismi)

Özellik: Ağırlık

Özellik: Genişlik

Özellik: Yükseklik

Özellik: Uzunluk

Özellik: Yük kapasitesi

Özellik: Fren mesafesi

....

Sınıf Otomobil / Üst Tür = Taşıt

Özellik: Model (aracın modelinin ismi)

Özellik: Fiyat

Özellik: Popülarite (satış sayısı)

Özellik: Renkler

Özellik: Taksitli satış imkanı

...

Yukarıda tanımlanan iki sınıf aynı tür olan otomobile aittir. Ancak aralarında büyük farklılıklar olduğunu görebiliriz. Bu farkların neden bahsettiğimizi gibi, programın ne amaca hizmet edeceğinden kaynaklanmaktadır. Soldaki tanımı kullanacak olan program, üretimle görevli mühendisleri hedef almaktadır. Yani bir otomobil üretim fabrikasında, otomobil tasarımı, testi, planlaması yapan mühendislerin evraklarını ve otomobillerin özelliklerini hafızada tutan bir program tasarladığımızı düşünelim. Bu programda kullanılacak olan sınıfta, o otomobilin, fiyat veya renk özelliklerinin olmasına gerek bulunmamaktadır. Çünkü, bu özellikler mühendisleri ilgilendirmemektedir. İlk bölümde, bir sistemi programlarken, amaca göre farklı çıktılar verebildiğini söylemiştik.

Galeri sahibi tarafından kullanılacak olan programda ise, yük kapasitesi, fren mesafesi gibi özellikler önemli olmadığından bulunmayacaktır. Bununla birlikte otomobilin fiyatı galeri sahibi için çok önemlidir ve bu özellik sınıf tanımında mutlaka bulunmalıdır.

İşte gördüğünüz üzere iki farklı program için iki farklı tanım kullandık ancak bu sınıf tanımları AYNİ türe aittir. Dikkat ederseniz bu iki tanımda ortak olan “Model” özelliği bulunmaktadır. Neden her ikisinde de bu özelliğin bulunduğuna gelince; sınıflarda, genelde bir veya birkaç TANIMLAYICI anahtarın bulunması gerekmektedir.

Tanımlayıcı Tanımlayıcı anahtar; bir küme içindeki nesnenin, diğerlerinden ayırt edilebilinmesi için, seçilmiş olan bir veya birden fazla özelliğe verilen addır. Peki bu ifade ne anlama gelir? Bir küme düşünelim, örneğin: Türkiye’deki insanlar. Bu küme içinden bir nesneyi yani bir kişiyi seçmek istiyoruz. Bu kişiyi seçebilmek için en az bir özellik ve bir değer ifade etmeliyiz. Örneğin; seçeceğimiz kişinin ismi “Ahmet” olsun. İstenilen kişiyi bulmak amacıyla, “isim” özelliğini kullandık ve seçmek istediğimiz kişinin isminin “Ahmet” olduğunu belirttik. Eğer, Türkiye’de (yani kümemizde) sadece 1 tane Ahmet olsaydı, o kişiyi rahatça seçmiş olurduk. Ancak, Türkiye’de yüzbinlerce Ahmet isminde kişi bulunmaktadır.

İşte bu nedenle, sadece isim özelliği istediğimiz kişiyi belirtmeye yeterli olmayacaktır. O halde, “isim” özelliği TEK başına bir “Tanımlayıcı anahtar” değildir. Peki, ifadeyi biraz daha genişletelim. İsmi Ahmet, Soyismi KÖROĞLU olan kişiyi seçmek istediğimizi varsayalım. Yine aynı şekilde, ismi Ahmet ve soyismi KÖROĞLU olan birden fazla kişi olabilir ve “isim + soyisim” özellikleri yine az önceki gibi tek başlarına bir tanımlayıcı anahtar değildir.

İfadeyi daha da genişletirsek; ismi Ahmet, soyismi KÖROĞLU, baba adı Mustafa, anne adı Esra, doğum Yılı 1984, cinsiyeti erkek, nüfusa kayıtlı olduğu il İstanbul, nüfusa kayıtlı olduğu ilçe Üsküdar olan kişiyi seçmek istersek eğer, o zaman bu özellikler (isim, soyisim, baba adı, anne adı, doğum yılı, cinsiyeti, nüfusa kayıtlı olduğu il, ilçe) belirttiğimiz kişi, seçebilmemize yeterli olacaktır ve bu özellikler bir tanımlayıcı anahtar olacaktır. Bu bilgiyi T.C kimlik no sorgulama sitesinden görebilirsiniz.

T.C. Kimlik No Sorgulama Bilgileri	
İl: İSTANBUL	İlçe: ÜSKÜDAR
Adı: Ahmet	Soyadı: KÖROĞLU
Lütfen, aşağıdaki resimdeki sayıyı Resim Doğrulama alanına girdikten sonra, sağ veya sol sorgulama alanlarından yalnız birini kullanarak işleme devam ediniz.	
Cilt No:	Baba Adı: Mustafa
Aile Sıra No:	Anne Adı: Esra
Birey Sıra No:	Doğum Yılı: 1984
	Cinsiyeti: Erkek
T.C. Kimlik No Sorgula	T.C. Kimlik No Sorgula

Figür 3.2 A: TC Kimlik Sorgulama ekranı - 1

Gördüğümüz gibi, bu özelliklerin hepsi aynı anda kullanılırsa ancak istediğiniz kişiye ulaşabilirsiniz. Tabi, yukardaki resmin sol alt bölümüne bakarsanız, aynı kişiyi bulmak için, farklı bir alternatif sunulmuştur ve aynı şekilde, bu özelliklerde bir kişiyi tanımladığı için, bir tanımlayıcı anahtardır.

T.C. Kimlik No Sorgulama Bilgileri	
İl: İSTANBUL	İlçe: ÜSKÜDAR
Adı: Ahmet	Soyadı: KÖROĞLU
Lütfen, aşağıdaki resimdeki sayıyı Resim Doğrulama alanına girdikten sonra, sağ veya sol sorgulama alanlarından yalnız birini kullanarak işleme devam ediniz.	
Cilt No:	Baba Adı: Mustafa
Aile Sıra No:	Anne Adı: Esra
Birey Sıra No:	Doğum Yılı: 1984
	Cinsiyeti: Erkek
T.C. Kimlik No Sorgula	T.C. Kimlik No Sorgula

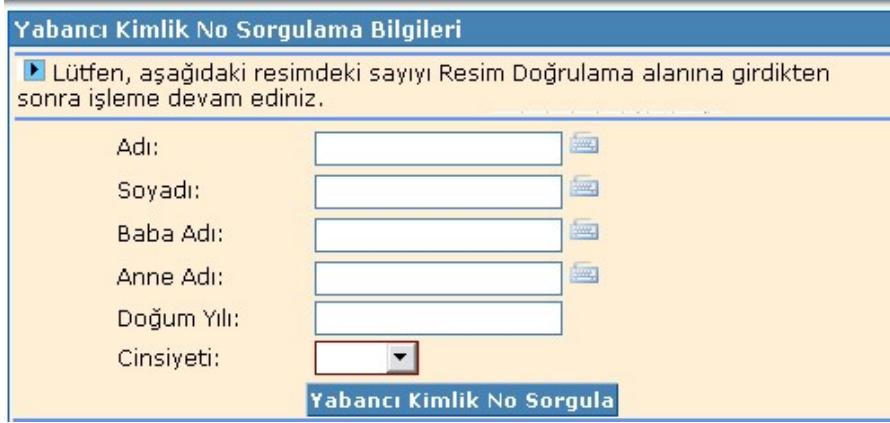
Figür 3.2 B: TC Kimlik Sorgulama ekranı - 2

Sonuç olarak;

İsim, Soyisim, İl, İlçe, Baba adı, Anne Adı, Doğum yılı, Cinsiyeti
özellikleri kullanılarak veya

İsim, Soyisim, İl, İlçe, Cilt No, Aile Sıra No, Birey Sıra No
bu iki tanımlayıcı anathar ile de istenilen kişiye ulaşabilirsiniz.

Tabi bir de ek olarak, TC Kimlik Numarası kullanarak istenilen kişiyi seçebilirsiniz. Türk olan, Türk vatandaşlarını seçmek için bu kadar ayrıntı gerekirken, yabancı uyruklu Türk vatandaşlarını seçmek için daha az özellik ve koşul yeterli olacaktır.



Figür 3.2 C: TC Kimlik Sorgulama ekranı - 3

Yukardaki resimde gördüğünüz gibi, sadece, Ad, Soyad, Baba adı, Anne Adı, Doğum Yılı ve Cinsiyeti özelliklerini belirttiğinizde, istediğiniz kişiye ulaşabilirsiniz. Gördüğünüz gibi, bir küme içinden bir nesne seçmek için yeterli özellikleri sağlayan koşula tanımlayıcı anathar denmektedir.

Bir sınıfa, o sınıfı diğerlerinden ayıran bir dizi özellik mutlaka eklememiz gerekmektedir. Yani sınıfların, tanımlayıcı anahtarları bulunması gerekmektedir.

Yukarıdaki iki sınıfta tanımlayıcı birer özellik kullanılmıştır. İlk sınıfta (soldaki) Model özelliği o sınıfın Tanımlayıcı anahtarıdır. Çünkü bir otomobil üreticisinde, aynı modele (model ismine: Audi **A8**, Toyota **Corolla**, Mazda **626**) sahip farklı otomobiller olamaz. Bir firma, her modele farklı isimler verdiği için sadece Model özelliği, bir otomobili tanımlamak için yeterli olacaktır. Aynı şekilde, sağ taraftaki sınıf tanımında da, model özelliği tanımlayıcı anahtartır.

Özellikleri belirlemede, özelliklerini belirleyeceğimiz sınıfın, sıfatlarını dikkate almak gerekmektedir. Dikkat edersek, sınıflara, sadece özelliklerin isimlerini yazarız, değerlerini değil. Daha sonra tanımladığımız sınıftan bir nesne oluşturduğumuzda, özelliklerin değerlerini yazabiliriz. 3.1 bölümünde, sınıfın bir kalıp, nesnenin o kalıptan üretilen bir ürün olduğunu söylemiştik. Aynı şekilde, sınıf tanımlanması bittikten sonra o sınıftan bir nesne oluşturup, bu oluşturulan nesnenin özelliklerine değer ataması yapılabilir.

Özellikler aslında birer değişkendirler. Biz sınıfların özelliklerini tanımlarken, o sınıf için kullanılacak değişkenleri belirlemiş oluruz. Yani bir sınıf, biri dizi değişkeni tutan bir kalıptır (yapıdır) denilebilir.

Sigortalının:	
Adı :	Anne Kızlık Soyadı :
Soyadı :	Mesleği :
Baba Adı :	İşyeri Sicil No :
Anne Adı :	T.C.Kimlik No :
Doğum Yeri :	Vergi Kimlik No :
Doğum Tarihi :	Vergi Dairesi :

BU BÖLÜM ÜYE TARAFINDAN DOLDURULACAKTIR.

Figür 3.2 D: Bir form örneği

Örneğin, bir tıbbi laboratuvar da kullanılmak üzere bir program yazdığınızı düşünelim. Burada kullanacağımız “hasta” sınıfında ne gibi özellikler olabilir? Bunu belirlemek için, en kolay yöntem, laboratuvarı bilgisayar kullanılmadan önce, tasarlanan formlara ne gibi bilgilerin girildiğini bulmaktır. Tıbbi laboratuvarı çalışan bir arkadaşımın aldığı form örneğinde aşağıdaki bilgiler bulunmaktaydı.

Hasta İsmi
Hasta Soyismi
İşlem Tarihi
İşlem (hangi tahlil yapılmış)
Kolestrol Seviyesi

Yukarıdaki verilerden, bilgilere göre, oluşturulmuş formun, sınıfını tasarlarken, yine aynı şekilde yukarıdaki özellikleri, sınıfın özellikleri olarak kullanacağız.

Sınıf Hasta İşlemi
Özellik : Hasta İsmi
Özellik : Hasta Soyismi
Özellik : İşlem Tarihi
Özellik : İşlem
Özellik : Kolestrol Seviyesi

Özellik türleri gerçek hayatta yazılı olarak ifade ettiğimiz sembollerin türlerinden kaynaklanmaktadır. Daha açık bir ifade ile, yazılar, metinlerle (karakterlerle) , sayılarla (rakamlarla) ve sembollerle ifade edilir.

Test 12341234 !?

Bu durumda, 3 farklı veri tipi ortaya çıkmaktadır: Metin, Sayı, İşaret (sembol). Ayrıca, bu veri tipleri farklı şekilde ifade edilerek yeni veri tipleri ortaya çıkmaktadır. Örneğin, Tarih, Ondalık Sayı, Saat, Tamam – İptal İşaretleri (✓ , ✗)

3.3.2007 Tarih
20:04 Saat
2341,43494 Ondalık sayı
✓ Tamam/İptal (doğru/yanlış)

Sınıfların özelliklerinin değerleri de bu veri tiplerinden ve bu veri tiplerinin birleşmesiyle oluşan veri tiplerinden oluşacaktır.

Sınıf Hasta İşlemi
Özellik : Hasta İsmi (metin)
Özellik : Hasta Soyismi (metin)

Özellik : İşlem Tarihi (tarih)
Özellik : İşlem (metin)
Özellik : Kolesterol Seviyesi (ondalık sayı)

Özelliklerden sonra, metodlara geçelim. Metodlar, bir türün yapabileceği veya onlara uygulanabilecek işlemlerdir.

Bir otomobil, kaza yapabilir, bakıma girebilir, satılabilir,... Az öncede belirttiğim gibi, yapabileceği işlemler yüzlerce de olsa, önemli olan konumuzla ilgisi olan metodlarını tespit etmektir.

Bir otomobil galerisini, otomobili sattıktan sonra, otomobilin kaza yapması, bakıma girmesi ilgilendirmez. Dolayısıyla programımızı da ilgilendirmez. Ancak, satılma işlemi, geri alınma işlemi, üreticisine geri gönderilme işlemi, otomobil galerisiyle ilgilidir.

Peki diyeceksiniz ki, program bu işlemleri kendi başına mı gerçekleştirecek? Tabi ki hayır. İlk bölümde bahsettiğim gibi, şirketlerin kullandığı programların genel amacı veri işlemektir.

Bir program bu tip işleri yapamaz. Ancak satılma işlemi, geri alınma işlemi, üreticisine geri gönderilme işlemi yapıldığında, bu işlemin yapıldığına dair veriyi kayıtlara girebilir. Böylece siz rapor çıkardığınızda hangi aracın satıldığını hangi aracın nerede olduğunu bilebilirsiniz.

Sınıfımızı tekrar yazalım bu sefer metodları ekleyelim.

```
Sınıf = Otomobil / Üst Tür = Taşıt  
Özellik : Hız  
Özellik : Motor Hacmi  
Özellik : Marka  
Özellik : Model  
Özellik : Fiyat  
Metod : Satış yap  
Metod : Üreticiye geri gönder
```

Eklenen metodlar, çağrıldıklarında (yani işleme konulduklarında), içlerinde belirtili işlemleri gerçekleştirirler. Örneğin, **satış yap** metodu çağrıldığında, galeride bulunan otomobillerden bir tanesini azaltır, toplam gelir miktarına, o otomobilin ücretini ekler ve yeni (aynı model) bir otomobilin gönderilmesi için üretici firmaya yazı gönderir.

Son olarak; sınıfımıza, bu otomobile, bizim tarafımızdan veya başkaları tarafından uygulanabilecek metodlar sonucunda, oluşacak olayları veya **durum(Bölüm 2.5)** değişikliklerini eklememiz gerekiyor.

Örneğin, satılmış olan bir otomobil, müşteri tarafından geri iade edilebilir. Böyle bir durumda, otomobil, bu eylemden etkilenen olacaktır. Yani etken değil edilgen özne olacaktır.

```
Sınıf = Otomobil / Üst Tür = Taşıt  
Özellik : Hız  
Özellik : Motor Hacmi  
Özellik : Marka
```

```
Özellik : Model
Özellik : Fiyat
Metod : Satış yap
Metod : Üreticiye geri gönder
Olay : Geri iade
```

Bir olay, genelde bir özelliğin değeri değiştirildikten sonra bu özelliğin değerinin değişmesi o nesnenin bir durumunun değişmesini gerektiriyorsa veya bir metod çağrıldıktan sonra durumda bir değişiklik oluyorsa gerçekleşir. Bir olayın gerçekleşip gerçekleşmeyeceğine yine bizim yazdığımız kurallar karar verecektir.

Örneğin, bir otomobilimiz var ve biz yol aldıkça, gidilen kilometre miktarı artacaktır. Bu özelliğin değişimi her zaman bir “olay” meydana getirmez . Ancak eğer gidilen yol miktarı, 10.000 kilometreye ulaşmışsa, aracın servise girmesi gerektiğini düşünelim. Böyle bir durumda, “servis gerekli” şeklinde bir “olay” çağırılabilir ve bu olay çağırıldığında yetkili kişi, servisi arayıp, otomobilin servise gitmesi gerektiğini belirtebilir.

Bu öğrendiklerimizi bir örnek üzerinde pekiştirelim: Varsayalım ki bir firmada sipariş işlemleri üzerine çalışmaktayız. Kullandığımız program, sipariş tabikini yapmakta olan bir program olsun. Girilen her bir sipariş için, “sipariş” adında bir nesne oluşturduğuna varsayalım.

```
Sınıf = Sipariş
```

İlk olarak (sipariş girişinden sorumlu kişi olduğumuzdan) sipariş bilgileriniz gireriz. Bu sipariş bilgileri: siparişin içeriği, kime teslim edileceği, hangi adrese teslim edileceği v.s. bilgilerinden oluşmaktadır.

```
Sınıf = Sipariş
Özellik: Siparişi veren firma
Özellik: Sipariş tarihi
Özellik: Sipariş teslim adresi
Özellik: Sipariş içeriği (hangi ürünlerden ne kadar miktarda olduğu)
```

Bu özelliklere ek olarak birde, siparişin hangi durumda olduğunu belirlemek amacıyla, bir “durum” özelliği de eklemeliyiz.

```
Sınıf = Sipariş
Özellik: Siparişi veren firma
Özellik: Sipariş tarihi
Özellik: Sipariş teslim adresi
Özellik: Sipariş içeriği (hangi ürünlerden ne kadar miktarda olduğu)
Özellik: Durum
```

Durum özelliği, o siparişin hangi aşamada olduğunu göstermek için gereklidir. Örneğin ilk olarak durum özelliğinin değeri “sipariş yeni girildi” olabilir. Sonrasında, “sipariş hazırlanıyor”, “sipariş teslim edildi”.... gibi değerlerde alabilir.

Daha sonrasında bu siparişi kayıt ederiz ve artık (sipariş giriş sorumlusu olarak) bizim işimiz tamamlanacaktır. Sonrasında, başka bir departmanda (tedarik) çalışan bir çalışan, siparişin içeriğini hazırlamak (içerikte bulunan ürünleri toplayıp getirmek) üzere sipariş

sınıfını görüntülemek üzere, siparişin bilgilerini yazıcıdan çıkarmak için, sipariş sınıfına ait “formu yazdır” metodunu çağırır. Bu metod, sipariş bilgilerini bir rapor olarak, yazıcıdan çıkarır ve siparişin içeriğindeki ürünleri toplayıp getirecek kişiye verilir.

```
Sınıf = Sipariş
Özellik: Siparişi veren firma
Özellik: Sipariş tarihi
Özellik: Sipariş teslim adresi
Özellik: Sipariş içeriği (hangi ürünlerden ne kadar miktarda olduğu)
Özellik: Durum
Metot: Yazdır
```

Siparişin içeriği toplantıktan sonra sipariş teslim edilmek üzere, belirtilen adrese gönderilir. Teslimden sonra, teslim eden kişi; teslim ettiği kişinin adını ve teslim saatini cep bilgisayarından merkeze bildirir yani o siparişin, durum özelliğinin değerini “tamamlandı” olarak değiştirir. İşte tam bu sırada, “sipariş sonuçlandı” şeklinde bir olay gerçekleşir.

```
Sınıf = Sipariş
Özellik: Siparişi veren firma
Özellik: Sipariş tarihi
Özellik: Sipariş teslim adresi
Özellik: Sipariş içeriği (hangi ürünlerden ne kadar miktarda olduğu)
Özellik: Durum
Metot: Yazdır
Olay: Sipariş Sonuçlandı
```

Durum özelliği değeri değiştirilerek, “bitti” haline getirildiğinde otomatik olarak sipariş sonuçlandı olayını çağırdığımızı düşünelim. Bu olay sonucunda da siparişin artık bittiği ve listeden çıkarılması gerekir ve siparişin listeden çıkarılma işlemi gerçekleşir diyebiliriz.

```
Sınıf = Sipariş
Özellik: Siparişi veren firma
Özellik: Sipariş tarihi
Özellik: Sipariş teslim adresi
Özellik: Sipariş içeriği (hangi ürünlerden ne kadar miktarda olduğu)
Özellik: Durum
(
    O anki durumu girilen değer olarak ata (ör. Yeni kayıt edildi,
    teslim edilmek üzere gönderiliyor....)
    Eğer durum “bitti” ise, o zaman “sipariş sonuçlandı” olayını
    çağır
)
Metot: Yazdır
(
    Uygun bölümü bul
    Uygun bölümün yazıcısını kullanarak bu sipariş hakkındaki
    bilgileri rapor olarak çıkar
)
Olay: Sipariş Sonuçlandı
(
    Siparişin tamamlandığına dair veriyi gir
```

Siparişi listeden çıkar
)

Aslında bir olay ile bir metod arasında çok büyük bir fark yoktur. Hatta bazı dillerde “olay” olarak tanımlanan bir sınıf elemanı bulunmamaktadır. Bunun yerine metotlar kullanılır.

Bu konuyu başka bir örnekle pekiştirelim. *İngiltere’deyken, bir süper markette, ürünleri daha çok sattırmak için, kabiliyeti ve masrafı az olan bir robot geliştirilmiş ve bu robotun üzerine son çıkan yada çok satılmak istenen ürünler konulmuştu. Böylece insanların ilgisini daha kolay çekiyor ve ürünler satılıyordu.* Varsayalımki bir süper markette çalışıyoruz ve uzaktan kontrol edebileceğimiz bir robotumuz bulunuyor. Bu robotun üzerine çok satılması istediğimiz ürünleri yerleştiriyoruz ve uzaktan kumanda ile, bu robotu market içinde dolaştırıyoruz. Tabi, robotun üzerinde bir kamera bulunuyor ve biz önümüze çıkan nesneleri böylece görebiliyor ve o nesnelere çarpmadan tanıtım yapabiliyoruz.

Robotu bir kumanda değilde bir bilgisayar tarafından yönettiğimizi düşünelim ve yönetebilmek içinde bir program yazıldığını varsayalım. Bu programda neler olmalıdır?

- Robotu sağa, sola, ileri, geri (isteğe göre hızlı yavaş) hareket ettirecek bir sistem

```
Sınıf = TürkRobot
Özellik: Kalan Şarj Miktarı
Özellik: Robotun üzerindeki ürünün ismi
Özellik: Robotun üzerindeki ürünün miktarı
Metot: Sola Dön
Metot: Sağa Dön
Metot: İlerle
Metot: Geri git
Metot: Dur
Olay: Şarj Bitiyor
Olay: Ürün Bitiyor
```

Yukardaki sınıf elemanlarını tek tek açıklayayım. Kalan şarj miktarı, robotun sahip olduğu pilin kalan şarj miktarını gösteren bir özelliktir. Robotun üzerine konulan ürünün ismi tanıtım açısından önemlidir. Zira program, hangi ürünün ne kadar süreyle tanıtıldığına dair istatistik tutmak istiyor olabilir. Robotun üzerine tüketilen (yenebilir veya içilebilir) bir ürün koyduğumuzu düşünelim. İşte bu özelliğinde değeri, kalan ürün miktarını gösterir.

Sağa dön, sola dön, ilerle, geri git, dur komutları robotu hareket ettirmeye yarar. Şarj bitiyor olayı, şarj seviyesi (varsayalımki) %10’un altına indiğinde, robotu yöneten kişiyi uyarmak adına, otomatik olarak çağrılır ve robot bu olay çağrıldığında otomatik olarak (programlandığı için) merkeze döner ve şarj olur. Mağazadaki müşteriler robotun üzerindeki üründen aldıkça (yedikçe veya içtikçe) bu ürün azalıp bitecektir. Boş boş dolaşan bir robotunda faydası yoktur. Bunun yerine, ürünün bitmesine çok az kala, bu olay çağrılacak, böylece ürün bitmeden hemen merkeze gidip yeniden doldurulabilecek.

```
Sınıf = TürkRobot
Özellik: Kalan Şarj Miktarı (pile bağlı bir devre olduğunu düşünelim)
(
    Şarj miktarını kontrol et
```

```
Eğer şarj miktarı %10'dan aşağıda ise, "şarj bitiyor" olayını
çağır
)
Özellik: Robotun üzerindeki ürünün ismi
Özellik: Robotun üzerindeki ürünün miktarı (robotun üzerinde bir
elektronik tartı olduğunu düşünelim ve bu tartıda her bir ağırlık
değişiminde bu özellik değerinin otomatik olarak değiştiğini
varsayıyoruz)
(
    Robotun üzerindeki ürün miktarını kontrol et
    Eğer ürün miktarı 100 gr.'dan azsa ürün bitiyor olayını çağır
)
Metot: Sola Dön
Metot: Sağa Dön
Metot: İlerle
Metot: Geri git
Metot: Dur
Olay: Şarj Bitiyor
(
    Kullanıcıya mesajı bildir
    Merkeze dön
)
Olay: Ürün Bitiyor
(
    Kullanıcıya mesajı bildir
    Ürün doldurma bölümüne dön
)
)
```

Bu şekilde, bir sınıfı tanımlamış oluyoruz. Tanımlanan işlemler, olaylar ve özellikler artık kullanılmaya hazırdır. Bu şekilde, sınıflar üzerine kurulmuş sistemlere, OOP yani object oriented programming, nesne tabanlı programlama denmektedir. Ayrıntılarını daha ileriki konularda göreceğiz.

Alıştırmalar

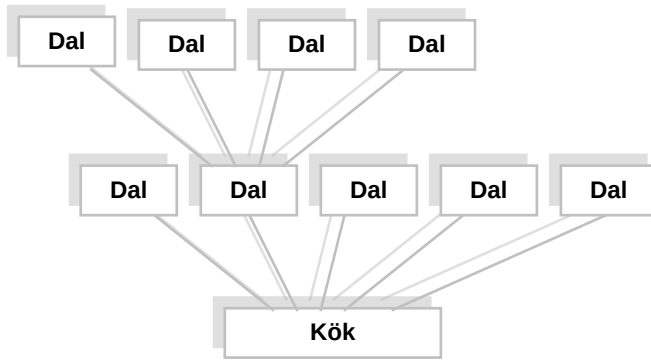
- ADT nedir?
- Sınıfların ne gibi elemanları bulunur?
- Tanımlayıcı nedir?
- Sınıf nasıl tanımlanır?
- OOP nedir?

[Ara verme bölümü]

3.3 Inheritance - Sub Class - Super Class

Bir önceki konuda, her kavramın, bir sınıftan türediğini ve onların daha alt sınıflarının olduğunu söylemiştik. En üst sınıf **varlıktır**. Bir alt sınıf olarak örneğin, Nesne'yi gösterebiliriz. Bu yapıyı bir tür ağaca benzetebiliriz. Nasıl bir ağacın bir kökü yada gövdesi bulunuyor ve onlarca dalı bulunuyorsa ve bu ağacın dallarında alt dalları bulunuyorsa, sınıf yapısı da benzer hiyerarşik bir şekildedir.

İşte bu örnekteki gibi, ağacın kökü, varlıktır. Her kavram, nesne, sistem **varlıktan** türemektedir. Yani, her ana dal, ağacın köküne bağlıdır, ve her dalda, ya ağacın direkt olarak köküne bağlıdır veya bağlı olduğu dal bir şekilde ağacın köküne bağlıdır. Bir ağacı ters çevirelim ve o şekilde hayal edelim.



Figür 3.3 A: Bir form örneği

Aslında bu diagram, daha önceki bölümlerde gördüğümüz kompleks sistem, basit sistem örneğindeki gibidir.

Her bir seviyeyi bir sınıf (hem öğrencilerin gittiği okuldaki sınıf, hemde, programlamadaki sınıf olarak) olarak düşünelim. Bu düşünceyi, ileriki paragraflarda kullanacağımız için aklınızda bulundurunuz.

OOP'deki kurallardan bir tanesi, bir üst sınıfın (yani tanımlanan sınıfın bağlı olduğu sınıf, örneğin **araç** sınıfı, nesne sınıfının bir alt sınıfıdır) sahip olduğu özelliklerin, metodların, olayların hepsi, bir alt sınıfta mutlaka olmak zorundadır, zaten üst sınıfta tanımlanmış olan özelliklerin, metodların, olayların, alt sınıfta tanımlanmasına gerek bulunmamaktadır.

Bu durumu daha açık belirtmek gerekirse; **varsayalım ki, üç öğrenci olsun. Bu üç öğrenci sırayla, ilkokul 1'e, ilkokul 2'e, ilkokul 3'e gidiyor olsun. Üçüncü sınıfa giden öğrenci, hem ikinci sınıfa giden öğrencilerin bildiklerini, hemde üçüncü sınıfa giden öğrencilerin bildiklerini bilmektedir.** Yani ikinci sınıfa daha önce gittiği ve aynı dersleri görüp öğrendiği için, hem kendi sınıfında gösterilen dersleri, hemde daha önceki senelerde (birinci ve ikinci sınıf) gösterilen konuları bilmektedir.

En üst sınıf olan VARLIK sınıfının, isim özelliği bulunmaktadır. Her varlığın, bir ismi vardır. Varlıktan türemiş sınıf olan NESNE sınıfında **isim** özelliği doğal olarak bulunmak zorundadır; öyle ki, her nesnenin bir ismi vardır. Bir kere daha devam ettirelim; Nesneden türemiş olan Araç sınıfı'nın da **isim** özelliği bulunmaktadır. Her aracın bir ismi vardır (örneğin "çekiç"). Şimdi de yukarıdaki örnek olay ile bu olayı birleştirelim.

- Varlığın ismi vardır. Varlıktan türemiş sınıfların (örneğin nesne) da ismi vardır.
- Birinci sınıfa giden bir öğrenci, çarpım tablosunu bilir. İkinci sınıfa giden bir öğrenci de çarpım tablosunu bilir. Yine doğal olarak, üçüncü sınıfa giden bir öğrenci de (birinci sınıfı okuduğu için) çarpım tablosunu bilir.

Sonuç olarak diyebiliriz ki, bir sınıfta (öğrencilerin gittiği sınıf) öğrenilen bir bilgi, daha sonra sınıflarda da öğrenilmiş sayılarak yola devam edilir. Örneğin eğer, bugüne kadar öğrendiklerimiz temel alınmasa, herşeye baştan başlamamız gerekirdi.

Dikkat etmemiz gereken konu; nesne sınıfının sahip olduğu başka bir özellik varsa, bu özelliğin aynısının araç sınıfında olacağıdır. Bu olayı katısal (aileden kalan) bir hastalık gibi düşünebilirsiniz. Ebeveynde olan bir kalıtsal hastalık doğal olarak çocuklara geçecektir. Yine başka bir hatırlatma olarak, bir müdür, bir çalışanın bildiği bilgiyi doğal olarak bilecektir. Başka bir örnek olarak, bir otomobil modelinin bir sonraki sürümünü yapmak istediğimizi düşünelim. Bir sonraki model, en az bir önceki modelin sahip olduğu bütün özelliklere sahip olacaktır.

Tür	Özellik
Varlık	İsim özelliği
Nesne	Yukarıdan (bağlı olduğu sınıftan - Varlık) gelen İsim özelliği
Nesne	En / Boy / Yükseklik özelliği
Araç	Yukarıdan (bağlı olduğu sınıftan - Varlık) gelen isim özelliği
Araç	Yukarıdan (bağlı olduğu sınıftan - Nesne) gelen En / Boy / Yükseklik özelliği
Araç	Amaç (hangi amaçla kullanıldığı)

Figür 3.3 B: Türler ve özellikleri

Daha farklı – hierarşik bir şekilde gösterirsek;

Tür	İsim	En Boy Yükseklik	Amaç
Varlık	X		
Nesne	Y	X	
Araç	Y	Y	X

Figür 3.3 C: Türler ve özellikleri - 2

Yukarıdaki tabloda bulunan;

X işareti: bu özellik, bu sınıfta yeni tanımlanıyor anlamına gelmektedir.

Y işareti ise, bu özellik zaten bir üst sınıfın özelliğidir ve otomatik olarak o sınıfa dahil bir özelliktir.

Özellikler için olan bu ilişki; metodlar ve olaylar içinde geçerlidir. Örneğin, varlıkların “yok olma” olayı vardır ve bu olay o varlık ortadan kaldırıldığında (örneğin: yakarak), yok edildiğinde çağrılır. Nesneler, bununla birlikte, “kırıldı” olayına sahiptir. Araçlar ise, belirli bir kullanımdan sonra körelebilecekleri için “köreldi” olayına sahiptir diyebiliriz.

Tür	Yok olma	Kırılma	Körelme
Varlık	X		
Nesne	Y	X	
Araç	Y	Y	X

Figür 3.3 D: Türler ve olayları

Birde metod örneği verelim ancak bu sefer en üst sınıf olan varlıkta herhangi bir metod olmasın. Nesneleri yakabiliriz, araçları hem yakabilir, hem de, bozulduklarında, tamir edebiliriz.

Tür	Yak	Tamir Et
Nesne	X	
Araç	Y	X

Figür 3.3 E: Türler ve metod

Buradan çıkaracağımız sonuç; alt sınıflar üst sınıfların sahip olduğu bütün elemanlara sahiptir. Ancak, alt sınıfların sahip olduğu her eleman, üst sınıflarda yoktur. Aşağıdaki tabloda kolonlarda 1'den 8'e yazılı olan sayılar, sınıfların sahip olabilecekleri elemanları (özellikler, olaylar, metodlar) göstermektedir. Her bir satır ise, sınıfları göstermektedir.

	1	2	3	4	5	6	7	8
Sınıf 1								
Sınıf 2								
Sınıf 3								
Sınıf 4								
Sınıf 5								
Sınıf 6								

Figür 3.3 F: Sınıflar ve elemanları

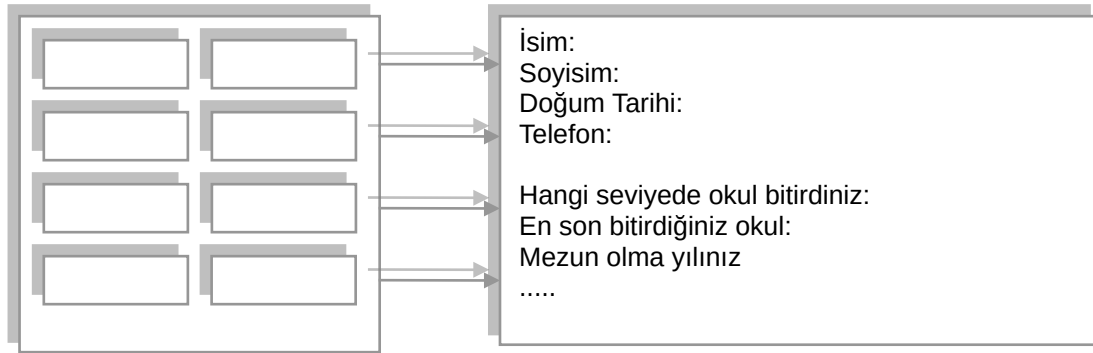
Siyah kutular, o sınıfın sahip olduğu yeni elemanı, yeşil kutular ise, o sınıfın, üst sınıfının sahip olduğu elemanları göstermektedir. Dikkat edersek, alt sınıf, üst sınıftan mutlaka daha fazla elemana sahip olmalıdır. Eğer bir sınıf (A), diğer bir sınıfın (B) alt sınıfı ise, ve eğer bu sınıfların elemanları sayısı ve elemanları eşitse, o zaman bu iki sınıf aynı sınıftır ($A = B$). Çünkü iki sınıfı programsal olarak birbirinden ayıran, sahip olduğu elemanlardır. Dikkat edelim !!! Az önce belirtmiş olduğum, iki sınıfın eleman sayıları aynı ise, o sınıflar aynı demek değildir. İki sınıftan biri diğerinin alt sınıfı ise (yani o şekilde belirtilmiş ise) ve eleman sayıları aynı ise o zaman bu iki sınıf aynıdır.

Bir bilimadamı, bir sistemi alır, geliştirir, diğer sistemlerle birleştirir, yeni eklemeler yapar ve sonuçta o güne kadar yapılmamış ve önceki sistemlerden birazda olsa farklı bir teknik icat eder, yapılan icadı, “ben buldum” diye sunmaz. Sonuçta yapılacak olan icadın, bir önceki sistemlerden farklı olması gerekmektedir. Aynı şekilde, yeni bir sınıf tanımlaması yaparken, eğer bir sınıf, önceki sınıflardan farklı ise, farklı özelliklere sahip ise, o yeni bir sınıftır diyebiliriz.

3.4 Object - Class (Nesne - Sınıf)

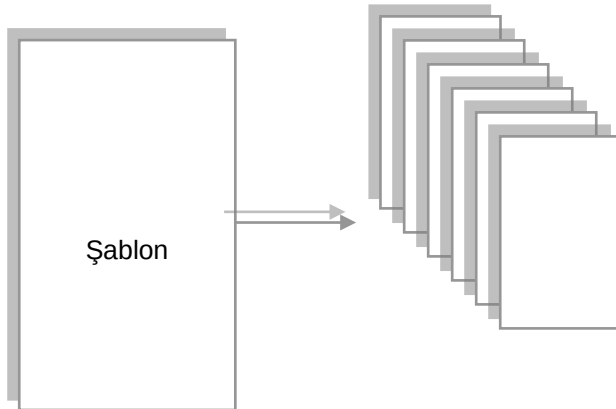
Daha önce sınıfın bir özet veri türü olduğunu, bir veya birden fazla değişkenin (özellik), metodlarının, olaylarının belirlenerek dış dünyadaki bir sistemin işlevlerini gerçekleştirdiğini söylemiştik.

Ancak, burada çok önemli bir nokta bulunuyor. Sınıf, bir şablondur. Yani, ismi ile belirtilen sistemin şablonunu tutan bir iskelettir. Bu iskeleti bir form gibi düşünebiliriz. Örneğin bizden, bir anket formu oluşturmamız isteniyor. Çizerek tasarlayacağımız anket formu, bir şablondur, yani sınıftır.



Figür 3.4 A: Bir form örneği

Hazırlanan bu anket formu, bir şablon olarak tasarlanır, ve bir tane örnek yazdırılır. Daha sonra, kaç kişiyle bu anket formu doldurulacaksa, o kadar adet çoğaltılır. Ancak şablon 1 tanedir. Çoğaltılan formların hepsi aynıdır ve şablondaki formatı kullanırlar.



Figür 3.4 B: Şablon ve Kopya örneği

Başka bir örnek vereyim. Varsayalım ki, bir otomobil firması, yeni bir model otomobil üretmek için, makineler üretti (veya satın aldı). Daha sonra makineleri, çizilen otomobil şablonunu üretecek hale getirdi.

Elimizdeki, otomobil çizimi, şablondur. Yani sınıftır. Daha sonra, o sınıf kullanarak, o şablona ait olan, otomobillerden ürettiğimiz her bir tanesi, bir NESNE'dir. Yani, sınıflar, şablon, nesneler ise, o şablonlardan türemiş varlıklardır. Örneğin, "insan" bir sınıftır. Albert Einstein, Isaac Newton veya herhangi bir insan ise o sınıftan türemiş varlıklardır, yani

programlama dilinde NESNE'dir. Nesnenin orjinal ismi "Object"tir.

Yani, biz önce, kalıpları - şablonları hazırlarız, daha sonra, o kalıplardan, o kalıplara uyan örnekler, yani nesneler oluştururuz. Buradaki fark çok açıktır ve şöyle ifade edilebilir.

Bir sınıfın özelliklerine, değerler atanamaz. Ancak ve ancak, o sınıftan türetilmiş bir örneğe yani nesneye özellik ataması yapılır. Sınıflar gerçek değildir!!! Örneğin, dünyaya, "İNSAN" ismiyle geçen bir varlık yoktur. Ancak, o varlıktan türemiş, oluşturulmuş nesneler vardır.

Sınıflar olmadan, gerçekler oluşturulamaz. Ancak, gerçekleri oluşturulmayan sınıflarda anlamsızdır. Bu konuyu, aynı beyin ile el arasındaki ilişki gibi düşünebiliriz. Beyni sınıf, elleri de nesne olarak görelim. Önce, bir işi beynimizde planlarız, şablonunu çıkarırız (işte bu sınıftır). Ancak, eğer onu ellerimizi kullanarak gerçekleştirmezsek, bir anlamı yoktur. Beynimizde bir kere tasarladığımız sınıfı, ellerimizle, binlerce kere gerçekleştirebiliriz (işte bu da nesnedir).

Aynı felsefedeki gibi, eğer bir kavramı düşünebiliyorsak, kesinlikle vardır. Çünkü, var olmasından dolayı, biz bu şekilde bir düşünce sahibi olabiliriz. İşte, bir nesnenin var olabilmesi içinde mutlaka onun tanımlanacağı bir sınıf olması gerekmektedir.

Şu âna kadar anlatılmış olan nesne – sınıf yapısı ve ayrıntıları OOP'nin temelini oluşturur. OOP temel olarak aşağıdaki elemanlardan oluşur.

Super class - Sub Class

Sub class, alt sınıf, super class ise, üst sınıf demektir. İki durum hariç her sınıfın üst ve alt sınıfları bulunmaktadır. Birinci durum, eğer sınıf, **en üst** sınıfsa, o sınıfın bir üst sınıfı zaten olamaz. İkinci durum; eğer sınıf **en alt** sınıfsa, yani daha altında başka bir sınıf yoksa, zaten alt sınıfı olamaz.

Sonuç olarak elde edebileceğimiz tanımlar;

Sınıf

Bir varlığın özelliklerini ve davranışlarını tanımlayan bir tür kalıptır. Başka bir deyimle; bir varlığın soyut özellik ve davranışlarını tanımlayan bir tür kalıptır.

Örneğin 'Araç' sınıfının, tüm araçlarda ortak olarak görülen özellikleri (yani bütün araçlarda olan özelliklerden olan tür, renk, vb), ve davranışları (sürmek, durmak...) içermektedir.

Sınıfa ait özellikler (attributes) ile davranışları tanımlayan yöntemlere (methods) topluca sınıf üyeleri (members) denir.

Nesne/Varlık

Bir sınıfa ait olan; bir varlığı, soyut değilde somut olarak; oluşturulan bir nesneyi ifade etmektedir. Örneğin, "Albert Einstein", "insan" sınıfına ait bir varlıktır. Bu varlık, insan sınıfının sahip olduğu bütün özelliklere sahiptir. Örneğin, isim, yaş, kilo, milliyet.... Aynı şekilde, bir nesne, sınıfında tanımlanan bütün metotları yani yöntemleri barındırır.

Yöntem (method)	Bir nesnenin uygulayabileceği işlemlere verilen addır. Herbir yöntem, nesnenin elemanı olduğu sınıfa ait tanımlanmış işlemi gerçekleştirebilir. Örneğin, İnsan sınıfına ait bir metot olduğunu varsaydığımız “koş” metodu, bu sınıfa ait bütün nesneler tarafından gerçekleştirilebilir
Kalıtım	Sınıflar arası organizasyonel yani hiyerarşik ilişkiyi tanımlamak için kullanılır. Daha önceki kısımlarda, alt sınıf üst sınıf hakkında bilgiler verilmişti. Birbirinden türeyen sınıfları tanımlamak için kullanılan bir ifadedir.
Sarma	<p>Bir sınıfın içeriğinde bulunan üyelerinin, erişim hakkını ifade etmektedir. Sınıfın içinde bulunan özellikler ve yöntemlerin, sınıf içinden, sınıf dışından veya alt sınıflardan erişilip yani kullanılıp kullanılmayacağını ifade etmektedir. İngilizcesi encapsulation (enkapsilasyon)’dur.</p> <p>Eğer, bir sınıfa ait üyenin, dışarı başka bir sınıf tarafından erişilebilmesine olanak sağlamak istiyorsak, tanımlayacağımız ifadeyi, public anahtar sözcüğünü kullanarak tanımlarız. Public, dışarıdan erişilebilen anlamına gelir.</p> <p>Eğer, bir sınıfa ait üyenin, dışarıdan değilde, sadece o sınıf içinden tanımlı yöntemler ve yordamlar tarafından erişilebilmesini istiyorsak, bu üyeyi, private anahtar sözcüğü kullanarak tanımlarız. Private, özel anlamına gelmektedir.</p> <p>Eğer, bir sınıfa ait üyenin, alt sınıflar tarafından ve tanımlandığı sınıflar içinden kullanılmasını istiyorsa, bu tanımlı, protected anahtar sözcüğü kullanarak yaparız.</p>

ÇEVİRME

Hatırlatma_1’e uyarlama yapacak olursak; gerçek hayatta nesne ve tür olarak ifade ettiğimiz, veya bir türe ait olan nesneleri tanımlamak için, programlamadaki, nesne ve sınıf ifadelerini kullanırız.

Bir **öğrencinin** kaydını yaptırmalıyız (buradaki öğrenci bir nesneye denk geliyor) Bütün **öğrencilerin** bu işlemi yapmasına izin ver (buradaki öğrenci, öğrenci sınıfına denk gelmektedir)

Bir sınıfı tasarlarlarken, yukarıda bahsettiğimiz gibi, sınıfın ismini, nereden türediğini (üst sınıfını), (gerekli) özelliklerini, olaylarını ve metotlarını da belirtmeliyiz.

Çeviri 7 : Sınıf ve elemanları

[Ara verme bölümü]

Üçüncü bölümde kullanılmış olan terimlerin İngilizceleri

beyin : brain	metot, yöntem : method
---------------	------------------------

sınıf : class nesne : object varlık : existence özü, soyut, özet : abstract özellik : property - attribute olay : event	miras - kalıtım : inheritance ağaç : tree dal : node özel : private genel : public
--	--

SONUÇ

Gerçek dünya ile bilgisayar dünya arasında bu kadar çok benzerlik olmasından ötürü, programlama yapmak bizim için çok ilginç olmayacaktır. Bir bakıma, Yaratıcı, evreni varlıklar ve sınıflar üzerine programlamıştır ve biz yaratılanlar olarak, sanal ortam içinde, gerçek dünyadakilerden esinlenerek, programlama yaparız.

Gerçek bir sınavda kopya çekme olanağımız yoktur. Daha doğrusu kopya çekme izniniz yoktur. Ancak programlamada gerçek dünyada olan sistemleri bilgisayar içine yani sanal dünyaya aktardığımız için önümüzde daima kopya çekebileceğimiz bir kaynak bulunmaktadır ve kopya çekmek serbesttir.

Programlama; bu açıdan bakıldığında, felsefe ile de alakalıdır. Hatta, hemen hemen her sistemi kopyalayıp bilgisayar ortamı içine aktarabildiğimize göre, bütün bilimlerle alakalıdır diyebiliriz.

4 Bilgisayar Sistemleri

İçerik

- 4 Bilgisayar Sistemleri
- 4.1 Bilgisayar Sistemleri
- 4.2 İşlemci
- 4.3 Programın hafızaya yazılması
- 4.4 Programlama yapısı
- 4.5 İşletim Sistemleri
- 4.6 Mimariler
- 4.7 Programlama dili kullanımı

Amaçlar

Dördüncü bölüm,

- Bilgisayar temel elektronik ve mantıksal sistemlerini,
- İşlemci, hafıza ve sabit disk gibi temel donanımları,
- Programlama dillerinin temel yapısını anlatmaktadır.

Anahtar sözcükler

anakart, sabit disk, bilgisayar sistemleri, işlemci, dil, derleyici, platform, mimari

4.1 Bilgisayar sistemleri

Bilgisayar sistemleri, yazılım ile donanımın mükemmel uyumunun üzerine kurulmuştur. Programlama yapmaktaki amacımız, bilgisayarın donanımının anlayacağı, talimatlara dönüşecek kodları yazmaktır. Yani yazdığımız programın kodları, bilgisayar donanımlarının anlayacağı bir talimat listesine dönüşür ve bilgisayar donanımlarını bu şekilde kontrol eder.

Beynimizde aslında aynı şekilde çalışmaktadır. Biz, bir kolumuzu hareket ettirmek istediğimizde, beynimizde, gerekli miktarda Potasyumu ve/veya Sodyumu, nöronların emriyle kimyasal elektriğe çevrilir yani bir çeşit belirli voltajda sinyale dönüşür, sonrasında sinir sistemi yoluyla, gerekli kaslara iletilir ve kasların belirtilen miktarda kasılmasını veya gevşemesini sağlar. (Elektrik çarptığında da kaslarımızın kasılması bu nedenledir. Yüksek voltajdaki elektrik, bütün kasları hareket ettirir.). Aynı şekilde, bu elektriksel sinyaller belirli manyetik alan oluşturmaktadırlar. Hepimizin televizyonlarda; film veya dizilerde gördüğümüz Kalp grafiği makinesini hatırlayalım. Tıpta ismi EKG (elektro kartiyo grafi) olarak geçen bu araç, kalp atışlarının grafiğini, kalbin oluştuğu sinyalleri tespit ederek grafik çizmektedir.



Figür 4.1 A: Bir EKG Makinesi

Aslında bizim için kolay görünen bu evrenin bu denli karışık olması şaşırtıcıdır. Aynı şekilde, bilgisayardaki en basit bir işlemde de aslında binlerce – milyonlarca elektriksel sinyalin bilgisayar donanımını kontrol etmesiyle mümkün olur.

Bilgisayar üzerinde, IO işlemleri yapılır. **IO, Input - Output** yani **girdi - çıktı** anlamına gelir. Yazdığımız hemen hemen bütün programlar, Girdi ve Çıktı aygıtlarıyla iletişim kurarak, o aygıtlar üzerinde işlemler yaparlar. Aynı şekilde beyinde, kırmızı (kendimiz tarafından hareket ettirilebilen kaslar : el – parmak – bacak gibi) ve beyaz (kendimiz tarafından değil de sadece beyin tarafından kontrol edilebilen kaslar : herhangi bir organ kası, kalp gibi) kaslarıyla iletişim kurarak, bu kasları hareket ettirir.

İşlemlerin bilgisayarda nasıl gerçekleştiğini görmeden önce, bilgisayar için önemli olan bazı bilgisayar parçalarını kısaca tanıyalım.



Figür 4.1 B : Anakart

Anakart

Anakart (mainboard), oldukça karmaşık bir yapıya sahip olan, bir bilgisayarın hemen hemen tüm parçalarını üzerinde barındıran ve bu parçalar arasındaki iletişimi sağlayan elektronik devredir. İnsan vücuduna benzeticek olursak, anakart, sinir sistemimizdir. Kulağı, eli, gözü olmayan bir insan yaşayabilmekte, ancak sinir sistemi olmayan bir insan yaşayamamaktadır. Aynı şekilde, anakart olmaksızın bir bilgisayarın çalışması imkansızdır. Anakartı bir telefon şebekesi gibi düşünebiliriz. Üstündeki bazıları beyaz ve bazıları siyah olan ince – uzun soketlere, **slot** denir (birde orta bölümde, kareye benzer bir şekilde işlemci için slot bulunmaktadır). Bu slotlara çeşitli bilgisayar donanımları yerleştirilir. Anakart, daha sonra bu aygıtların arasında konuşmasını (iletişim kurmasını) sağlamaktadır.



Figür 4.1 C: Ekran kartı, ses kartı, tv kartı

Hafıza ve sabit diskten daha önce değişkenler bölümünde bahsetmiştim. Kısa bir hatırlatma yapayım. Sabit disk verileri, uzun süreli olarak saklayan (elektrik kesilse bile), çok fazla büyük boyutlarda verileri tutabilen ve hafızaya göre yavaş çalışabilen bir tür depodur. Yani siz bir dosyayı kayıt ettiğinizde, o dosya sabit diske kayıt edilir. Hafıza ise, o an işlenmekte olan programları ve dosyaları, geçici olarak, tutan (sadece işlendiği süre boyunca) ve çok hızlı çalışan bir aygıttır. Yani siz bir programı açtığınızda, o programın bir kopyası hafızaya alınır ve oradan işlem görür.

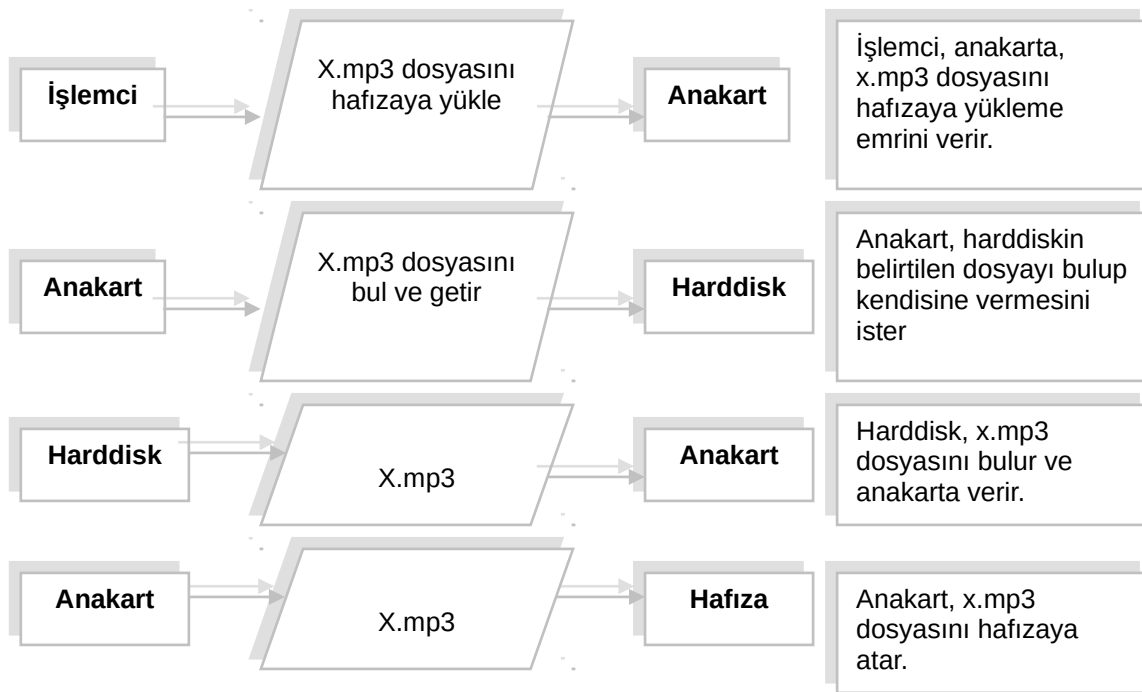
Bunun haricinde, ekran kartı: monitörünüzün bağlandığı ve görüntü işlemlerini yaparak size sunan, ses kartı : hoparlörünüzün bağlandığı ve ses işlemlerini yapan aygıtlardır.

Son olarak ve en önemlisi, işlemci: bilgisayarın beynidir ve bilgisayarın gerçekleştirdiği işlemlere temel bir altyapı oluşturan, bütün işlemleri koordine eden ve yönlendiren parçasıdır (ayrıntılı bilgi ileriki bölümde).

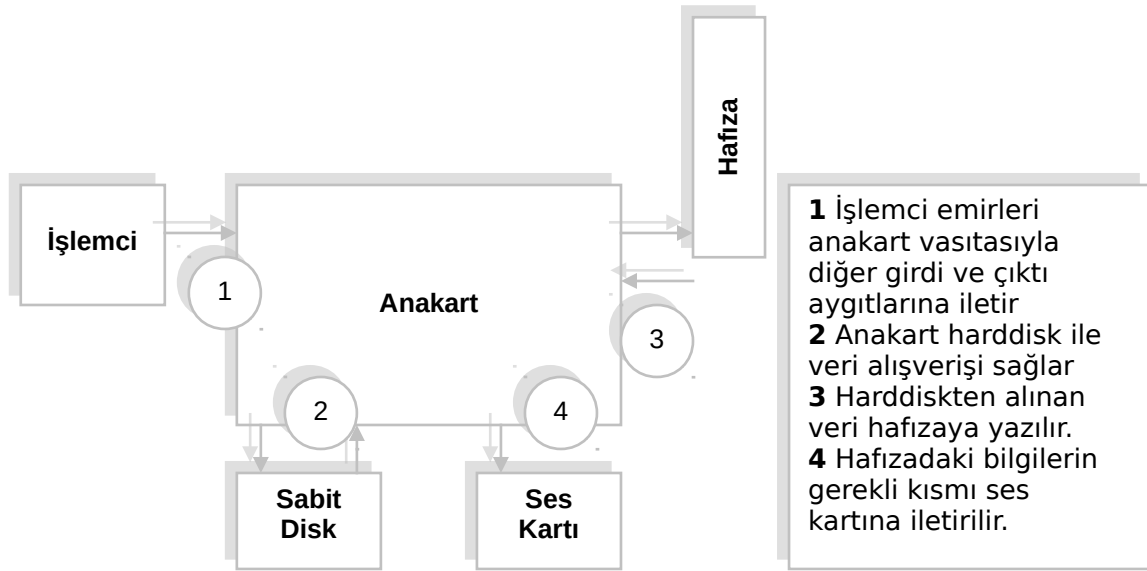
Peki bu aygıtlar nasıl çalışıyor? Örneğin, sizin bir şarkıyı dinlemeniz için,

- öncelikle, işlemci, şarkı dosyasını çalışacak olan programın (winamp / windows media player / real player), sabit diskten bulunup, hafızaya atılmasını emreder,
- bu emir anakarta iletilir
- anakart bu emri, sabit diske iletir,
- sabit disk, istenilen programı bulur ve anakarta verir,
- anakart bu programı hafızaya atar,
- daha sonra, işlemci harddiskten, o şarkı dosyasının hafızaya alınmasını emreder,
- bu emir, anakarta gider, sabitdiske emiri iletir ve o dosya, harddiskten bulunup, anakarta iletilir,
- anakart bu dosyayı, hafızaya atar,
- daha sonra, işlemci, programı çalıştırır,
- program ise, dosya içindeki veriyi okur (Hafızadan) ve ses kartına, istenilen şarkıyı duyuracak melodileri - sesleri çıkarmasını emreder,
- bu emir, yine anakart aracılığıyla ses kartına iletilir.

(aslında bu işlem biraz daha farklı gerçekleşmektedir ancak, okuyucuların daha kolay anlayabilmesi için bu yöntem seçilmiştir)



Figür 4.1 D: Anakart, donanımlar ve işlemcinin çalışma yapısına bir örnek



Figür 4.1 E: Anakart, donanımlar ve işlemcinin çalışma yapısına bir örnek

- 2

Tabi ki bu işlemlerde başka elemanlar da görev alır, ayrıca farklı mimarilerde bazı değişiklikler olabilir. Ancak burada belirtilmek istenen sistemin çalışma yapısıdır.

Bit (parça -ısırık) Bilgisayardaki en küçük veri formatı bit'tir(hayvan olan bit ile karıştırmayalım). Bit ya 1, yada 0 değerini alabilen bir veri formatıdır. Ünlü matematikçi, George Boole, yaptığı çalışmalar sonucunda, bilgisayara ya low (düşük - 0), ya da high (yüksek - 1) değeri gönderilerek, yada bir dizi bu düşük-yüksek değerler gönderilerek işlem yaptırılabilindiğini keşfetmiştir. Doğal olarak, 2 farklı değer bulunabileceği sayı düzlemi, 2'lik sayı düzlemidir. Genellikle, tek başına bit bir anlam ifade etmez, bunun yerine bir dizi bit kullanılır. Bit'teki değerin 1 olması, o değeri tutan donanımın içinde elektriksel verinin olması anlamına gelir. Aynı şekilde, değerin 0 olması, o değeri tutan donanım içindeki elektriksel verinin olmaması anlamına gelir. Yani bir lambanın yanması 1, yanmaması için 0 değeri kullanılır.

Byte 8 Bitlik veriye, 1 byte adı verilir. Byte anlam ifade eden en küçük veri formatıdır. Bitlerden oluştuğu için 2lik sayı formatındadır (tabanıdır).

H	G	F	E	D	C	B	A
---	---	---	---	---	---	---	---

Figür 4.1 F: 8 bitlik veri (her harfin olduğu bloğa bir veya sıfır gelebilir)

Bilindiği gibi, sayıların basamak değerleri sağdan sola doğru artar. Yukarıdaki tabloda, her bir harf bir biti gösterir ve herbirinin değeri ya 1'dir, ya da 0'dır. Onluk sayı tabanında, 8 haneli bir sayının en büyük değeri **99999999'dur**. İkilik düzlemde ise, **11111111'dir**. İkilik sayı düzleminde, en büyük değer (8 haneli), onluk sayı düzleminde ifade edilirse, 255 olur.

Nasıl yazı yazarken harf-harf yazıyor isek, bilgisayarda da işlemler, byte-byte yapılır.

Byte'lar, programlamanın ve bilgisayar sistemlerinin mimarisini oluşturur.

Ek Konu: Haberleşme

Bilgisayar ile aygıtlar yani fare, klayve, monitor gibi girdi çıktı aygıtları arasında 3 çeşit haberleşme bulunmaktadır: Interrupt, Polling, DMA

Interrupt Aslında bu yöntem, daha çok robotlarla uğraşanlar tarafından kullanılan sistemlerde sıklıkla görülür. Interrupt, kesme anlamına gelir. Yani bir işlemi geçici olarak iptal etme, kesme anlamındadır.

Aygıttan okunan değer hafıza da (sabit ve işlemci tarafından bilinen) bir yere yazılır (Bu bütün yöntemler için aynıdır).

İşlemci açılırken bir dilde, Assembly veya C (genelde assembly dir ancak bazı işlemciler direk c dilini okuyabilmektedir.), bir tür interrupt rutini yazılır ve bu interrupt rutini, hafızanın o noktasında bir veri değişmesi olduğunda tetiklenir.

İşlemci bu tetiği aldığı anda son yaptığı işlemi Stack Point (yığın noktası) denilen, hafızada bir noktaya yazar, işlemcinin son durumunu da aynı şekilde kayıt edilir. Sonra ilgili interrupt rutini yani prosedürü çağırılır. İşlem bitince, kayıt edilen nokta ve durum geri yüklenir, işlem devam eder.

Örnek olarak klayve bu şekilde çalışmaktadır. Yani siz klayvede herhangi bir tuşa bastığınızda, işlemci o an yapıyor olduğu işlemi durdurur, daha sonra (daha önceden işletim sistemi tarafından atanmış olan) klayveden bir tuşa basıldığında çalıştırılması gerekli olan prosedürü çağırır yani işlemi gerçekleştirir, sonrasında yine kaldığı yerden işleme devam eder. Ancak bu olayların hepsi inanılmaz derecede hızlı yaşanır ve siz klayve tuşuna bastığınız herhangi bir bekleme yaşamazsınız.

DMA Direct Memory Access (Doğrudan Hafızaya erişim): Aslında burda da bir tür thread bulunmaktadır. (örneğin) Ekran kartı, direk olarak hafızada ilgili bölümden verileri okur, kendi işlemcisi üzerinde (yani bu işlem ayrı bir thread gibi işlenir) işler! daha sonra monitöre yazdırır. Bazen bilgisayar tıkanınca (aslında çok oluyor) ekranda değişiklik olmaz. Bunun nedeni, işlemcinin, görüntülemeadaki değişiklikleri, hafızaya yazamamasından kaynaklanmaktadır. Çünkü işlemci o an başka bir işle uğraşmakta ve bu işleme sıra gelmemektedir. İşte bu yüzden hafızadaki görüntü ile ilgili bölümde değişiklik olmaz. Sonuç olarak ekran kartı da aynı veriyi aldığı için bir değişiklik görünmez.

Polling Polling ise, bir aygıtlarla iletişimi sağlamak için sürekli olarak çalışan ve her zaman aralığında (duruma göre salise, bazen daha kısa aralık) kontrol ederek, “bu aygıt bir mesaj göndermiş mi?” ve “bu aygıt bir mesaj gönderilmiş mi?” sorularını sormaktadır. Eğer bu soruların birinden “evet” cevabı gelirse, bu mesaj gerekli bölüme iletilir. Örnek

olarak mouse bu şekilde çalışır. İşlemci sürekli olarak, “mouse’un koordinatları ne” diye soru sorar ve o sorunun cevabına göre, işletim sistemine gerekli bilgileri gönderir.

[Ara verme bölümü]

4.2 İşlemci

İşlemci Merkezi işlem birimi (MİB veya CPU) bir bilgisayarın en önemli parçasıdır. Çalıştırılmakta olan yazılımın içinde bulunan komutları işler.

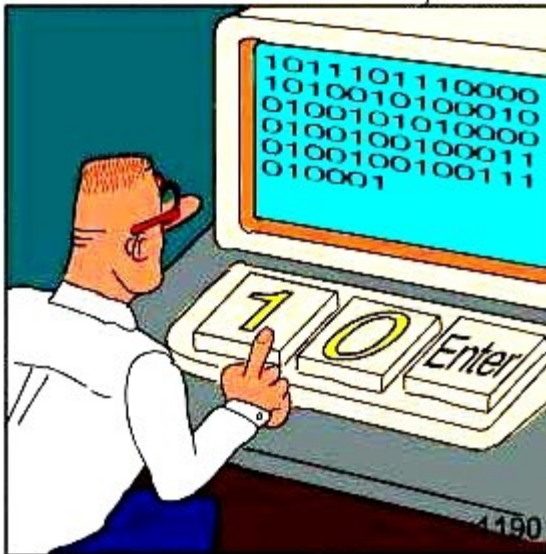
Cpu (Merkezi işlem birimi), veya basitçe işlemci, dijital bilgisayarların veri işleyen ve program komutlarını gerçekleştiren bölümüdür. İşlemciler, ana depolama ve giriş/çıkış işlemleri ile birlikte bilgisayarların en gerekli parçaları arasında yer alır. [e17]

Mikro işlemci Mikroişlemci (ya da CPU) (Central Processing Unit, Merkezi İşlem Birimi) çevresel birimlerden gelen sayısal verileri işleyerek gelen kodların yapısına göre sanal ortama ya da diğer çevresel birimlere uyarlayan işlevsel ve programlanabilir bilgi ünitesi.[e18]

Hepimizi, bilgisayar alırken, ilk ilgilendiren donanım: CPU (İşlemci)'dur. CPU; Central Processing Unit yani, **merkezi işlem birimi** anlamına gelmektedir. Daha önceki bölümlerde, işlemlerin, önce hafızaya daha sonra hafızadan işlemci üzerine alınarak işlendiğini söylemiştik. Bilgisayarlarda, iki, birbirleriyle iletişimli sistem bulunmaktadır: Hardware (Donanım), Software (Yazılım). Bilgisayar sistemleri yazılım (program) ve donanımların (aygıtlar) mükemmel uyumu ile büyük başarı elde ederler.

Donanım sadece elektronik aygıtlardan meydana gelen bir sistemdir. Yazılım ise, donanıma, binlerce farklı işlemi yaptırmak üzere tasarlanmış bir plandır. Bu iki sistemin aralarında haberleşmesi programlamaya başlayan herkes tarafından merak edilen bir konudur.

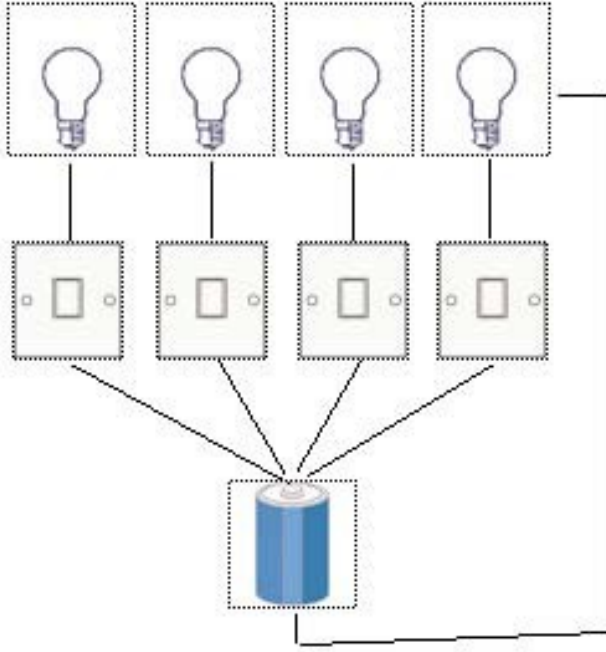
Yazılım, sadece elektriksel bir ifadedir. Daha önce, sadece bir seri 1 ve 0 dan oluştuğunu; öyle ki bu 1 ve 0'ın, elektriksel sinyaller olduğunu (ya elektrik var yada yok – ya düşük voltajda elektrik var, ya yüksek voltajda) söylemiştik.



Figür 4.2 A: Programlama, en sonunda 1-0 lara dönüşür.

Peki nasıl oluyorda, kodlayarak yazdığımız bir ifade (program - yazılım) elektriksel bir veri haline geliyor? Yani nasıl oluyorda, bir dizi kod satırı, elektriksel ifadeye dönüşüp,

işlemleri gerçekleştiriyor? Tabiki bu konuyu iyice kavramak için biraz elektroniğe girmemiz gerekiyor.



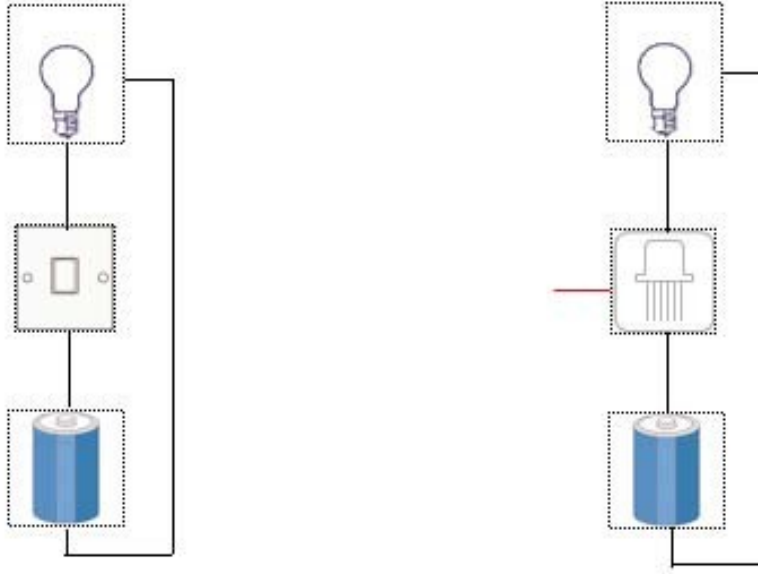
Figür 4.2 B: Basit bir devre örneği

Yukarıdaki şekilde, 4 lamba, ve bu 4 lambaya bağlı birer anahtar bulunmaktadır. Bu 4 lambanın diğer ucu (- eksi, alt kısımdaki) ise, pile bağlanmıştır. Tahmin edeceğimiz üzere, herhangi bir anahtarın açılması durumunda, o anahtarın bağlı olduğu lamba yanacaktır.

Şöyle düşünelim: Enerji, bilgisayar açma tuşuna bastığımız anda, bütün anahtarlara gelecektir. Açıp - Kapamak için parmağımızı kullandığımız anahtarlar yerine, birer transistör kullandığımızı düşünelim.

Transistör Transistör, küçük bir elektronik aygıt olup, devrede anahtar görevi görebilmektedir. Anahtarın açılıp kapanması ise, transistöre giriş yapan 2. girdi ile kontrol edilmektedir.

Yani, bir devreyi anahtarla kapatıp açmak yerine yine elektrikle çalışan bir aygıt olan, transistör kullanıyoruz.



Figür 4.2 C: Anahtarlı devre - Transistörlü devre

Soldaki devrede, bir anahtar, bir lamba ve bir pil bulunmaktadır. Pil ile lamba arasında, pilin negatif kutbundan (altından) lambaya direkt olarak bir bağlantı yapılmıştır. Pilin pozitif kutbu ile arasındaki bağlantı ise, bir anahtar ile kesilmiştir. Eğer, anahtar açılırsa, devre bağlanmış olur ve lamba yanar.

Sağdaki devrede, yine sistem aynı şekilde bağlanmıştır. Ancak, burada, transistörü **PARMAĞIMIZI** kullanarak açmak yerine, transistöre giriş yapan 2. girdiye elektrik veriyoruz (kırmızı). Eğer kırmızı girdide elektrik var ise, devre açılır ve lamba yanar.

Konuya devam etmek için bilgisayar açıldığında sırayla hangi işlemlerin yapıldığını bilmemiz gerekiyor.

İlk olarak, bilgisayarımızı çalışır duruma getirdiğimizde yani elektrik düğmesini açtığımızda, bilgisayarın bütün gerekli güç işlemlerini yapan güç kaynağı devreye girer ve voltajı – akımı kontrol eder. Sonrasında, eğer voltajda problem yoksa, işlemciye Power Good sinyali gönderir. Açılıştan Power Good'a kadar geçen süre genellikle 0.1 ve 0.5 saniye arasındadır. Sonrasında elektrik akımı anakarta yüklenir, bütün devreleri ve aygıtları çalıştırmak için gerekli olan 12, 5 ve 3 voltluk enerji hazır hale getirilir.

Bios Basic Input / Output System, yani basit (yada temel) giriş – çıkış sistemi, bir bilgisayar açılırken çalışan, değiştirilemeyen bir koddur. Bu kodun temel görevi, bilgisayara bağlı olan donanımların ve bilgisayar üzerinde çalışacak olan sistemlerin (örneğin işletim sistemi) çalışabilmesi için ortamı hazır hale getirmektir. BIOS EPROM çipi üzerinde yer alır.

ROM Read Only Memory, yani sadece okunabilir bellek anlamına gelen bir tür donanımdır. Bildiğimiz hafızaya benzer. Ancak üzerine kullanıcı veya programlar tarafından veri yazılamaz.

RAM Random Access Memory, Rasgele erişimli bellek anlamına gelir ve değişkenlerin, programların, açık (çalışmakta olan) dosyaların dinamik olarak depolandığı Hafıza adı verilen donanımın bir bölümüdür.

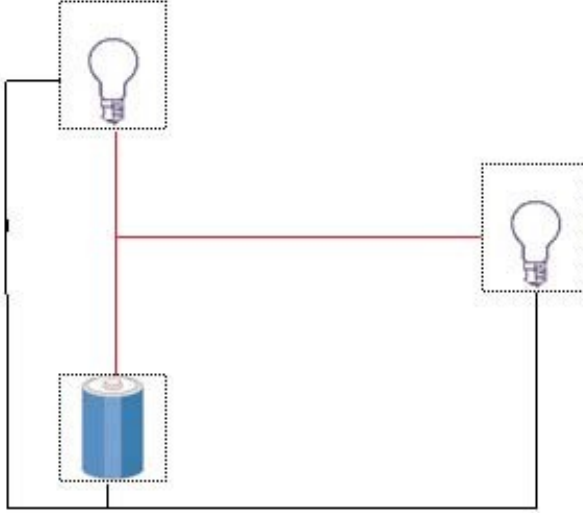
1 - Bundan sonra CPU, ROM Bios kodunu çalıştırmaya başlar (bu işleme bootstrap denir). Yani biostaki yazılı işlemler CPU üzerine yazılır (aktarılır). [Açıklama sonrasında yapılacaktır] *

2 - Bios'a yazılı olan işlemler içinde ilk olarak donanımların temel bir testi bulunmaktadır. Yani CPU, bu işlemten sonra, bütün donanımları bir testten geçirir. Bu teste "Power On Self Test" adı verilir ve işlemin ardından öncelikle ekran kartının durumu kontrol edilir ve RAM (bellek) hafızası denetlenir.

3 - Bütün bu testlerin ardından, işletim sisteminin sabitdiskten yüklenmesi için BIOS Master Boot Record (MRB) kısmı taranır (bir işletim sabitdiskin ilk önce okunacak bölümüdür). Bu bölümde, kurulu işletim sistemleri (Windows, Linux, Mac.. vb) ve sürücüler hakkında bilgiler bulunur.

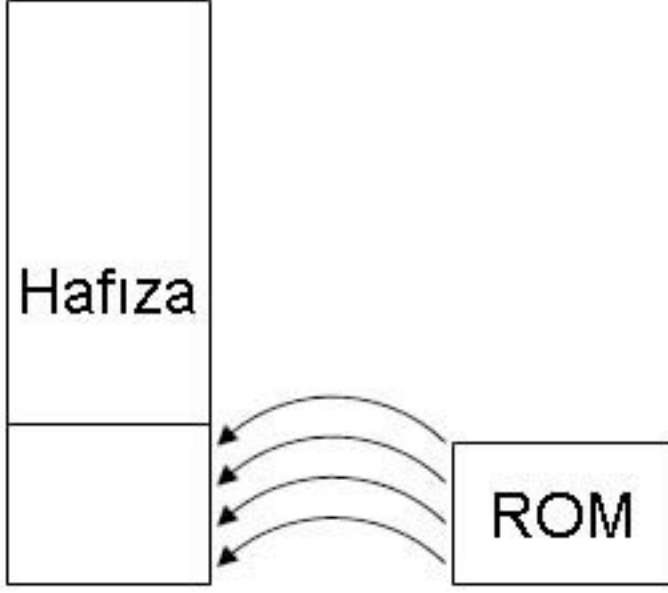
4 - Bu evreden sonra yüklenen işletim sistemi, işletim sisteminin açılışı için gerekli olan işlemleri gerçekleştirmeye başlar. En uzun süreç, bu işlemlerin gerçekleştirilmesinde geçer.

* Peki bu yazım işlemi nasıl gerçekleşir. Sadece basit bir lamba yakmak için tasarladığımız bir devre düşünelim. Eğer pilden, lambaya giden tellerden birine bağlı başka bir paralel tel daha eklersek, o teldede o yük bulunur ve eğer devre sağlanırsa, sağlanan devredeki lamba da yanar.



Figür 4.2 D: Seri devre örneği

Devrede, normalde, pil, üst lambaya bağlanmaktadır. Ancak, aralarındaki kırmızı tele, başka bir lamba bağlanırsa, doğal olarak, elektrik akımı o telden de geçecek ve diğer lambayı da yakacaktır. İşte, ROM üzerindeki hazır (daha önceden anakart üreticisi tarafından işlemci standartlarına göre hazırlanmış) verilerde aynı bu şekilde Hafızaya yazılır. Yani ROM ile hafıza arasında her bir veri için (öyle ki bu veriler bittir) bir tel olduğunu düşünebiliriz.



Figür 4.2 E: Hafıza - rom ilişkisi

Şekilde görüldüğü gibi, ROM'daki veriler, hafızanın belirli bir bölümüne, direkt olarak yazılır. Daha sonra, ROM, açılış prosedüründe ne yapılması gerektiğini bildiği için, işlemciye gerekli işlemleri yapması için emir verir (örneğin, kurulu işletim sistemini açmak gibi).



Figür 4.2 F: Hafıza bloğu

Hafıza yukarıdaki gibi, bir ızgaraya benzer. Her bir satırda ya veriler yada işlem emirleri bulunur. Hafızadaki satırlar bölünemez, satır satır işlenir.

Daha farklı bir örnek ile anlatayım: Eski otomatik (elektronik olmayan) çamaşır makinelerini düşünelim. Bu makinelerde,

- suyu deterjan bölümünden yükle
- suyu musluktan yükle
- su boşalt
- sağa döndür
- sola döndür
- hızlı döndür

işlemlerinin gerçekleştiğini düşünelim. Bu işlemleri yapan, (yani devreyi açıp

kapayan) anahtarlar olduğunu düşünelim. Aşağıdaki resim, eski bir çamaşır makinesinin panel üzerindeki, döndürülen ayar düğmesinin içinde kalan kısmıdır.



Figür 4.2 G: Eski çamaşır makinelerindeki ayar düğmesinin iç kısmı

Sol alttaki, kırmızı renk ile belirtilmiş bölümde küçük mandalların olduğunu görürüz. Bu çark sürekli belirli bir hızda dönmektedir. Dönme esnasında bu mandallar, yine resimde görmüş olduğunuz mavi çubuk üzerindeki anahtarları tetiklemekte, ve yukarıdaki listelenmiş işlemleri, belirtildiği sıraya göre yapmaktadırlar.

İşte işlemci de aynı şekilde çalışmakta, ancak mandal ve anahtarların yaptığı işlemleri de elektronik olarak yapmaktadır.

İşlemci işlemleri nasıl gerçekleştirir?

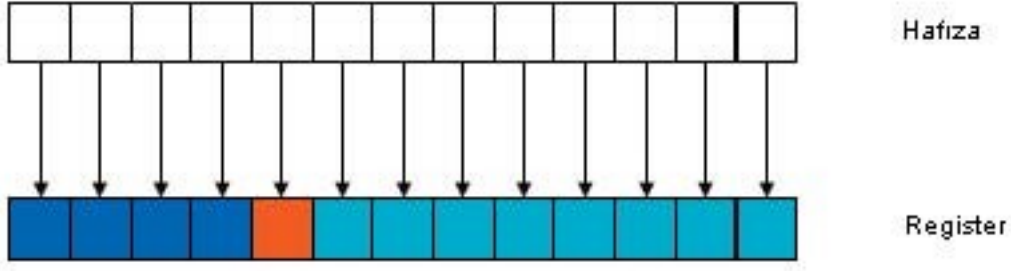
Önemli olan konu da zaten işlemcinin işlemleri nasıl gerçekleştirdiğidir. İşlemcinin yaptığı işlem sayısı iki yüzden azdır. İşlemcinin yapabildiği işlemlerin grupları:

Transfer(aktarma),
Aritmetik,
Mantık ve diğerlerdir.

Yani işlemci, genelde ya üzerine yazılan veriyi değiştirir, ya hafızadan okur-hafızaya yazar ya da aygıtlarla iletişimi sağlar. İşlemcinin anlayacağı İşlem denilen, aslında birkaç tane 1 veya 0 dan oluşan elektriksel veridir. Hafızaya alınan bu veri, her işlemcide farklı formatta işlenir.

Şuanki bilgisayarların çoğunda, 32 bit işlemci bulunmaktadır. Yani, bu işlemciler, 32 tane 1 veya 0'ın yan yana gelmesiyle oluşan bir dizi, emri tanımlayabilecek işlemcileri anlayabilmektedir. Ancak daha sonra, AMD işlemci firmasının 64 bit lik işlemci ürettiğini gördük. Bu işlemcide de 64 tane 1 veya 0 yan yana gelerek bir dizi oluştururlar.

İşlemci bu hafızaya atılan veriyi, tek tek, kendi üstündeki **register'lara** atar (aynı bilgisayarın açılışında, ROM'daki verilerin hafızaya alınması gibi). **Register'lar**, işlemcinin özel hafızasıdır. Normal hafızadan çok daha hızlı çalışmaktadır. Registerler üzerine alınan emir, belirli gruplara bölünür. Daha sonra, belirli bölümlerdeki transistörlere iletilirler. Bu veri bir dizi anahtar olacağı için belirli işlemleri gerçekleştirir.



Figür 4.2 H: İşlemci registerleri

Yukarıdaki şekilde, görüldüğü gibi, emirler, hafızadan register'lere alınır. Daha sonra belirli gruplara bölünür (her rengin bir grup olduğunu düşünebiliriz). Bu bölme işlemi ardından, işlemci tarafından belirlenen bir bloğu (örneğin mavileri) yapılacak işlemi, başka bir bloğu (örneğin yeşilleri) yapılacak işlemin argümanını temsil eder.

Örneğin, işlemcinin anlayabildiği bir komut olduğunu varsaydığımız, **sil** komutunu düşünelim. Bu komut SİL ADRES şeklinde ifade edilen bir komut olsun. **Sil**, komutun ismi, **Adres'te** hangi adresteki (hafızadaki adres) veriyi sileceği olsun. İlk 4 byte'a girilen veri, sil komutu anlamına gelsin ve sonraki byte'lara girilen veride, bu işlemin yapılacak olduğu adres olsun.

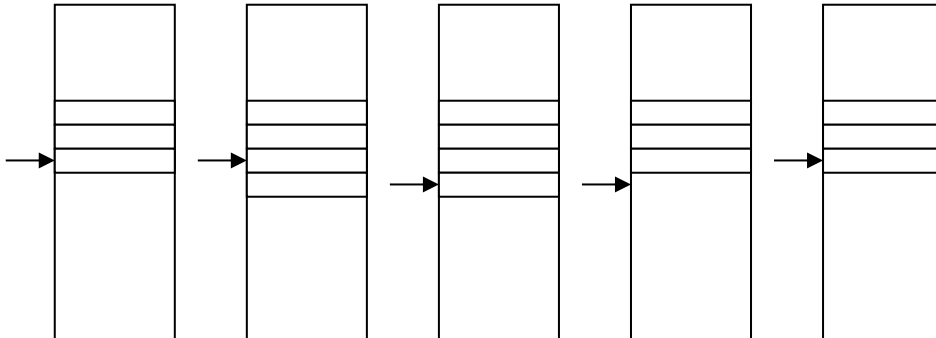
0010 : Sil
00110111 : Adress

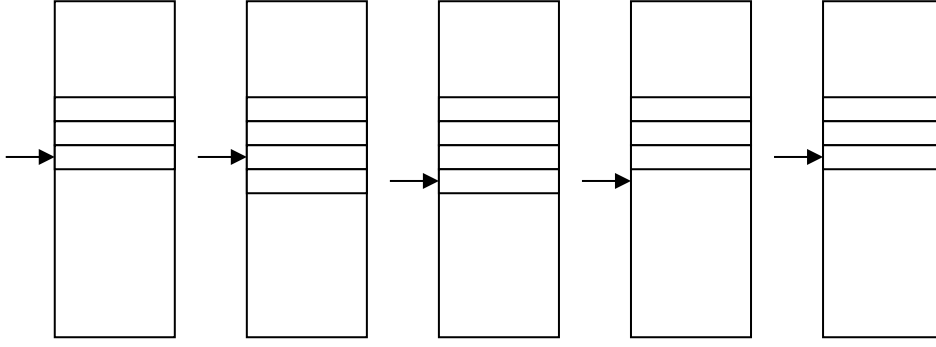
Sonuçta; eğer işlemciye gönderilen emir

001000110111
İşlem Argüman

şeklini alır

Tabiki bu işlemler, birçoğunuzun duyduğunda “saçma” olarak nitelendirebileceği bir şekilde gerçekleşmektedir. İşlemci, kendisine son verilen işlemi, gerçekleştirdikten sonra, başka bir işlemin eklenip eklenmediğini görmek için yine en son kaldığı yere döner, sonra eğer eklenmiş bir işlem var ise, o işlemi yapar ve daha sonra bittiğinde yine başladığı yere döner, eğer, yeni bir işlem eklenmemişse, YERİNDE sayar ve yeni bir işlemin eklenmesini bekler.





Figür 4.2 I: İşlem sırası

Yukardaki şekillerde, uzun dikey dikdörtgenler, hafızayı, yatay dikdörtgenler işlemleri, oklar ise, işlemcinin o an hangi işlemi yaptığını gösteriyor. İlk durumda, işlemci, için sonraki bir işlem bulunmadığı için yerinde sayıyor. Elektrik akımı sürekli olarak geldiği için, işlemci çok kısa aralıklarla, bir sonraki hafıza bloğuna bir işlemin eklenip eklenmediğini kontrol eder. Alarmlı bir saati düşünün, her bir dakika değiştiğinde, o vakite atanmış bir alarm var mı yok mu diye kontrol etmek zorundadır. İşte işlemcilerde, her “kısa süre” de yeni bir emir eklenmiş mi eklenmemiş mi kontrol etmektedirler.

İkinci durumda, yeni bir işlem hafızaya eklenir, üçüncü durumda ise, hafızayı yeni emir için kontrol eden işlemci yeni bir işlem eklendiğini görür ve o işlemi yapmak için bir sonraki hafıza bloğuna gider. İşlemi yaptıktan sonra işlemi yapıldığı ve bittiği için o işlemi hafızadan siler (4. durum). Daha sonra, yine kendi yerine döner ve yeni işlem gelmesini bekler.

Bir alarmlı saatte aynı şekilde çalışır. Her dakika değişiminde hafızasına yazılı olan alarm saatini kontrol eder, eğer o saate atanmış alarm var ise, alarmı çaldırır yok ise, sadece dakikayı artırır ve 1 dakika sonra yine aynı işlemi tekrarlar.

Yani bir program, bir ve sıfırlardan oluşan bir matristir.

1	0	0	1
0	1	1	0
1	0	1	0
0	1	0	1
0	0	1	1
0	1	0	0

Komut 1
Komut 2
Komut 3
Komut 4
Komut 5
Komut 6

Figür 4.2 J: Kod - Komut

Bu bir ve sıfırlar işlemcinin register'lerine yazıldıktan sonra gerekli işlemleri gerçekleştirmek üzere, işlemci içindeki sistemleri kontrol etmektedir.

Alıştırmalar

- İşlemci nedir? Nasıl çalışır?
- Transistör nedir?
- Bilgisayarda işlemler nasıl gerçekleşir?

- Sabit disk ile hafıza arasındaki fark nedir?
- RAM/ROM hakkında bilgi veriniz
- İşlemci hangi kategorilerde işlemler gerçekleştirir?

[Ara verme bölümü]

4.3 Programın hafızaya yazılması

Bir program çalıştığında, işlemi, programı otomatik olarak hafızaya yazar. Peki, bir program çalıştığında ilk yapacağı işlem nedir, yada, işlemci programın ilk başta neresini okur? Bu soruya benzer bir soruyu interneti öğrenirken sormuştum: “Eğer bir web sitesinde birden fazla sayfa var ise (yada birden fazla sayfa olabiliyorsa), bu sayfalardan hangisinin ana sayfa olduğu nasıl anlaşılıp ilk o sayfa yayınlanıyor?”. Daha sonra, ilk başta görüntülenecek olan sayfanın **index.html** olduğunu öğrendim.

Aynı şekilde, daha önceki konumuzda, bilgisayar açılırken, bir sabitdisk’in, Master Boot Record bölümünde yazılı olan işlemleri yaptığını söylemiştim.

Main İşte aynı şekilde her programda bir **main** prosedürü bulunmaktadır ve bu main (ana) prosedürü, herhangi bir program açıldığında ilk çağrılacak olan işlemdir. Bir program hafızaya atılırken, ilk başta main prosedürünün kodu hafızaya yazılır ve işlemci, programın ilk noktası olan mainden okumaya ve işlemeye başlar.

Program içinde yapılacak olan bütün işlemler, doğrudan veya dolaylı olarak main prosedürü tarafından çağrılmalıdır.

Main içine ne yazılır?

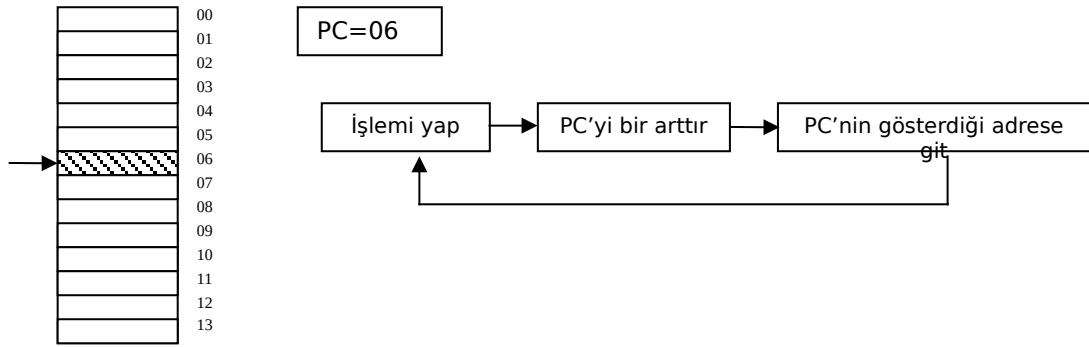
Birgün tanıdıklarımın birinin 6-7 yaşlarındaki çocuğu bir soru sormuştu: **“Acaba bir binayı yapmaya üstten(çatı) mı başlıyorlar, alttan(bodrum) mı?”** Tabi bu bizim için gülünç bir soru olsada, yerçekimi kanunlarını bilmeyen bir çocuğa göre mantıklı bir soru olabilir.

Bir binayı oluşturmaya en alttan başlarız. İşte o en alt, main prosedürüdür. Bir binanın en alt katında pabuç denilen ve binanın bütün üst katlarını taşımakta yükümlü olan kalıplar bulunur. Bu kalıplar olmadan veya yeterince sağlam olmadan binanın diğer katlarının yapılması imkansızdır.

İşte main, bir binanın temeli kadar önemli bir prosedürdür. Main prosedürü içine yazdıklarını yapıyor olduğu sürece program çalışır. Yani, main içine bir işlem yazdınız, program çalıştı, main yüklendi, main içine yazılmış olan işlem gerçekleştirildi. İşte bu işlem bittikten sonra program kapanır. Eğer main içinde yazılı bir işlem yoksa, program açıldığı gibi kapancaktır.

Bu işlemi, eline ne kitap verilirse verilsin okuyan birine benzetebiliriz. İşlemci de, o an hangi programın açılma emri verilirse, o programı açıp ilk satırından okumaya başlar ve içindeki talimatları yerine getirir.

Program counter (program sayacı) PC (program counter – program sayacının kısaltılmış hali), işlemcinin o an yapmakta olduğu işlemin adresini tutan bir değişkendir. Yani aktif yapılmakta olan işlem adresi (hafıza adresi) program counter’da tutulur ve, her işlem gerçekleştikten sonra, PC bir artırılır.



Figür 4.3 A : Program counter örneği

Böylece, işlemler, peşpeşe gerçekleştirilir. İşlemci belirli bir iş yok ise, o zaman pc'yi arttırmaz, yani işlemci kendi üstünde dönmüş olur ve işlem gerçekleştirmez. Aslında olaylar, biraz daha farklı gerçekleşmektedir. Anca, sizin öğrenmeniz için en uygun anlatım böyle olacaktır.

Program hafızası

Bir program açıldığında, işlemci o program için belirli bir bölüm hafızayı ayırır. Bu ayrılmış hafıza; iki bölümden oluşmaktadır.

İlk bölüm, daha program çalışmadan hafızaya yazılan ve program çalışıyor olduğu sürece hafızada kalacak bölüm olan **STATİK (STACK)** alandır. Bu alana sabit değişkenler (global) ve sabit değerler (constant), program sınıflarının temel yapısı ve benzeri statik yani değişmeyen veriler yazılır.

İkinci bölüm ise, programın çalışması esnasında, bir prosedür çağrıldığında, bir nesne oluşturulduğunda kullanılan alan olan **DİNAMİK (HEAP)** alandır. Bu alana atanmış olan değişkenler ise fonksiyon bittiğinde veya nesne yok olduğunda silinir.

Alıştırmalar

- PC (program counter) nedir ?
- Main nedir? Ne işe yarar?

4.4 Programlama yapısı

Low Level Programlama dilleri öncelikle High Level / Low Level (üst seviye / alt seviye) olarak ikiye ayrılır. Low level programlama dilleri, cpu'nun direkt olarak anlayacağı şekilde tasarlanmıştır. Komplike ve uzun işlemler bulunmamaktadır. En alt "low level" programlama dili Assembly'dir. Assembly, CPU ile direkt olarak konuşabilen tek dildir diyebiliriz. Her işlemcinin (intel - amd...), bazende her modelinin farklı assembly dili bulunmaktadır.

High Level High Level diller (Java, VB.Net, C++,), güncel hayatta en çok kullanılan ve programaların yapıldığı dillerdir. Günümüzde gördüğümüz birçok program, High Level yani yüksek katmanda bulunan bu dillerle yazılır.

High Level diller:

- mantıksal ifadeleri destekler
- dilde kullanım kolaylığı sağlar
- işlemlerin mümkün olduğunca kısa ve çabuk yapılmasını sağlar
- yazılan kodlardaki hataları bulup gösterirler
- bunun haricinde, yüksek katmandaki diller, birçok sistemle birlikte çalışmak için bir arabirim sağlarlar.

Bilgisayar, temel olarak sadece Assembly denilen dili algılayabilir. High Level dillerin hepsi yazılan kodları Assembly diline çevirir, ve ancak bu şekilde bilgisayar, yazılan programın ne ifade ettiğini, ne gibi emirler verdiğini anlayabilir.

Assembly kullanarak, günümüzde kullanılabilecek (örneğin bir stok programı) yazmaya kalksak, yüzbinlerce satır kod yazmamız, ve bunun geliştirilmesi için aylar hatta yıllar harcamamız gerekmektedir.

Tabi ki, projenin büyük ve sürenin uzun olmasından dolayı, proje tamamlandığında içinde binlerce hata ve giderilmesi gereken onlarca sorun oluşacaktır.

Burada önemli olan, assembly ile program yazmaya kalkarsak, en ufak bir uygulamanın bile, haftalar hatta aylar alabileceğidir. Çünkü, işlemleri, tek tek işlemciye anlatmamız gerekecektir. Low level diller genelde, daha high level dilleri tanımlamak için kullanılır. Yani, alt seviye dilleri, üst seviye dillerini tanımlar.

Üst seviye diller, daha okunabilir, mantık kurulabilir, daha kısa, ve daha komplike ifadeler içerebilir halde tasarlanmışlardır. Bu dillere örnek olarak, .Net, Delphi, Java, ASP, PHP dillerini verebiliriz. Üst seviye dillerde tanımlanan bir işlem, assembly'ye otomatik çevrilir. Böylece, biz kodcular, her bir işlem için binlerce satır kod yazmaktansa 1-2 satır ile istediğimizi tanımlayabiliriz.

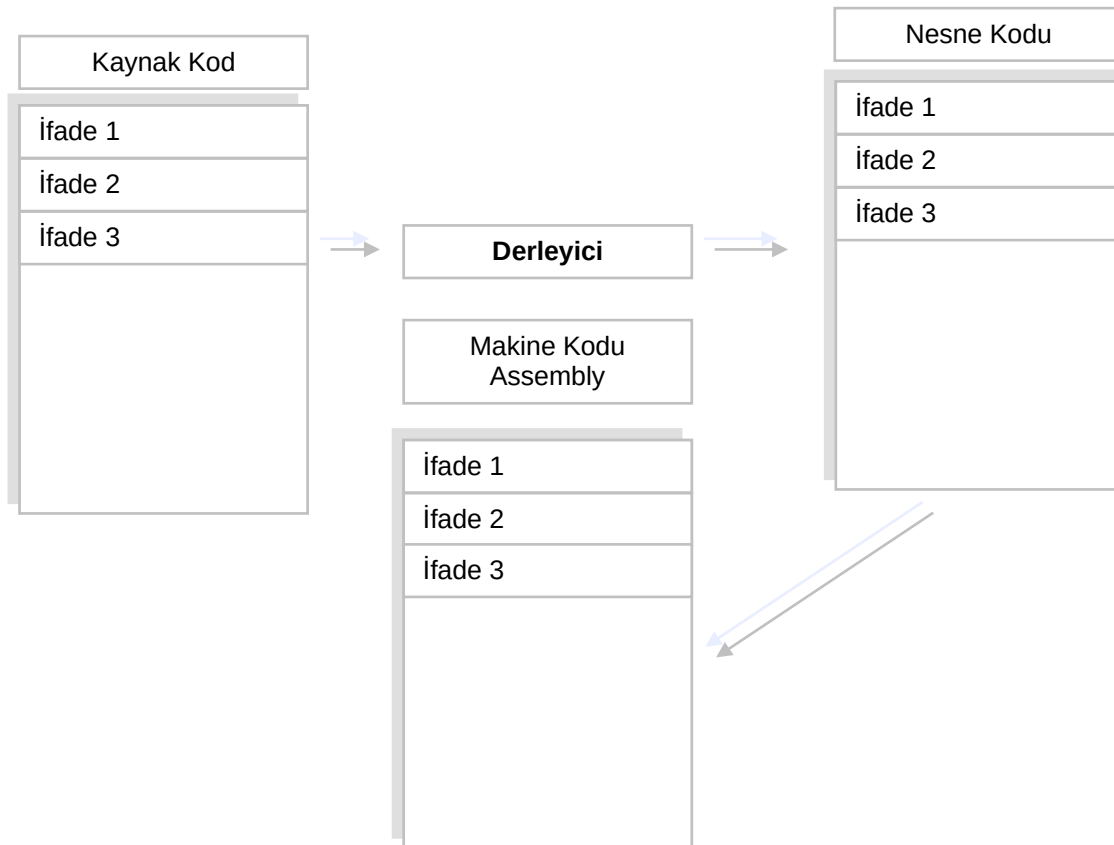
Assembly dili, bilgisayarın anlayabileceği tek dildir. Assembly dili içinde, sadece direk emirler bulunur, mantıksal ifadeler bulunmaz. İşlemler, sadece hafıza üzerinde, matematiksel olarak yapılabilir.

Assembly dili, temelde, 1 ve 0 lardan oluşur, daha doğrusu, Assembly dili, bir dizi 1 ve 0 a çevrilir. Daha açıkcası, Assembly'den yazılan bir komut, 1001100101110001 gibi sayısal bir ifadeye dönüştürülür.

Assembly dili bilgisayarda, elektriksel yük olarak ifade edilir. Yukarıda belirtildiği gibi, bir dizi 1 ve 0 larla elektriksel yükün olup olmadığı ifade edilir. Bilgisayar bu yüklerin değerlerine göre, microchip leri çalıştırır.

Compiler : Derleyici

Derleyici, üst seviyede yazılmış olan bir kodu, daha alt seviyelere çeviren bir programdır. Yani bizim üst seviye bir dilde yazdığımız kod, CPU tarafından direkt olarak anlaşılamayacağı için öncelikle bir program haline çevrilir. Bu program çalıştırıldığında, genelde assembly diline direkt olarak dönüşür ve emredilen işlemleri gerçekleştirmek üzere hafızaya yazılır. Yani insanların, yaptırmak istedikleri işlemler, öncelikle bir üst düzey dile, daha sonrada daha alt dil olan assembly'ye çevrilir. Sonuçta arada 2 farklı tercüman kullanılmış olunur.



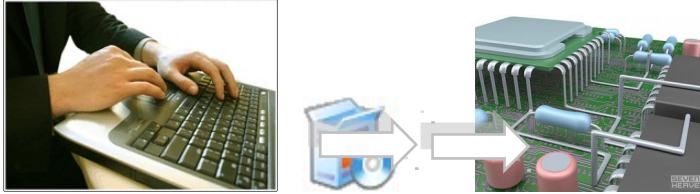
Figür 4.4 A : Derleyicinin çalışması

Daha önceki konularımızda, Kaynak koddan bahsetmiştik. Kaynak kod, dir kodcu tarafından görüldüğünde anlaşılabilir, ve üzerinde değişiklikler yapılabilir, ancak kullanıcılara gönderilmemesi gereken koddur. Nasıl Kola'ların formülü gizli tutuluyorsa, kaynak kodunda gizli tutulması gerekmektedir.

Nesne kodu Nesne kodu, 3. bölümde gördüğümüz sınıfların metodların ve diğer işlemlerin, karmaşık bir formatta tutulan, kaynak kodun derlenmesinden sonra, bilgisayarın anlayabileceği bir şekle dönüştürülmüş koda verilen addır. Çalıştırdığınız veya bilgisayarınıza kurduğunuz herhangi bir program, nesne kodu formatındadır. Üretici firmalar size programın çalışan halini gönderirler.

Makine kodu denilen ise, daha önce bahsettiğimiz, assembly dili komutlarıdır. Yani bir dizi 1 veya sıfırdan oluşan komutlardır.

Nesne kodu, çalışma anında, makine koduna çevrilir. Bu olayı şu şekilde düşünebiliriz: Bir derin dondurucu aldığımızı düşünelim. Derin dondurucunun nasıl yapıldığı, üretimi, mühendisliği o işin, kaynak kodudur. Ürün ise, nesne kodudur. Aldığınız ürünü çalıştırdığınızda işlev gerçekleştirmesi ise, o ürünün, makine kodudur (yani çalışma anında gerçekleştirdiği işlemlerdir).



Figür 4.4 B : Kaynak kod, Program (nesne kodu), Assembly kodu

Aşağıdaki tablonun ilk kolonunda bir programın kaynak kodu, ikinci kolonunda, aynı programın derleyici tarafından işlenmiş çıktısı yani nesne kodu bulunmaktadır.

<pre>#include <stdio.h> void main(int argc, char *argv[]) { int i, result; result = 1; for (i = 1; i<6; i++) { result = result * i; } printf ("Result is: %d.\n", result); }</pre>	<pre>.file "fact.c" gcc2_compiled.: .global .umul .section ".rodata" .align 8 .LLC0: .asciz "Result is: %d.\n" .section ".text" .align 4 .global main .type main,#function .proc 020 main: !#PROLOGUE# 0 save %sp, -120, %sp !#PROLOGUE# 1 st %i0, [%fp+68] st %i1, [%fp+72] mov 1, %o0 st %o0, [%fp-24] mov 1, %o0 st %o0, [%fp-20] .LL3: ld [%fp-20], %o0 cmp %o0, 5 ble .LL6 nop b .LL4 nop .LL6: ld [%fp-24], %o0 ld [%fp-20], %o1 call .umul, 0 nop st %o0, [%fp-24] .LL5: ld [%fp-20], %o0 add %o0, 1, %o1 st %o1, [%fp-20] b .LL3</pre>
---	---

	<pre> nop .LL4: sethi %hi(.LLC0), %o1 or %o1, %lo(.LLC0), %o0 ld [%fp-24], %o1 call printf, 0 nop .LL2: ret restore .LLfe1: .size main, .LLfe1-main .ident "GCC: (GNU) 2.95.2 19991024 (release)" </pre>
--	--

Yukarıdaki (sol kolon) kaynak koddan oluşturulmuş olan nesne kodunun (sağ kolon), makine kodu aşağıda verilmiştir.

```

00000000 7f45 4c46 0102 0100 0000 0000 0000 0000
00000020 0001 0002 0000 0001 0000 0000 0000 0000
00000040 0000 0234 0000 0000 0034 0000 0000 0028
00000060 0008 0001 002e 7368 7374 7274 6162 002e
00000100 7465 7874 002e 726f 6461 7461 002e 7379
00000120 6d74 6162 002e 7374 7274 6162 002e 7265
00000140 6c61 2e74 6578 7400 2e63 6f6d 6d65 6e74
00000160 0000 0000 9de3 bf88 f027 a044 f227 a048
00000200 9010 2001 d027 bfe8 9010 2001 d027 bfec
00000220 d007 bfec 80a2 2005 0480 0004 0100 0000
00000240 1080 000c 0100 0000 d007 bfe8 d207 bfec
00000260 4000 0000 0100 0000 d027 bfe8 d007 bfec
00000300 9202 2001 d227 bfec 10bf fff2 0100 0000
00000320 1300 0000 9012 6000 d207 bfe8 4000 0000
00000340 0100 0000 81c7 e008 81e8 0000 0000 0000
00000360 5265 7375 6c74 2069 733a 2020 2564 2e0a
00000400 0000 0000 0000 0001 0000 0000 0000 0000
00000420 0400 fff1 0000 0001 0000 0000 0000 0000
00000440 0400 fff1 0000 0000 0000 0000 0000 0000
00000460 0300 0003 0000 0008 0000 0000 0000 0000
00000500 0000 0002 0000 0000 0000 0000 0000 0000
00000520 0300 0002 0000 0017 0000 0000 0000 0000
00000540 1000 0000 0000 001d 0000 0000 0000 0000
00000560 1000 0000 0000 0024 0000 0000 0000 0078
00000600 1200 0002 0066 6163 742e 6300 6763 6332
00000620 5f63 6f6d 7069 6c65 642e 002e 756d 756c
00000640 0070 7269 6e74 6600 6d61 696e 0000 0000
00000660 0000 003c 0000 0507 0000 0000 0000 005c
00000700 0000 0209 0000 0000 0000 0060 0000 020c
00000720 0000 0000 0000 0068 0000 0607 0000 0000
00000740 0061 733a 2057 6f72 6b53 686f 7020 436f
00000760 6d70 696c 6572 7320 342e 3220 6465 7620
00010000 3133 204d 6179 2031 3939 360a 0047 4343
00010020 3a20 2847 4e55 2920 322e 3935 2e32 2031
00010040 3939 3931 3032 3420 2872 656c 6561 7365
00010060 2900 0000 0000 0000 0000 0000 0000 0000
00010100 0000 0000 0000 0000 0000 0000 0000 0000
00010120 0000 0000 0000 0000 0000 0000 0000 0001
00010140 0000 0003 0000 0000 0000 0000 0000 0034
00010160 0000 003d 0000 0000 0000 0000 0000 0001

```

0001200	0000	0000	0000	000b	0000	0001	0000	0006
0001220	0000	0000	0000	0074	0000	0078	0000	0000
0001240	0000	0000	0000	0004	0000	0000	0000	0011
0001260	0000	0001	0000	0002	0000	0000	0000	00f0
0001300	0000	0011	0000	0000	0000	0000	0000	0008
0001320	0000	0000	0000	0019	0000	0002	0000	0002
0001340	0000	0000	0000	0104	0000	0080	0000	0005
0001360	0000	0005	0000	0004	0000	0010	0000	0021
0001400	0000	0003	0000	0002	0000	0000	0000	0184
0001420	0000	0029	0000	0000	0000	0000	0000	0001
0001440	0000	0000	0000	0029	0000	0004	0000	0002
0001460	0000	0000	0000	01b0	0000	0030	0000	0004
0001500	0000	0002	0000	0004	0000	000c	0000	0034
0001520	0000	0001	0000	0000	0000	0000	0000	01e0
0001540	0000	0052	0000	0000	0000	0000	0000	0001

Bu üç koda dikkat ettiyseniz, ilk kodun daha açık, mantıklı, anlaşılabilir, kolay bir şekilde değiştirilebilir olduğunu, ikinci kodun daha çok ayrıntı içerdiğini ve karmaşık olduğunu, son kodun ise, anlaşılabilmesi için saatler gerektiğini görebiliriz.

Derleyicinin Avantajları,

- programın sadece bir kere derlenmiş olması (böylece, bir kere dönüştürülmüş – derlenmiş program istenilen sayıda çalıştırılabilecektir)
- çok hızlı çalışması
- kodun optimize edilerek daha da hızlı çalıştırılabilir olması şeklinde sıralanabilir.

Bununla birlikte; çalışma anındaki kontroller çok zordur. Yani çalışmakta olan bir programın hangi ifadesinde hata olduğunu bulmak oldukça zordur. Ayrıca derleyiciler çok büyük programlardır.

Interpreter : Yorumlayıcı Yorumlayıcı, derleyici gibi, bir dilde yazılmış olan ifadeleri, makine koduna çeviren bir programdır, ancak bu çevrimi, o an yapmaktadır. Derleyici, bütün kodu, satır satır çevirip bir PROGRAM dosyası yapar ve sabit diske kayıt eder. Yorumlayıcı ise, kodu satır satır çevirip hafızada o an çalıştırır ve, programın doğru çalışıp çalışmadığının anlaşılması için ortam oluşturur. Böylece, çalışma anında bir hata olacaksa, bu hata önceden, yorumlayıcı tarafından tespit edilmiş olunur.

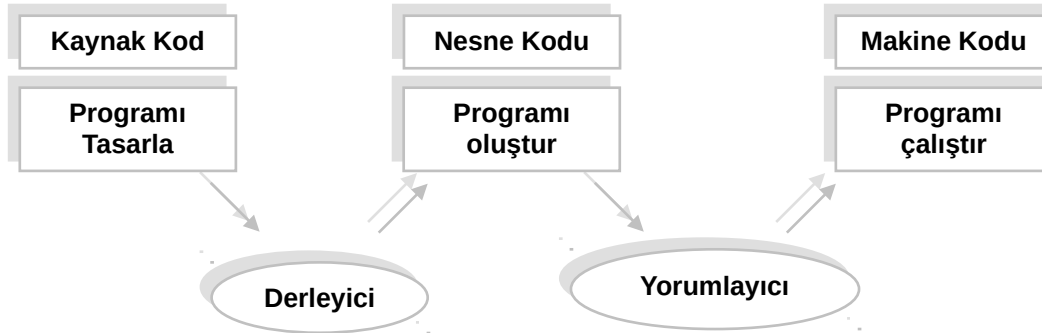
Yorumlayıcı, aşağıdaki işlemleri, peş peşe tekrarlar;

- kaynak koddan tek bir ifadeyi okur
- kaynak koddan okunan ifadeyi, bir veya birkaç makine koduna çevirir
- çevrilmiş olan makine kodunu çalıştırır

Yorumlayıcılar genelde çok küçük programlardır ayrıca çalışma anında, hangi ifade de olduğunu bildiği için, hatanın tespiti daha kolay yapılır. Yorumlayıcı, derleyiciye göre, hız bakımından daha düşüktür. Ayrıca, yorumlayıcı, çalıştırma esnasında önüne gelen, yani karşılaştığı ilk hatayı tespit edebilmektedir. Bir sonraki hatayı, ancak o anki hatayı düzeltip, tekrar çalıştırdığınızda tespit edebilirsiniz. Derleyici ise, kodu tamamen okuduğu için daha çok hata bulabilir. Ancak, çalışmadan anlaşılacak hatalar daha azdır.

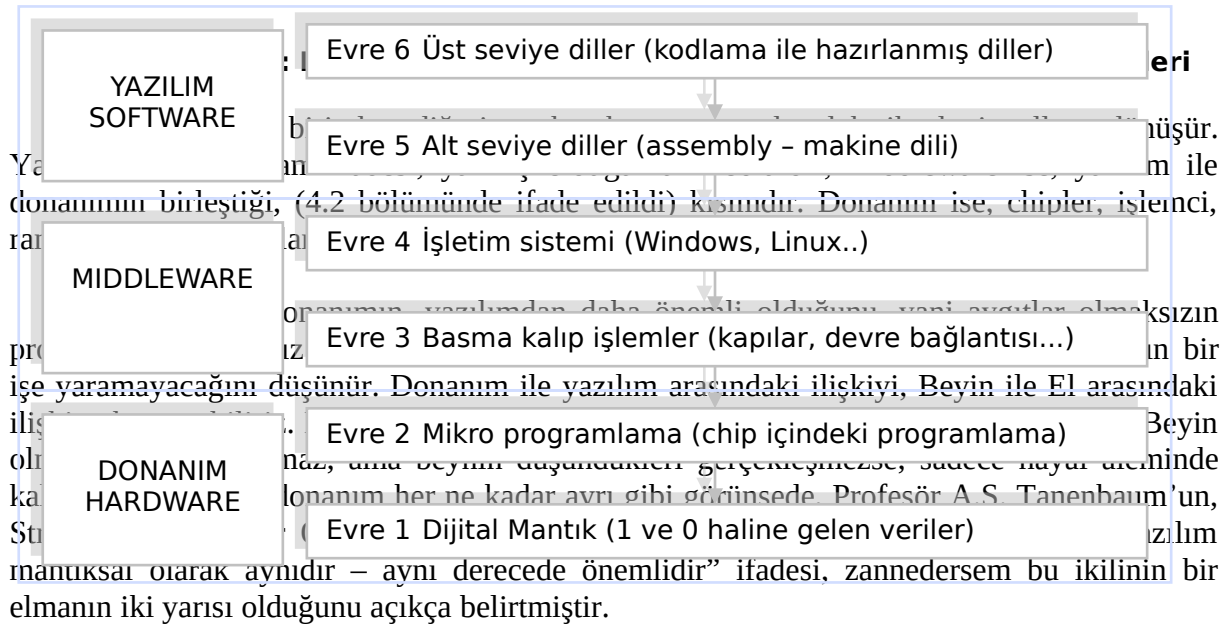
Sonuç olarak diyebiliriz ki, yorumlayıcı daha çok hata bulabilir ancak tek tek görüntüler, derleyici, (daha az hata bularak) bütün hataları gösterir.

Hem derleyici, hem de yorumlayıcı birlikte kullanılabilir. Böylece, ilk olarak hatalar tespit edilip düzeltilir, daha sonra, kalan hatalarda, satır satır çalıştırılarak tespit edilebilir.



Figür 4.4 C : Programın gelişimi

Programlar, bir dilde ifade edilmiş kodlardan, 6 evreden geçerek, elektriksel sinyallere dönüştürülür. (aşağıdaki liste, yukarıdan aşağıya doğru sıralanmıştır)



İşletim sistemi, Assembly diline dönüştürülmüş kodunu, gerekli işlemlerde kullanmak üzere yönetir ve yönlendirir. İşletim sistemi, yazılım ile donanım arasında bir arabirim kurar ve yazılım ile donanım arasında iletişimi mümkün kılar.

İşletim sistemi evresi, Assembly dilinden gelen bir dizi 1 ve 0'ı elektriksel yük olarak hafızaya yazarak donanımla konuşur, yani yazılımdan gelen mantıksal ifade şeklindeki emirleri, donanıma iletir.

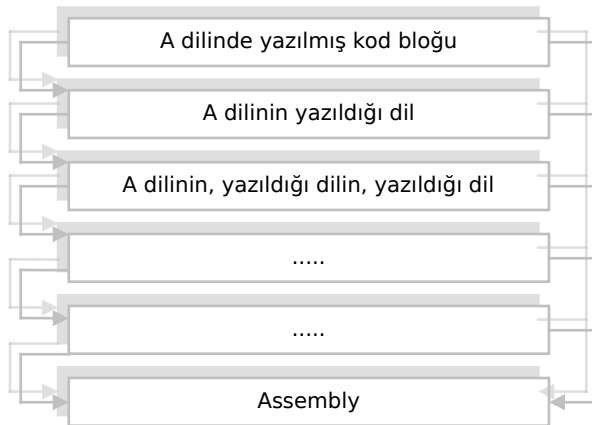
İşletim sistemi tarafından Assembly kodu olarak çalıştırılmak üzere gönderilen elektriksel yük, Micro programlama tarafından gerekli işlemleri gerçekleştirebilir. Micro Programlama ile yapılan işlemler; Girdi Çıktı aygıtlarını çalıştırma – veri alma – veri gönderme ve hafıza işlemleridir.

Micro programlamadaki microchipler sayesinde, gelen emir, en basit devre parçalarıyla ifade edilebilecek hale getirilir.

Dijital mantık evresi; algoritma – mantık dersinde üstünde durduğumuz mantıksal işlemlerin, mantıksal kapıların ve elektronik devrelerin çalıştığı evredir. Dijital Mantık evresi tamamen donanım üzerinde gerçekleşen bir evredir, bir standart çalışan bir makineden farkı yoktur.

Programlama dilleri, genelde, emir olarak gönderilen kodu, kendi yazıldığı dile çevirir. Ancak bazı diller, kodları, bir veya iki üst hatta bazen direk olarak Assembly diline çevirir.

Örnek : ‘A’ adında bir programlama dilinde bir program yazıyor olduğumuzu düşünelim



Figür 4.4 E : Dillerin birbirine çevrilmesi

Her dil, kodu ya kendi yazıldığı dile, yada direkt olarak assembly’ye çevirir. Bu işlemi, fabrika yöneticisinden emir alan bir mühendisin, bu emri yerine getirmek için bir veya birkaç işçiye başka emirler (daha basit) vermesine benzetebiliriz. Ayrıca, bir fabrika yöneticisi, bir işçiye direkt olarak emirde verebilir. İşte aynen yukarıda gösterildiği gibi, üst dil, bir alt dile veya birkaç alt dile veya direkt olarak o işi yapacak olan işlemciye assembly kodu olarak çevrilebilir.

Alıştırmalar

- Derleyici nedir?
- Assembly nedir?

[Ara verme bölümü]

4.5 İşletim sistemleri

İşletim sistemi İşletim sistemi, bir bilgisayarın temel, teknik işlemlerini gerçekleştirmek amacıyla tasarlanmış bir tür yazılımdır. İşletim sistemleri, kullanıcıları, donanımlarla yapılacak olan işlemlerin uğraşından, ayrı tutmaya yarar. Low-level programlamada bu seviyede yapılır. Yani üst seviye programlama dillerinin, yapacak olduğu işlemlerin, ve bu işlemlerin yapacak olduğu işlemlerin tasarlandığı bölümdür. İşletim sistemleri kullanıcıya, donanımsal işlemlerin kendisi tarafından yapıldığı bir arabirim sunmaktadır. Günlük hayatta kullandığımız binlerce program, bu TEMEL program olan işletim sistemleri üzerine kurulur. En çok kullanılan işletim sistemleri, Window ve Linux'tır. Bilgisayarlarınızda kurulu olan, Windows XP, Windows 2000, Windows Vista... birer işletim sistemidir.

Bir bilgisayar, temel olarak, 4 bileşenden oluşmaktadır:

Donanım (hardware)	Daha önceden bahsettiğimiz gibi, bir bilgisayarın fiziksel parçalarına verilen addır. (İşlemci, bellek üniteleri gibi)
İşletim Sistemi	(Linux, Windows.. gibi)
Sistem Yazılımları	(Taban programlar, API'ler, Derleyiciler, Veritabanı ve Network Yazılımları)
Uygulama Yazılımları	(Programcıların kendi geliştirdikleri yazılımlar, grafik yazılımları, muhasebe yazılımları)

İşletim sistemleri, daha önceki konularda bahsedilen anakart gibi (nasıl anakart, işlemci ile diğer kartlar arasında iletişimi sağlıyorsa), bilgisayar kullanıcılarıyla bilgisayar donanımı arasındaki bağlantıyı kurar ve kullanıcılardan alınan girdileri yorumlayıp – değerlendirip, bu değerlendirmeye göre bilgisayar donanımlarını, gerekli işlemleri yapmak üzere çalıştırır.

İşletim sistemlerin kısa tarihi

a) Birinci nesil işletim sistemleri

Daha önceki bölümlerde, ilk sayısal bilgisayarın ingiliz matematikçi “Charles Babbage” tarafından tasarlandığını belirtmiştim. Ancak Babbage bilgisayar çalışmalarında istediği başarıyı elde edememiştir. Bu süreç diğer bilimadamlarını da etkilemiş ve II. Dünya savaşı yıllarına kadar bu konuda çok az gelişmeler söz konusu olabilmıştır. II. Dünya savaşı yıllarında, Princeton ve Harvard Üniversitelerinden (dünyanın en iyi iki üniversitesi), bazı bilimadamlarının yaptığı çalışmalar sonucunda, vakum tüpleri kullanan sayısal bazı makineler geliştirilmiştir. Tabi bu makineler, binalar büyüklüğünde ve onbinlerce vakum tüpleri kullanarak geliştirildiğinden dolayı inanılmaz büyük masraflar sonucunda ortaya çıkabiliyordu. Üstelik, şuanki standart bilgisayarın yaptığı işlemlerin milyonda birini ancak yapabiliyordu.

Bu dönemde, programcı – yazılımcı – teknik sorumlu – işletim sistemi uzmanı ... gibi meslek dalları bulunmuyordu. Bilgisayarlarla ilgili bütün çalışmaları bütün işlemleri bu bilimadamlarından oluşan ekip yapıyordu.

b) İkinci nesil işletim sistemleri

II. dünya savaşından sonraki süreçte, transistörlerin geliştirilmesi yeni bir devrim başlattı. Bu dönemde, artık bilgisayarlar, üretilip büyük firmalara satılmaya başlandı. Ayrıca, bilgisayar üreticileri, operatörler, programcılar ve teknik personel meslekleri oluştu. Her ne kadar birçok işlem gerçekleştirebilselerde, çok büyük olduklarından dolayı sadece büyük şirketlerde veya devlet dairelerinde kullanılabiliniyorlardı.

Şuan sabitdisk ismi ile kullandığımız aygıt yerine, bu zamanlarda, kart ve manyetik teypler bulunuyordu. Bu dönemdeki en büyük gelişme, derleyicilerin bir kere bilgisayara kurulmasından sonra, çok sayıda farklı programı bant veya kart üzerinde arka arkaya yüklenip çalıştırılabilmesidir. Bu kavram, Yığın İşlem(Batch Processing) olarak adlandırılır.

c) Üçüncü nesil işletim sistemleri

Bu süreçte, liderliğini IBM'in yaptığı, 360 mimarisi sayesinde, bilgisayarlarda en önemli yenilik olan, transistör kullanımı yerine entegre devrelerin kullanılması oldu. Böylece, makine boyutları küçüldü, masrafları azaldı ve kapasiteleri arttı.

Üçüncü nesil işletim sistemlerinin en büyük özelliği, birden fazla programı aynı anda çalıştırabilme özelliği idi. Böylece, arka arkaya biriken ve sırayla çalıştırılan programlar yerine, işlemci; bu programları bir tür önem sırasına göre çalıştırır. Bu olanağa **Spooling** adı verilir.

d) Dördüncü nesil işletim sistemleri

Bu dönemde, LSI (Large Scale Integration circuits) entegre devrelerinin geliştirilmesi sayesinde, binlerce transistörü içinde barındıran ve buna karşı 1cm² yer kaplayan kişisel bilgisayarlar devri açılmıştı. Maliyet ve boyut azaldığından, artık bilgisayarlar kullanıcıların evlerine ulaşmaktaydı.

Dönemin işletim sistemleri arasında MS-DOS ve Unix vardı. 1985'ten sonra, değişik bir teknolojik yapılanma ile birlikte, PC(personal computer – kişisel bilgisayar)'ler; Ağ işletim sistemi ve Dağıtık işletim sistemi kullanmaya başlamıştır.

4.6 Mimariler

Bu bölümde, genelde masaüstü ve web uygulamalarında kullanılan en yaygın olan dilleri anlatacağım. En çok kullanılan mimarilerden bir tanesi doğal olarak, dünyanın en büyük yazılım firmalarından biri olan (hatta en büyüğü) **Microsoft'un Visual Studio'sudur**. Microsoft, C++, C#, VB, ASP, ASP.Net, AJAX, VB.Net, J# ... gibi dilleri desteklemektedir.

Microsoft dillerinin özellikle son senelerde çıkardığı .NET sisteminin en önemli özelliklerinden biri, çok güvenli olması ve çok fazla sayıda ek özelliği olmasıdır. Ancak Microsoft ürünleri (kendi geliştirdiği .net) bazı uzman programcılar tarafından tercih edilmez. Microsoft ürünlerinin pozitif yanı, diğerlerine göre daha kolay olması, internette çok daha fazla döküman ve örnek bulunması ve geniş özellik yelpazesine sahip olmasıdır. Negatif yanı ise, bu dillerin çok sağlam (sağlam : yeterince hatasız ve mantıklı) bir yapı üzerine oturmuş olmamasıdır. Bir sonraki bölümde, en çok kullanılan diller listesinde, Java dilinin en üstte, Nicorosoft ürünlerinden en kolay olan Visual Basic'in ise, 5. sırada olduğunu, Microsoft'un diğer dillerinden olan C# in ise çok daha düşük seviyede olduğunu görebilirsiniz. Bunun nedeni, az önce bahsettiğim gibi sağlam bir alt yapının olmamasından kaynaklanır. Microsoft, .net (2.0) 2005'te daha sağlam bir yapı oluşturduysa da, daha önceki sistemlerinden dolayı beklediği ilerlemeye elde edemedi.

Bunun yanı sıra, JAVA ve DELPHİ gibi diller biraz daha uğraş isteyen, ancak oldukça - hatta inanılmaz derece de sağlam dillerdir. Sağlam dil ifadesini yukarıda tanımlamıştık yeni bir açıklama olarak; eğer bir sistem çok sık ve hızlı değişirse ve çok bug (hata) içeriyorsa, bu, o dilin veya yazılımın zaten iyi olmadığı, daha oturtulamadığı anlamına gelmektedir. Özellikle Java dili, sahip olduğu sistemler bakımından çok mükemmel bir dildir. Hemen hemen her platformda kullanılabilir. Java karşımızda sadece bilgisayarlarda değil, internette, cep telefonlarında çıkar. Hatta, java ile çalışan işlemciler bile bulunmaktadır. Java piyasaya ilk 1996 da yılında çıkmıştır ve çıkmasından bu yana, çok az yenilik yapmıştır. Bu, sistemin yeterliliğini göstermeye bir delildir.

Bu sağlam alt yapıyı, Microsoft bile kullanmaktadır. VB6 (visual basic 6 - Microsoft'un Visual Studio 6 daki dillerinden biri) kullanmış (yada kullanacak) olanlar, Vb.Net (2002- 2003)'in tamamen Java'ya benzetildiğini, Vb.Net 2005 in ise tamamen Java'nın özelliklerini aldığını görebileceklerdir. Aynı durum C# 2003, içinde geçerlidir. Hatta, Microsoft, Visual Studio 6'da çıkardığı, J# (Java'ya çok daha benzer bir dil) ile birlikte, Java kullanıcılarını, kendi editörünü – kendi sistemini kullanmasını amaçlamıştır. Java bu teknolojileri ve sistemleri yıllar önce sağlam bir şekilde geliştirmiştir. Microsoft, yeni çıkardığı ajax teknolojisinde bile Javascript'i terk edemedi.

Ancak Microsoft hızla yükselmeye devam etmektedir. Microsoft'un hedefi “ne olursa olsun en iyi dil bizim olsun” iken, Java, “sadece kendi geliştirdiğimiz sistemlerde, kendi mimarimizde, en iyi dil bizim olsun” hedefini benimsemektedir. Yani Java kendi içine kapanık ama sağlam, Microsoft dilleri ise sonradan mükemmelleşen ve dışa dönük dillerdir. İşte bu yüzden, gelecek 5 sene içinde (eğer farklı bir gelişme söz konusu olmaz ise), Microsoft'un galibiyetine (en çok kullanılan dillerden birinin Microsoft dili olması) kesin denilebilir.

Son yıllarda iyice artan open source dilleri ve bu dillerle geliştirilen open source yazılımlar, ücretsiz ve bütün içeriğinin kodlmayacıya sunulmasından dolayı inanılmaz büyük bir çıkış yapmıştır. PHP, MYSQL, Linux gibi birçok sistemin open source olması pazar

payını arttırıp, kalıcı olmasını sağlamaktadır. Kullanıcılar; yüzlerce, bazı ürünlerde binlerce dolar ödeyerek, yazılım geliştirmektense, ücretsiz sistemlerle çalışmayı tercih etmektedirler.

4.7 Programlama dili kullanımı

Bir önceki konuda belirtilen bilgileri, bilimsel dayanaklarla desteklemek için aşağıdaki tablolara bakabiliriz. Aşağıdaki ilk tablo (solda ki tablo) da, Tiobe Yazılım sitesinden alınan, programlama dillerinin kullanım oranlarıyla ilgili istatistikler gösterilmiştir. Sağdaki tablo ise, aynı verilerin developer.com adresindeki tablosudur. [e19] [e20]

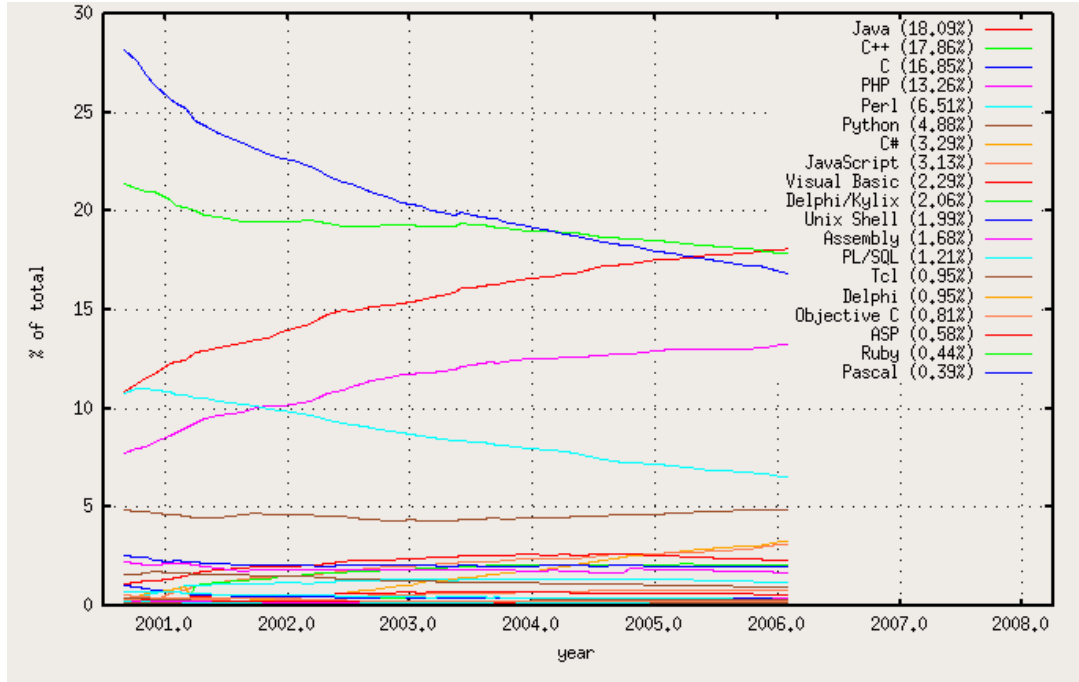
Mayıs 2007 Sıra	Mayıs 2006 Sıra	Dil	Mayıs 2007 Oran	Fark	Sıra	Dil	Oran	Fark
1	1	Java	19,14	-2.18%	1	Java	17,00%	-6.21%
2	2	C	15,15	-2.54%	2	C	16,34%	-1.64%
3	3	C++	10,11	-0.82%	3	C++	15,31%	-1.27%
4	4	PHP	8,738	-1.48%	4	PHP	10,43%	+5.75%
5	5	(Visual) Basic	8,431	-1.13%	5	(Visual) Basic	10,14%	+2.67%
6	6	Perl	6,152	+0.10%	6	Perl	8,44%	-0.54%
7	8	Python	3,779	+0.74%	7	Delphi/Pascal/Kylix	4,81%	+2.94%
8	7	C#	3,656	+0.38%	8	Python	4,70%	+3.41%
9	9	JavaScript	3,072	+0.88%	9	SQL	2,86%	-0.14%
10	19	Ruby	2,632	+2.18%	10	JavaScript	1,68%	-0.11%
11	10	Delphi	2,13	+0.36%	11	C#	1,63%	-0.37%
12	11	SAS	2,076	+0.60%	12	SAS	0,73%	-0.25%
13	12	PL/SQL	1,979	+0.97%	13	COBOL	0,53%	-0.54%
14	18	D	1,347	+0.87%	14	IDL	0,35%	-0.12%
15	21	ABAP	0,731	+0.31%	15	Lisp	0,33%	-0.34%
16	14	Lisp/Scheme	0,698	-0.19%	16	Fortran	0,33%	-0.54%
17	17	Ada	0,679	+0.19%	17	Ada	0,32%	-0.19%
18	13	FoxPro/xBase	0,637	-0.37%	18	MATLAB	0,28%	-0.05%
19	20	Fortran	0,63	+0.20%	19	RPG	0,28%	-0.38%
20	15	COBOL	0,627	-0.04%	20	Prolog	0,26%	-0.24%

Figür 4.7 A : Dillerin kullanım oranları

Tablolarda görüldüğü üzere, oranlar birebir aynı olmasada, ilk altı dil, aynı şekilde sıralanmıştır, bu, tablolardaki verilerin ne kadar doğru olduğunu göstermektedir. Soldaki

tablo, Mayıs 2006'dan bu yana değişiklikleri ve Mayıs 2007 deki verileri içermektedir. Sağdaki tablo ise, 2007 yılının ortalamasını içermektedir. Bu yüzden aralarında fark olması olağandır.

Aşağıdaki grafik, Berkeley üniversitesi tarafından hazırlanmıştır. [e21]



Figür 4.7 B : Dillerin kullanım oranları (2)

Dördüncü bölümde kullanılmış olan terimlerin ingilizceleri

anakart : Mainboard	dil : language
sabit disk : harddisc	platform : platform
işlemci : processor	mimari : architecture
mikro işlemci : mikro processor	işetim sistemi : operation system
yorumlayıcı : interpreter	
derleyici : compiler	

SONUÇ

Bilgisayar sistemlerini iyi kavramak, bir sistemin işlemlerinin nasıl yapıldığını anlamak için çok önemlidir. Dış dünyadaki bir sistemin analiz edilmesine yardımcı olur. Çünkü işlemciler, sadece ve sadece en basit ifadeleri anlayabildiğinden dolayı, en basit programlar bile, daha basit ifadelere dönüşür ve işlemciler, ancak bu şekilde işlemleri anlayabilir. Bizde gerçek dünyadaki sistemleri aynı işlemcinin anlaması için bölündüğü gibi, programlama dilinin anlayacağı sisteme göre analiz etmeliyiz.

5 İleri Programlama

İçerik

- 5.1 Oluşturucu
- 5.2 Değişkenlerin süreklilik alanı
- 5.3 İleri operatörler
- 5.4 Değişkenlerin ömrü
- 5.5 Programsal bloklar ve syntax
- 5.6 Programsal ifadeler
 - 5.6.1 Do / Loop / While / Until
 - 5.6.2 If Then Else Else
- 5.7 Prosedürler
- 5.8 Argümanlar
- 5.9 Diğer gerekli bilgiler
 - 5.9.1 İsimlendirme
 - 5.9.2 Dosya - Program ilişkisi
 - 5.9.3 Yorum
- 5.10 Matematiksel fonksiyonlar
- 5.11 Karakterel ve Metinsel fonksiyonlar
- 5.12 Tarih & saat Fonksiyonları
- 5.13 İşaretler
 - 5.13.1 Parantezler
 - 5.13.2 Nokta, İki nokta, Virgöl, Noktalı Virgöl
- 5.14 Dosya işlemleri
- 5.15 Anahtar sözcük

Amaçlar

- Beşinci bölüm,
- İleri nesne programlama ilkelerini,
 - İleri değişken işlemlerini, programsal anlatmaktadır.

Anahtar sözcükler

yeni, oluşturucu, özel, genel, sabit, dosya, yorum, doğru, yanlış

5.1 Oluşturucu (Constructor)

İkinci bölümde, gördüğümüz değişken tanımlama ifadeleri sadece ve sadece BASİT VERİ TÜRLERİ için geçerlidir. Bir sınıf türünde değişken belirlemek çok farklı gerçekleşmektedir.

Bir sınıf türünde, değişken tanımladığımızda, yine uçak bileti örneğimizde verdiğimiz gibi bir rezervasyon yapılı, ancak hiçbir dil, bir sınıf için VARSAYILAN değer atamaz. Yani, bir **sınıf** türünde tanımladığımız bir değişkenin, bir kopyasını, bizzat oluşturmamız gerekmektedir. Üçüncü bölümde sınıf ve nesne arasındaki ilişkiyi görmüştük. İşte, tanımlanan bir sınıf türündeki değişken, mutlaka bir NESNE tutar.

Bir nesne oluşturmak için, YENİ (new) anahtar sözcüğü ile birlikte, o sınıf türünün, “oluşturucu”larından birini çağırmalıyız.

```
Sınıf1 yeni;
```

Yukarıdaki ifade yeni adında, Sınıf1 türünde bir değişken tanımladı. Ancak daha önce belirttiğimiz gibi, bu değişkenin bir değeri yoktur. Bu değişken NULL’a eşittir. Şimdi yeni bir NESNE oluşturalım.

```
yeni = new Sınıf1();
```

Yukarıdaki ifade de, YENİ (new) anahtar sözcüğünü kullandık ve dile, bize yeni bir nesne oluşturması emrini verdik. New anahtar sözcüğünden sonra kullandığımız Sınıf ismi, oluşturacak olduğumuz nesnenin sınıfıdır.

Böylelikle, yeni değişkenine, Sınıf1 in yeni bir kopyası (nesne) oluşturulmuş ve yazılmış olur. Yukarıdaki iki ifade birleştirilip, tek ifade haline getirilebilir.

```
Sınıf1 yeni = new Sınıf1();
```

Peki neden hem tanımlarken hemde oluştururken aynı isim iki kere kullanılıyor? Yani, neden, yukarıdaki ifade de, değişken türü olarak Sınıf1 yazılıyor, o değişkene kopyası oluşturulup yazılacak olan nesnenin türünde Sınıf1 olarak yazılıyor?

Daha önceki bölümlerde, değişkenin bir POINTER olduğunu söylemiştik. Siz değişkeni tanımladığınızda, bu tanımlama, hiçbir anlam ifade etmez. Ancak ve ancak, o değişkene bir değer atarsanız (o prosedür içinde) bir anlam ifade edecektir. Aksi takdirde, kullanmadığınız (yani hiçbir değer atamadığınız) bir değişkenin olduğu kodu derlemeye çalıştığınızda, zaten derleyici hata verecektir.

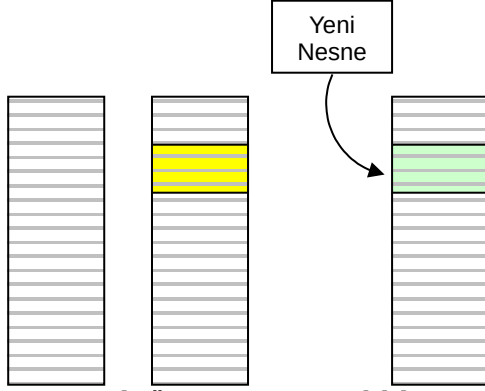
```
int i;  
int k;  
k=3;
```

Yukarıdaki kodda i değişkeni tanımlanmış fakat kullanılmamıştır. Derleyici böyle bir durumda, hata verir ve değişkene bir değer atamamızı söyler. Aynı durum nesneler içinde geçerlidir. Eğer bir sınıf türünde değişken tanımlayacak isek, o değişkene yeni bir nesne

ataması yapmalıyız.

Sınıf1 yeni = new Sınıf1();

Yukarıdaki ifadede, ilk Sınıf1, “Sınıf1 türünde, bir değişken tanımlamak istiyorum. Bana, hafıza üzerinde, bu değişkenin değerini tutabilecek bir yer aç” anlamına gelmektedir. İkinci Sınıf1 ise, “Sınıf1 türünde bir nesne oluştur” anlamına gelmektedir.



Figür 5.1 A : Yeni bir nesne için hafızada yer açılması ve yeni nesnenin yerleştirilmesi

Yukarıdaki çizgili uzun dikdörtgenleri hafıza gibi düşünelim. İlk başta hafızaya yeni bir nesne oluşturulduğunda, yerleştirebilmek için yer açılıyor (sarı). Daha sonra, yeni oluşturulan nesne, bu daha önceden açılmış olan yere, yerleştiriliyor (yeşil).

İkinci Sınıf1 ifadesinden sonra () parantezlerin olduğunu görüyoruz. Bu parantezler, “o sınıfın, OLUŞTURUCU metodunu çağır” anlamına gelmektedir. O halde, her sınıfın oluşturulabilmesi için bir Oluşturucu metodu olması gerekmektedir.

```
Class Taşıt
    Özellik : İsim
    Özellik : Model

    Sub New()
    End Sub
End Class
```

Yukarıdaki sınıfta bulunan, New metodu bu sınıfın, oluşturucusudur. Bir sınıfın kopyası oluşturulduğunda, ilk olarak bu metod çağrılmaktadır.

Taşıt yeni = new Taşıt();

Yukarıdaki ifade, çağrıldığında, akabinde çağrılacak olan ilk metod, Taşıt sınıfının içinde bulunan New metodudur.

Eğer, bir sınıfa, oluşturucu metod yazmaz iseniz, derleyici sizin için otomatik olarak bir boş (içi boş – hiç bir işlev gerçekleştirmeyen) oluşturucu metod yazacaktır.

Bazı sınıflarda, oluşturucu metoda bir işlem yazmanız gerekmemektedir. Böyle durumlarda herhangi bir oluşturucu tanımlamanıza gerek yoktur, az önce belirttiğim gibi,

derleyici sizin için bir tane boş oluşturucu yazacaktır. Bazı durumlarda ise, oluşturucunun içine kod yazmak gerekebilir. Örneğin, uçak bileti kayıtlarını tutan bir program yazdığımız ve bu program içinde bilet adında bir sınıf olduğunu düşünelim

```
Class Bilet
    Özellik: Sefer No
    Özellik: Tarih
    ....
End Class
```

Bu sınıfın, oluşturma metoduna, bir kod ekleyerek, belirtili seferde uygun bir yer olup olmadığını kontrol edebilir ve ona göre işlem yaptırabiliriz.

```
Class Bilet
    Özellik: Sefer No
    Özellik: Tarih
    ....
    Sub New()
        Eğer boş yer var ise;           bir tanesini ayır
        Eğer boş yer yok ise;           bu kaydı beklemeye al
    End If
End Class
```

Bir sınıf oluşturucusu, ek olarak argümanlara sahip olabilir.

```
Class Bilet
    Özellik: Sefer No
    Özellik: Tarih
    ....
    Sub New(SeferNo As Integer, Tarih As Date)
        Eğer boş yer var ise;           bir tanesini ayır
        Eğer boş yer yok ise;           bu kaydı beklemeye al
    End Sub
End Class
```

Bir sınıf oluşturucusu, eğer argümanlara sahip ise, o zaman, o sınıfı oluştururken kullanacağımız oluşturucu da aynı argümanlara sahip olmalıdır. Örneğin yukarıdaki sınıftan bir nesne oluşturmak için

```
Bilet yenibilet = new Bilet ( SeferNo, Tarih )
```

Abstract class nedir

Abstract class, özet sınıf anlamına gelmektedir. Bildiğimiz gibi tasarladığımız bir sınıftan istediğimiz kadar nesne üretebiliriz. Ancak bazen, oluşturduğunuz bir sınıftan sadece bir tane nesne türetilmesini isteyebiliriz. Yani bir tür limitleme yapabiliriz.

Gerçek dünyada da bazı sınıflardan binlerce nesne türetilirken, bazı sınıflardan sadece bir tane nesne türetilmektedir. Örneğin, istediğimiz kadar t-shirt üretebilirken, sadece bir tane başbakan vardır. Aynı şekilde, binlerce gezegen olmasına karşın tek bir uzay vardır. İşte

bazen bir sınıftan sadece bir tane üretilmesini isteyebiliriz. Varsayalım ki uzay mühendisliği ile ilgili bir program tasarlıyoruz. Az önce belirttiğim gibi, böyle bir programda, sadece ve sadece bir tane uzay nesnesi olmalıdır. Yani uzay sınıfından türetebilen sadece bir tane nesne olmalıdır ki, evrenin temel ilkelerinden biri olan “bir tane uzay vardır” ilkesi bozulmasın.

İşte bu tip durumlarda Abstract class kullanılır. Özet sınıfı tanımlarken bazı dillerde “abstract” anahtar sözcüğü ile ifade edilir. Bazı dillerde de, constructor yazılmaz. Böylece o sınıftan bir nesne oluşturulamaz.

Alıştırmalar

- Abstract Class nedir?
- Sınıf nasıl oluşturulur?
- Neden oluşturucu kullanılır?

5.2 Değişkenlerin süreklilik (geçerlilik) alanı

[3. Bölümdeki “sarma” terimini tekrar okuyunuz.] Çok gizli projeler üreten bir araştırma – geliştirme ekibini düşünelim. Şüphesizdir ki bu tip projeleri geliştiren bir ekip, güvenliği yüksek olan bir binada çalışmak zorundadır ve yine şüphesizdir ki bina içersindeki her bölüme, her kişi giremeyecektir. Sadece yetki verilen kişiler, izinli oldukları odalara veya bölümlere girebilecektir.

Başka bir örnek olarak, bir istihbarat teşkilatının gizli bir binasında bulunan çalışanlarda belirli erişim seviye izinlerine sahip olacaktır ve ancak izinli oldukları (Yetkilerinin yeterli olduğu) odalara ve bölümlere giriş yapabilecektir. Ayrıca, bilgisayarlarda bulunan dökümanların sadece izin verilenlerine erişebileceklerdir.

Aynı durum değişkenler içinde geçerlidir yani değişkenlere programın her bölümünden istediğiniz şekilde erişimi engelleyebilirsiniz. Sadece bazı sınıfların erişmelerine veya bazı değişkenlerin dışarıdan erişilememesine olanak sağlayabilirsiniz. Yani bazı değişkenleri özel, bazı değişkenleri de diğer sınıflardan erişebilir hale getirebilirsiniz.

Genelde bir çok dilde “Public” olarak ifade edilen anahtar sözcük kullanarak bir değişkeni dışarı sınıflar tarafından erişilebilir hale getirebilirsiniz. “Public” yerine “Private” anahtar sözcüğü kullanırsanız, tanımladığınız değişken, sadece o sınıf içinden erişilebilir.

Yani, Public sözcüğünü herkeste olan bir anahtara benzetebilirsiniz. Herkeste olan anahtar sayesinde isteyen, gerekli odaya girip istediği bilgiye ulaşabilir. Ancak Private yani özel anlamına gelen anahtar ise, sadece o bina içindekilerde veya o bölüm içindekilerde varolan bir tür anahtardır.

```
Sınıf Araç{  
    Private MotorSeriNumarası  
    Public Plaka  
}
```

Yukarıdaki sınıfta, herhangi bir dış sınıftan – ifadeden, Plaka değişkenine ulaşabilir, bu değişkenin değerinin değiştirebilir veya okuyabilirsiniz ancak aynı durum Motor seri numarası adlı değişken için geçerli değildir.

Örnek olarak;

```
Araç Aracım = new Araç();  
Aracım.Plaka = “34 PLT 34”  
Aracım.MotorSeriNumarası = “FXTGGYHKEFK36109”
```

Yukarıda ilk önce araç sınıfından türeyen yeni bir nesne oluşturduk. Daha sonra bu yeni oluşturulan nesnenin plaka özelliğine bir değer atadık. Akabinde, MotorSeriNumarası değişkenine değer atamaya çalıştık! Ancak bu yanlış bir ifadedir ve geçerli olmayacaktır hatta hata oluşmasına sebep olacaktır.

Public ve Private ifadeleri aslında sadece değişkenler için kullanılmaz. Aynı zamanda, özellikler, metodlar, fonksiyonlar ve olaylar içinde kullanılır. Yani Public olarak tanımlanan

bir metod, olay veya özellik, dış sınıflar tarafında çağrılabilir veya GÖRÜLEBİLİR. Ancak tanımlanan metod, olay veya özellik private olarak tanımlanmışsa, sadece o sınıf içinden çağrılabilir.

Aşağıdaki sınıfta; kaydını_yap metodunu dışarıdan çağırabiliriz.

```
Araç Aracım = new Araç();  
Aracım.kaydını_yap();
```

```
Sınıf Araç{  
  
    Private MotorSeriNumarası;  
    Public Metod kaydını_yap(){  
        MotorSeriNumarası = Yeni motor seri numarası atama işlemini yap!  
    }  
  
    Public Metod bakım_hatırlatma(){  
        if (AraçArızalıMı = True) { // eğer arızalı ise, bakıma gönder  
            bakım_yap();  
        }  
        elseif (bakım_kontrol () = True){ // arıza yoksa bile 10 bin bakım  
            gerekiyorsa...  
            bakım_yap();  
        }  
    }  
  
    Private Metod bakım_kontrol(){  
        if (GidilenKilometre Mod 10000 = 0 ) { // her on binde bir bakım  
            gerekir  
            return true;  
        }  
        Else{  
            return false;  
        }  
    }  
  
    Public AraçArızalıMı;  
    Private GidilenKilometre;  
}
```

Aynı şekilde, belirli periyotlarda, bakım_hatırlatma metodunu çağırabiliriz. Ancak bakım_kontrol metodunu çağıramayız. Çünkü bu metod private olarak tanımlanmıştır. Ek olarak, bakım_kontrol metodu, bakım_hatırlatma metodu içinden çağrılabilir.

Bir sınıf içinden o sınıfa ait olan bütün elemanlara ulaşmanız söz konusudur. Public veya private olması durumu değiştirmez.

Alıştırmalar

- Public & Private nedir?

[Ara verme bölümü]

5.3 İleri operatörler

Operatörler konusundan ikinci bölümde bahsetmiştim. Dikkat ederseniz kitapta hemen hemen her konudan birden fazla kere bahsediyorum. Eğer giriş kısımlarını okuduysanız, kitabın bir spiral şeklinde yani sırayla her konudan biraz öğrettikten sonra konuların ileri bölümlerini öğrettiğini söylemiştim. Matematiksel ve Programsal operatörler aldıkları argüman sayısı bakımından 3'e ayrılırlar.

Eğer bir operatör tek bir argüman alıyorsa, yani dışarıdan tek bir veri alıp bu veri ile işlem yapıyorsa, bu operatör bir unary (birli – tekli) operatördür. Eğer operatörün aldığı argüman sayısı iki ise, o zaman bu operatör binary (ikili) operatördür. Son olarak eğer bir operatör 3 tane argüman alıyorsa, bu operatöre ternary (üçlü) operatör denilir.

Sayı Üssü

Bir sayının üssünü ifade etmek için $^$ işaretini kullanılır. A üssü B şeklindeki bir ifade, matematikte, A^B şeklinde, programlama da ise, $A \wedge B$ şeklinde ifade edilir. Bazı dillerde $^$ işaretinin farklı anlamları bulunmaktadır, bu işaretin hangi dilde ne anlama geldiğini o dilin syntax ıyla ilgili araştırma yaparak bulabiliriz.

$$\begin{array}{ll} X = 3 \wedge 2 & X = 9 \text{ (3'ün 2. kuvveti)} \\ X = 2 \wedge 3 & X = 8 \text{ (2'nin 3. kuvveti)} \\ X = 4 \wedge 5 & X = 1024 \text{ (4'ün 5. kuvveti)} \end{array}$$

% (mod) operatörü

Matematikten hatırladığımız mod (modüler aritmetik) operatörü, bir sayı, bir sayıya bölündüğünde kalan sayıyı döndürür. Varsayalım ki X sayısını, Y sayısına böldüğümüzde, bölüm (önemli değil) Z, kalan C olsun.

$$\begin{array}{rcl} & X & \\ X = Y * Z + C & & Y \\ & & \\ X \text{ MOD } Y \equiv C & & Z \\ & & \\ \text{yada } C & & \\ & & \\ X \% Y \equiv C & & \end{array}$$

şeklinde ifade edilir.

17 mod 3 = 2 (17'yi 3'e bölersek, bölüm 5, kalan 2 olur)
17 MOD 2 bize 1 değerini
15 MOD 8 bize 7 değerini
90 MOD 10 bize 0 değerini verecektir.

(mod işlemi bazı dillerde mod bazı dillerde % işareti olarak ifade edilir).

! (değil) operatörü veya NOT operatörü

Bu operatör, DEĞİL anlamına gelmektedir. Yani bir ifadeyi tam tersine çevirmektedir. Özellikle boolean ifadelerde, yani true / false olan ifadelerde kullanılmaktadır.

Örneğin bir değeri kontrol ederken “bu değişken, bu değere sahip **değilse**”, ifadesini anlatmak için kullanılabilir.

if (a != 0)

yukarıdaki ifadede belirtilmek istenen, “eğer a sıfır değilse”’dir. Aynı ifade

if (NOT a = 0)

şeklinde de tanımlanabilir.

And ve Or

Matematikteki mantık konusundan "veya", "ve" tanımlarını hatırlayalım.

A	B	A ve B
0	1	0
1	0	0
0	0	0
1	1	1

A	B	A veya B
0	1	1
1	0	1
0	0	0
1	1	1

Figür 5.3 A : VE / VEYA Tabloları

Bu "ve", "veya" terimlerinin ingilizceleri sırayla, "and", ve "or" dur ve aynı matematiksel mantıkta olduğu işlevi yaparak, aynı sonuçları elde etmemizi sağlar.

Örneğin;

X = 1 AND 0 X = 0

Y = 1 OR 0 Y = 1

Z = (Yaşım 22'dir) AND (Otomobilim vardır)

Yukardaki z eşitliğinde, sonucun 1 yani doğru olabilmesi için, AND'in sağ ve sol tarafının 1 (yani doğru) olması gerekir. Sonuçta eğer, 22 yaşında isem ve otomobile sahipsem, Z'nin değeri 1 olur. Aksi takdirde, Z'nin değeri sıfır olacaktır.

Derse girelim mi? = (Zil çaldımı) AND (Öğretmen sınıftamı)

Eğer zil çaldıysa, **ve** eğer öğretmen sınıfta ise, o zaman "derse girelim mi"nin değeri 1 olur yani DOĞRU olur.

Sinemaya gidelim mi ? = (Biletimiz var mı) OR (Bilet alacak paramız var mı)

Yine aynı şekilde, sinemaya gidebilmek için ya biletimizin olması VEYA bilet alabilecek paramızın olması gerekmektedir.

Bu operatörleri AND ve OR şeklinde yazabildiğimiz gibi "&" ve "|" şeklinde de yazabiliriz. Aşağıdaki iki ifade aynıdır

H = (Kilom 75'tir) & (Boyum 1.74'tür)

H = (Kilom 75'tir) AND (Boyum 1.74'tür)

Yine aynı şekilde aşağıdaki ifade de aynıdır

$J = (\text{Otomobilim eskidir}) \mid (\text{Otomobilim bozuktur})$

$J = (\text{Otomobilim eskidir}) \text{ OR } (\text{Otomobilim bozuktur})$

Programlama da bunun haricinde daha farklı iki tip AND ve OR operatörü vardır. Bazı dillerde ORELSE, ANDALSO şeklinde, bazı dillerde de $\&\&$ ve \parallel şeklinde ifade edilir

Bu ifadelerin anlamı: ifadeyi oluşturan koşulların birincisi, ifadenin sonucunu sağlıyorsa, ikinciyi yapmanıza gerek bulunmuyor anlamına gelir.

Örneğin;

Gidebilir miyiz ? = (birinci ifade) & (ikinci ifade)

Gidebilir miyiz ? = (Otomobilin benzini varsa) & (Otomobilin anahtarı varsa)

Eğer otomobilin benzini yoksa (birinci ifadeye bakarak), otomobilin anahtarının olmasında önemi yoktur. Çünkü anahtar olsa bile sonuç yine "0" yani hayır olacaktır. And tablosunu hatırlarsak, eğer koşullardan biri 0 ise, sonuç mutlaka sıfır olur.

A	B	A ve B
0	1	0
1	0	0
0	0	0
1	1	1

Figür 5.3 B : VE tablosu

O zaman böyle bir durumda, ifadeyi değiştirip aşağıdaki hale getirebiliriz.

Gidebilir miyiz ? = (Otomobilin benzini varsa) $\&\&$ (Otomobilin anahtarı varsa)

Aynı durumu OR operatörü için uygulayalım

Gidebilirmiyiz ? = (Otobüs biletimiz varmı) \mid (Taksi paramız var mı)

İkisinden birinin olması durumunda diğerinin önemi yoktur. Çünkü, OR tablosunda, eğer koşullardan biri doğru ise, sonuç mutlaka doğru olur.

A	B	A veya B
0	1	1
1	0	1
0	0	0
1	1	1

Figür 5.3 C : VEYA tablosu

O halde bu ifadeyi aşağıdaki gibi değiştiriyoruz

Gidebilirmiyiz ? = (Otobüs biletimiz varmı) \parallel (Taksi paramız var mı)

NOT (devamı)

Başka bir ifade, olumsuzluk anlamına daha doğrusu "değil" anlamına gelen NOT ifadesidir. Not ifadesi, bir koşulun sonucunu tersine çevirir. Sonuç bir ise, sıfır, sıfır ise, bir yapar.

A	NOT A
0	1
1	0

Figür 5.3 D : DEĞİL tablosu

Müsait misin? = NOT (Çalışıyorum)

Yukarıdaki ifade de, eğer çalışıyor iseniz, müsait olamazsınız. Yani birbirinin zıttı durumlarda, bu şekilde eşitleme yapabilirsiniz.

Büyüktür, küçüktür

Denklemlerden hatırladığımız $<$, $>$ işaretleri, programlama da yine aynı şekilde kullanılır.

Eğer yaş $>$ 18 ise, ehliyet alabilirsin

Eğer hava sıcaklığı $<$ 15 (santigrat derece) ise, yağmur yağabilir.

Eğer param $<$ 20 (YTL) ise, otobüsle eve dön

\leq , \geq ifadelerini kullanmak içinse bu ifadeleri bölmek gerekir. Bu işaretler yerine \leq , \geq işaretleri kullanılır.

Eğer yaş \geq 18 ise, ehliyet alabilirsin

Eğer param \leq 20 (YTL) ise, otobüsle eve dön

Daha önce koşul ifadelerinden bahsetmiştim. Koşul ifadeleri, çeşitli durumlarda neler yapılacağını bildirdiğinden birkaç farklı modeli bulunur. Bunlardan ilkini az önce belirttim. Bir başka model olarak “**eğer** durum böyle **değilde**, **böyleyse**” ifadesi verilebilir.

Lisedeyken bir arkadaşım, not hesaplama programı yapmıştı. O zamanlar, not ortalaması 5 üstünden 3.5’u geçenler teşekkür belgesi, 4.5’u geçenler takdir belgesi alıyordu. Arkadaşımın tasarladığı programa, bütün notlarınızı (yazılı sonuçlarınızı – sözlülerinizi....) girdiğinizde bu hesaplamayı otomatik olarak yapabiliyordu. Ancak, programda bir hata vardı. Arkadaşım, takdir veya teşekkür alınıp alınmadığı hesaplamak için 2 ifade yazmıştı (doğal olarak):

Eğer Not ortalaması 3.5’tan büyükse Teşekkür aldı

Eğer Not ortalaması 4.5’tan büyükse Takdir aldı

Biraz daha programsal ifade şekline çevirirsek;

Eğer NotOrtalaması $>$ 3.5 İse MesajGöster(‘Teşekkür Aldınız’)

Eğer NotOrtalaması $>$ 4.5 İse MesajGöster(‘Takdir Aldınız’)

Birçoğunuz, yukarıdaki ifadelerde bir problem olmadığını düşünüyor olabilirsiniz. Ancak malesef hayat bu kadar kolay değil.

Eğer öğrenci 3.5 ve altında bir not alırsa, program hiçbir mesaj göstermiyor.. Doğru.

Eğer öğrenci 3.5 üzeri bir not alırsa, program “Teşekkür Aldınız” mesajı gösteriyor..

Doğru

Eğer öğrenci 4.5 üzeri bir not alırsa, program hem “Teşekkür aldınız” hemde “Takdir aldınız” mesajı gösteriyor.. Yanlış. Çünkü bir öğrenci hem takdir hem teşekkür belgeleri alamaz. Yukarıdaki ilk ifadede, bir yanlışlık bulunuyor. “Eğer NotOrtalaması 3.5’tan büyükse” ifadesi için, takdir alan bir kişinin notu zaten 3.5’tan büyük olacağı için, takdir alan bir öğrenciye bu iki mesajda gösterilecektir.

Bu yanlışlığı engellemek için ifadelerimiz daha farklı tasarlanmalıdır.

- 1 Eğer NotOrtalaması > 3.5 VE NotOrtalaması <= 4.5 ise “Teşekkür Aldınız”
- 2 Eğer NotOrtalaması > 4.5 ise “Takdir Aldınız”

5.4 Değişkenlerin Ömrü

“Başlangıcı olan herşeyin bir sonu vardır” (Oracle), sözünden de anlayacağımız gibi, değişkenlerin de bir ömrü bulunmaktadır. Daha önce 2. bölümde, değişkenlerin bir süreklilik süresi olduğundan bahsetmiştik. Nasıl insanlar doğar – büyür – ölür ise, değişkenlerde tanımlanır, kullanılır ve yok olur. İşte bu olaya değişkenlerin ömrü denilir.

Basit türde (integer, string, char, byte gibi, yani sınıf olmayan türler) tanımlanmış bir değişken 2 yolla yok olur. İlk yol, bu değişkeni nothing veya null’a eşitleyebilirsiniz. Bu durum her dil tarafından desteklenmeyebilir. İkinci yol ise, Garbage Collector’u çalıştırmaktır.

Genel olarak, bir değişkenin değerini tutan herhangi bir pointer kalmamışsa, Garbage Collector çalıştırılır veya otomatik olarak belirli aralıklarda veya değişken tanımlamada çalışır ve artık kullanılmayan değişkenleri hafızadan siler. Böylece, boşuna hafıza alanı işgal edilmemiş olunur.

Garbage collector (çöp toplayıcı), yukarıda bahsedildiği gibi, kullanılmakta olmayan – işlevi bitmiş olan değişkenleri silen bir araçtır. Yine bahsedildiği gibi hem otomatik çağrılabilir hemde kendi belirli işlemler arkasından otomatik olarak çağrılır.

Prosedürler içinde belirlenen değişkenler, prosedür bittiğinde yani prosedür işlemi tamamlandığında, hafızadan silinir. Çünkü, prosedürler içinde tanımlanan değişkenler sadece prosedür içinden geçerli olacağından, prosedürün son noktasına ulaşıldığında artık gerekli olmayacaktır ve doğal olarak garbage collector tarafından hafızadan silinecektir.

```
Metod herhangi_bir_metod{  
  
    int a=0;  
    a=1;  
    //a değişkenini farklı işlemlerde kullan  
    //...  
    //işlem bittiğinde, a değişkenini hafızadan sil  
}
```

Bir sınıf içinde tanımlanan değişkenlerde, sınıf yok oluncaya kadar hafızada kalır. Yani öncelikle bir sınıfı tanımladığınızda (yani new anahtar sözcüğü ile oluşturduğunuzda) hafızaya yazılır. Sınıf içindeki kendi değişkenleri de sırayla hafızaya yazılır ve sınıf yok olduğunda doğal olarak sınıf içindeki değişkenlerde yok olur.

```
Sınıf araba{  
    public plaka as string;  
    public model as string;  
}
```

```
Metod dışarıdan_çağrılan_bir_metod{  
    araba arabam = new araba();  
    arabam.plaka = “34 PLT 34”;  
    arabam.model = “mercedes”;
```

```
//Burada metod bittiğinden yani metottaki işlemler bittiğinden, tanımlanmış  
//olduğumuz arabam adlı değişkende yok olur. Dolayısıyla, araba adlı sınıf  
//içinde bu değişkenle birlikte tanımlanmış olan iki değişken (plaka ve model)  
//de hafızadan otomatik olarak silinir  
}
```

Eğer bir değişkeni, bazı dillerde static bazı dillerde shared olarak tanımlarsanız, o değişken hafızadan program bitene yani kapanana kadar silinmeyecektir.

Örneğin;

```
Private static FaizOranı = 3.56;
```

Aynı sabit değerler gibi hafızadan silinmeyen bu tip değişkenlerin, const lardan farkı değerlerinin değişebilir olmasıdır.

Tanımladığınız değişkenin programın içinden herhangi bir yerinden ulaşılabilmesini istiyorsanız, tanımladığınız değişkeni global kullanarak tanımlarsınız. Böylece, değişken, program kapanıncaya kadar hafızada kalır.

```
Global ayar = 3.467
```

Alıştırmalar

- Garbage collector ne işe yarar?

5.5 Programsal Bloklar ve Syntax

Syntax Syntax, bir programlama dilinde, program yazma esnasında uyulacak kurallar dizisidir. Nasıl bir kompozisyonu yazarken, giriş - gelişme - sonuç şeklinde bir yol izliyorsak veya nasıl cümlelerin sonunda nokta bulunuyorsa veya cümlelere başlarken büyük harfle başlıyorsak, veya Türkçe’de, özne başta, tümleç ortada, yüklem sonda ise programlamada da belirli bir format bulunmaktadır. Bu formata o dilin syntax'ı denir, ve hemen hemen her dilin syntax'ı değişiktir. Syntax kodları yazma düzeni de demektir.

Örnek syntax'lar (Aşağıdaki kodlar, farklı kaynaklardan amaçsız olarak alınmıştır.)

VB.Net

```
Private Function RemoveLast(ByVal ds() As Double) As Double()  
Dim dk(ds.Length - 2) As Double  
For i As Integer = 0 To ds.Length - 2  
    dk(i) = ds(i)  
Next  
Return dk  
End Function
```

PHP

```
<html>  
<body>  
<?php  
echo "Hello World";  
?>  
</body>  
</html>
```

Asp.Net

```
<html>  
<body>  
<%  
response.write("Hello World!")  
%>  
</body>  
</html>
```

Java

```
String QueryString="";  
for (int i=0; i<Queries.length-1; i++){  
    QueryString = QueryString + Queries[i] + " OR ";  
}  
QueryString = QueryString + Queries[Queries.length-1];
```

Her programlama dilinin syntax'ını, internetten bulabiliriz.

Bloklar

Bir kompozisyon yazarken, yazıyı oluşturan paragrafları belirli bir formata göre yazarız. Nasıl kompozisyonlar giriş, gelişme ve sonuç bölümünden oluşuyorsa, paragraflarda; paragrafı açıklayan bir açılış cümlesi, akabinde, birkaç cümle ile o paragrafta anlatılmak

istenen düşüncenin ifadesi, son bir cümle olarak paragrafın amacından (sonucu) oluşmaktadır.

Giriş cümlesi

Açıklayıcı cümle 1

Açıklayıcı cümle 2

.....

Bitiş cümlesi

Programlama içinde de, aynı şekilde, belirli kalıplar – bloklar bulunmaktadır. Yukarıda görüldüğü gibi bir (paragrafı) bloğu, başlatan ve bitiren bir ifade bulunmaktadır. Aynı durum programlama içinde geçerlidir.

Blok Başlangıç

İşlemler 1

İşlemler 2

...

Blok Bitiş

5.6 Prosedürler

Teknik rapor görenlerimiz bilir, raporda neredeyse bilgi kadar işin prosedürsel kısmı (yani resimlerin tabloların numaralandırılması – isimlendirilmesi – açıklanması, referanslar, şekiller, bölümlerin belirlenmesi, indeks, ekler,) yer kaplamaktadır. Başka bir örnek olarak, bir devlet dairesinde işlem yaptıracağınız zaman, her ne kadar önemli olan evraklar başkaları da olsa, sizden nüfus cüzdanı sureti, ikametgah, gibi evraklar isterler.

Nasıl raporlarda, veriler tablolar içine yazılıyorsa, programlamada da, kodlar belirli kalıplar içine yazılmak zorundadır. İkinci bölümde programsal bloklar ve syntax konusunda bilgiler verilmişti. Nasıl ifadeler, paragraflar içine yazılıyorsa, aynı şekilde, yazılacak olan kodlar da, prosedürler içine yazılmak zorundadır.

Daha önce üçüncü bölümde gördüğümüz sınıfların, metodları birer prosedürdür. Metodların içlerinde kodlar bulunur ve isimleri vardır. Aynı şekilde özelliklerde prosedür olarak kullanılabilirler. Bir olayı gerçekleştirmek için bir dizi işlem uygulanır. Ancak bu işlemler bir prosedür içine yazılmalı ve işaretlenmelidir ki; hem düzgün bir şekilde çalışabilsin hemde, diğer prosedürlerle karışmasın.

İşte bu nedenle, kodları yazarken, ismini belirlediğimiz prosedürler içine yazarız. Matematik, fizik, kimya ders notlarını ortak olarak yazdığınız bir defteri düşünün; bu defterler içinde yüzlerce satırlık bilgi olabilir. Ancak eğer siz, bu defter içinde, belirli bölümler yapmaz iseniz, veya defter içinde yazılı olan notun hangi dersin hangi konusuna ait olduğunu belirtmezseniz, ilk bakışta yazdıklarınızın hangi dersin notu olduğunu anlayamayabilirsiniz. Kaldı ki, insan beynine göre aptal diye tabir ettiğimiz bilgisayarlar hiç anlayamayacaktır.

İşte bu nedenle, defteri bölümlere ayırıp her bölüme o bölümün notlarını yazabiliriz. Aynı şekilde, yapılacak olan işlemleri gerçekleştirecek olan bir prosedüre isim vermeli ve işlemleri bu isimle tanımladığımız prosedürün içine yazmalıyız.

Bir prosedür belirtmek için;

```
Metod metod_ismi {  
    // işlem 1  
    // işlem 2  
    // işlem 3  
    // ...  
}
```

Şekline benzer bir ifade kullanılır. Yukarıda belirtilen syntax bazı çok kullanılan dillerde kullanılmaktadır. Yukarıdaki kod bloğunda, { işareti, tanımlanan metodu başlatır } işareti tanımlanan metodu bitirir. Metod çağrıldığında, sırayla işlemleri yapacaktır.

Prosedürlerin türleri bulunmamaktadır: Metodlar, Fonksiyonlar. Yukarıda belirttiğimiz gibi bir dizi işlemi gerçekleştirmek için tasarlanmış yapı, bir metoddur. Eğer bir dizi işlem gerçekleştirdikten sonra o prosedürün bize bir sonuç döndürmesini istiyorsak, metod yerine fonksiyon yazarız. Fonksiyonlar konusundan daha önce ikinci bölümde bahsetmiştim. Metodlar ile fonksiyonlar arasındaki tek fark, metodların bir sonuç döndürmediği, fonksiyonların ise yapılan işlemler sonrasında bir değer döndürdüğüdür.

Prosedür : Fonksiyon / Metod arasındaki fark

Bir prosedür, programların içine yazıldığı paragraflardır diyebiliriz. Genelde iki çeşit prosedür bulunmaktadır: Fonksiyon ve Metod. Fonksiyonlar genelde bir işlem yapıp sonuç döndürmek için kullanılır. Metodlar ise bir değer döndürmezler.

ÇEVİRME	<p>Hatırlatma_1'i hatırlayalım, amacımız, gerçek dünyada metinsel olarak ifade ettiğimiz bir işlemi bilgisayar içine programsal kod olarak aktarabilmek. Gerçek dünyada, “bu işin prosedürü böyle”, “bu işin kuralı böyle”, “bu iş böyle yapılır” diye ifade ettiğimiz veya, birden fazla işlemi sırayla yazarak oluşturduğumuz metinleri ifade ederken, prosedür yani 3.bölümden gördüğümüz metotları kullanırız.</p> <p>Bir işlemin nasıl yapıldığını, yada, o işlemi gerçekleştirmek için yapılması gerekli olan alt işlemlerin bir arada toplu tutulmasına yarayan terime prosedür denir.</p>
----------------	---

Çeviri 8: Prosedür

Alıştırmalar

- Prosedürler ne işe yarar?

5.7 Argumanlar

Matematiki fonksiyonları hatırlayalım.

$$Y=F(X)$$

şeklinde bir fonksiyonu değerlendirmeye aldığımızda, X değişkeninin bu fonksiyona bir değer girdiğini söyleyebiliriz. X değişkeni burada bir tür argümandır. Bir fonksiyona işlem görmek üzere girdi olarak verdiğimiz her değişken birer argümandır.

Faiz problemlerinden $f = \text{ANT}/100$ fonksiyonunu hepimiz hatırlarız. Bu fonksiyon, A yani anapara, N yani faiz oranı, T yani zaman değişkenlerini alır, hesaplamayı yapar ve sonuç olarak f yani faiz miktarını döndürür. Dikkat ettiyse, bu fonksiyon, işlem yapabilmek için 3 tane değişkene yani argümana gereksinim duymaktadır. Sonuç olarak diyebiliriz ki bir fonksiyona veya metoda, dışarıdan değişken olarak bir veri girişi yapılıyorsa, bu değişkenlerin her biri birer argümandır.

Prosedürler yani ya fonksiyonlar yada metodlar argümanlara sahiptirler ancak argümansız olan fonksiyon veya metodlarda bulunmaktadır. Argümanlar yapılacak olan işlemin; kime veya neye uygulanacağını ve bazı ayarları belirtir.

Byval / Byref

Byval (by value), “Değer vasıtasıyla”, “Değer ile” anlamına gelmektedir. Bir prosedürü çağırırken, prosedürün ismini yazdığımızı sonrasında ise (varsa) argümanlarını yazdığımızı belirtmişim. Bu şekilde belirttiğimiz argümanların **sadece değerleri, kopyalanıp**, prosedüre iletilir. Daha açık bir ifade ile, bir prosedürü çağırırken, o prosedürün argümanı olarak kullanacağınız değişkenlerin değerlerinin birer kopyası alınıp, prosedüre iletilir ve sanki o prosedürün içinde tanımlanan ayrı bir değişkenmiş gibi işlenir.

```
Metod1{
    int i=4;
    Metod2 ( i );
}

Metod2( argüman){

}
```

Yukarıdaki ifadede, Metod1 içinden çağrılan Metod2 prosedürüne, Metod1 prosedürü içinde tanımlanmış olan i değişkeninin değeri iletilir. Daha sonra, bu iletilen değer argüman adlı değişkenin (argümanın) üstüne yazılır. Sanki Metod2 prosedürü içinde tanımlanmış bir “argüman” adında değişken olduğunu ve bu değişkene, çağrılan bir prosedürden gönderilen “4” değerinin yazıldığını düşünebiliriz.

Böylelikle, çağrılan metod içinde, “argüman” argümanda yapacağınız değişiklikler, çağırılan prosedürdeki, “i” değişkeninin değeri üzerine yazılmaz ve sanki ayrı birer değişken gibi işlenirler.

Bir prosedürü çağırırken, argümanlarda değişken isimlerinden başka ek bir ifade belirtmiyorsa, o argümanın türü byval olarak algılanır.

ByRef (by reference), “Referans olarak”, “Referanstan vasıtasıyla” anlamına gelmektedir. Burada belirtilen “referans” pointer anlamına gelmektedir yani bir değişkenin hafızadaki yerini belirtmektedir. Bir prosedürü çağırırken belirttiğimiz argümanları byref ile tanımlarsak o değişkenlerin, ByVal da olduğu gibi değerleri değil kendileri prosedüre gönderilir.

Böylece, çağırılan metod içindeki değişken ile çağırılan metod içinde argüman aynı değişken olur ve, çağırılan metod içinde, değişkene yaptığınız değişiklikler, çağırılan metod içindeki değişken içinde geçerli olur. Sonuç olarak Byref, değişken değeri yerine değişkenin adresini gönderir ve işlemleri bu adres üstünde yap emrini verir.

Bir fonksiyona istediğiniz sayıda argüman girebilirsiniz. Ancak bir fonksiyon normalde sadece bir değer döndürebilir.

$$y = f(x_1, x_2, x_3, x_4, \dots)$$

Yukarıdaki ifadede bir fonksiyona (f) birden fazla argüman değeri girilmiştir ($x_1, x_2, x_3, x_4, \dots$) ancak tek bir sonuç dönmüştür (y).

Bazı durumlarda, bir fonksiyondan birden fazla değer döndürmesini bekleyebilirsiniz. Örneğin, girilen sıcaklık celsius değerini ($^{\circ}\text{C}$), fahrenheit ($^{\circ}\text{F}$) ve kelvin ($^{\circ}\text{K}$)’a çeviren bir fonksiyon yazarken, bir girdi (sıcaklığın celsius değeri) ve iki çıktı olacağını söyleyebiliriz.

İşte bu gibi durumlarda eğer bir değişkeni argüman olarak kullanacaksak, başına byref kelimesi ekleriz ve o değişkenin değerini değil adresini kullanırız.

```
Metod1 (){  
  
    int sıcaklık = 34;  
    int f;  
    int k;  
  
    Değiştir (sıcaklık, f, k)  
}  
  
Değiştir( Celcius, Byref Fahrenheit, Byref Kelvin){  
  
    Fahrenheit = Celcius * 1.8 + 32  
    Kelvin = Celcius + 273  
}
```

Optional

Optional, seçeneysel anlamına gelir. Bazı durumlarda, yazdığınız bir program içinde bir argümana otomatik olarak bir varsayılan değer atayabilirsiniz. Yazdığınız argümanı çağırarak olan kişi eğer bir değer belirtmezse, bu argümanın varsayılan değeri kullanılacaktır. Aksi takdirde (yani bir değer belirtmesi halinde), belirtilen değer kullanılacaktır.

```
ÇevreHesapla ( YarıÇap, Optional pi = 3.14){  
....  
}
```

Yukarıdaki prosedürde, pi argümanı optional olarak belirtilmiştir yani bu prosedür çağrılırken, pi argümanı için bir değer belirtilmezse 3.14 değeri kullanılacaktır.

ÇevreHesapla (10)

Gördüğünüz gibi yukarıdaki ifadede, sadece yarıçap argümanı belirtilmiştir. Bu durumda, pi argümanın değeri otomatik olarak 3.13 olur. (bildiğimiz üzere bazı durumlarda pi sayısı 3.14 yerine 3.1415 veya 3 olarakta kullanılabilir)

ÇevreHesapla (10, 3.1415)

Yukarıdaki ifadede ise, pi argümanının değeri 3.1415 olur.

Recursive fonksiyonlar

Bazı fonksiyonlarda, aynı işlem sadece parametreleri değiştirilerek peşpeşe çağırabilir. Yani bir fonksiyon kendi kendini çağırıp belirli işlemleri aynen (Sadce argümanlarını değiştirerek) gerçekleştirebilir.

Varsayalım ki; faktöryel bulan bir fonksiyon yazıyoruz. Matematikten hatırlarsak; $5! = 1.2.3.4.5 = 120$ değeri elde edilir. Faktöryel, 1'den N'e kadar olan sayıların çarpımı anlamına gelmektedir.

Faktöryelin fonksiyonel tanımını yapalım (matematikten);

Faktöryel (X) =	Faktöryel (X-1) . X	Eğer X>0 ise
	1	Eğer X=0 ise

Yukarıdaki fonksiyon tanımından, eğer girilen sayı 0 ise, bu sayının faktöryelinin 1 olduğu eğer sayı 0'dan büyükse, o zaman, girilen sayı ile bir eksiğinin faktöryelinin çarpımı olduğunu anlayabiliriz.

O halde bu fonksiyonu tanımlamak için;

```
Fonksiyon faktöryel (X) {  
    if X = 0 then  
        return 1  
    else  
        return X * faktöryel (X-1)  
    end if  
}
```

Yukarıdaki fonksiyonda, faktöryel kendini çağırmaktadır.

[Ara verme bölümü]

Alıřtırmalar

- Optional nedir?
- Byref nedir?

5.8 Yorum

Şüphesizdir ki, onlarca yüzlerce hatta binlerce satır kod yazdığınızda, hangi işlemin hangi nedenle yapıldığını hatırlamak uzun süre alır. Ayrıca bazı işlemlerin dizilişini ve sırasını unutabilirsiniz. İşte bu amaçlardan ötürü pro-gram kodları içinde, açıklama metinleri kullanılabilir. Bu açıklama metinleri program içine katılmaz, sadece yazdığınız kodda görünür. Programın işleyişine veya çalışmasına etkisi yoktur. Açıklamaları belirtmek için her dilde farklı farklı semboller kullanılır.

Örneğin;

Kod satırı

Kod satırı 2

Kod satırı 3

//Açıklama Kod satırı 4

4. satırdaki ifade bir açıklama yazısıdır. Bir anlamı yoktur ve istediğiniz şekilde, dilde, açıklama yazmanıza olanak sağlar. Bu satırlar, yani açıklama satırları programda önemsenmez, belirttiğimiz gibi sadece kodu okuyan kişiye (size veya başkasına) yazılmış olan açıklamayı gösterir.

Açıklama yazısı tanımlamak için kullanılan semboller: (programlama dillerinin ayrıntılı syntax'ı bu kitabın ekinde mevcuttur)

Kod Satırı 1 //Açıklama satırı 1 - TEK SATIR İFADE

Kod Satırı 2

/* Açıklama satırı 2

Açıklama satırı 3

*/

Kod Satırı 3 'Açıklama satırı 4 - TEK SATIR İFADE

-- Açıklama satırı 5 - TEK SATIR İFADE

Yukarıda gördüğünüz yeşil alanların hepsi açıklamadır. /* ve */ işaretleri: /* işaretinden sonra bir sonraki */ işaretine kadar olan bütün alanı açıklama yapar. İsteddiğiniz kadar satır ekleyebilirsiniz. Bu iki işaret arasındaki bölümde kodlar yazılı olsa bile işlenmez!

Kod Satırı 1 /*Açıklama satırı 1 - TEK SATIR İFADE

Kod Satırı 2 - İPTAL İŞLENMEZ

Açıklama satırı 2

Açıklama satırı 3

*/

Kod Satırı 3

/*Açıklama satırı 4*/

Kod Satırı 4

Açıklama alanları, programı test aşamasında da sıklıkla kullanılır. Bu konu ile ilgili ayrıntılı bilgi ileriki bölümlerde anlatılacaktır.

Alıştırmalar

- Yorumlar ne amaçla kullanılır?

5.9 Matematiksel Fonksiyonlar

İlk bölümden beri matematiğin programlamanın temelini oluşturan bilimlerden biri olduğunu söylemiştim. Şüphesiz matematik üzerine kurulu bir sistem olan programlamada hemen hemen her türlü matematiksel işlemi yapmanız mümkün olacaktır ve bu işlemler için bir çok fonksiyon bulunmaktadır.

ABS Absolute (mutlak değer)

Abs, fonksiyonun açılım hali, absolute yani mutlak anlamına gelir. Anlaşılacağı gibi, girilen bir sayının mutlak değerini döndürür. Daha önce bu fonksiyonu kendimiz yazmıştık ancak programlama dilinde bu fonksiyon var ise, yazmamıza gerek kalmayacaktır.

$$\text{Abs}(3.4) = 3.4$$

$$\text{Abs}(-2,5) = 2.5$$

$$\text{Abs}(0) = 0$$

ACOS Arc Cosinus

Bu fonksiyon girilen cosinus değerinin açısını döndürmektedir. Yani $x = \cos y$ ifadesindeki, x değerini ACos fonksiyonuna girersek çıktı olarak, y değerini alırız. Yani matematikte, $\text{ArcCos}X$ anlamına gelmektedir.

$$\cos 60^\circ = 0.5$$

$$\text{ACos}(0.5) = 60^\circ$$

ASIN, ATAN Arc Sinus, Arc Tangent

Bu fonksiyonlar cosinüste olduğu gibi sırasıyla, sinüs ve tanjant ın değerlerine karşılık gelen açılarını döndürmektedir.

$$\sin 30^\circ = 0.5$$

$$\text{Asin}(0.5) = 30^\circ$$

$$\tan 45^\circ = 1$$

$$\text{Atan}(1) = 45^\circ$$

COS, SIN Cosinus, Sinus

Bu fonksiyonlar, matematikteki, Cosinüs, sinüs, tanjant işlemlerini gerçekleştirmektedir. Girilen açının o fonksiyondaki değerini döndürür.

$$\cos(60^\circ) = 0.5$$

$$\sin(30^\circ) = 0.5$$

Ceiling

Ceiling tavan anlamına gelmektedir. Girilen bir ondalık sayıya en yakın bir üst sayıyı döndürmektedir. Eğer girdi tamsayı ise, girilen sayıyı aynen döndürmektedir.

$$\text{Ceiling}(3) = 3$$

$$\text{Ceiling}(3.4) = 4$$

$$\text{Ceiling}(3.1) = 4$$

$$\text{Ceiling}(34.9) = 35$$

DivMem

DivMem (Bölünen, Bölen, Kalan) : Bu işlem, girilen sayıyı (bölünen), girilen sayıya (bölen) böler. Kalan kısım, Kalan adlı değişkene yazılır, sonuç ise fonksiyonun sonucu olarak döndürülür.

Dim Kalan As Integer
Sonuc = DivMem (14, 3, Kalan)

Burada, sonuç, 4 olacaktır. 14'ün içinde, 3, 4 kere vardır. $3 \times 4 = 12$. Kalan ise, 2 olacaktır.
 $14 - 12 = 2$

Exp

Bu fonksiyon, bir ifadeyi, matematikteki e üssü x halinde yazar.

$$e^x = \exp(x)$$

Floor

Floor, tavan anlamına gelmektedir. Girilen sayıya en yakın, en büyük, sayıyı döndürür. Eğer girilen sayı ondalık ise, o sayının tam sayı kısmını döndürür. Eğer girilen sayı tam sayı ise, o sayıyı döndürür.

Floor (3) = 3
Floor (3.2) = 3
Floor (76.4) = 76

Log

Bu fonksiyon girilen sayının, girilen tabandaki logaritmasını döndürür.

Log(10, 10) = 1
Log (100, 10) = 2

Max, Min

Max fonksiyonu girilen iki değer içinden, eğer X büyükse X'i, Y büyükse, Y'yi döndürür.

Max (3, 4) => 4
Max (15, 5) => 15

Min fonksiyonu girilen iki değer içinden, eğer X küçükse, X'i, Y küçükse, Y'yi döndürür.

Min (3, 1) = 1
Min (34, 9) = 9

Pow

Bu fonksiyon, girilen sayının, girilen üssünü döndürür. Yani matematikteki, Sayı^{Üs} ifadesiyle aynı anlama gelmektedir.

Pow (3, 2) = 9
Pow (2, 3) = 8

Sign

Signum anlamına gelen bu fonksiyon girilen sayı 0 ise, 0, pozitif ise, 1, negatif ise -1 değerini

döndürür.

$\text{Sign}(-3) = -1$

$\text{Sign}(0) = 0$

$\text{Sign}(60) = 1$

Sqrt

Bu fonksiyon girilen sayının kökünü alır. Matematikteki karekök işlemiyle aynı anlama gelir.

$\text{Sqrt}(9) = 3$

$\text{Sqrt}(16) = 4$

Int

Bu fonksiyon, girilen sayının tam sayı kısmını döndürür, ondalık kısmını önemsemez.

$\text{Int}(3.2) = 3$

$\text{Int}(7.9) = 7$

Round

Bu fonksiyon, girilen sayıyı yuvarlar. Matematikteki yuvarlama ile aynı anlama gelmektedir.

$\text{Round}(3.7, 0) = 4$

$\text{Round}(3.2, 0) = 3$

Matematikte bilindiği üzere, bir sayının virgülden sonraki kısmı, 5 veya üzeri ise, sayı bir üste, değilse sayı bir alta yuvarlanır. İkinci argüman olan hane, çıkacak olan sayının virgülden sonra kaç hane devam edeceğini belirtir .

$\text{Round}(3.264, 1) = 3.3$

$\text{Round}(6.256, 2) = 6.26$

5.10 Karakterel ve Metinsel Fonksiyonlar

Asc (char) : Integer

Bir karakterin, karakter kodunu döndürür. Bilgisayarda, her bir karaktere karşılık gelen bir kod bulunmaktadır. Bu karakter değerleri, 0-255 arasındadır. Aşağıdaki tabloda karakterlerin kodları görüntülenmektedir.

65	A	78	N	97	a	110	n
66	B	79	O	98	b	111	o
67	C	80	P	99	c	112	p
68	D	81	Q	100	d	113	q
69	E	82	R	101	e	114	r
70	F	83	S	102	f	115	s
71	G	84	T	103	g	116	t
72	H	85	U	104	h	117	u
73	I	86	V	105	i	118	v
74	J	87	W	106	j	119	w
75	K	88	X	107	k	120	x
76	L	89	Y	108	l	121	y
77	M	90	Z	109	m	122	z

Figür 5.10 A : Karakter tablosu

Bu tabloda belirtilenlerin haricinde, 0-9 arası rakamlar, Türkçe karakterler, noktalama işaretleri de bulunmaktadır.

Asc ("A") => 65

Asc ("a") => 97

Chr (Integer) : Char

Girilen karakter koduna karşılık gelen karakteri döndürür. Yani Asc fonksiyonun tam tersi işlemini yapmaktadır.

Chr (65) => A

Chr (97) => a

LCase (String) : String / UCase (String) : String

LCase, Lower case - küçük harf anlamına gelmektedir. Girilen bir metnin içindeki bütün harfleri küçük harf yapar, diğer karakterleri aynen bırakır.

LCase ("DENEME") => deneme

LCase ("DeNeMe") => deneme

LCase ("deneme") => deneme

LCase ("Deneme 123") => deneme 123

UCase, Upper case fonksiyonu ise, girilen metni, büyük harfler olarak döndürür.

UCase ("deneme") => DENEME

UCase ("Deneme123") => DENEME123

Trim – Ltrim – Rtrim

Trim fonksiyonu, girilen metnin başındaki ve sonundaki boşlukları siler, kalan metni döndürür.

Trim (“ deneme “) => deneme

Trim (“deneme”) => deneme

Trim (“deneme “) => deneme

Trim (“ deneme”) => deneme

LTrim, girilen metnin sadece solundaki boşlukları, RTrim ise, girilen metnin sadece sağındaki boşlukları siler.

LTrim (“ deneme”) => “deneme”

LTrim (“ deneme “) => “deneme “

LTrim (“deneme “) => “deneme “

RTrim (“deneme “) => “deneme”

RTrim (“ deneme “) => “ deneme”

Left / Right

Left fonksiyonu, girilen bir metinden, soldan başlamak üzere, girilen uzunluktaki kısmını seçer ve döndürür.

Left (“deneme”, 1) => “d” //soldan başlayarak 1 karakter seç

Left (“deneme”, 2) => “de” //soldan başlayarak 2 karakter seç

Left (“deneme”, 3) => “den”

Left (“deneme”, 5) => “denem”

Right fonksiyonu da aynı, left fonksiyonu gibi çalışmaktadır ancak, girilen metnin solundan değil sağından başlamak üzere, seçer.

Right (“deneme”, 1) => “e” //sağdan başlayarak 1 karakter seç

Right (“deneme”, 2) => “me” //sağdan başlayarak 2 karakter seç

Right (“deneme”, 4) => “neme” //sağdan başlayarak 4 karakter seç

Len

Len, Length (uzunluk)’in kısaltılmış halidir. Girilen bir metnin uzunluğunu döndürür.

Len (“mustafa”) => 7

Len(“şanslı s”) => 8

Len (“”) => 0

Len (“pala”) => 4

Instr, Indexof

Instr veya indexof fonksiyonları (bazı dillerde instr, bazı dillerde indexof) girilen bir metinde, girilen başka bir metin başlangıç noktasını döndürür.

Instr(“denizin ortasında ne vardır?”, “ne”) => 20 Soldan başlamak üzere, 20. karakter “n”, 21. karakter “e” harfidir.

Yani, “denizin ortasında ne vardır?” ifadesinde, görünen ilk “ne” 20. karakterden başlar ve bu yüzden fonksiyon 20 değerini döndürür.

Instr (“deneme”, “e”) => 2

Burada görüldüğü üzere, ilk “e” metni, 2. karakterde başlamaktadır.

Instr (“menemen”, “me”) => 1

Metin içinde (“menemen”) ilk görünen “me” ifadesi, ilk karakterden başlamaktadır.

Replace

Replace değiştirmek anlamına gelmektedir. Girilen bir metinde (1), bir metni bulur (2), ve başka bir metinle değiştirir. Bu işlem, ofis programlarında, Düzen menüsü altında, Bul/Değiştir özelliğinde kullanılmaktadır.

Replace (“Ahmet ayvayı yedi”, “Ahmet”, “Haluk”) => “Haluk ayvayı yedi”

Fonksiyon, ilk girilmiş olan metin “Ahmet ayvayı yedi”, deki, “Ahmet” i bulup, “Haluk” ile değiştirir.

Replace (“Sınav iyi geçti”, “iyi”, “kötü”) => “Sınav kötü geçti”

Replace (“Ben geldim. Ben seni aramıştım.”, “Ben”, “Ben dün”) => “Ben dün geldim. Ben dün seni aramıştım”

Split

Girilen bir metni, girilen bir ayraca göre, böler ve her bir parçayı döndürür.

Split (“Kavun,Karpuz,Elma,Muz”, “,”)

“Kavun”

“Karpuz”

“Elma”

“Muz”

Bu örnekte, girilen metin, virgüllerden bölünmüştür. Bu fonksiyon, microsoft office excelde, veri menüsü altında, TextToColumns özelliğinde kullanılmaktadır.

Split (“Deneme”, “e”)

“D”

“n”

“m”

Split (“Ayşe, Ahmet ile Mustafa ile Harun ile bayramlaşmadı”, “ile”)

“Ayşe, Ahmet”

“Mustafa”

“Harun”

“bayramlaşmadı”

Join

Join, birleştirmek anlamına gelir. Join array halinde, metinleri girilen birleştiriciyi kullanarak birleştirir. Yani split fonksiyonunun yaptığı işin tersini yapar.

Join ([“Ayşe, Ahmet”, “Mustafa”, “Harun”, “bayramlaşmadı”], “ile”)=> “Ayşe, Ahmet ile Mustafa ile Harun ile bayramlaşmadı”

Join ([“Kavun”, “Karpuz”, “Elma”, “Muz”], “,”) “Kavun,Karpuz,Elma,Muz”

Mid / Substring

Mid ve SubString (bazende SubStr) farklı dillerde kullanılır ve aynı işlevi gerçekleştirir. Girilen bir metinden, girilen karakterden başlamak üzere, girilen uzunluk kadar olan kısmını seçer ve döndürür.

Mid (“deneme”, 1, 2) => “de” 1. Karakterden başla ve 2 karakter seç

Mid (“Ortadoğu”, 3, 2) => “ta” 3. karakterden başla ve 2 karakter seç

Mid (“düşünmek”, 4, 3) => “ünm” 4. karakterden başla ve 3 karakter seç

5.11 Tarih & Saat Fonksiyonları

Tarih ve saat işlemlerini yapmak üzere tasarlanmış olan bir veri türünün olduğundan bahsetmiştik.

Date (Yıl, Ay, Gün)

Dim tarih as New Date (2006, 11, 23)
23 Kasım 2006

Date (Yıl, Ay, Gün, Saat, Dakika, Saniye)
Dim tarih as New Date (2007, 12, 25, 14, 30, 23)
25 Aralık 2007, Saat 14:30:23

Tarih ve saat işlemlerini, bu veri türünü kullanarak yaparız.

DateAdd (DateIntervalType, Number, Date) : Date

Bir zamana, belirli bir gün, ay, yıl, hafta, saat, dakika, saniye eklemek amacıyla kullanılır. Örneğin beş gün sonraki tarihi bulmak veya 3 yıl 12 gün önceki tarihi bulmak amacıyla kullanılabilir.

Tarih = DateAdd(Day, 3, New Date (2006, 3, 4))
4 Mart 2006 tarihine, 3 gün ekle
Tarih : 7 Mart 2006

Tarih = DateAdd (Month, 1, New Date (2003, 2, 17))
17 Şubat 2003 tarihine, 1 ay ekle
Tarih : 17 Mart 2003

Tarih = DateAdd (Week, 2, New Date (2002, 5, 12))
12 Mayıs 2002 tarihine, 2 hafta ekle
Tarih : 26 Mayıs 2002

DateAdd ifadesinden sonra ilk kullanılacak olan ifade, bir enumdur. Bu enum içindeki seçenekler aşağıda görüntülenmektedir.

Enum DateIntervalType

Day	:	Gün
Week	:	Hafta
Month	:	Ay
Year	:	Yıl
Hour	:	Saat
Minute	:	Dakika
Second	:	Saniye

End Enum

DateDiff (DateIntervalType, Date1, Date2) : Integer

Bu fonksiyon girilen iki tarih arasındaki farkı, gü-y/yıl/saat/dakika/saniye olarak döndürür. Yani yukardaki fonksiyonun tersini yapar.

Tarih = DateDiff (Day, New Date(2006,3, 4), New Date (2006, 3, 10))

Tarih : 6

[Ara verme bölümü]

5.12 İşaretler

5.12.1 Parantezler

Matematikten bildiğimiz 3 çeşit parantez bulunmaktadır. Parantezlerin kullanıldığı bir çok yer bulunmaktadır. Örneğin

-Bir kelimenin veya olayın, anlamı veya açıklaması yapılacağı zaman normal parantez içine yazılır. Ahmet evden geliyorken eşyaları (takım çantası, fener, çakmak) aldı.

-Bir madde, öge belirtirken tek parantez kullanılır.

1) Can kurtaran

2) Itfaiye

3) Polis

-Bir sayı aralığı bildirirken

a = [3, 989) a; 3'e eşit veya büyük, 989'dan küçüktür.

.....

Ancak parantezler programlamada sadece ve sadece ikili olarak kullanılır. Yani, ikinci örnekte gösterilen gibi madde listelemede tek parantezin kullanıldığı bir durum söz konusu değildir. Programlama da parantezler, [], {}, () olarak kullanılır, üçüncü örnekteki gibi [) olarak veya benzeri bir şekilde kullanılamazlar. Buradan anlayacağımız gibi, parantezler, programlamada, bir başlangıcı ve bir bitişi olan ifadeler için kullanılır.

Örneğin; bir metodu çağırırken, metodun ismini yazdıktan sonra, () parantezlerini eklemeliyiz. “(“ ın anlamı, “bu metodu çağırırken kullanılacak olan argümanların listesi başladı” demektir. “)” nın anlamı ise, belirtili metodu çağırırken kullanılacak olan argümanların listesi bitti” demektir.

Metod1 (argüman1, argüman2, argüman3)

Ancak bazı durumlarda, metodların argümanları yoktur. Böyle bir durumda ise, içi boş olan bir parantez kullanmalıyız.

Metod2 ()

Normal parantez, koşullu ifadelerde de kullanılmaktadır.

if (değer == 3) Normal parantez haricinde, [] köşeli parantez bulunmaktadır. Bu parantez bazı dillerde array tanımlamamamızı, ve tanımlanmış olan array'e ulaşmamızı sağlar.

```
int a[ ] = new int[3] ;
```

```
a[ 0 ] = 7
```

```
a[ 1 ] = 8
```

Köşeli parantez bazı dillerde, bir sınıfın varsayılan özelliğine ulaşmak için kullanılabilir.

```
List listem = new ArrayList();  
listem.add( "x" )  
listem.add( "deneme" )  
String s = listem [ 1 ] //bu ifade listem.getItem ( 1 ) ile aynı anlama gelmektedir.
```

Tırnak parantezler, bazı dillerde, tanımlanan bir arrayin içindeki nesneleri belirtmek amacıyla kullanılır.

```
Dim isimler as String () = New String () { "Ahmet", "Mehmet", "Hüseyin" }
```

Tırnak parantezler, bazı dillerde ise, blok başlatmak ve bitirmek amacıyla kullanılırlar.

```
for (int i=0; i<4;i++){
```

```
class deneme{}
```

5.12.2 Nokta, İki nokta, Virgöl, Noktalı Virgöl

Yazısal ifadelerde, sayısal ifadelerde kullanılmakta olan çeşitli karakterler veya noktalama işaretleri bulunmaktadır. Şüphesiz programlamada da birçok farklı ifadeyi belirtmek için, semboller veya karakterler kullanılmaktadır.

Noktanın kullanıldığı yerler:

-Bir sınıftan türetilmiş olan bir nesnenin, bir özelliğine, veya metoduna ulaşabilmek için, nokta kullanılır.

Yani, öncelikle, bir nesneyi tanımlarız. Daha sonra, o nesnenin sahip türetildiği sınıfa ait özelliklerden veya metodlardan birine ulaşabilmek için, nesnenin ismini yazdıktan sonra, nokta kullanılır, daha sonra ise, erişmek istediğimiz, özellik veya metodun ismi yazılır.

```
Class Öğrenci  
Isim : String  
Soyisim : String  
Sınıf : Byte  
Şube : Char  
Numara : Short
```

```
void SınıfDeğiştir (Sınıf As Byte, Şube As Char)
```

```
.....
```

```
end
```

```
End Class
```

Şimdi tanımlamış olduğumuz bu sınıftan bir nesne oluşturuyoruz.

```
Dim öğrenci1 as New Öğrenci  
öğrenci1.Isim = "Sencer"  
öğrenci1.Soyisim = "YAZICI"  
öğrenci1.Sınıf = 3
```

öğrenci1.Şube = "A"
öğrenci1.Numara = 401

Yukarıdaki örnekte, öğrenci1 adında bir değişken tanımlanmıştır. Bu değişken, öğrenci sınıfından türemiş bir obje'ye point etmektedir. 1. satırdan sonraki satırlarda bütün satırlarda, kullanılmakta olan nokta, o değişkenin tanımlandığı sınıfın, birer özelliğine ulaşmak için kullanılmıştır.

Aynı şekilde, bir sınıfa ait metodlara ulaşabilmek için yine, nokta kullanılmaktadır.

öğrenci1.SınıfDeğiştir (3, "B")

Yukarıdaki ifade de kullanılan nokta ile akabinde yazılan metod ismi ile birlikte o metod çağrılmış olur.

-Bunun haricinde, nokta, ondalık sayıları belirtmek için kullanılmaktadır.

dim i as double

i = 3.4

ifadesinde kullanılan nokta, aslında Türkiye'de matematikte ondalık sayıları ifade etmek için kullanılan virgölün yerine yazılmıştır. Biz ondalık sayıları okurken 3,4 (üç virgöl dört) şeklinde okuruz. Halbuki programlama da 3.4 (üç nokta dört) şeklinde ifade edilir.

-Bunun haricinde, nokta, dillerde özel anlamlar ifade edebilir. Bu ifadeler, kitabın ekinde mevcuttur.

"İki nokta üst üste"nin kullanıldığı yerler:

-İki nokta üst üste ifadesi, daha önce gördüğümüz goto veya jump ifadelerinde bildiğimiz label (etiket)larda kullanılır. Prosedür içinde bir label belirtmek için iki nokta üst üste kullanılır.

etiket1:

etiket2:

-Bazı programlama dillerinde ise, eğer iki veya daha fazla ifadeyi bir satırda yazmak istiyorsak noktalı

virgöl kullanırız.

Dim s as string : dim k as integer : k = 3

-Bazı programlama dillerinde, bir namespace'e ait olan bir sınıf tanımlanacağı zaman iki tane, iki nokta üst üste kullanılır. Öncelikle namespace 'in adı, daha sonra, iki tane, iki nokta üst üste yazılır. Sonrada kullanılacak olan sınıf / kalıp veya arayüz ismi yazılır.

Örneğin

Namespace Uzay

Class Jupiter

End Class

End Namespace

şeklinde

tanımlanan namespace ve sınıf için; yapacak olduğumuz bir tanımlama;

Dim x as Uzay::Jupiter

şeklinde yapılır.

Bazı programlama dillerinde ise, yine sınıfların özelliklerin ve metodlarına ulaşmak için kullanılan nokta kullanılır.

Dim x as Uzay.Jupiter

Virgülün kullanıldığı yerler: Virgüller program içinde, argümanlar arasında, arraylerin her bir nesnelerinin arasında kullanılır. Virgül kullanımı hemen hemen her dilde aynıdır. Bütün dillerin argümanlarının arasında virgül bulunur.

void SınıfDeğiştir (Sınıf As Byte, Şube As Char)
dim i as String() = new String () {"deneme", "test"}

Noktalı virgülün kullanıldığı yerler:

Bazı dillerde, her bir ifadeden sonra, noktalı virgül kullanmak bazı dillerde de her ifadeden sonra yeni bir satıra geçmek gerekmektedir. Yani program derleyicileri, bir ifadenin bittiğini, ya noktalı virgülden yada yeni bir satıra geçilmesinden anlamaktadırlar. Noktalı virgül sadece bir ifadenin bittiğini belirtmek amacıyla kullanılmaktadır.

int i=0;

5.13 Dosya işlemleri / Program

Bir program, programlanmış olan kodu, çalıştırıp, hizmet edeceği amacı gerçekleştiren bir uygulamadır. Bazı programlar, çalıştıklarında, dışardan veri okur veya dışarı veri yazarlar. Yazdıkları bu veriye de dosya denir.

Yani dosyalar veriler – değerlerdir, programlar da onları işleyen fonksiyonlardır diyebiliriz.

F(x) işleminde, F fonksiyonu programı, x değeri de dosyayı temsil eder.

Programlar olmaksızın dosyaları görüntülemek, işlemek, değiştirmek imkansızdır. Dosyaları kitaplar, programları beynimiz gibi düşünelim. Beynimizi kullanarak kitapları okuruz ve o bilgileri işleriz. Ya da beynimizi kullanarak kitap yazabiliriz. Beynimiz - yada insan beyni olmadan kitapların bir anlamı yoktur. Burada dikkat edilmesi gereken nokta, kitaplar olmasada (ilk insanları veya yazının keşfinden önceki yılları düşündüğümüzde) insan beyni yine de bazı bilgiler üretebilecek kapasiteye sahiptir.

İşte aynı beynimizin ve kitaplar arasındaki ilişki gibi, Word'u açıp, daha önceden yazmış olduğumuz veya başkası tarafından yazılmış olan bir dosyayı açıp inceleyebiliriz. Dosyadaki bilgileri okuyup anlayabiliriz (ve işleyebiliriz). Bununla birlikte, önceden çalıştığımız bir dosya olmasa bile, Word'u kullanarak bir dosya oluşturabiliriz. Buradan anlaşılacağı gibi, Word bir programdır, fakat word dökümanları birer dosyadır.

Her dosyanın formatının bir uzantısı bulunmaktadır (Bazı dosya tipleri aynı uzantıyı kullanabilir). Uzantılar genelde 3 harflidir. Örneğin Word dökümanlarının uzantısı "DOC" tur. Bir dosyayı ifade ederken, dosyanın yolu, dosyanın ismi ve dosyanın uzantısı şeklinde tanımlarız.

Programların uzantısı, com ve exe'dir. Com uzantılı dosyalar, Microsoft Dos için tasarlanmış programlardır. Exe'ler ise, şuanda kullanmakta olduğumuz onlarca programın formatıdır. Örneğin, Word, Excel, Paint, Winamp.....

Çok çeşitli formatta dosya bulunmaktadır. Bunlar içinde, txt (metin dosyaları), bmp (resim dosyaları bitmap), jpg (resim dosyaları) ve bunun gibi standart formatlar olduğu gibi, programların,kendi dosya formatları da olabilir. Örneğin, autocad çizim programının kendi formatı vardır ve uzantısı dwg'dir veya Powert Point programının formatının uzantısı ppt'dir.

Dosya işlemleri

Dosyaların verileri saklamak için kullanılan veri dökümanları olduğunu belirtmiştim. Ayrıca bilgisayar programlarının da verileri işlediğini, yani verileri diske veya hafızaya yazdığını, diskten veya hafızadan okuduğunu ve benzeri işlemleri gerçekleştirdiğini belirtmiştim. Bu kadar yoğun bir şekilde veri işlemesi üzerine çalışan bir sistem olan programlamada, dosya işlemleride çok büyük rol oynamaktadır.

Daha öncede, dosyaların bir deftere benzetilebileceğini belirtmiştim. Peki, bir defter üzerinde işlem gerçekleştirmek için ne gibi bir prosedür uygulanır? Diyebiliriz ki;

- defter açılır, gerekli sayfalara gidilir, gerekli sayfa okunur, defter kapatılır

- defter açılır, en son sayfaya gidilir, gerekli bilgiler yazılır, defter kapatılır

Yukarıdaki ifade gerçek dünyada gerçekleşen günlük işlemlerden ikisidir. Yukarıdaki ifadede bulunan, “defter” kelimelerinin hepsini “dosya” ile değiştirelim.

- dosya açılır, gerekli sayfalara gidilir, gerekli sayfa okunur, dosya kapatılır
- dosya açılır, en son sayfaya gidilir, gerekli bilgiler yazılır, dosya kapatılır

Satır ve sayfa

Satır, kelimelerin, yazılması sonucunda yatay olarak oluşan, karakterlerden oluşan bir dizidir. Peki neden satır’a gerek duyulmuştur? Şüphesiz birkaç kelime tutan dökümanlar olduğu gibi on binlerce sayfa tutan dökümanlarda bulunmaktadır. Eğer yanyana binlerce – milyonlarca kelimeyi yazarsanız, yazıyı yazıyor olduğunuz sayfanın uzunluğu metreler olurdu. Bir ansiklopedide yaklaşık 500 sayfa yazı bulunduğunu, her sayfada 4 kolon bulunduğunu, her kolonda, yaklaşık 20 harf bulunduğunu ve toplamda satır olarak bir sayfada 60 satır olduğunu varsayarsak; bir ciltte; 2.400.000 karakter olduğunu söyleyebiliriz. Bu kadar çok sayıda karakteri satır kullanmadan yanyana yazsa idik; 6 km uzunluğunda ve 2-3 cm yüksekliğinde bir kağıda gereksinim duyardık. Aynı durum sayfalar içinde geçerlidir. Eğer bir cilt içindeki bütün bilgileri tek bir sayfaya yazmaya kalksak; 150 m yüksekliğinde ve 20 cm genişliğinde bir ansiklopedi cildimiz olurdu.

İşte böyle çok büyük boyutlarda ciltler taşımak yerine, boyut probleminden dolayı (insanlar yukarıda bahsedilen boyutlardaki dökümanları veya kitapları hem taşıyamaz hem okuyamaz) satırlar ve sayfalar kullanılır.

Halbuki, bilgisayarlar, boyut kapsamı bakımından çok geniş olduğundan ve zaten geniş olan bu boyut (sabit disk) çok düşük fiyatlarla daha da çoğaltılabileceği için, satır ve sayfa gibi tanımlara gerek bulunmamaktadır. Yani bir bilgisayar dökümanında bütün karakterler, tek bir sayfada ve tek bir satırda ifade edilir. Daha doğrusu, kaydedilirken veya işlenirken bu şekilde tek bir sayfa ve satır üzerinde işlenir. Ancak kullanıcıya gösterilirken, belirtili noktalardan bölünerek satırları ve sayfaları oluşturur (monitöründe bir genişliği olduğu için).

İşte bu nedenle, yukarıdaki ifadede geçen “.... gerekli sayfalara gidilir....”, “sayfa” kelimelerini nokta olarak değiştirebiliriz.

- dosya açılır, gerekli noktalara (bölümlere) gidilir, gerekli noktalar (yani bölümler) okunur, dosya kapatılır
- dosya açılır, en son bölüme gidilir, gerekli bilgiler yazılır, dosya kapatılır

Bilgisayardaki dosya işlemleri de aynı yukarıda belirtildiği gibi gerçekleşmektedir. Yani bir dosya ile işlem gerçekleştirebilmek için, öncelikle o dosya açılır, daha sonra işlem gerçekleştirilir ve işlemin bitmesinden sonra dosya kapatılır.

Tabi dosya işlemlerinde dikkat edilmesi gereken bir durum söz konusudur. Açtığınız bir dosya, kapatılana kadar, başka bir program tarafından açılmaz. Yani, siz dosyadan okuma yaparken veya veri girerken, başka bir işlem yapmak üzere belirtili dosyayı açamazsınız.

Yaklaşık 10-15 yıl önce, bir tanıdığımızın, peynir deposu vardı. Bu depo, yerin 1 kat altındaydı ve aşağıya inmek için bir asansör bulunuyordu. Bir süre sonra asansör takılmaya başlamış ve orada bulunan bir yetkili, asansörün taşıma işlemini yapan dişlerini temizlemek için, kaldırma işlemini gerçekleştiren dişleri, asansörün üzerindeyken, açmış. Tabi asansörü taşıyan bu dişler boşalınca, asansörün tabanı ve bu işi yapan kişi bir kat aşağıya düşmüş. Hatta bu kişi bacağını kırmış.

Belirtmeye çalıştığım, bir dosya üzerinde, bir işlem yaparken, diğer bir işlemi gerçekleştiremezsiniz. Yukarıdaki örnekte, asansörün, temizlemek üzere açılmasından dolayı, asansör, üzerinde bulunan kişiyi, “kaldırma” eylemi gerçekleştirememiş ve düşmüştür.

Aynı şekilde, bir defteri veya kitabı bir kişi okurken herhangi bir problem oluşmaz. Ancak aynı kitabı iki kişi okumaya ve farklı sayfalarını okumaya kalkarsa, karışıklık olacaktır. Varsayalım ki, A kişisi, bir sayfayı okuyor, aynı anda B kişisi, başka bir sayfayı okuyor. Bu işlemin normalden daha zor olacağını rahatlıkla tahmin edebiliriz. Belki, aynı sayfayı okuyorsanız, bu durum pek problem olmayabilir ancak yinede okuyucular yeterince rahat olmayacaktır.

İşte programlamada da, aynı şekilde, bir dosya, bir program tarafından açılmış ve kullanılıyorken, başka bir program veya aynı program, açılı olan bir dosyayı tekrar açıp işleyemez.



Figür 5.13 A : Bir dosyanın aynı anda açılmaya çalışılması

Dosya işlemleri, tahmin edeceğimiz gibi, veri ekleme, veri silme, veri değiştirme ve veri görüntüleme olmak üzere dört adettir.

5.14 Keyword (anahtar sözcük)

Bilgisayar dünyasında bir anahtar sözcük, kullanılan programlama dilinde özel bir anlamı olan bir ifadedir. Bir konuşma dilinde on binlerce kelime bulunur. Biz bu kelimelerin bir kısmını bilmeli yani tanımalıyız ki o dili konuşabilelim. Kelime türleri arasında isimler bulunur ve zaten bir dilin kelimelerin %50'ye yakını isimleri ifade eder. İsimlerde kendi aralarında ikiye ayrılır:

- özel isim
- cins isim

Örneğin, yabancı bir dil öğreniyorsak, ve “john” kelimesinin bir özel isim olduğunu bilmiyorsa, sözlüğe bakarız, ancak yine anlamını bulamayız. Bu ayrımı yapmak için, özel isimler büyük harfle yazılır. Ancak almancada bütün isimler büyük harfle yazılacağı için, almanca bir isim gördüğümüzde bunun bir insan – şehir .. gibi özel bir isim mi yoksa, masa, sandalye... gibi bir isim olup olmadığını anlayamayız.

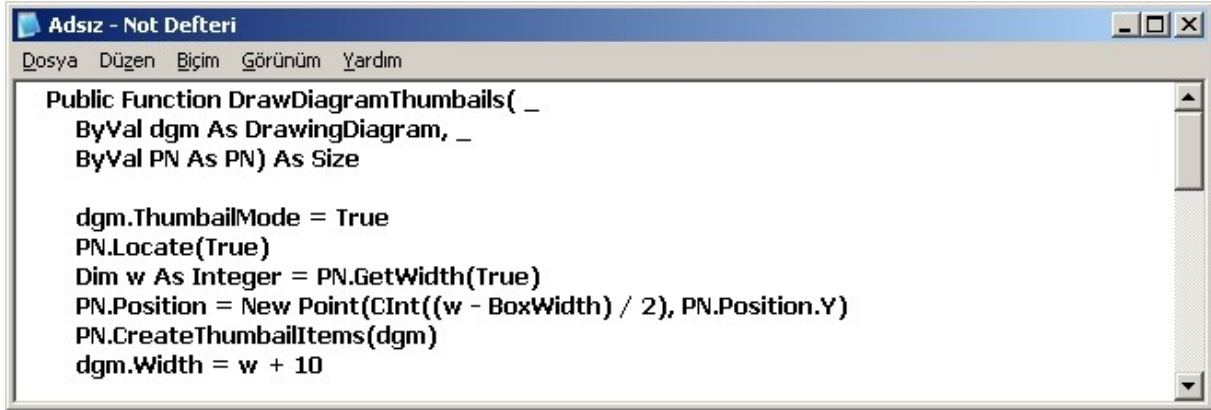
İşte böyle bir durumda o dile ait olan isimleri bilmemiz gerekir ki, karşılaştığımız bu ismin bir özel isim olduğunu anlayabilelim.

Aslında, anahtar sözcük bir tür terimdir. Hemen hemen her bilgisayar programında, birden çok anahtar sözcük bulunmaktadır. Bu anahtar sözcükler kullanıldıklarında, birer anlam ifade etmektedirler. Bazı dillerde anahtar sözcükler benzer veya aynı olabilir veya benzer kullanılabilir. Bazen, ezberlediğimiz bir anahtar sözcük başka dilde de kullanılabilir ve bizde bir daha aynı anahtar sözcüğü ezberlemek zorunda kalmayız. Aşağıda bir dilin kullandığı anahtar sözcüklerin listesi bulunmaktadır.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	int	short	try	extends
char	final	interface	static	void
class	finally	long	strictfp	super
const	float	native	while	

5.15 IDE (Integrated development environment)

Programlama yapmak için aslında ihtiyacımız olan üç araç bulunmaktadır. Bunlardan ilk ikisi daha önceki bölümlerde bahsettiğimiz derleyici ve yorumlayıcı araçlarıdır. Bu iki araç olmaksızın yazdığımız kodun derlenmesi ve yorumlanması ve sonuç olarak programa dönüşmesi imkansızdır. Daha önce bu konu üzerine bilgi verdiğim için, detaylı olarak tekrarlamadan, kodlama için gerekli olan üçüncü aracı anlatacağım. Üçüncü aracımız, kodu yazacağımız bir metin editörüdür. Yani kodu yazmamız ve kayıt etmemiz için gerekli olan araçtır. Bu araca en basit örnek olarak, not defteri uygulamasını verebiliriz.



Figür 5.15 A : Bir kod dosyası

Ancak programlama git gide karışan ve karışıklığı artan bir şekilde vaktinizi boşa harcatan bir uğraştır. Kodlar siz yazdıkça karışık bir hal alır.

Kodların karışmaması ve göz tarafından kolay görünebilir – ayırt edilebilir olması için anlaşılır olması gerekmektedir. İşte bu nedenle kodların hep bir hizada olması amacılığıyla, fixed-size yazı tipleri yani aynı genişliğe sahip harfler içeren yazı tipleri kullanılır. Bu yazı tiplerinde, en geniş “m” harfi ile en dar “i” harfi bile aynı genişliğe sahiptir. Normal bir yazı tipine baktığınızda;

m
i

bu iki harfin genişlikleri birbirinden farklıdır. Hatta o kadar fark bulunur ki, 3 tane “i” harfinin toplam genişliği ancak bir tane m harfine denk gelmektedir.

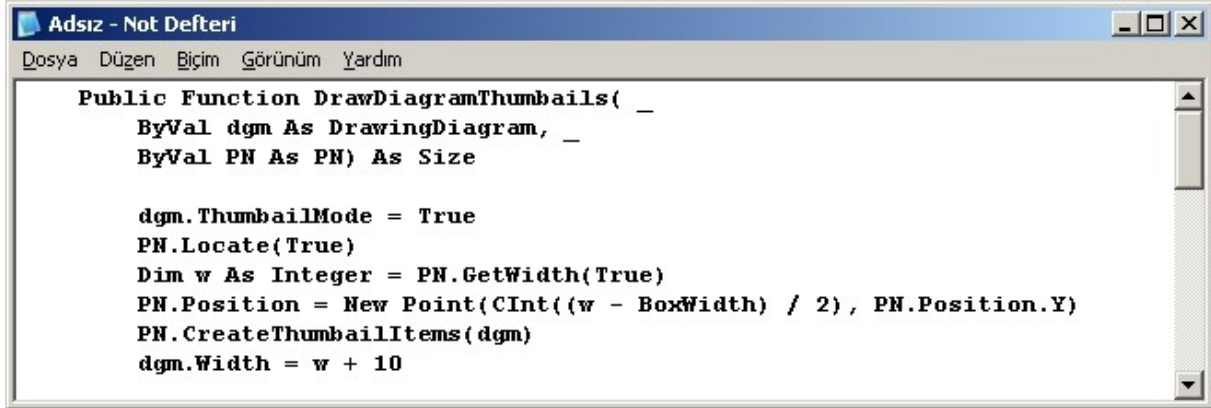
m
iii

Fakat sabit genişliğe sahip yazı tiplerinde durum farklıdır. Harflerin genişlikleri aynıdır.

m
i

abcdefghijklmnoprstuvwxyzqw

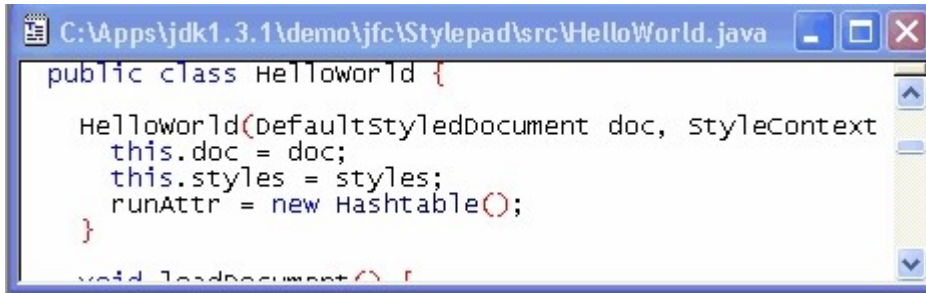
Yukarıdaki harflerin hepsinin genişliği aynıdır ve kod bu veya benzer bir yazı tipi ile yazıldığında, bütün satırdaki bütün karakterler, yukarıdan aşağıya aynı hizada olacaktır. Bu nedenle, kod yazarken, fixed size yazı tipi kullanmalıyız. Fixed size yazı tiplerine örnek olarak; Terminal, Fixedsys, Courier, Courier New, Lucida Console, Andale Mono yazı tiplerini verebiliriz.



```
Public Function DrawDiagramThumbnails( _  
    ByVal dgm As DrawingDiagram, _  
    ByVal PN As PN) As Size  
  
    dgm.ThumbnailMode = True  
    PN.Locate(True)  
    Dim w As Integer = PN.GetWidth(True)  
    PN.Position = New Point(CInt((w - BoxWidth) / 2), PN.Position.Y)  
    PN.CreateThumbnailItems(dgm)  
    dgm.Width = w + 10
```

Figür 5.15 B : Bir kod dosyası

Programlama çoğu zaman (bir noktadan sonra) o kadar karmaşık bir hâl alır ki, kodlar birbirine karışır ve hangi kodun nerede yazılı olduğunu, bir kodun hangi kod tarafından kullanıldığını, kodun hangi satırında ne olduğunu hatırlayamazsınız. İşte burada, yazılan kodları, biraz daha anlaşılır ve farkedilir hale getirmek için, daha gelişmiş metin editörleri kullanılabilir.



```
public class HelloWorld {  
    HelloWorld(DefaultStyledDocument doc, StyleContext  
        this.doc = doc;  
        this.styles = styles;  
        runAttr = new Hashtable();  
}  
  
void loadDocument() {
```

Figür 5.15 C : Başka bir editörde gösterilmiş kod dosyası

Yukarıdaki figürde bir editör içine yazılmış olan birkaç satırlık kod görünmektedir. Dikkat ederseniz bu kodda, parantezler, anahtar sözcükler farklı renklerle ifade edilmiştir. Bu renk ayrımı, gözümüzün, belirli bölümleri kolayca anlaşılmasını, anahtar sözcükler – parantezler ve işaretlerin, dolayısıyla, blokları belirlemesini kolaylaştırır.


İşi biraz daha ilerletip, spaghetti kod yazdığınızda, bu kodu otomatik olarak düzenleyen ve bulunduğu bölgeye göre, gerekli sayıda satır başı ekleyen bir özellik eklersek işimiz daha da kolaylaşacaktır. Bu özelliklere, birde, hata ayıklamaya yardımcı olan bir araç eklediğimizde, elde edeceğimiz sonuca IDE denir. Entegre Uygulama Geliştirme Ortamı'nın kısaltılmış halidir.

Girişte belirttiğim gibi aslında sadece bir metin editörü kodlama alanı olabilmektedir ancak, kod ile uğraşmak istemiyorsanız ve yazdıklarının daha anlaşılır olmasını istiyorsanız ayrıca zamandan tasarruf etmek istiyorsanız, IDE kullanmanız gerekmektedir.

Bir IDE genelde 4 veya 5 bölümden oluşmaktadır.

Bunlardan ilk ikisi zaten olması gereken Derleyici ve Yorumlayıcıdır. Biz kodu yazdıktan sonra, o kodu bir program haline getirebilmek için yorumluyucu ve derleyiciyi kullanarak bir dizi işlem yaparız (komut satırında). Bu işlemler içinde kullanılan bütün dosyaların derleyiciye belirli parametrelerle gönderilmesi ve ardından yine belirli parametrelerde yorumlayıcıya gönderilmesi bulunur. IDE, üstümüzden bu yükü alarak, sadece bir tuşa bastığımızda, bu gerekli işlemleri bizim yerimize yapacaktır.

Bir sonraki araç ise “kaynak kod editörü”dür. Yani kodları yazdığımız bölümdür.

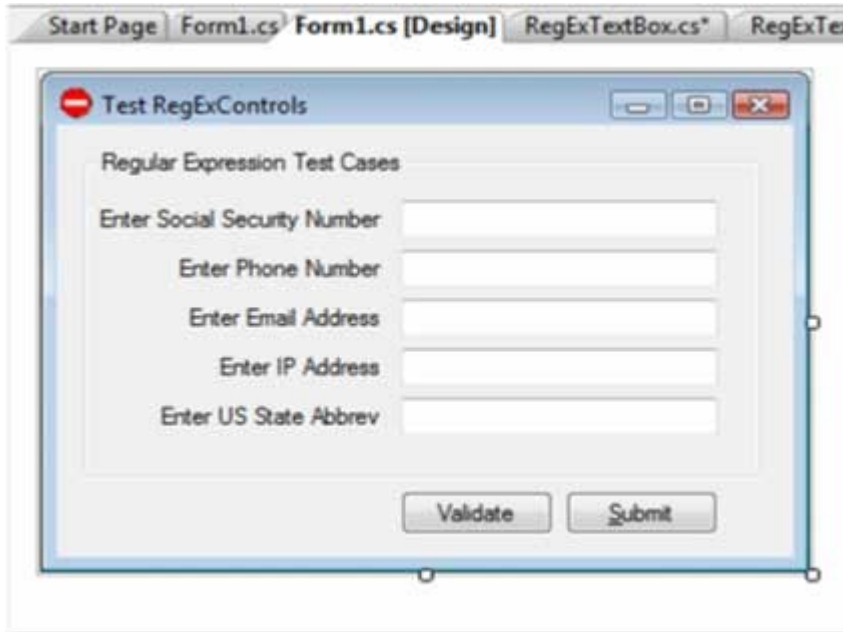


```
6 class Program : Object
7 {
8     static int _I = 1;
9
10    /// <summary>
11    /// The quick brown fox jumps over the lazy dog
12    /// THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
13    /// </summary>
14    static void Main(string[] args)
15    {
16        Uri IllegalUri = new Uri("http://packmyboxwith/jugs.html?q=five-dozen&t=liquor");
17        Regex OperatorRegex = new Regex(@"\S#$", RegexOptions.IgnorePatternWhitespace);
18
19        for (int 0 = 0; 0 < 123456789; 0++)
20        {
21            _I += (0 % 3) * ((0 / 1) ^ 2) - 5;
22            if (!OperatorRegex.IsMatch(IllegalUri.ToString()))
23            {
24                Console.WriteLine(IllegalUri);
25            }
26        }
27    }
28 }
```

Figür 5.15 D : Bir IDE’nin kod yazım bölümü

Dördüncü araç, debugger dediğimiz hata ayıklayıcı araçtır. Bu araç hataları tespit etmek, hataların nerede oluştuğunu – neden oluştuğunu göstermek için ve oluşan hataların düzeltilmesi için kullanılır.

Son araç ise, tasarım bölümüdür. Hazırlamakta olduğunuz programın kodunu, kaynak kod editörüne yazarsınız ve program içinde kullanılacak olan pencereleri de (aşağıda görünüyor) tasarlama ekranında tasarlıyorsunuz.



Figür 5.15 E : Bir IDE'nin tasarım bölümü

5.16 Timer

Timer, Türkçesi zamanlayıcı, belirli bir işlemi belirli bir süre aralıkla, gerçekleştirmeye yarayan bir araçtır. Belirtilen bir prosedür içindeki işlemleri, belirtilen aralıkta (saniye veya salise cinsinden) gerçekleştirir yani belirtili prosedürü vakti geldiği her seferde çalışması için çağırır. Daha açık bir ifade ile, belirtili süre bekler, belirli prosedürü çağırır, sonra yine belirli bir süre bekler ve yine belirtili prosedürü çağırır.

Bu gibi bir işleme örnek olarak, periyodik olarak yaptığınız her tür işlemi verebilirsiniz. Bölüm 2.4'te program – durum ilişkisini anlatmıştım ve bir programı, belirli bir X durumunda, Y işlemi gerçekleştirir gibi talimatlar belirterek, karşılaşılabileceği durumlar için tembihleriz yani programlarımız demiştim. Ancak bazen, her ne durum olursa olsun, yani bir durum belirtmeksizin, belirli aralıklarla bir işlemi gerçekleştirmek istiyorsak Timer kullanırız. Timer genelde kontrol veya toplu işlemler için kullanılır.

Bir hatırlatma programı yaptığımızı ve bu programa, “beni saat 15:25'te uyar” emrini verdiğimizizi düşünelim. Programın bu işlemi gerçekleştirmesi için her 60 saniyede bir saatin 15:25 olup olmadığını kontrol etmesi gerekmekte ve eğer kontrol sırasında saat 15:25 ise, kullanıcıya bir mesaj vermesi gerekmektedir.

İşte böyle bir durumda, 60 saniyelik periyotla, (örnek olarak) “AlarmSaatiniKontrolEt” adlı bir prosedürü çağırmanız ve alarmı kontrol etmemiz gerekmektedir. Tabiki alarm uyarısı verildikten sonra Timer'ın çalışmasına gerek bulunmamaktadır. Yani alarm uyarısı verildikten sonra Timer'ı kapatabiliriz.

Timer'lar programlar içinde, belirli periyotlarla yapılan kontrol – takip gibi her tür işlem tarafından kullanılırlar.

ÇEVİRME	Hatırlatma_1'i hatırlayalım, amacımız, gerçek dünyada metinsel olarak ifade ettiğimiz bir işlemi bilgisayar içine programsal kod olarak aktarabilmek. Gerçek dünyada, “... şu kadar sürede bir bu işlemi gerçekleştir v.b ifadelerini kullandığımız durumlarda, programlamadaki timer'ı kullanırız ve belirtilen işlemlerin, belirtilen aralıklarla sürekli olarak gerçekleştirilmesini sağlayabiliriz.
----------------	--

Çeviri 9 : Timer

5.17 Enum

Enum, bir sınıf gibi, özet veri türüdür. Enum, enumeration 'ın kısaltılmış halidir. Enumeration, sıralama sayma anlamına gelmektedir. Çoğu zaman, hayatta kısaltmalar kullanmak için, maddeleri veya öğeleri numaralandırırız.

MADDE 1. – Türkiye Devleti bir Cumhuriyettir.

MADDE 2. – Türkiye Cumhuriyeti, toplumun huzuru, millî dayanışma ve adalet anlayışı içinde, insan haklarına saygılı, Atatürk milliyetçiliğine bağlı, başlangıçta belirtilen temel ilkelere dayanan, demokratik, lâik ve sosyal bir hukuk Devletidir. ...

Maddeleri, öğeleri, bir kere tanımladıktan sonra, bir daha o madde, öğe hakkında bahsederken, sadece numarasını söylememiz yeterli olacaktır. Yani, bir konuda, iki tarafında bildiği bir liste bulunuyorsa ve bu iki taraf birbirleri arasında haberleşecek ise, listedeki maddeler yerine, maddelerin numaralarıyla da karşı tarafa anlatması mümkündür.

Bazı anketlerde, eğer bir sorunun cevabı veya bir bilginin karşılığı olarak, metinsel, sayısal veya tarihsel bir değer değilse, seçeneysel bir değer var ise, işte bu enum'un en iyi örneğidir.

Hangi şehirde çalışıyorsunuz?

Istanbul
Istanbul
Izmir
Ankara

Figür 5.17 A : Bir seçenek örneği

DERŞANEMİZİ TERCİH ETME NEDENLERİNİZ	
Okuldaki öğretmenlerimin tavsiyesi	<input type="checkbox"/>
Okuldaki idarecilerimin tavsiyesi	<input type="checkbox"/>
Derşanenin bina yapı	<input type="checkbox"/>
Derşanenin öğretmen kadrosu	<input type="checkbox"/>
Arkadaşlarımın bu derşaneye geliyor olması	<input type="checkbox"/>
Kayıt görüldüğünde ilna olmam	<input type="checkbox"/>
Reklam ve afişlerden etkilenmem	Derşanenin hedef kitlelerine uygun olmam
Ücretin uygunluğu	<input type="checkbox"/>
Derşanenin bulunduğu semt	Daha önce bu derşanede sınava girmem
Ulaşımın kolay olması	<input type="checkbox"/>
Derşanenin fiziksel koşulları	Ödüllü sınavdan indirim kazanmam
Derşanenin imajı	<input type="checkbox"/>
Derşanenin başarıları	Teknolojik ve bilimsel yenilikler yapılması
Derşanenin yayınları	<input type="checkbox"/>
	Derşanede bir yakınımın olması
	<input type="checkbox"/>
	Önceden bu derşanenin öğrencisi olmam
	<input type="checkbox"/>
	Derşanenin öğrencilerinin tavsiyesi
	<input type="checkbox"/>
	Ailemin veya yakınlarımın isteği
	<input type="checkbox"/>

Figür 5.17 B : Bir seçenek örneği

Bu olaya verebileceğimiz en iyi örnek telefon numaralarıdır. Bir kişiyi arayacağımız zaman ismini - oturduğu yeri değil, telefon numarasını gireriz ve o numaraya karşılık gelen kişi aranır. Başka bir örnek olarak, mahkemelerde, avukatlar, ceza yasasına ait konulardan bahsederken, “Türk Ceza Kanununun 15. maddesi uyarınca,” gibi ifadeler kullanırlar. Tabi birde, okullarda yoklama yapılırken, “Emre”, “Ahmet”, “Özge” ... gibi isimlerin hatta bazen soyisimlerin (aynı isimdekiler için) kullanılması yerine, “427”, “542”, “403” gibi numaralar kullanılır.

Varsayalım ki, bir öğrencinin öğrenim durumunu gösteren bir veri tipi hazırlamak istiyoruz. Bu öğrencinin öğrenim durumu için birkaç seçenek bulunmaktadır: İlköğretim, Lise, Üniversite, Yüksek lisans. Böyle, bir değişken için, seçenekler bizim tarafımızdan belirleniyorsa, enum kullanırız.

```
Enum ÖğrenciÖğrenim
    İlköğretim
    Lise
    Üniversite
    YüksekLisans
End Enum
```

Yukarıdaki ifade, bir değişken veri türü olan ÖğrenciÖğrenim'i belirtir. ÖğrenciÖğrenim türünde tanımlanmış bütün değişkenler bu ifade de Enum ÖğrenciÖğrenim ile, End Enum arasındaki değerlerden birini alabilir.

```
Dim öğrenim as ÖğrenciÖğrenim
öğrenim = İlköğretim
```

Yukarıdaki örnekte, öğrenim adında bir değişken tanımlanıyor, öyle ki bu değişkenin türü ÖğrenciÖğrenim'dir. Daha sonra, bu değişkenin değerini, olası olasılıklardan biri olan İlköğretim olarak atama yapıyoruz.

Bu işlemi aslında, bir değişkene GLOBAL bir constant değeri atamak olarak gösterebiliriz. Yani varsayalım ki, değeri 1 olan, ismi İlköğretim olan, bir global constant bulunuyor ve biz bu constantın değerini öğrenim adlı değişkenimize atama yapıyoruz.

```
Global Const İlköğretim as Byte = 1
Global Const Lise as Byte = 2
Global Const Üniversite as Byte = 3
Global Const YüksekLisans as Byte = 4
```

```
Daha sonra,
Dim öğrenim as Byte
```

```
öğrenim = İlköğretim
```

Yukarıdaki örnekte, global constantlardaki değerler, tanımlanmış değişkene atanır. İşte, tek tek global constant tanımlayıp hepsine değer atamaktansa böyle durumlarda enum kullanırız. Ayrıca, enum kullandığımızda, bir gruba ait olan öğe diğer grupla karışmayacaktır. Örneğin; bir sınavın zorluğunu tanımlamak için global constantlar kullanalım.

```
Global Const Kolay As Byte = 1
Global Const Normal As Byte = 2
Global Const Zor As Byte = 3
```

Birde, peynirin yağlılık oranını tanımlamak için global constantlar kullanalım.

```
Global Const Yağsız As Byte = 1
Global Const AzYağlı As Byte = 2
Global Const Normal As Byte = 3
Global Const Yağlı As Byte = 4
```

Dikkat ederseniz aynı isimde, iki farklı değişken kullandık. Bu durum program için bir

tanımlama hatası teşkil edecektir. O yüzden program, bu değişkenlerden birini silmemiz gerektiği uyarısını verecektir.

Dikkat edersek, ilk constantlar arasındaki Normal'in değeri 2, ikinci constantlar arasındaki normal'in değeri 3'tür. Eğer birini silersek, diğerinin değeri okunacağından dolayı, hata oluşacaktır, karışacaktır.

İşte enum, bütün bu problemleri ortadan kaldırmak için tasarlanmıştır. Enum içindeki öğelere kendimiz değer ataması yapabiliriz

```
Enum Lastik  
YüksekKalite = 3  
DüşükKalite = 6  
OrtaKalite = 7
```

```
End Enum
```

Enumların değerleri her zaman sayısalıdır. Sayısal veri türleri içinde, Byte, Short, Integer ve Long türünde tanımlanabilir.

```
Enum Lastik As Byte  
YüksekKalite = 3  
DüşükKalite = 6  
OrtaKalite = 7  
End Enum
```

```
Enum Motor as Short  
V6_3000CC = 3000  
V6_2000CC = 2000  
End Enum
```

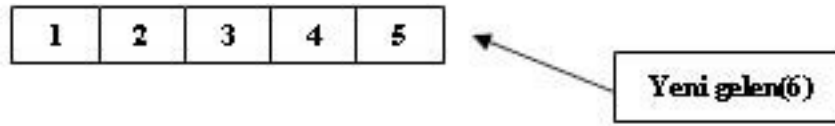
5.18 Programsal sınıflar

İkinci bölümde, programlama dillerinin bizim için bazı işlemleri yaptığını ve kolaylaştırdığını söylemiştim. Beşinci bölümde, programlama dilleri tarafından ortak olarak kullanılan fonksiyonları işlemiştik. Aynı şekilde, programlama dilleri, çok sık kullanılan bazı ortak sınıflar (aynı işlevi gerçekleştiren) içermektedir.

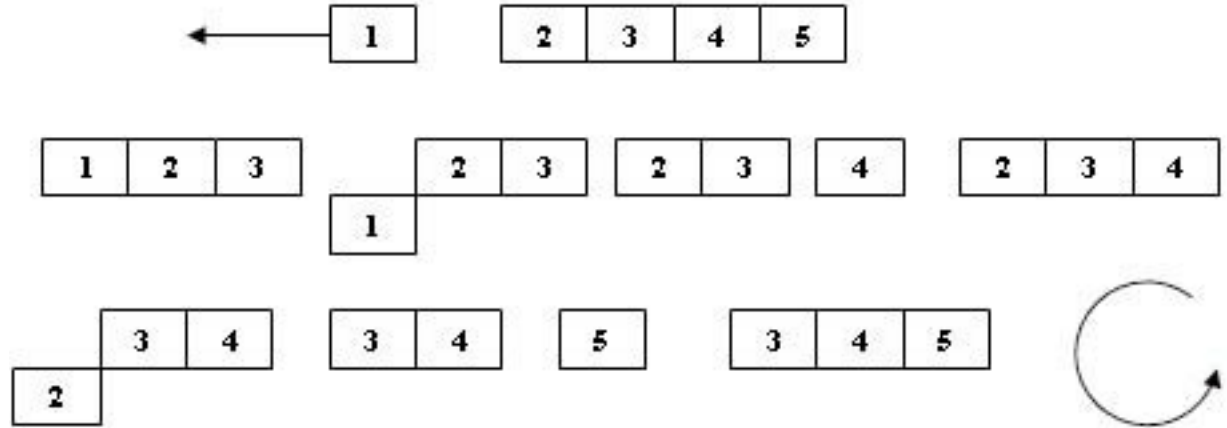
5.18.1 Queue

Queue, kuyruk demektir. Bir banka önünde işlem sırasını bekleyen bir kuyruk (Bir dizi insan) düşünelim, veya bir devlet dairesinde işlemini gerçekleştirmek için sırada bekleyen bir dizi insan (kuyruk) düşünelim. Bu insanlar, belirli bir sıraya girerler, ve işlem yapabilecek bir memur (vezne görevlisi – görevli - memur) boş konuma geçtiğinde, sıranın (kuyruğun) **en başındaki kişi** çağrılır. Sonradan gelip sıraya girmek isteyen herkes kuyruğun en sonuna gider (**en sona eklenir**).

Yukarda kalınlaştırılmış olan iki ifade çok önemlidir. Yeni bir kişi geldiğinde sona eklenir



İşlem yapılacağı zaman, kuyruğun en başındaki kişi çağrılır.



Figür 5.18.1 A : Kuyruk

Yukardaki evrelerin ilkinde, sırada bekleyen 3 kişi bulunmaktadır. Daha sonra, ilk kişi işlemini yaptırmaya gider (2. evre), bu sırada yeni bir kişi bankaya gelir (3. evre) daha sonra bu yeni gelen sıraya girer (4.evre), daha sonra 2 Nolu kişi işlemini yaptırmaya gider.....

Sonuç olarak bir kuyruğa ilk giren, ilk işlem yaptırır son giren son işlem yaptırır. Kuyruk sınıfı, FIFO mantığı ile çalışmaktadır. First In - First Out, ilk giren, ilk çıkar anlamına gelen bu ifade, bir kuyruğa ilk giren kişinin veya işlemin o kuyruktan ilk çıkacağı anlamına gelmektedir.

Kuyruklar bilgisayarın ve programlamanın birçok yerinde kullanılır. Örneğin, hafızaya bir dizi işlem yazdınız (yani bir dizi işlemi yapacak bir program yazıp bu programı çalıştırdınız).

Programın içindeki işlemler, tek tek hafızaya yazılır, ve daha sonra yine tek tek işlenir. Tabiki bu işlemler sırasına göre yapılır. Hafızaya ilk yazılan ifade ilk çalıştırılır son yazılan son çalıştırılır.

Kuyrukların özellikleri / işlemleri;

-Count / Length as integer: Bu özellik veya metod, kuyrukta kaç elemanın olduğunu döndürür.

-Pop as item : Bu metod ilk baştaki nesneyi, kuyruktan siler ve döndürür. Yani ilk başta olan nesneyi kuyruktan çıkartıp bu çıkarttığı nesneyi döndürür. Böylece ilk nesne alınıp işlenebilir.

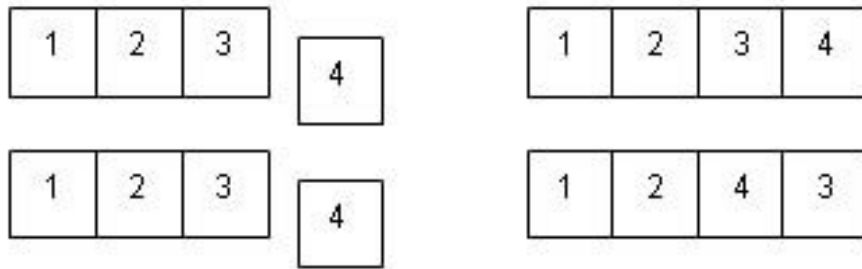
-Add (item) : En sona bir nesne eklemeye yarar. Yeni eklenecek olan nesneyi bu metodun parametresi olarak girebiliriz.

-Clear : Kuyruktaki bütün nesneleri siler.

5.18.2 List

Listeler, günlük hayatta kullandığımız ve içine bir dizi işlem – eleman veya benzeri nesne yazabildiğimiz dizelgelerle aynı işlevi görmektedir. Listeler, kuyruklardan daha fazla işlem yapabileceğiniz, ve bu işlemleri, liste içerisindeki her bir nesneye uygulayabileceğiniz, sınıflardır. Bu sınıfra örnek olarak, bir alış veriş listesini düşünelim. Bir alış veriş listesi hazırladınız... Bu listeye uygulanabilecek bütün işlemler, bir (program sınıfı) listeye de uygulanabilir. Örneğin, bu listeye nesne eklenebilir, nesne çıkarılabilir (iptal edilir), kaç tane nesne olduğu öğrenilebilir, belirli bir sıradaki nesne okunabilir (işlenebilir).....

Listelerde de, yeni nesneler sona eklenir. Ancak istenildiği takdirde, liste içinde, iki nesne arasına bir başka nesne eklenebilir.



Figür 5.18.2 A : Liste

Listelerin özellikleri ve işlemleri:

- Count / Length as integer : Liste içinde kaç tane nesne olduğunu döndürür.
- Add (item) : Yeni bir nesneyi sona ekler
- Remove (item) : Listedeki belirtilen nesneyi çıkarır - kaldırır
- RemoveAt(index as integer) : Listedeki belirtilen sıradaki (index teki) nesneyi çıkarır
- Insert (item, index as integer) : Belirtilen index'e, belirtilen item'ı ekler (sona değilse, belirtilen index'e)
- Clear : Listedeki bütün nesneleri temizler
- Contains (item) as boolean : Belirtilen nesnenin liste içinde olup olmadığını döndürür. Eğer bu belirtilen nesne, liste içinde bulunuyor ise true, bulunmuyor ise false değeri döndürür.
- Item (index) : Belirtilen index'teki, nesneyi döndürür.

Her liste bir kuyruk gibidir. Yani bir liste kullanarak bir kuyruk yapabiliriz. Dikkat ederseniz, kuyruksa olupta, listede olmayan, tek bir işlem vardır, POP. Bu işlem yukarıda belirttiğimiz gibi, baştaki nesneyi alır - kuyruktan çıkarır, ve o çıkarılan nesneyi döndürür.

```
Function Pop () : Object
Begin Baş: Object; //Bir değişken tanımla.
Baş = Item(0); //Baştaki nesneyi al ve değişkene yaz
RemoveAt(0); //Baştan alınan bu nesneyi, kuyruktan çıkar
Return Baş; //Değişkene yazılmış olan nesneyi döndür
End
```

Böylece bir listeyi bir kuyruk haline getirebiliriz.

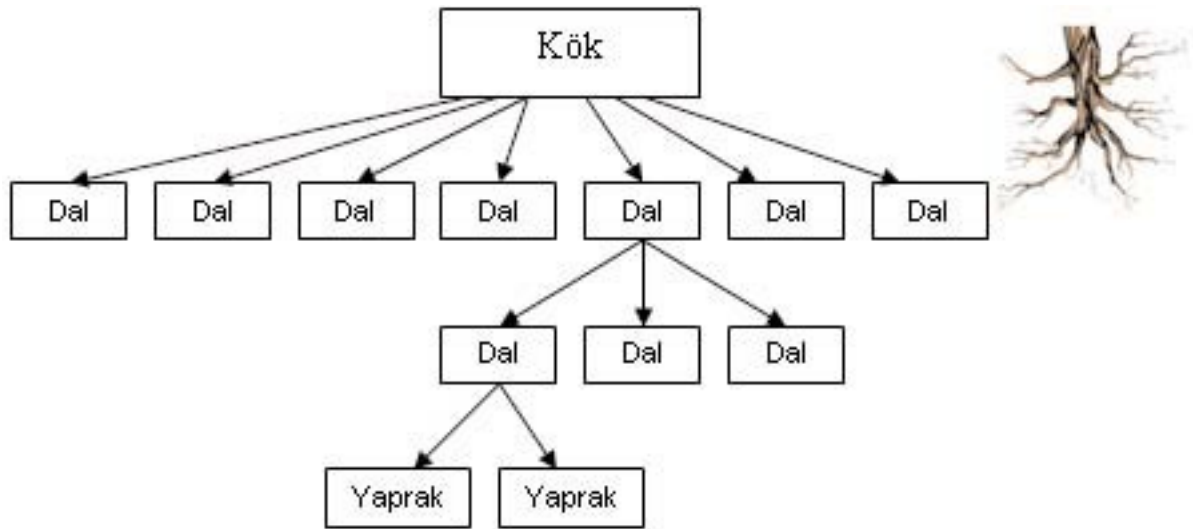
5.18.3 Tree

Tree, ağaç anlamına gelmektedir. Bunu bir aile ağacı (soy ağacı) gibi düşünebiliriz. Bir ağaçta, mutlaka bir kök vardır, dallar köke bağlıdır. Dallara bağlı başka dallarda olabilir ve o dallara bağlı başka dallarda. En sonunda ise, dallara bağlı yapraklar vardır ve yapraklara herhangi bir başka kısım bağlı değildir.



Figür 5.18.3 A : Ağaç

Bu ağacı bir ağaç diagramına çevirmek için: Bir ağacın dal sayısı veya dallarının kaç kere daha dalları olduğu sayısını bilemeyiz. Burada tek sabit olarak alabileceğimiz köktür. İşte bizde o yüzden, kökü ters çevirip, daha sonra dalları yerleştiriyoruz.



Figür 5.18.3 B : Ağaç yapısı

Dikkat ederseniz, yaprakların daha alt bölümleri olamaz. Sadece dalların alt bölümleri olabilir ve bu birçok kere devam edebilir.

Kökün özellikleri ve işlemleri:

- Add (dal): Kök'e bir dal eklenebilir. (node = dal)
- Remove : Kökten bir dal çıkarılabilir.-Length : Köke bağlı dalların sayısını döndürür
- Item (index) As Node: Belirtilen indexteki node 'u (dalı) döndürür.

Her bir dalın sahip olduğu özellikler ve işlemler:

- Add (dal) : Belirtili dala başka yeni bir alt dal ekler
- Remove : Belirtili daldan bir alt dalı çıkarır
- Length : Dala bağlı alt dalların sayısını döndürür
- Item (index) As Node: Belirtilen indexteki alt dalı döndürür
- Parent : Bağlı olduğu dalı veya kökü döndürür

Ağaçlar programların içinde genelde bir hiyeraşi tutmak için kullanılırlar. Bu şekilde bir ağaç sınıfını, bir liste kullanarak oluşturalım.

Dikkat ederseniz, bir KÖK için, liste kullanabiliriz, çünkü kökün sahip olduğu bütün özellikler listede de bulunmaktadır.

Class : Ağaç Inherits : List

End Class

Doğal olarak; Add, Remove, Length, Item gibi prosedürleri tekrar yazmamıza gerek bulunmamaktadır. Bu ağaç sınıfı, bir liste olarak, NODE'ları yani dalları tutacaktır. Peki node'u nasıl tanımlarız?

Node'u, yine aynı şekilde bir liste olarak tanımlayabiliriz ancak, ek olarak, Parent özelliğini eklemeliyiz.

Class : Dal Inherits : List

Property : Parent

End Class

5.18.4 Set

Set, matematikten bildiğimiz küme anlamına gelmektedir. Set, bir tür listedir. Ama kümelerden hatırladığımız üzere, bir küme içerisinde aynı iki nesne olamaz. Listeler içinde ise aynı nesneden istenilen kadar bulunabilir.

Bir listede aşağıdaki gibi aynı nesneler bulunabilir

Ahmet
Ahmet
Mehmet
Mustafa
Ergin
Erhan
Ergin

Bir listede ise, aynı nesneler bulunamaz, her nesne tekil olmak zorundadır.

Ahmet
~~Ahmet~~
Mehmet
Mustafa
Ergin
Erhan
~~Ergin~~

Bir kümeye uygulanabilecek işlemler, ve bir kümenin özellikleri, bir listeninkilerle aynıdır.

Kümeler, aynı tür nesnelerin olamayacağı – yada olmaması gereken her yerde kullanılabilir. Örneğin; bir ürün listesi oluşturuyoruz ve ürünler içinde aynı isme sahip iki farklı ürün olamayacaktır. Bu nedenle böyle bir durumda set yani küme kullanabiliriz.

5.19 Aduzayı & Imports

3. bölümde, evrende milyonlarca sınıf olduğunu belirtmiştim. Bu kadar çok sınıfı listelemek, onlar üzerinde işlem yapmak, bu sınıflar içinden aradıklarımızı bulmak epeyce zor olacaktır. Samanlık içinde iğne aramak gibi...

Aynı durum, müşteriler, ürünler, otomobiller içinde geçerlidir. İşte bu kadar sınıfın veya üyenin olduğu bir kümede karışıklığı önlemek için, kategorilendirme işlemi yapılacaktır. Yani bir tür, hiyerarşik yapı kurularak, belirtilen küme içersindeki elemanlar, bu hiyerarşi üzerindeki uygun yere yerleştirilecektir. Sonuçta, 6 vitesli otomobiller dediğinizde size binlerce otomobil yerine sadece istediğiniz kategoriye ait olan sonuçlar görüntülenecektir.

İşte gerçek dünyada, “kategori” veya “hiyerarşi” olarak belirttiğimiz ifadeleri, programlama dünyasında “aduzayı” (namespace) olarak adlandırırız. Aduzayları, programlama dünyasında, oluşturulan sınıfları, (daha sonradan göreceğimiz) arabirimleri, kalıpları ve diğer elemanları gruplandırmak için kullanılır.

Özellikle programlama dillerinde, yüzlerce hatta binlerce sınıf olduğundan dolayı, karışıklığı engellemek için, kategorilendirme şarttır. Aşağıdaki tabloda, Microsoft.Net platformunun (Vb.Net / C#.Net dillerinin kullandığı ortak yapı) aduzayları gösterilmiştir.

Microsoft.CSharp	System.Data
Microsoft.JScript	System.Data.Common
Microsoft.VisualBasic	System.Data.OleDb
Microsoft.Vsa	System.Data.SqlClient
Microsoft.Win32	System.Data.SqlTypes
System	System.Diagnostics
System.CodeDom	System.Diagnostics.SymbolStore
System.CodeDom.Compiler	System.DirectoryServices
System.Collections	System.Drawing
System.Collections.Specialized	System.Drawing.Design
System.ComponentModel	System.Drawing.Drawing2D
System.ComponentModel.Design	System.Drawing.Imaging
System.ComponentModel.Design.Serialization	System.Drawing.Printing
System.Configuration	System.Drawing.Text
System.Configuration.Assemblies	System.EnterpriseServices
System.Configuration.Install	

Beşinci bölümde kullanılmış olan terimlerin ingilizceleri

yeni : new	veya : or
oluşturucu : constructor	doğru : true
özet : abstract	yanlış : false
özel : private	yorum : comment
genel : public	isim : name
sabit : static	dosya : file
ve : and	zamanlayıcı : timer

SONUÇ

Varsayalım ki çok uzman bir programcısınız ve çalıştığınız firmadaki müdürünüz yada yetkili olduğunuz kişi sizden, bir fabrika yönetim programı yapmanızı istedi. Doğal olarak, bu programı yapamayacaksınız. Çünkü, fabrika yönetimi konusunda bilginiz bulunmuyor. Yani, iyi bir programcı olmanız, içeriğini bilmediğiniz bir programı yapabileceğiniz anlamına gelmeyecektir.

Aynı şekilde, her ne kadar iyi bir programcı olursanız olun, doğal olarak, bazı konularda bilgisayarı tanımanız ve bilgisayarla içli – dışlı olmanı gerekmektedir. Bu nedenle, bilgisayar sistemlerini, bazı işlemlerin bilgisayarda nasıl tanımlanacağını zamanla ve araştırarak öğrenmeniz gerekmektedir.

6 Veritabanı

İçerik

- 6.1 Veri tabanları
- 6.2 Veri tabanı yönetim sistemi
- 6.3 Tablolar / Kolonlar / Kayıtlar
- 6.4 Entity – Relation yapısı

Amaçlar

Altıncı bölüm,

- v,
- İleri değişken işlemlerini, programsal anlatmaktadır.

Anahtar sözcükler

yeni, oluşturucu, özel, genel, sabit, dosya, yorum, doğru, yanlış

6.1 Veri tabanları

İnsanların, öğrendiklerini bir sonraki nesillere aktarmadıkları sürece ilerleme kaydedemeyeceklerini belirtmiştim. Bilimsel, kültürel ve sanatsal bilgi ileriki kuşaklara aktarıldığı sürece, insanlık ilerleme kayıt edebilmiştir.

Tarih, yazının bulunmasıyla başlamaktadır. İnsanlar, öğrendiklerini önce taşlara, kayalara sonra derilere, kağıtlara yazarak, bir sonraki nesillere aktarmışlardır. Bilgisayar biliminin ortaya çıkmasından sonra, artık bu bilgiler bilgisayarların içine taşınmaya başladı.

Şüphesiz ki, gelişen teknoloji sayesinde, artık çok fazla veriyi saklayabiliyor ve işleyebiliyoruz. Verileri saklamak ve işlemek için kullandığımız her sisteme **veritabanı** denir.

Google’da çalışmak isteyen gençlere, mülakat sırasında sorulan sorulardan biri: “Veritabanının ne anlama geldiğini sekiz yaşındaki kuzeninizin anlayacağı bir şekilde açıklayın” şeklindedir.

Zannediyorum ki, kitap okurları arasında hiç kimse sekiz yaşında değildir. Ancak ben yine de, bu soruyu cevaplamak adına, size sanki sekiz yaşında bir çocukmuşsunuz gibi, veri tabanının ne olduğunu anlatacağım.

Anlatmadan önce, sekiz yaşındaki bu çocuğu (kuzenimi) bir kütüphaneye götürürüm. Kütüphaneye dikkatlice bakmasını isterim. İnceleme sonucunda göreceğimiz: kütüphanenin; sütunlar, raflar ve kitaplardan oluştuğudur.



Figür 6.1 A : Bir kütüphane

Daha sonra, kütüphaneye getirdiğim kuzenime, kütüphanenin, binlerce kitabı barındıran bir tür bilgi bankası olduğunu, çeşitli türlerde, farklı konularda binlerce bilgiyi öğrenmek için, en ideal yer olduğunu, cevabını merak ettiğimiz sorulara, gerekli sütun – raf – kitaba bakarak ulaşabileceğimizi belirtirim. Ayrıca eski kitapların raftan kaldırıldığını yazarların yeni yazdıkları kitapların yer aldığını ifade ederim.

7-12 yaş arası bütün çocuklar kendilerinden daha büyüklere hemen hemen gördükleri

ve duydukları her konu hakkında soru sorarlar. İşte, kütüphane; bütün bu soruların cevaplanabileceği bir merkezdir diyebiliriz.

Bu bilgiyi verdikten sonra, veritabanını anlatmak çok daha kolay olacaktır. Çünkü, veritabanı, sadece elektronik bir kütüphanedir.

Veri tabanı Veri tabanı düzenli bilgiler topluluğudur. Kelimenin anlamı bilgisayar ortamında saklanan düzenli verilerle sınırlı olmamakla birlikte, daha çok bu anlamda kullanılmaktadır. Bilgisayar terminolojisinde, sistematik erişim imkanı olan, yönetilebilir, güncellenebilir, taşınabilir, birbirleri arasında tanımlı ilişkiler bulunabilen bilgiler kümesidir. Bir başka tanımı da, bir bilgisayarda sistematik şekilde saklanmış, programlarca istenebilecek veri yığınıdır. (TDK sözlüğüne göre "veri tabanı" olarak ayrı ayrı yazılır) [e22]

Veri tabanları; verileri, uzun süreli saklayan ve onlar üstünde işlem gerçekleştirilmesine olanak sağlayan sistemlerdir. Daha önceki bölümlerde bir değişkenin ne olduğunu görmüştük. Verileri (değerleri) belirli bir süre (HAFIZADA) saklayan tutamaçlara değişken deniyordu. Veritabanları da birden fazla, hatta binlerce - milyonlarca veriyi, sabit disk üzerinde tutan (böylece veri kayıt edilmiş olur ve daha sonra yeniden geri yüklenebilir) sistemlerdir.

Dünya üzerinde şuanda kullanılmakta olan yüzbinlerce veritabanı vardır. Devlet arşivlerini, polis sabıka kayıtlarını, nüfus kayıtlarını, sağlık bakanlığı kayıtlarını, vergi kayıtlarını, kullanıcı listelerini..... düşünelim. Eğer bu kayıtlar belirli bir formata uygun kayıt edilmeseydi ve bir veritabanı sistemi kullanılmasaydı, en basit bir işlem için bile haftalarca hatta aylarca beklememiz gerekirdi. Öyle ki bu sürede istenilen kayıtlara ulaşılabilinsin ve işlem yapılabilinsin.

Kayıtların elektronik ortamda tutulmasını sağlayan veritabanları, arama işlemlerini, çok fazla sayıda kayıt olsa bile birkaç saniyelik sürede yapabilirler. Ayrıca, binlerce kağıt ve dosyanın harcanacağı saklama işleminin maliyeti ve bu kağıtları elde etmek için kesilen ağaçların maliyetini tahmin bile edemeyiz... Sonuç olarak, veritabanları, hızlı çalışmakta, ucuz oluşturulmakta (kağıt masrafı bulunmamaktadır), hızlı işlenmekte ve veriler çok kolay bir şekilde değiştirilmektedir.

Veri değiştirme, sabit kayıtlar için her zaman bir problem olmuştur. Bir listeyi düşünün. Listedeki nesneler arasına bilgi eklemek imkansızdır. Sadece silip değiştirebilirsiniz, ekleme yapamazsınız. Veritabanlarında ise bu işlem yani bir kaydın araya eklenmesi çok kolaydır.

Linear Search (Lineer arama). Lineer arama, kör arama olarakta bilinir. Elinizde bir dizi kutu olsun ve bu kutuların bir tanesinin içinde istediğiniz bir eşyanın olduğunu varsayalım. Aradığınız eşyayı bulabilmek için, kutuları teker teker açıp içlerine bakmanız gerekmektedir. Aradığınız eşyayı, ilk kutu içindedey bulabilirsiniz, son kutu içinde de... Bu yüzden, ortalama olarak, kutu sayısının yarısı kadar kutuyu açmanız gerekmektedir. Yani N tane kutu bulunuyorsa, ortalama olarak N/2 tanesine bakarak bulabilirsiniz.

Bir kitap düşünün, eğer bu kitapta bir kelime arıyorsanız, ortalama 350 kelime/sayfa (sayfa başına düşen kelime sayısı) dan 500 sayfalık kitaptaki 175 bin kelime bulunur. Tek tek

bu kelimeleri okuyarak istediğinizi bulmak çok uzun süre alır. İnsan dakikada ortalama 150-200 kelime okur. 175 bin kelimeyi (dakikada) 175 kelimeye bölersek, 1000 dakika yani 16.67 saat (neredeyse 1.5 gün - 9 saatlik çalışma zamanına göre) olur. Ancak, aradığımız kelimeyi kitabın başında veya sonunda bulabiliriz. Yani ilk kelime istediğimiz kelime olabilir, son kelime... O zaman linear search (lineer arama) mantığındaki ortalama işlemi uygulayacağız. Yani ortalama, 16.67 / 2 saat harcayarak istenilen kelimeyi bulabiliriz.

Bu süreyi azaltmak için kitapların sonunda anahtar sözcüklerin listesi, ve bu sözcüklerin geçtiği sayfalar eklendi. Böylece, istediğimiz kelimenin anında nerede geçtiğini bulabiliyoruz. Tabiki, aradığımız kelime, anahtar sözcüklerden biri değilse, o zaman bu işlemin anlamı kalmayacaktır.

Başka bir örnek vereyim. Bir arama motorunu düşünün (google, yahoo....). İnternette milyonlarca site, o sitelerinde, onlarca alt sayfaları vardır. Bu site sayfalarında ise, yüzlerce bilgi (cümle veya paragraf) bulunur. Bu sayıları yan yana yazıp çarptığınızda, oldukça büyük bir sayı elde edersiniz. Lyman & Varian'ın 2000 yılında yaptığı saptamaya göre, internetteki bilgi sayısı her yıl 1-2 exabyte arasında artmaktadır [1,000,000,000,000,000,000 (10¹⁸) byte]. Yani bu rakam, dünyadaki her kişi için (dünya nüfusu yaklaşık 6,64 milyar - 2007) 250 MB demektir. Yazılı kaynaklar ise sadece, bu miktarın %0.003 (yüz binde 3 = internetteki 100.000 kaynak için, 3 basılı kaynak vardır) kadardır. Bu kadar büyük sayıda bilgilerin arasından istediğiniz bulmak imkansız olacaktır. Samanlıkta iğne aramak inanılmaz çok daha kolay olacaktır.

İşte bu yüzden, veritabanları kullanılır ve gerekebilecek her tür veri bu veritabanlarına kayıt edilir. Aynı şekilde, kamu veya şirket bilgilerinin tutulduğu bir veritabanı sistemi olmasaydı, kişilere ve olaylara ait bilgilerin bulunması aylar hatta yıllar alabilirdi. Veritabanlarının öncelikli amacı bilgiyi saklamak değil, bilgiyi gerektiğinde, kolay bir şekilde işlemek üzere belirli bir formatta saklamaktır.

Veritabanını illa ki bir bilgisayar üzerindeki sistem olarak düşünmeyin. Kayıtların tutulduğu bir kütüphane, bir dosya, olarakta düşünebilirsiniz. Bilgisayar keşfedilmeden önce de veri tabanları kullanılırdı. Bir liste, bir çizelge; bir tablo şeklinde, hangi kayıtların nerede olduğunu, kayıtların numaralarının / renklerinin ne olduğunu belirten bir döküman olarak kullanıldığında, bir veritabanıdır.

İlk bölümde programların genelde 5 işlem üzerinde yoğunlaştığını söylemiştim: Veri yazma, veri okuma, veri silme, veri değiştirme, veri işleme... Saymakta olduğum ilk dört işlem, genelde (%90) veritabanlarında kayıt altında tutulur. Yani saymakta olduğum işlemlerin dört tanesi veritabanı ile ilişkilidir ve veri tabanı olmadan bu işlemler yapılamaz.

Hepimiz, bir liste hazırlamış ve bir Excel dökümanı ile çalışmışızdır. İşte bu dosyalar, aslında birer veritabanıdır.

Ad	Soyad	Vize1	Final	Ortalama
FATİH	POYRAZ	40	34	27,0
NECMİ	GÖCEK	50	65	45,0
YUSUF	ÖZDEMİR	30	76	45,5
KERİM	HIZARCI	70	72	53,5
SELİM	UYSAL	34	34	25,5
SABİHA	DEMİR	65	87	59,8
DİLEK	ÇANKAYA	23	54	32,8

OSMAN	GÖKHAN	76	87	62,5
İSMAİL	GÜRAN	100	100	75,0

Yukarıda gördüğünüz tablo, bir tür veritabanıdır. Biz bu veritabanına ait dört işlemi, klavye ve mouse yardımıyla kendimiz yaparız. Yani yukarıdaki listeye bir öğrenci girişini kendimiz yaparız, değişiklikleri – görüntülemeyi ve silmeyi de aynı şekilde... Programlar ise, bu işlemleri bizim için veritabanlarına girer – okur – siler – değiştirir. Yani manuel olarak yapıyor olduğumuz işlemi program biraz otomatikleştirir.

Alıştırmalar

Veritabanı nedir?

Veritabanları ne işe yarar?

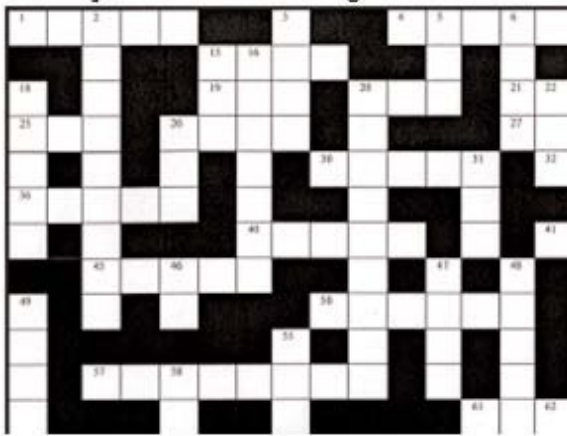
Linear search nedir?

6.2 Veri tabanı Yönetim Sistemi

DBMS DBMS (database management system, veritabanı yönetim sistemi), veritabanlarını, yönetmek için kullanılan yazılımlara verilen addır. Basit olarak; veri tabanları veriyi tutan dosyalardır. Ancak bir veritabanı yönetim sistemi, veritabanının tuttuğu veriyi yöneten, üzerinde işlemler gerçekleştirilmesini sağlayan bir sistemdir.

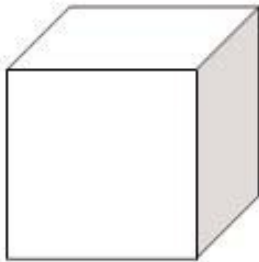
Tek bir veriyi tutmak istesek, bir sabit değişkene (constant) bu veriyi atayıp kullanabiliriz. Ancak birden fazla ve temel veri türü formatında bulunmayan verileri saklamak, onlar üzerinde işlemler yapmak için bir DBMS, veritabanı yönetim sistemi kullanmalıyız. Ayrıca, bildiğimiz üzere, değişkenler, hafızada tutulur (memory – RAM), bilgisayarı kapattığınızda, hafızaya elektrik gelmeyeceğinden, değişkenlerde otomatik olarak silinecektir. İşte bu yüzden verilerimizi, sabit diske kayıt etmeliyiz. Ayrıca, hafızada, en fazla 1GB, 2GB gibi değerlerde veri tutabiliriz halbuki sabit disklerde, 40-50 GB gibi büyük değerlerde veriler tutabiliriz.

Dünyada en yaygın olarak kullanılan veri saklama format Matris, yani tablodur. Matris, NxM (N tane kolonu, M tane satırı olan) farklı boyutta bir tablodur. Örneğin bir bulmaca 2 farklı boyutta (en/boy) bir matristir.



Figür 6.2 A: Bulmaca

Başka bir örnek bir küp şeker, 3 farklı boyutta (en/boy/genişlik) bir matristir.

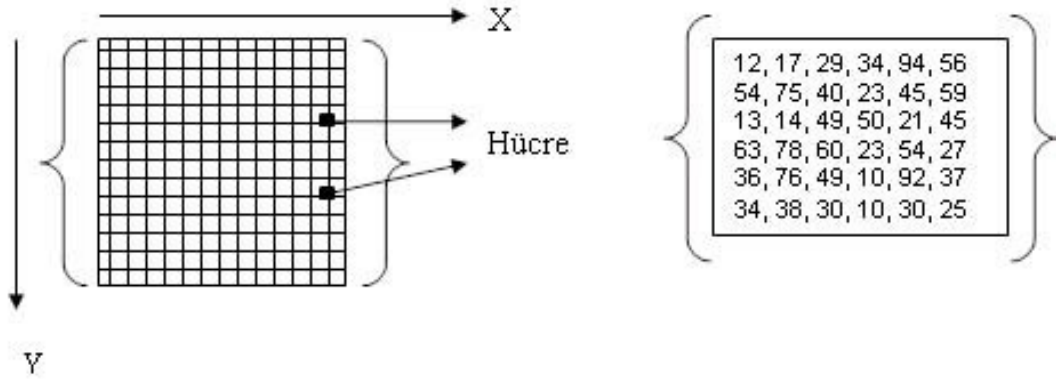


Figür 6.2 B: Küp

Tablolarda, yine bulmacalar gibi 2 farklı boyutta bir matristir. Tabloları hepimiz, Excel, Word, Access gibi uygulamalardan veya gerçek hayatta, en basit örnek olarak verebileceğimiz, haftalık ders programından biliriz.

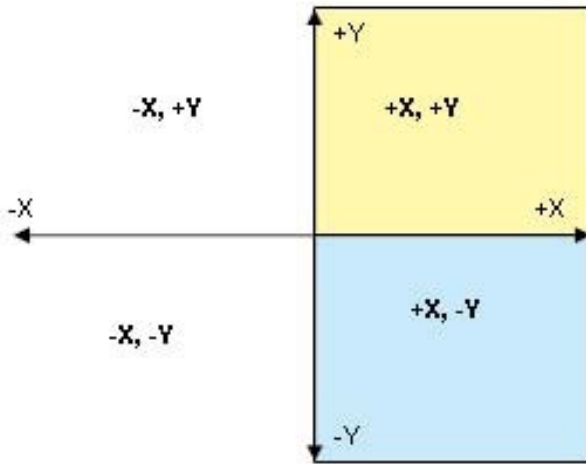
Ders No	Pazartesi	Salı	Çarşamba	Perşembe	Cuma
1	Matematik	Edebiyat	Coğrafya	Fizik	Tarih
2	Kimya	Edebiyat	Matematik	Edebiyat	Fizik
3	Biyoloji	Beden Eğitimi	Müzik	Matematik	Kimya
4	Geometri	Tarih	Kimya	Biyoloji	Geometri

Bir matriste, önemli olan koordinatlardır. Koordinatlar bilindikten sonra bu koordinatlardaki hücreye veri yazılabilir veya okunabilir. Örneğin elimizde bir aşağıdaki gibi matris bulunuyor



Figür 6.2 C : Çizelge Örneği

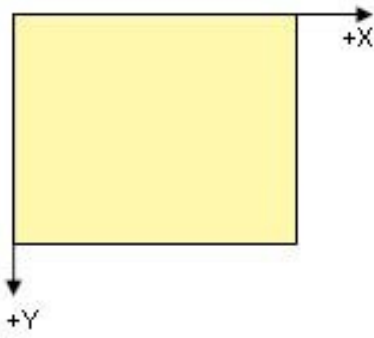
Burada dikkat etmeniz gereken bir nokta bulunuyor. Analitik geometriden hatırlarsınız, koordinat düzlemi aşağıdaki gibidir.



Figür 6.2 D : Koordinat düzlemi

X ve Y koordinatlarının pozitif olduğu bölge yani sağ üst bölgeye (taralı alan) 1. Bölge denir. (Mavi bölgeye de 4. bölge denir) Bir matriste, negatif bölümler bulunmaz. Çünkü matristeki, kolon ve satır değerleri pozitifdir. Yani mutlak değerdir. Nasıl bir uzaklığın negatifi olamıyorsa aynı şekilde, sıra nında negatifi olamaz.

Yukarıdaki koordinat düzlemi işte bu nedenle, bazı değişikliklerle aşağıdaki hali almaktadır.



Figür 6.2 E : Koordinat düzlemi

Yani, normal koordinat düzleminde, 1. bölge olan bölüm, 4. bölgenin üzerine gelir. Bundan sonra X ve Y koordinatları için herhangi bir negatif değer söz konusu olmaz. Ayrıca, buradan, aşağıya doğru gidildikçe satır numarasının arttığını, sağa doğru gidildikçe de kolon numarasının arttığını söyleyebiliriz.

Aşağıdaki matriste, 6 satır 6 kolon bulunmaktadır. Koordinatlarını verdiğimiz bir değeri bulalım:

12, 17, 29, 34, 94, 56
54, 75, 40, 23, 45, 59
13, 14, 49, 50, 21, 45
63, 78, 60, 23, 54, 27
36, 76, 49, 10, 92, 37
34, 38, 30, 10, 30, 25

Figür 6.2 F : Matris

Matris(Kolon, Satır) = Değer

Matris(1,2) => 54

Matris(2,1) => 17

Matris(1,2) => 12

Matris(4,1) => 34

Matris(2,6) => 38

şeklinde olacaktır. Tablolar yapı olarak matrislere çok benzer ancak, kolonlarda bazı değişiklikler söz konusudur. Örneğin kolonların indexleri yani numaraları yoktur, numaralar yerine, isimleri vardır.

Bir tabloda önemli olanlar:

-Kolonlar

o Kolonların değerlerinin türleri

o Kolonların isimleri

o Satırlar (her satır bir kayıt anlamına gelir)

o Hücrelerdeki veriler

Veritabanları, tablolardan oluşur ve tablo mimarisi üzerine kurulmuştur. Aşağıda bir örnek tablo görünmektedir.

Şirket İsmi	Yetkili Kişi	Adres
Alfredo Futerkioto	Maria Anders	Obero Str. 57
Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222
Arlunio Murenu Taquería	Arlunio Murenu	Maladeros 2312
Around the Horn	Thomas Hardy	120 Hannover Sq
Berglunds snabbköp	Christina Berglund	Berguvsvägen 8
Blauer See Delikatessen	Hanna Moos	Forsterstr. 57
Blondelcoel père et fils	Frédérique Citeaux	
Bólido Comidas preparadas	Marlin Sommer	C/ Araquil
Run appi'	Laurence Leblanc	
Rullman-Dollar Markets	Elizabeth Lincoln	23 Tsawassen Blvd

Figür 6.2 G : Tablo örneği

Burada görüldüğü üzere, kolon numaraları yerine kolon isimleri bulunmaktadır. Her bir satır, bir kaydı göstermektedir - ifade etmektedir. Ayrıca, sınıf tanımlamasındaki gibi, her kolona girilecek verinin belirli formatları bulunmaktadır. Hücrelere bu formatlar dışında veri girilmez (aynı sınıflardaki özellikler gibi)

Veri tutan her sistem veri tabanıdır. Buna, bir excel dosyasını, bir metin dosyasını örnek gösterebiliriz. Aşağıda bazı formatlarda veritabanı örnekleri bulunmaktadır.

Metin Dosyası

Name	Owner	Type	Create Date
syscolumns	dbo	System	06.08.2000 01:29:12
syscomments	dbo	System	06.08.2000 01:29:12
sysdepends	dbo	System	06.08.2000 01:29:12
sysfilegroups	dbo	System	06.08.2000 01:29:12
sysforeignkeys	dbo	System	06.08.2000 01:29:12
sysindexes	dbo	System	06.08.2000 01:29:12
sysindexkeys	dbo	System	06.08.2000 01:29:12
sysmembers	dbo	System	06.08.2000 01:29:12
sysobjects	dbo	System	06.08.2000 01:29:12
syspermissions	dbo	System	06.08.2000 01:29:12
sysproperties	dbo	System	06.08.2000 01:29:12
sysprotects	dbo	System	06.08.2000 01:29:12
sysreferences	dbo	System	06.08.2000 01:29:12

Excel Dosyası

Şirket	Yetkili	Yetki
Ricardo Adocibados	Janete Limeira	Assistant Sales Agent
Richter Supermarkt	Michael Holz	Sales Manager
Romero y tomillo	Alejandra Camino	Accounting Manager
Santé Gourmet	Jonas Bergulfsen	Owner
Save-a-lot Markets	Jose Pavarotti	Sales Representative
Seven Seas Imports	Hari Kumar	Sales Manager
Simons bistro	Jytte Petersen	Owner
Spécialités du monde	Dominique Perrier	Marketing Manager
Split Rail Beer & Ale	Art Braunschweiger	Sales Manager

Figür 6.2 H : Tablo örneği - 2

XML

```

<?xml version="1.0" encoding="us-ascii"?> <doc>
<assembly>
<name>exread</name>
<version>1.0.1347.40148</version>
<fullname>exread, Version=1.0.1347.40148, Culture=neutral,
PublicKeyToken=null</fullname>
</assembly>
<members>
<member name="T:AltairCommunications.ExifReader.ExifReader">
</member>
</members> </doc>

```

SQLServer

City	Region	PostalCode	Country
London		EC1 4SD	UK
New Orleans	LA	70117	USA
Ann Arbor	MI	48104	USA
Tokyo	<NULL>	100	Japan
Oviedo	Asturias	33007	Spain
Osaka	<NULL>	545	Japan
Melbourne	Victoria	3058	Australia
Manchester	<NULL>	M14 6SD	UK
Göteborg	<NULL>	S-345 67	Sweden
Sao Paulo	<NULL>	5442	Brazil
Berlin	<NULL>	10785	Germany
Frankfurt	<NULL>	60439	Germany
Cuxhaven	<NULL>	27478	Germany

Figür 6.2 I : Tablo örneği

6.3 Tablolar / Kolonlar / Kayıtlar

Tabloların, veri tabanı mimarisini oluşturduğunu söylemiştim. Veritabanında, veriler, tablolar üzerine eklenir ve tablolar üzerinde saklanır. Bunun için, öncelikle tabloları tasarlamak gerekir, daha sonra bu tabloları hazırladığımız formata göre, tablolara veri girişi - okuması - silmesi, değişikliği gibi işlemleri gerçekleştiririz.

Aslında, tabloların sınıflardan, kolonların sınıfın özelliklerinden neredeyse hiçbir farkı yoktur. Şöyle ifade edeyim; bir sınıf tasarladık, sınıfın ismi dediğimiz tablonun ismidir yada tablonun ismi, sınıfın ismidir. Bir sınıfın sahip olduğu özellikler, o sınıfa karşılık gelen tablonun özellikleridir, yani o tablonun kolonlarıdır.

Bir kütüphanedeki sütunları tablolar, rafları kolonlar, kitapları da bu yapı içerisine yerleştirilen veriler olarak görebiliriz.

Bir tabloda en az bir kolon bulunmak zorundadır. Bir tabloda bulunabilecek en fazla kolon sayısı kullanılacak olan veritabanı sistemine göre değişmektedir.

Veritabanı sistemlerinde de aynı, programlama olduğu gibi kayıt edilecek olan verilerin türleri bulunmaktadır. Bu türlerle ilgili ayrıntılı bilgi aşağıdaki tabloda verilmiştir

Türün ismi	Programsal Türdeki karşılığı	Açıklaması
varchar	String	<p>Varchar (variable charachter, değişebilir uzunlukta karakter) metin tipi verileri saklar. Bu tür tanımlanırken, bir sayı belirtilir. Bu sayı, o kolonun (yada alanın) alabileceği en çok miktardaki karakter sayısıdır. Programlamada metin tipindeki değişken olan String'i tanımlarken bu şekilde bir limitleme kullanmayız. Hafıza çok esnek ve hızlı olduğundan dolayı, bir metnin uzunluğunu değiştirmemiz kolay olur ve istediğimiz uzunlukta (en üst limit geçmediği sürece) uzun metin değer kullanabiliriz.</p> <p>Varchar'ın akabinde yazılan sayı, o kolona girilebilecek en çok karakteri belirtir, bu sayıdan daha az karakter kullanılabilir ancak daha fazla karakter kullanılamaz.</p> <p>Örneğin : isim varchar(20) İsim adında, bir kolon tanımladık ve uzunluğu en fazla 20 karakter olarak belirttik. Bu limitin dışına çıkmadığı sürece bu kolona, istenildiği uzunlukta isim girebiliriz.</p> <p>Örneğin;</p> <p>Ali, Ahmet, Fuat Tamer, Mustafa, Aylin, Ayşe, Nil</p>

char	String, Char []	<p>Char veri tipi, varchar'a benzer ancak uzunluğu değişmeyen, sabit olan veriler için kullanılır. Yani char veri tipi ile tanımlama yaparken yine varchardaki gibi maksimum uzunluğu belirleyen bir sayı belirtilir. Ancak bu sayı hem maksimum hem minimum değerdir. Yani bu alana girilecek olan verilerin uzunluğunu sabitler</p> <p>Örneğin; Nüfus cüzdanımızda ki T.C. Kimlik Numarası, sabit uzunluktadır ve 11 karakterden oluşur. Eğer bizde T.C. Kimlik numarası girmek için bir alan oluşturuyorsak, o zaman char veri tipini kullanıp uzunluk olarak 11'i belirtebiliriz.</p> <p>Tckimlik char(11) Daha sonra aşağıdaki gibi 11 karakterli her tür veri bu alana girilebilir:</p> <p>45330967312 49087024329 42390808903 41092897893</p>
Bigint, int, smallint, tinyint, long, short, byte, Numeric, Decimal	Bigint, long, int, integer, tinyint, byte, short, numeric, decimal	<p>Bu tip veri türleri tamsayıları belirtmek için kullanılır. Daha önceden programsal türlerde belirttiğimiz gibi, farklı aralıkları, yani limitleri vardır. Bu limitler ve kullanılan türler, veritabanı sistemlerine göre değişmektedir. Bu veri türlerinin aralıkları programsal türlerde belirtilmiştir.</p>
Binary, VarBinary, Image	Byte[]	<p>Binary daha öncedende gördüğümüz gibi ikili anlamına gelmektedir. Bu veri türleri; bir ikili veriyi veya komple bir dosyayı, veritabanı içine kayıt etmek için kullanılan bir türlerdir. Kullanılan veritabanı yönetim sistemine göre aralıkları ve özellikleri değişmektedir. Yine daha önceki bölümde belirttiğimiz gibi, ikilik düzendeki veriler, ilk başta onaltılık veriler olarak DBMS (veritabanı yönetim sistemi)'ye atılır. Daha sonra bu veriler veritabanında ikilik düzlemde tutulur.</p>
Datetime, smalldatetime, date, time	Date, datetime	<p>Bu veri türleri isminden de anlaşılacağı gibi, tarih ve/veya saat verilerini tutmak için kullanılır. Belirli formatları bulunmaktadır ve bu formatlar (ayrıca limitler) kullanılmakta olan DBMS'ye göre değişmektedir.</p>
Float, Real, Double	Double, single	<p>Bu veri türleri, programsal türlerde belirttiğimiz gibi ondalık sayıları tutmak için kullanılır. Kullanılan DBMS'ye göre özellikleri ve limitleri</p>

Bit, Bool	Boolean	değişir. Sadece 1 veya 0 değeri girilebilen bir veri formatıdır.
-----------	---------	---

Not: Daha farklı veri tipleri de bulunmaktadır.

Kayıtları da şu şekilde düşünelim: Bir sınıf oluşturduk, o sınıfa ait olarak oluşturduğumuz her nesneyi birer kayıt olarak görebiliriz. Yani her **Object** bir **kayıt** olabilir. Tabiki bu oluşturduğumuz her sınıfı veritabanında kayıt edeceğiz anlamına gelmemektedir. Ayrıca, veritabanına kayıt edeceğimiz her tipte veri için sınıf oluşturmaya da gerek yoktur. Buradaki amaç, tabloların da aynı sınıflar gibi tasarlandığını göstermektir.



Figür 6.3 A : OOP - Veritabanı

Örneğin bir sınıf oluşturduk ve varsayalım ki bu sınıf program içindeki hatalar üzerinde işlemler yapsın. Bu sınıfın nesnelerinin, veri tabanına kayıt edilmesine gerek yoktur. Yani eğer, oluşturulan bir sınıfta bir özellik yok ise, bu sınıf kayıt edilemez.

Aynı şekilde, varsayalım ki, kullanıcının programı hangi saatte açıp kapattığını kayıt eden bir tablo bulunsun. Bu tabloyu bir "günlük rapor" gibi düşünebiliriz. Bu gibi bir durumda belirtili işlem için bir sınıf tasarlamaya gerek bulunmamaktadır.

Sınıflarda, Özellikler, Olaylar ve metodlar bulunurken, tablolarda sadece özellikler bulunur. Çünkü, metodlar ve olaylar, kayıt edilemez. Ancak özelliklerin değerleri kayıt edilebilir.



Figür 6.3 B : UML'de tablo gösterimi

Yukarıda soldaki şekil, bir sınıfı, sağdaki şekil bir tabloyu ifade etmektedir. Verilerini kayıt etmek istediğimiz bir sınıf bulunuyorsa, o sınıfın özelliklerine uyan bir tablo tasarlarız.

İlk bölümlerde bahsettiğimiz bir anket vardı, ankette kişilerin isim-soyisim-doğum tarihi ve meslek bilgileriyle birlikte destekliyor olduğu siyasi parti bulunuyordu. Bu anketin formunu tasarlayalım.

İsim	<input type="text"/>
Soyisim	<input type="text"/>
D. Tarihi	<input type="text"/>
Meslek	<input type="text"/>
S. Parti	<input type="text"/>

Figür 6.3 C : Bir anket örneği

Yukarıda gördüğünüz anketin, veritabanı içindeki tablosunu tasarlırsak:

Anket		
İsim		VARCHAR(20)
Soyisim		VARCHAR(20)
DoğumTarihi		DATETIME
Meslek		VARCHAR(30)
DesteklediğiSiyasiParti		VARCHAR(30)

Figür 6.3 D : Anket Tablosu

Dikkat ederseniz; isim ve soyisim için, en fazla 20’şer karakter, meslek ve desteklediği parti isimleri içinde en fazla 30’ar karakter kullanılıyor. Birde “DateTime” formatında bir doğum tarihi özelliği ekleniyor. Bu tasarımı yaptıktan sonra, yapmamız gereken sadece kağıtlara doldurulmuş olan anketleri, bu oluşturduğumuz tabloya girmek olacaktır.

Üniversite başvuru formları, askerlik müracat formları, ... binlerce farklı form aynı şekilde, tablo oluşturularak bilgisayara kayıt edilir.

6.4 Entity - Relation yapısı

Daha önceki bölümlerde, birçok veri tipini incelemiş; metinlerin ve tamsayıların özelliklerini belirtmiştik. Bir nesneyi tüm ayrıntılarıyla belirtmek yerine, o nesneyi tanımlamızı sağlayan bir tür anahtar kullanabiliriz. Daha önceki konularda anahtarın; bir küme içinde, bir nesneyi diğerlerinden ayıran bir özellik olduğunu söylemiştik. Örneğin, TCKimlik numarası, Türk vatandaşları (kümesi) içinde, her bir kişiyi (nesneyi) tanımlayabilir yani her bir TCKimlik numarası, bir kişiye özeldir, numarası verilen kişinin bilgileri bulunabilir, belirtilen kişinin TCKimlik numarası bulunabilir.

İlkokulda – lisede yoklama yaparken isim yerine, numara okunur ve bizde “burada” diye onaylama cevabı veririz. Peki neden? Tabiki, öğretmenin veya yoklamayı yapan kişinin, tek tek isimleri (hatta söyisimleri) okuması uzun süreceği için.

Bu durum, bilgisayar dünyasında da aynı şekilde kullanılmaktadır. Yani siz bir öğrenciyi belirtmek için, isim soyisim sınıf gibi verileri girmek yerine, sadece numarasını girerseniz, program, o kişiyi çok daha kolay bir şekilde bulacaktır.

Bilgisayar (daha doğrusu işlemci), tamsayı değerlerini, metinsel değerlerden daha hızlı işler. Okullarda, öğrencilere verilen numaralar 1’den başlar ve tek tek artar. Bir öğrenci mezun olsa, okuldan atılsa, başka okula gitse bile, o numaraya tekrar geri dönülmez. Örneğin 512 numaralı bir öğrenci olduğunu varsayalım ve bu kişiden sonra kayıt olan 10 kişinin olduğunu varsayalım. Sonuçta en sonra öğrenci numarası, 522’dir. Varsayalım ki, 512 numaralı öğrenci, başka bir okula gitti, okuldan atıldı ... veya herhangi bir nedenle okuldan çıktı. Daha sonra gelen bir öğrenci boş olan 512 numarasını ALAMAZ! Yeni gelen öğrenciye 523 numarası verilir, sonrakilere 524, 525, 526... Bu işlemin amacı, ileri de birgün o öğrenci (512 numaralı) ile ilgili bir belge istendiğinde, karışıklık olmasını engellemektir.

Bilgisayar dünyasında, veritabanında, her nesneye (yani kayıda) otomatik bir numara verdirebilirsiniz. Böylece, o kayıda ulaşmak için, isim, soyisim veya benzeri metinsel alanları kullanmak yerine, sadece o kaydın numarasını belirtmeniz yeterli olacaktır.

Entity: bir sistemi oluşturan bir parça; varlık anlamına gelmektedir. Bir tabloyu oluşturan alanları temsil etmektedir. Yani bir tablonun alanları (kolonları) o birer entity’dir.

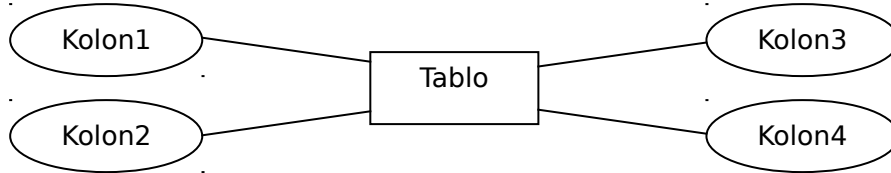
Relationship (ilişki, akrabalık): entity’ler arasındaki ilişkilere verilen addır.

Bir veritabanını tasarlarken, metinsel ifadeler veya bir kaydın tamamını belirtmekten, o kaydı tanımlayan bir numara kullanmanın daha mantıklı olduğunu anlatmıştım. Dördüncü bölümde, pointer’ın ne olduğunu öğrenmiştik. Pointer bir nesneyi gösteren bir tür işaretir yani o nesneyi, işaret eden bir tür anahtar – numara veya belirteçtir. İşte aynı şekilde, veritabanı sistemlerinde de, pointer, bir kaydı belirten numara olarak ifade edilebilir.

Sonuç olarak, Relationship = Pointer ifadesini kullanabiliriz. İlişki, numara türünde bir “alandır – (kolondur)” ve pointer görevi görmektedir. Başka bir tablodaki başka bir kaydı işaret etmektedir.

Relation ise; veritabanında sınıfları belirtmeye yarayan ifadedir, yani tabloları simgelemektedir.

Relation, dikdörtgen ile, entity ise elips ile ifade edilir. Aşağıda, örnek bir E-R diagramı gösterilmiştir.



Dikdörtgenler ve elipsler içine, o tablonun veya kolonun isimleri yazılır ve kolonlar hangi tablolara ait ise, o tablolar ile arasına birer çizgi çizilir.

Altıncı bölümde kullanılmış olan terimlerin ingilizceleri

Veri : data Veritabanı : database Kayıt : Record Satır : Row Kolon: Column Özellik : Attribute Alan : Field Tablo : Table	
--	--

SONUÇ

Veritabanları gelişmiş işlemlere olanak tanıyan dosyalardır. Daha önceki bölümlerde dosyaların ne kadar önemli olduğunu, zaten bilgisayarın yaptığı işlemin veri işlemek olduğunu söylemiştim. Bundan dolayı, veri tabanları inanılmaz önemli bir konuma gelir.

Günümüzde, neredeyse, her uygulamanın, her kurumun, her büyük şirketin veritabanları vardır. Askerlik kayıtları, ÖSS kayıtları, Vergi dairesi kayıtları, firma faturaları, e-posta sistemleri.... Hepsi birer veritabanıdır.

Özellikle endüstri ve iş dünyasında, veritabanlarının önemi çok büyüktür. Bu nedenle, yapılan birçok program veritabanları üzerine kurulur. Daha doğrusu, veritabanıyla birlikte tasarlandıktan sonra, program ile veritabanı arasında bir bağlantı oluşturulur.

7 Hata Düzeltme

İçerik

- 6.1 Veri tabanları
- 6.2 Veri tabanı yönetim sistemi
- 6.3 Tablolar / Kolonlar / Kayıtlar
- 6.4 Entity - Relation yapısı

Amaçlar

Altıncı bölüm,

- v,
- İleri değişken işlemlerini, programsal anlatmaktadır.

Anahtar sözcükler

yeni, oluşturucu, özel, genel, sabit, dosya, yorum, doğru, yanlış

7.1 Hatalar

Bu konuya, "Hatalar, programlamacıların ve sistem tasarlayıcıların kabusudur" şeklinde bir girişle başlamayacağım. Çünkü çok büyük firmaların bile bitirdikleri ürünlerde binlerce hata bulunabiliyor. Onlar yapabiliyorsa sizde yapabilirsiniz. Tabiki en önemlisi en az hatalı şekilde program yazabilmektir. Büyük firmaların bazıları, bu hataları bile bile yapmaktadır. Bile bile yapıyorlar ki, bir sonraki daha düzgün çalışan (daha az hatalı) ürünü de satın alınsın.

Ancak bir bilişim profesyoneli olarak iki davranışta; hatalı kod yazmak, bir sonraki ürünü aldirmaya çalışmak, yanlışır. Bir profesyonel olarak, hatasız kod yazabilmeliyiz, ancak baştan bilelim ki tamamen hatasız kod yazımı olamaz. Fakat, hatalarımızı tespit edebiliriz, test ederek hata sayısını azaltabiliriz, ve hataları düzeltebiliriz. Ayrıca sistemli bir şekilde çalışırsak, olası hataları önleyebiliriz.

Anlaşma Yazıyor olduğunuz her programın, her işleminin bir “**anlaşma**” sı olmak zorundadır. Daha doğrusu bu anlaşmanın olması, profesyonel programlama için gereklidir.

Bu anlaşma, bir programın hangi durumlarda, nasıl kullanılacağı, kullanım ortamını, ve kullanım şartlarını bildirmektedir. Günlük hayatta bir ürün aldığımızda, bu ürünün paketinden çıkan “kullanım kılavuzu” aynı amaca hizmet etmektedir.

Bir mikro dalga fırın aldığımızı düşünelim, eğer kullanma kılavuzunu okumaz (veya okusak bile) ve yanlış bir işlem gerçekleştirirsek (örneğin 5 dakika içinde hazırlanması gereken bir yiyeceği, 15 dk boyunca mikro dalga fırın içinde tutarsak), hatalar oluşacaktır (örneğin yemeğin yanması gibi).

Amerikalı kadının biri evine yeni bir mikrodalga fırın satın alır. Kadının bir de çok sevdiği kedisi vardır. Bir gün kadının kedisini yıkar. Tabii yıkadıktan sonra da kurutması gerekmiş... Kadının aklına bu işi çabucak halledebileceği parlak bir fikir gelir ve ıslak kediye alıp mikrodalga fırının içine koyar. Zavallı kedi, fırının kapağı tekrar açıldığında ölü bir şekilde fırının içinde yatıyordur. Bu durum karşısında kadın sevgili kedisini kaybetmenin intikamını almak için mikrodalga fırını üreten firmanın aleyhine yüklü bir tazminat davası açar.

Mahkemenin aldığı karar şöyledir : “Üretici firma, fırının kullanma kılavuzunda ‘içinde kedinizi kurutmayınız’ yazmadığı için suçludur ve istenen tazminatı ödemekle yükümlüdür.”

Haberlerden alıntıdır.

İşte yazılan her bir işlemde kullanılacak arabirimin, argümanların ve değerlerin hangi kurallara uygun olması gerektiğini belirtmek için hazırlanan metne, **anlaşma** denir.

Bu duruma basit bir örnek olarak, yazıyor olduğumuz bir programı düşünelim. Bu program, girilen sayıları sıralasın (küçükten büyüğe). Programın anlaşmasına da,

Programın sıralaması için girilen sayılardan herhangi bir tanesi negatif sayı olamaz
Programın sıralaması için girilen sayılar en fazla 10 tane olabilir

şartlarını belirtiyoruz. Eğer kullanıcı bu şartlara uymazsa, program doğal olarak hata verecektir. Anlaşma, kodu yazan ile kullanacak olanlar arasında bir bağlantı oluşturur.

Anlaşma metninin maddelerini, programı yazan kişi veya kişiler belirlemek zorundadır. 2.4 bölümünde, program durum ilişkisinden bahsetmiştik. Bir bilgisayar programının, insanlar gibi karar verme yeteneği olmadığı için, programı tasarlarken, karşılaşılabileceği her durumda, ne gibi işlemler gerçekleştirilmesi gerektiğinin belirtilmesi gerekmektedir. Program, kullanıcı tarafından kullanılmaya başladığında, daha önceden belirtilmemiş bir durumla karşılaşır, ya hata verir, ya da işlem gerçekleştirmez.

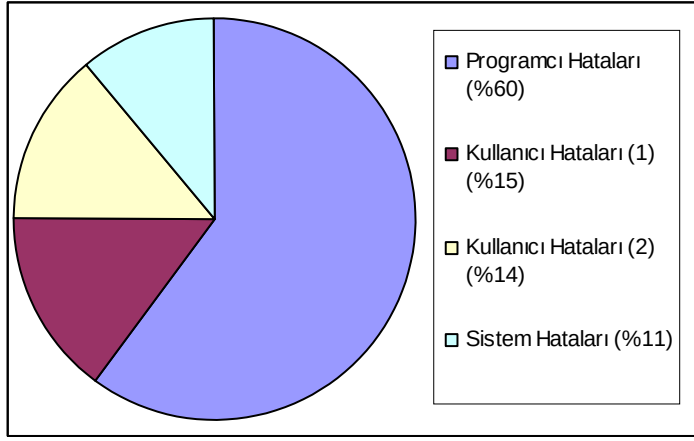
Bu duruma daha iyi bir açıklama olarak, ingilizce öğrenen bir öğrenciyi düşünelim. Öğrenciye, belirtme cümlesi kurmayı öğretirseniz ve soru sormayı öğretmezseniz, doğal olarak öğrenci, ingilizce cümle kurabilse bile, soru soramayacaktır.

Bir programı hazırlarken de, programda belirtilmemiş durumların, kullanma kılavuzunda ifade edilmesi gerekmektedir ki, kullanıcı, programı bu belirtilmemiş durumlar için kullanmasın. Aksi takdirde hatalar oluşabilir.

Kurallar belirlenerek, bir tür kısıtlama yapılır. 3. Bölümde, beynimiz içinde milyarlarca nöron olduğunu söylemiştik. Evrende de, milyarlarca, nesne, sistem, kavram, teori, durum söz konusu olduğundan, istediğimiz varlığa ancak bazı limitlemeler yaparak ulaşabiliriz. Bir program yazarkende (örneğin muhasebe programı), program, muhasebe ile ilgili olan herşeyi yapmayacaktır. Bazı durumlar ve bazı sistemler tasarlanmayacaktır. İşte bu, tasarlanmayan, program içine katılmayan bölümler, kullanıcıya belirtilmelidir ki, kullanıcı bunların farkında olsun. Basit bir örnek olarak, programın, yapılacak işlemlerde sadece YTL ile çalıştığını düşünelim (yani döviz ile ilgili işlem yapılamadığını varsayıyoruz). Bu durum kullanıcıya bildirilmelidir ki; kullanıcı döviz ile işlem yapamayacağını bilsin ve ona göre çözüm üretebilsin.

Bu anlattıklarımın “hatalarla ne alakası var” diyenleriniz olmuştur. Hatalar genelde 2 farklı konseptte toplanmaktadır: **Kullanıcı tabanlı hatalar** ve **Programcı tabanlı hatalar**. Tabi birde, sistem ile ilgili hatalar bulunmaktadır. Eğer, kullanıcıya, bir kılavuz vermezsek, doğal olarak, hata yapabilecektir. Kullanıcı tabanlı hataların önlenmesi veya azaltılması için, bir kılavuz hazırlamamız gerekmektedir. Tabi kullanıcılar bunların haricinde başka hatalarda yapabilir, bu tip hataları ileriki bölümlerde göreceğiz.

Kullanıcı tabanlı hataların ve programcı tabanlı hataların hepimiz kolay bir şekilde ne olduğunu anlamışızdır. Biri kullanıcının eksik bilgilendirmesinden dolayı oluşan hata tipi, diğeri de programcının hatalı – eksik kod yazmasından kaynaklanan hata tipidir. Sistemden kaynaklanan hatalarda kullanılan programlama dili, veritabanı sistemi, işletim sistemi gibi program geliştirme ortamından kaynaklanan hatalardır.



Figür 7.1 A : Hataların yüzdelerik dağılımı

Yukarıdaki figürde, program kullanımı sırasında oluşan hataların oranları gösterilmiştir. Kullanıcı hataları 1 diye ifade edilen oran, kullanıcıya eksik bilgi veya yanlış (anlaşma) verilmesinden kaynaklanan hataları (yani aslında bu hatalarda kodcunun hatalarıdır), kullanıcı hataları 2 diye ifade edilen oran ise, kullanıcının dikkatsizliğinden veya verilen anlaşmayı okumamasından kaynaklanan hataları göstermektedir. Görüldüğü üzere, hataların %75'i yani dörtte üçü, programcılar tarafından kaynaklanmaktadır. Aslında bu hataları da yine ikiye bölebiliriz: sistem planlayıcılar tarafından kaynaklanan hatalar [%25] (sistemi tasarlayan kişilerin hataları) ve tasarlanan sistemi koda döken kodcular tarafından kaynaklanan hatalar [%35 + %15 (kullanıcı hataları 1)].

Hataların oluşmasının suçunu kodculara atmamızın nedeni, sistemi tasarladıktan sonra testlerde ilk olarak sistem hatalarının kolay olarak saptanabilmesidir. Böylece, sistem hataları tespit edilmiş ve düzeltilmiş olacağından, sistem tasarım hataları azalmış olur.

Sistem hataları konusunda yapılabilecek çok fazla bir işlem yoktur. Çünkü sisteme (programlama dili – veritabanı..) müdahale olanağı bulunmamaktadır. Bu gibi durumlarda yani kullandığımız sistemde hata olması sonucunda yapabileceğimiz iki işlem bulunmaktadır. İlk işlem doğru sistemi – doğru versiyonu seçmek. Böylece kullanacağınız sistem içinde çakışma ve uyumsuzluk olmayacaktır. İkinci işlem ise kullandığımız sistemi tasarlayanların, sistemlerdeki hataları gidermek için hazırladığı yamaları (patch) bulup, bilgisayarımıza kurmak olacaktır.

Yama Bir bilgisayar üzerinde kurulu olan bir sistemin veya programın, içindeki hataları düzeltmek için, üretici firma tarafından tasarlanmış bir tür küçük programa verilen addır.

Kullanıcı hatalarını (2) azaltmanın yolu, programlama ile alakalı değildir. Bunun için, gerekli eğitimlerin firma personeline (kullanıcılara) verilmesi gerekmektedir. Tabiki program kullanımı hakkında verilecek olan bu eğitimde, programı tasarlayanlardan birisinin bulunması gerekmektedir.

Eğer her bir işlem için düzgün ve açıklayıcı bir anlaşma hazırlanır ve kullanıcıya sunulursa, kullanıcıdan kaynaklanan hatalar (1) önlenecektir. Bazı firmalar, yazılımla birlikte bir tür katalog vermektedirler bazıları da teknik destek servisi ile müşterilerine yardımcı olurlar.

Bu şekilde hataların %40'ını çok zor olmayan bir şekilde önlemeniz mümkün olacaktır. Geri kalan %60'lık bölümün saptanması, saptanma için izlenecek yol hakkında bilgiler sonraki bölümlerde mevcuttur.

Alıřtırmalar

- Anlaşma nedir? Örnek veriniz
- Hata çeşitleri nelerdir?
- Oluşan hataların nedenleri nedir?
- Yama nedir?

7.2 Hataların saptanması

Depremleri engelleyemeyiz, bu nedenle, depremleri saptamayı amaçlarız. Aynı şekilde, hataların bir kısmını da neredeyse engelleyemeyiz, mutlaka gözden kaçan 1-2 nokta olacaktır. Önemli olan bu hataların tespit edilmesi yani saptanmasıdır. Bazı hatalar, siz geliştirme ortamındayken otomatik olarak saptanabilir. Daha önceki konularda, IDE olarak tanımladığımız kod geliştirme ortamında yazdığımız kod eğer syntax hatası içeriyorsa yani yazım hatası varsa, bu hataları otomatik olarak göstermektedir. Bazı IDE'ler bu hata saptamasına ek olarak kodda yazdığınız (sadece bazı) mantıksal hataları da bulup göstermektedirler. Örneğin; hiç kullanılmayan bir değişkeni işaret edebilirler, değişken hafızadan alındıktan sonra o değişkenin değerini kullanmaya çalışırsanız bu değişkende işaret edilebilir (hafızadan atıldıktan sonra kullanılamaz... şeklinde bir mesaj ile).

Programcı tabanlı hatalarda, suç tamamen bizde, yani programı tasarlayan ve kodlayan kişilerdedir. Malesef hataların tamamen önlenmesi neredeyse imkansızdır. Ancak eğer, programın çalışacak olduğu her durum için bir test yapılırsa, o zaman hatasız bir program yazmak mümkün olacaktır.

Bir program içinde, binlerce, onbinlerce hatta yüz binlerce satır kod olabilir. Önemli olan bu binlerce satırlık programın içinde, hatanın nerede olduğunun tespitidir.

Runtime Error

Description: An application error occurred on the server. The current custom error settings for this the details of the application error from being viewed.

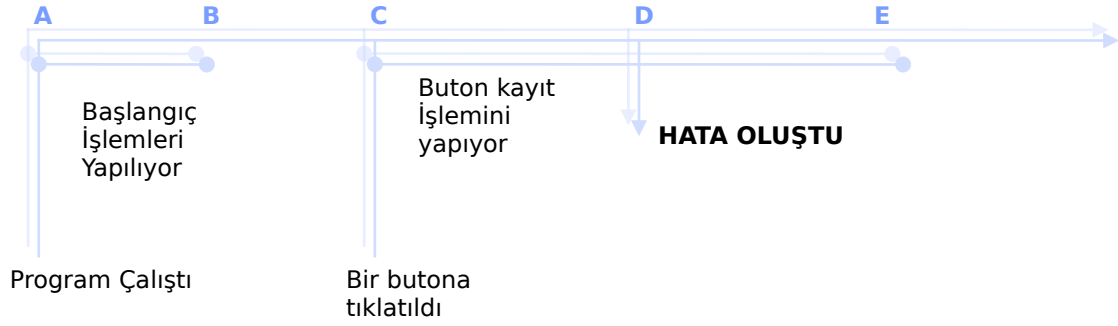
Details: To enable the details of this specific error message to be viewable on the local server machir <customErrors> tag within a "web.config" configuration file located in the root directory of the current w <customErrors> tag should then have its "mode" attribute set to "RemoteOnly". To enable the details to b machines, please set "mode" to "Off".

```
<!-- Web.Config Configuration File -->

<configuration>
  <system.web>
    <customErrors mode="RemoteOnly"/>
  </system.web>
</configuration>
```

Figür 7.2 A : Hata vermiş bir program örneği

Hata oluştuğu anda, hata mesajı görüntülenir, ve program (genellikle) çalışmayı durdurur. Programın çalışma evresini bir sayı doğrusu gibi düşünelim.



Figür 7.2 B : Bir programın işlem - zaman grafiği

Yukarıdaki doğruda, A noktası programın başladığı noktayı temsil ediyor. A-B arasında, programın ilk açılışında yapılması gereken işlemlerin gerçekleştirildiğini varsayalım (veritabanına bağlanma, ayarları yapma gibi). Bu açılış işlemleri B noktasına kadar sürüyor. Yani A-B noktası arasındaki işlemler gerçekleşiyor.

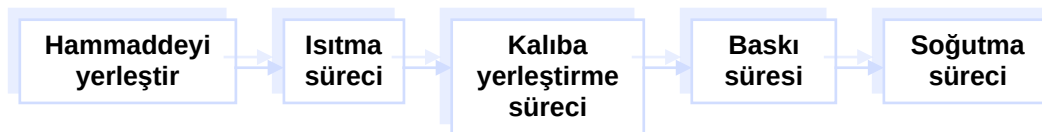
Daha sonra, kullanıcı, bir kayıt et butonuna basıyor (C noktası). Normal şartlara göre, C noktasında başlayan bu kayıt etme işlemi E noktasında bitmelidir. Ancak, C ile E noktası arasında bir yerde – D noktasında, hata oluşuyor ve program hata mesajını gösterdikten sonra duruyor.

Hatanın nerede olduğunu (yani D noktasının yerini) ilk bakışta keşfedemsek bile, kesin olarak bilebileceğimiz bir durum: hatanın C ile E noktası arasında olduğudur, yani butona tıklatıldığı ve tıklatmadan sonra işlemlerin başladığı andan, işlemlerin bittiği âna kadar olan bölümde olduğudur.

```
Void butona_tıklat(){
    //Formu kontrol et
    //Yeni kayıt oluştur
    //Verileri yeni kayıda gir
    //Formu temizle
}
```

Varsayalım ki, butona tıklatıldığında (C noktasında), yukarıdaki dört işlem gerçekleşiyor. Bildiğimiz, hatanın, programın geri kalan binlerce satırda olduğu değil bu dört işlemde birinde olduğudur.

Hataların tespiti için, endüstriden yardım alalım. Seri ürün üreten bir fabrikayı düşünelim. Fabrikada, ürünlerin uygun olup olmadığını belirlemek için KALİTE KONTROL işlemi yapılır. Uygun kalite düzeyinde olan ürünler bu kontrolden olumlu not alırken, uygun kalite düzeyinde olmayanlar sınıfta kalır. Aşağıda, basit bir ürünün üretimindeki evreler gösterilmiştir.



Figür 7.2 C : Bir ürünün üretim süreci

Profesyonel üretimde, hata olabileceği düşünülen her evrede kontrol yapılması gerekir

ve en sonunda, ürün tekrar kontrol edilir. Daha açık bir tabirle, yukarıdaki ilk işlemin (hammaddeyi yerleştir) ardından hammaddenin doğru yerleştirilip yerleştirilmediği, ikinci işlemin ardından, kazanın yeterli ısıya ulaşp ulaşmadığı, üçüncü işlem ardından, ısıtılan hammaddenin kalıba düzgün yerleştirilip yerleştirilmediği kontrol edilmelidir. Eğer, bu kontrollerin herhangi birinden olumsuz sonuç çıkarsa, o ürünün üretimi iptal edilir ve bir sonrakine geçilir. En sonunda, ürün tekrardan temel bir kontrole alınmalı ve ona göre, sonuç belirlenmelidir.

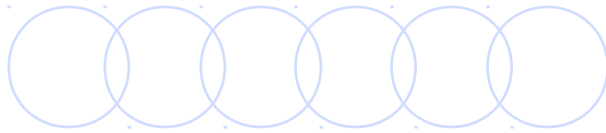
Önceki yıllarda (70’lerde), liselerde, her ne kadar sene içinde, sınava girip başarılı olunsu bile, sene sonunda yapılan büyük (final sınav) sınavdan geçer not alınmadığı takdirde, sınıfta kalınırdı. Aynen bu şekilde hem endüstride, hemde programlamada, final test çok önemlidir.

İşte bizde yazıyor olduğumuz her kodu, hem yazdığımız anda hemde, diğer kodlarla birleştirildiğinde kontrol etmeliyiz. Bunun için yukarıdaki gibi bir işlem zinciri düşünelim.



Figür 7.2 D : İşlem zincirleri

Hazırladığımız bu iki zinciri, tamamladıktan sonra, sağlam mı değil mi diye kontrol ettiğimizi düşünelim. Daha sonra bu iki zinciri birleştirip yeni ve daha uzun bir zincir oluşturalım.



Figür 7.2 E : Birleştirilmiş işlem zincirleri

Elde ettiğimiz bu yeni zinciri, tekrar, sağlam mı değil mi diye kontrol etmeliyiz. Aynı durum, programlama içinde geçerlidir. Yazdığımız bir prosedürü, fonksiyonu tamamladıktan sonra kontrol ederiz ve diğer elemanlarla – fonksiyonlarla – sınıflarla birleştirdiğimizde tekrar kontrol ederiz. Bu kontroller sonucunda, sistemin %90 doğru çalıştığına kanaat getirebiliriz.

Yine temel bir ilke olarak kullanabileceğiniz bir söz belirtmek istiyorum: “One system built on another, if one fails, so must the others”, yani “Bir sistem, başka bir sistemin üzerine kurulmuştur, eğer birisi bozulursa – yıkılırsa – hata verirse, diğerleri de yıkılmak zorundadır.” Bir binada, üst katlar, alt katların üstüne inşa edilmiştir. Eğer bir binanın alt katı yıkılırsa, diğer katlar bu alt katın üstüne kurulu olduğu için, diğer katlarda yıkılacaktır. Buradan çıkarmamız gereken sonuç, düzgün test edilmeyen bir kodu kullandığımızda, bu kod hata içerebileceğinden, kodun bağlı olduğu diğer bütün kodlarda çalışmayacaktır veya hata verecektir. Yazdığınız ikinci kod her ne kadar doğru olursa olsun, ilk kodda problem varsa, ve ikinci kod, ilk kodu kullanıyorsa, bu iki kodda, hatalıdır anlamına gelir.

Testte önemli olan her bir durumu göz önünde tutmak ve bu bütün durumlar için kontrolü sağlamaktır. Basit bir ifade ile, N adet kod birleştirilirken, 2 faktöryel (2!) tane

kombinasyon oluşur. Örneğin, A ve B adında iki fonksiyon yazdığımızı düşünelim. Bu fonksiyonların kombinasyon sayısı (2!) 4 olacaktır. Yani program tarafından bu fonksiyonlar 3 farklı şekilde çağrılabilir (yani kullanılabilir)

A fonksiyonunu kullan
B fonksiyonunu kullan
A fonksiyonunu, akabinde B fonksiyonunu kullan
B fonksiyonunu, akabinde A fonksiyonunu kullan

Yukarıda gördüğümüz gibi, 2 fonksiyon için 4 farklı durum oluşabilir. Yanlış dikkat etmemiz gereken önemli bir nokta bulunuyor, A ve/veya B fonksiyonu birbiri ile bağlantılı değilse, son iki durumun test edilmesine gerek yoktur. Yani, A fonksiyonu ile B fonksiyonu birbiri ile alakalı değilse, kendi içlerinde birbirlerini çağırıyorlarsa, birbirlerinin peşpeşe kullanılması durumunun test edilmesine gerek yoktur. Böyle bir durumda sadece A fonksiyonunu ve B fonksiyonunu test etmeliyiz.

Bu durumu biraz daha açık ifade edeyim. A fonksiyonunu bir açının (geometrideki) cotanjant değerini bulmak için yazdığımızı düşünelim. B fonksiyonunu da, girilen bir e-posta adresinin doğru olup olmadığını kontrol etmek için yazdığımızı düşünelim. Mantıksal olarak bu iki fonksiyon arasında bir bağlantı yoktur. O yüzden bu fonksiyonları, sadece tek birer kere test etmemiz yeterli olacaktır. Ancak varsayalım ki bir kullanıcı giriş formu tasarlıyoruz ve içinde, C adında bir fonksiyon bulunuyor ve bu C fonksiyonu, kullanıcı girişi yapan kişinin, adını – soyadını – e-posta adresini ve şifresini kontrol ediyor. Bu kontrol içerisinde de, e-posta adresini kontrol etmek için B fonksiyonunu çağırıyor.

```
function c() {  
    İsmi kontrol et  
    Soyismi kontrol et  
    b();  
    Şifreyi kontrol et  
}
```

Dikkat edersek, C fonksiyonu içinden, B fonksiyonu çağırıyor. Bu nedenle; A, B, C fonksiyonlarını test ettikten sonra, C – B fonksiyonlarını da test etmeliyiz. Çünkü, C fonksiyonu kendi başına doğru çalışıyor olabilir, B fonksiyonu kendi başına doğru çalışıyor olabilir, ancak ikisi birlikte çalıştıklarında yani C fonksiyonu, B fonksiyonunu çağırıldığında farklı bir durum oluşabilir. Bu nedenle her durumu kontrol etmek adına, aşağıdaki işlemleri kontrol etmeliyiz.

A : A fonksiyonu düzgün çalışıyor mu?
B : B fonksiyonu düzgün çalışıyor mu?
C : C fonksiyonu düzgün çalışıyor mu?
C - B : C fonksiyonu çalışıyor mu ve C fonksiyonu içinden çağrılan B fonksiyonu doğru çalışıyor mu?

Dikkat etmemiz gereken nokta, 3. sırada C fonksiyonunu test ederken, B fonksiyonunu çağırma yapacağımızdır. Yani, C fonksiyonunu, B fonksiyonunu çağırmadanki halini test ediyoruz.

```
function c() {
```



```
İsmi kontrol et  
Soyismi kontrol et  
Şifreyi kontrol et  
}
```

Yukarıdaki 3. sırada test edilecek olan kod yukarıdakidir.

```
function c() {  
    İsmi kontrol et  
    Soyismi kontrol et  
    b();  
    Şifreyi kontrol et  
}
```

Yukarıdaki 4. sırada test edilecek olan kod yukarıdakidir.

Konumuza geri dönelim. Varsayalım ki yazdığımız bir programda, bir işlemi gerçekleştirirken bir hata oluşuyor ancak bu hatanın neden kaynaklandığını bilmiyoruz. Hata tespiti için, bazı kriterlerin belirlenmesi gerekir.

Olay İlk kriter hatanın hangi işlemi gerçekleştirirken veya hangi olay gerçekleştikten sonra, hangi periyotta, hangi zamanlarda olduğunun tespitinin yapılmasıdır. Olayın saptanması en kolay kriterlerden biridir. Yani hatanın hangi işlem sırasında olduğunun bulunmasıdır.

Olayın saptanmasında dikkat etmemiz gereken, Program açıldığı anda mı hata veriyor? Eğer program çalıştığı anda bir hata oluşuyorsa, o zaman programın açılış prosedüründe bir hata bulunuyor demektir. Olayın gerçekleştiği nokta main prosedürüdür.

Program bir bölümde mi hata veriyor? Eğer hata sadece kullanmakta olduğunuz bir pencerede gerçekleşiyorsa, o zaman hatanın bulunduğu nokta, işlemleri yapıyor olduğunuz penceredir.

Program bir butona tıklattığımızda veya bir eylem gerçekleştirdiğimizde mi (bir tuşa basmak, fare ile tıklamak, fareyi hareket ettirmek....) hata veriyor? Eğer böyle bir durum söz konusu ise, yaptığımız işlemde hangi prosedürün çağrıldığını buluruz ve hatanın oradan kaynaklandığını söyleyebiliriz.

Program belirli bir periyotta veya belirli bir süre bekledikten sonra mı hata veriyor? Böyle bir durum söz konusu ise, periyodik olarak çalışan işlemleri, timer'ları kontrol etmeliyiz.

Aynı yukarıdaki figürde verdiğimiz örnekteki gibi, hata, hangi işlemin ardından gerçekleşiyorsa, hatanın başlangıç noktası, o eylemi başlattığınız noktadır.

Hata olayın nerede olduğunun saptanmasından sonra, hangi işlemde hata verdiğini bulmak gerekir, bunun için, Hata türü – Hata bilgisi gerekmektedir.

Hata Türü Hatanın nerede (hangi prosedürde) olduğunu tespit ettikten sonra, prosedür içinde hangi işlemde hata olduğu tespit edilirken, oluşan hata türüne ve

bilgisine göre limitleme yapılır.

Bu işlem için hata türleri hakkında bilgi sahibi olmalıyız ve hangi hataların hangi tip işlemlerde gerçekleşebileceğini bilmeliyiz. Hata türleriyle ilgili ayrıntılı bilgiler 8.4 bölümünde belirtilmektedir.

Eğer bu iki kriter hatayı bulmamıza yardım etmezse, yani hatanın yerini saptamakta zorluk çekiyorsak, o zaman en amatör yöntem olan, blok belirleme yöntemini kullanabiliriz.

```
metod herhangi_bir_metod () {  
    işlem1();  
    işlem2();  
    işlem3();  
    işlem4();  
    işlem5();  
}
```

Varsayalım ki yukarıdaki metotta bir hata bulunuyor ve hatanın hangi metottan kaynaklandığını bilmiyoruz. O zaman yapacağımız, her metodun öncesine bir Mesaj eklemektir.

```
metod herhangi_bir_metod () {  
    debug ('işlem1 çağrılıyor');  
    işlem1();  
    debug ('işlem2 çağrılıyor');  
    işlem2();  
    debug ('işlem3 çağrılıyor');  
    işlem3();  
    debug ('işlem4 çağrılıyor');  
    işlem4();  
    debug ('işlem5 çağrılıyor');  
    işlem5();  
}
```

Metodu yukarıdaki hale getirdiğimizde ve çalıştırdığımızda, sırasıyla, “işlem N çağrılıyor” ifadesi görüntülenecek ve akabinde belirtili işlem gerçekleştirilecektir. Ancak bu işlemler gerçekleştirilirken bir noktada hata verecektir. Hangi “işlem çağrılıyor” ifadesinden sonra hata mesajı görüntüleniyorsa, hata, o işlemten kaynaklanmaktadır.

Örneğin;

```
işlem1 çağrılıyor  
işlem2 çağrılıyor  
işlem3 çağrılıyor  
işlem4 çağrılıyor  
hata mesajı : (hata oluştu...)
```

Yukarıdaki örnekte gördüğümüz gibi, 4. işlem çağrılıyor ifadesinden hemen sonra hata oluşmuştur. Yani, bu metottaki hata, 4. işlemten kaynaklanmaktadır. Bu yöntem en basit en

amatör yöntemdir ancak hatanın nereden kaynaklandığını tespit edemiyorsak bu yöntemi kullanabiliriz.

Sistemi test etmek

Bilimadamları bir teori oluştururken veya icat yaparken, öncelikle bir fikir ortaya atarlar ve o fikrin yanlış olduğunu ispatlamaya çalışırlar. Eğer yanlış olarak ispatlayamıyorlarsa (tabi ki bu seviyeye, birçok test ve deneyden sonra ulaşılır), teori doğru olarak kabul edilir. İşte sizinde teoriniz, yaptığını programın doğru çalıştığı olacaktır. Daha sonra programı: “bu program doğru çalışmıyor” düşüncesi ile test ederek, çalışmadığını veya hata verdiğini ispatlamaya çalışacaksınız. Eğer ispatlarsanız içindeki hatayı tespit ettiniz demektir, daha sonra tespit ettiğiniz hatayı düzeltirsiniz ve tekrar denersiniz. Eğer bir hata bulamıyorsanız yani program içinde bir hata olduğunu ispatlayamıyorsanız o zaman, program düzgün çalışıyor diyebilirsiniz.

Yani sonuç olarak sistemi test etmek için sisteme düşman biri gibi davranmalı ve sisteme yanlış girdiler sokarak hata vermesini sağlamak için uğraşmalısınız. Taaa ki “program içinde hata yoktur” şeklinde tatmin oluncaya kadar.

Bunun haricinde dikkat edilmesi gereken nokta, kodu yazarken ne kadar sıklıkla test etmemiz gerektiğidir. Eğer süre olarak nitelendirirsek, her 6-7 dakikada bir test etmemiz gerekir. Yani, 6-7 dakika boyunca kod yazmalı akabinde test etmeliyiz. Eğer test etmek için çok kod yazıp beklersek, bir hata oluştuğunda bu hatanın nereden kaynaklandığını bulmakta zaman alacaktır. Eğer satır sayısı olarak kontrol ediyorsak, yazdığımız kodlar 20 satırı geçmeden test etmeliyiz. Eğer prosedürel olarak kontrol ediyorsak, her ana işlemi gerçekleştiren prosedür yazıldığında test etmek doğru olacaktır.

Alıştırmalar

- Hataların saptanması nasıl gerçekleşir?
- Hata türü nedir?
- Debug nedir?
- Sistem nasıl test edilir?

7.3 Test Yöntemleri

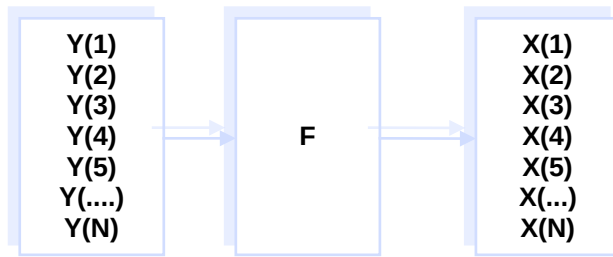
İnsanlar bugüne kadar test için birçok yol izlemişlerdir. Ancak, “birçok yol izlemişler” deyişinden de anladığımız gibi bu uygulanan yöntemler insanları tam olarak tatmin etmemiştir. Yani tam olarak bütün hataları bulan bir algoritma kuramamışlardır. Test edilme de, farklı proje türlerinde (program türlerinde) farklı metodlar kullanılır. Aslında bu öğreneceğiniz test metodları sadece programlama için değil, karşılaştığınız sistemleri test etmek içinde kullanılabilir.

7.3.1 Blackbox testi

Black box, kara kutu anlamına gelmektedir. Fonksiyonların testi için kullanılan ve oldukça başarılı sonuçlar elde eden bir yöntemdir. Aynı bir uçağın kara kutusu gibi, içini açmadan hangi eylemlerin gerçekleştiğini bilemezsiniz ve kara kutu testinde amaç, fonksiyonun içini görmeden, fonksiyonun doğru çalışıp çalışmadığını anlamaktır. Basit olarak bir fonksiyon için, belirli girdilere karşı belirli çıktıları bulunmaktadır. Örneğin;

$$y = f(x)$$

ifadesinde, x 'e bir değer verildiğinde, bu işlemin sonucunun y 'nin olması gereken değeri bildiğimizi düşünelim. Buradan, bir fonksiyonun girdi ve çıktı kümelerinin olduğunu söyleyebiliriz.



Figür 7.3.1 A : Bir fonksiyonun girdileri ve çıktıları

Yukarıdaki figürde görüldüğü üzere, eğer, belirli bir fonksiyonda hangi girdilerde, hangi çıktıları biliyorsak, bu girdi çıktı çiftlerini (yani $X(N) - Y(N)$) test ederek, olası hataları tespit edebiliriz.

Daha sonra, f fonksiyonuna x değerini verdiğimizde, eğer istenen çıktı olan y değerini veriyorsa, o zaman fonksiyonun bu durum için doğru çalıştığını söyleyebiliriz. Ancak fonksiyonlara girilebilecek girdi-çıkı sayısı çoğunlukla 1 den fazladır. Çünkü, eğer bir fonksiyonun çıktısı hep aynı olursa, o fonksiyonu kullanmamıza gerek kalmaz, o fonksiyon yerine bir sabit değer kullanılırız.

O halde, fonksiyonun girdi ve çıktıları, birden fazla durum için test etmeliyiz. Matematiksel bir örnek verelim;

$$Y = F(X)$$
$$F(X) = X + 1$$

$X, Y \in \mathbb{N}$ (doğal sayılar, 0 dan + sonsuza kadarki tam sayılar)

Yukardaki fonksiyon, girilen değerin 1 fazlasını döndürür. Tabiki bu fonksiyon herhangi bir işe yaramaz çünkü, bir sayının bir fazlasını zaten o sayıya bir ekleyerek bulabiliriz. Bunun sadece bir örnek olduğunu düşünelim.

Bu fonksiyona girilebilecek girdi sayısı sonsuzdur. Girdi olarak, her pozitif veya 0, tam sayıyı, alabilir, olası çıktı sayısı da aynı şekilde, sonsuz tane dir. Yani X için sonsuz tane varyasyon, Y için sonsuz tane varyasyon bulunmaktadır.

Bazen, bir fonksiyonun girdilerinin ve çıktılarının kombinasyonu (girdilerin alabilecek değ er sayısı X çıktılarının olabilecek değ er sayısı) çok fazla sayıda olabilir. Böyle durumlarda, her bir tanesini test etmek imkansızdır. O zaman sadece önemli, birbirinden farklı, değ iş ik türde girdi ve çıktıları test etmeliyiz.

Peki bu değ iş ik türde girdi ve çıktıları nasıl belirleyeceğiz? Test etmek istediğ iniz, fonksiyon için;

- Ç ıktı olarak NULL –NOTHING değ erlerini veren girdileri test etmeliyiz.
- Girdi olarak NULL –NOTHING değ erini vermeliyiz
- Argümanların alabilecekleri limitler dışında değ erler vermeliyiz
- Fonksiyon içinde, farklı bloklarla işlem yapılmasını sađlayan değ erler vermeliyiz
- Birkaç tane, standart girdi – çıktı denemeliyiz
- Fonksiyonun çalışmasını engelleyen bir durumu test etmeliyiz.
- Verilen girdi ile fonksiyonun çalışmayacağı (çalışmasının anlamının olmadığı) bir örnek test edilmelidir. Girdiler içinde beklenmeyecek durumları test etmeliyiz.

Bu ifadeleri açıklamadan önce önemli bir noktaya değ inmek istiyorum. Black Box testing, kara kutu anlamına gelir, yani fonksiyonun içinde hangi kodların yazdığını görmeyiz ve bilmeyiz. Sadece ne işlem gerçekleştirdiğini biliriz. Bu durum aynı, ÖSS sınavına veya çoklu seçenekten meydana gelen sınavlara benzer. Bu sınavlar bizim beynimizde ne olduğunu değ il, belirtili girdiye göre ne çıktı vereceğimizi ölçerler. Black box testing de, aynı şekilde, fonksiyonun içinde hangi satırların yazıldığını bilmeyiz, ne iş e yaradığını biliriz ve hangi girdilerde hangi çıktıları vermesi gerektiğini biliriz, ve bu girdi – çıktı ikililerini test ederiz.

Dördüncü ifadeyi daha sonra açıklayacağım. Şimdi bir örnekle bu ifadeleri açıklayalım. Bir fonksiyona bir liste olarak pozitif sayılar girdiğimizi ve fonksiyonun bu sayıları sıralayıp bize döndürdüğ ünü varsayalım. Eğer bir hata oluşursa, yanlış bir ifade girilirse, bize sonuç olarak NOTHING – NULL döndürmesini istiyoruz. Ayrıca, varsayalım ki; sayı listesi en fazla 10 tane sayıdan oluşabilmektedir.

Aşağıdaki talimatname programlamayı yapan kişiye (veya kendimize) vereceğimiz emirdir.

Hedef : Sırala
İşlem : Bir liste içinde tamsayıları girdi olarak al, ve sıralanmış liste olarak döndür
Limitler : Girdi uzunluğu en fazla 10 tane olacaktır.
: Girdi listesindeki bütün sayılar pozitif olacaktır

Function Sırala(girdi As List) As List

....

End Function

```
List Sırala( List girdi){  
....  
}
```

Method: Sırala(@girdi:list) Returns:list

Test edeceğimiz girdi - çıktılar;

Girdi Null ise (1. Madde)
Sırala (Null) => Null olmalı

Girdinin uzunluğu 1 ise yani eğer girdi listesinde 1 tane sayı bulunuyorsa (7. Madde)
Sırala ([7]) => Null olmalı

Girdiler limitler dışında ise
Sırala ([-3.2 , 8]) => Null olmalı (3. Madde)
Sırala ([3.2 , Null]) => Null olmalı (3. Madde)

Standart girdiler; (5.Madde)
Sırala ([5,2,7,9,3,86,0,12]) => [0,2,3,5,7,9,12,86]
Sırala ([2,1,98,4]) => [1,2,4,98]

Beklenmeyecek durumlar; (7.madde)
Sırala ([5, 2, 5, 0, 12]) => Null olmalı (aynı değerden 2 tane var)

Başka bir örnekle, diğer maddeleri de öğrenelim. Örneğin; bir liste içinde verilen pozitif tam sayıları sırayalan, daha sonra, ardaşık olanlarını döndüren bir fonksiyonu test edelim. Yani, bir dizi sayı verdikten sonra, bu sayıları önce sıralayıp daha sonra aralarında birer fark olanlarını (peş peşe takip edenlerini) sonuç olarak döndürecektir.

Hedef : Sırala2
İşlem : Bir liste içinde tamsayıları girdi olarak al, ve sırala, daha sonra ardaşık olanları döndür
Limitler : Girdi uzunluğu en fazla 10 tane olacaktır.
: Girdi listesindeki bütün sayılar pozitif olacaktır

```
Function SıralaveArdaşıklarıDöndür( girdi As List) As List  
....  
End Function
```

```
List SıralaveArdaşıklarıDöndür(List girdi){  
....  
}
```

Method: SıralaveArdaşıklarıDöndür(@girdi:list) Returns:list

Test edeceğimiz girdi - çıktılar;

Standart girdiler; (5.Madde)
Sırala2 ([5,2,7,9,3,86,0,12])=> [2,3]
Sırala2 ([2,1,98,4]) => [1,2]
Girdi Null ise (1.madde)
Sırala2(Null) => Null olmalı
Girdi uzunluğu 1 ise (7.madde)
Sırala2 ([23]) => Null olmalı
Girdiler limitler dışında ise
Sırala2 ([-3.2 , 8]) => Null olmalı (3. Madde)
Sırala2 ([3.2 , Null]) => Null olmalı (3. Madde)
Beklenmeyecek durumlar; (7.madde)
Sırala2 ([5, 2, 5, 0, 12]) => Null olmalı (aynı değerden 2 tane var)
Çalışmasına engel (6.madde)
Sırala2 ([1,3,5,7,8]) => Null olmalı

Bu durumda, peşpeşe olan hiç bir veri bulunmamaktadır. Görüldüğü üzere, bu tip bazı örnekler denenedikten sonra, istenilen sonuçların elde edilmesiyle, test, "geçti" sonucunu alır. Her fonksiyonu tek tek kontrol etmek tabiki en doğru olanıdır, ancak her fonksiyonu aynı yöntemlerle test etmemize gerek yoktur.

Black box testing, programlamacı bir kişinin (sizin değil - çünkü, eğer kodu siz yazmış olsaydınız, zaten içinde geçen satırları biliyor olurdunuz. Black box - kara kutu, test yönteminde kodun içeriği bilinmez) yazdığı fonksiyonu sizin, olası girdi - çıktıları deneyerek kontrol etmeniz demektir.

7.3.2 White box testi

Bu test tipinde, test edeceğimiz fonksiyonunun, içerisinde yazılı olan kodu görebilmekteyiz. Kodun içeriğini görebildiğimiz için, test edeceğimiz girdi ve çıktıları, kod içeriğine göre belirleyebiliriz.

Bu şekilde, kullanılacak olan, test örneklerini, işlemin koduna bakarak hazırlamalıyız. Kodu biz yazıyor ve içerisinde hangi işlemler olacağını biliyor olduğumuzdan dolayı, test edilecek örnekleri bizim seçmemiz daha doğru olacaktır. Bu metod, programcılar tarafından kullanılır, yazılımcılar ise, black box yöntemini kullanmaktadır. Çünkü, yazılım uzmanları normal (bizim ülkemizde olmasa bile) kod yazmazlar. Sadece, “şu işlemi, şu şekilde yapan bir kod yaz” talimatı verirler. Gerisini programcılar yapar ve en son testi, yazılım mühendisleri blackbox test ile yaparlar.

Yukarıdaki 4.maddenin kullanılabileceği bir örnek verelim. Bir fonksiyon yazmamız gerektiğini ve bu fonksiyonun, bir sayının mutlak değerini döndüreceğini varsayalım.

Örnek $y = f(x) \Rightarrow f(x) = |x|$

```
function Mutlak( x )  
if x < 0 then  
return -1 * x  
else  
return x  
end if
```

end function

Yukarıdaki ifade; eğer girilen arguman olan x değeri sıfırdan küçükse, sayıyı -1 ile çarp (Yani işaretini değiştir, negatif sayıyı pozitif yapar) eğer değilse, girilen sayıyı direk olarak döndür. Dikkat ederseniz burada 2 blok bulunmaktadır. Birinci blok, if ifadesinin doğru olduğu bloktur, ikincisi ise, else bloğudur. Yapacağımız testin bu iki bloğa girdiğinden emin olmanızı. İlk bloğa girebilmesi için, sayının negatif olması gerekmektedir. İkinci bloğa girebilmesi içinse pozitif olması gerekmektedir.

Mutlak (-1) $\Rightarrow 1$ (ilk bloğa girer)

Mutlak (1) $\Rightarrow 1$ (ikinci bloğa girer)

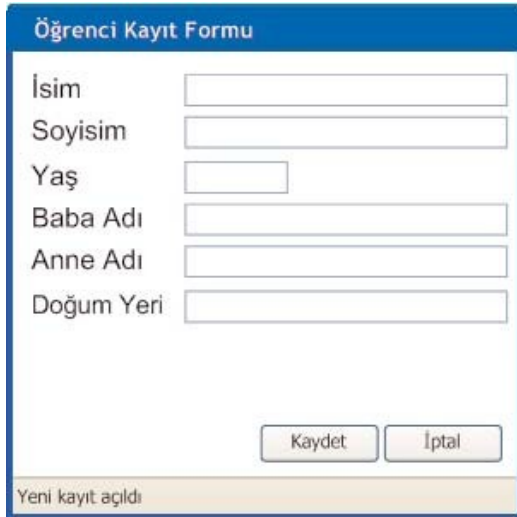
Mutlak (0) $\Rightarrow 0$ (ikinci bloğa girer)

Böylece, 7 farklı durumda, ne gibi girdileri test etmemiz gerektiğini öğrenmiş olduk. Tabiki, fonksiyonların iç kısımlarını da (programlamacı olarak) test etmemiz gerekmektedir, ayrıca, bazı fonksiyonların hiç argumanları yok ise, o zaman, yine fonksiyonunun içerisindeki kodu test etmemiz gerekmektedir.

7.4 Hata türleri:

7.4.1 Dış ortam hataları – Kullanıcı hataları

Öncelikle, dış ortam hatalarını inceleyelim. Bu tip hatalar, genelde, bir kullanıcının, programda işlev gerçekleştirecek olan bir bölümüne gönderilecek olan bir veriyi yanlış girmesiyle oluşmaktadır. Örneğin, bir öğrencinin adını, yaşını ve diğer bilgilerini, veritabanına kayıt eden bir program yazdığımızı düşünün. Kullanıcının, bir öğrenciyle alakalı olan bilgileri bir kullanıcı arayüzüne girdiğini varsayalım.



The image shows a web form titled "Öğrenci Kayıt Formu" (Student Registration Form). It contains several input fields: "İsim" (Name), "Soyisim" (Surname), "Yaş" (Age), "Baba Adı" (Father's Name), "Anne Adı" (Mother's Name), and "Doğum Yeri" (Place of Birth). Below these fields are two buttons: "Kaydet" (Save) and "İptal" (Cancel). At the bottom of the form, there is a status message: "Yeni kayıt açıldı" (New registration opened).

Figür 7.4.1 A : Bir form örneği

Eğer kullanıcı, YAŞ diye adlandırılan kutuya bir 3 haneli bir sayı girerse, bu bizim için bir hata oluşturacaktır. Çünkü, (3 haneli en küçük sayı 100, en büyük sayı 999'dur) hiçbir öğrenci 100 ile 999 yaşı arasında olamaz. Başka bir örnek verelim, eğer kullanıcı, yaş kutusuna değer olarak 3 girerse, bu durum bizim yine aynı şekilde, hata oluşturur. Çünkü, 3 yaşında bir öğrenci olamaz. Yine başka bir örnek olarak; yaş kutusuna değer olarak, "a" veya bir metin değeri girilirse, program yine hata verecektir. Çünkü, "a" veya başka metin sayıya çevrilemez (Bazı durumlar hariç). Bir kayıt sırasında oluşabilecek birçok hata olabilir. Bu hatalara bir ek olarak, isim kutusuna herhangi bir değer yazılmaması verilebilir. İsmi olmayan bir insan yok olacağından bu durum yine bir problem teşkil eder. Bu hatalara "Kullanıcı Girdi Hataları" denir. Dikkate ederseniz, kullanıcı girdi hataları da 4'e ayrılır.

- 1 - Yanlış veri formatı. Örnek : Yaş yazılması gereken yere (bir tam sayı), metin yazmak
- 2 - Limit dışı veri. Örnek : Yaş yazılacak alana, 236 yazılması (bir kişi 236 yaşında olamaz)
- 3 - Boş alan. Örnek : Yukarıdaki kayıt formunda, isim, soyisim gibi kutular asla boş bırakılamaz.
- 4 - Kodcu tarafından yapılan limitleme : Örneğin, kodcu, sadece 15 ile 18 yaş arasındaki öğrencilerin kaydının yapılmasına izin veriyorsa ve bu aralık dışında bir veri girilirse, bu durum program için hata teşkil eder. Başka bir örnek olarak, aynı isim soyisim yaş anne adı baba adı doğum yeri bilgileri ile bir öğrenci daha kayıt edilmek istenebilir. Kodcu doğal olarak bu duruma izin vermez.

Kullanıcıların yapabilecekleri başka tip hatalarda bulunmaktadır. Örneğin, eğer bir kullanıcı programın çalışması için gerekli bir dosyayı silerse, veritabanını değiştirirse, veya genel olarak, programın dışından okunacak olan bir veriyi manuel olarak değiştirirse, bu

durum yine bir problem teşkil edebilir.

Bunların haricinde, ayarlama (configuration) ve kurulum hataları bulunmaktadır. Bazı durumlarda, programın veya bilgisayarın ayarlarında gerekli değişikliklerin yapılması gerekebilir. Eğer bu ayarlamalar eksik veya yanlış yapılmışsa programda hata oluşabilir.

Bir programı kurarken bazı dosyaların yüklenmesi veya bazı programların daha kurulması gerekmektedir. Kurulma esnasında eğer bu belirtilenler tam yapılmamış ise, bu durum program için hata oluşturabilir.

Peki bu hatalardan nasıl korunuruz? Kullanıcının, girdilerini kontrol etmek için, her kutuya giriş yaptığında, kontrol etme uygulanabilir. Örneğin, isim kutusuna yazı yazıp, diğer kutuya geçtiği anda, isim kutusuna yazılmış olan veri kontrol edilebilir. Aynı şekilde, bu durum her bir kutu veya girdi için yapılır. Eğer bir kutuya uygun olmayan bir değer girmişse, o kutudaki değer silinebilir.

Ancak kontrolün tamamen sizde olduğundan emin olmak için, kullanıcı verileri girerken, tek tek her girdiyi kontrol etseniz bile, kaydet butonuna tıklatıldığında, TEKRAR bir kontrol yapmanızda fayda vardır. Çünkü, bazen bir yolla, girilen bu veriler, doğrulansa bile daha sonra, değiştirilebilir. Peki hem veriyi girdikten sonra, hemde kayıt butonuna bastığımız anda, iki kere kontrol etmemize gerek var mı? Tabi ki yoktur. İşte bu yüzden, sadece ve sadece kayıt butonuna basıldığı anda kontrolleri yapıp, eğer bütün kontroller geçerse, kayıt işlemini gerçekleştiririz, eğer geçmezse, kullanıcıya hatayı bildiririz.

Dosya ve ayarların kontrolü, biraz daha karışıktır. Örneğin programın çalışması için gereken bir dosyanın kullanıcı tarafından silindiğini veya değiştirildiğini düşünelim. Bu şekilde, program çalışamayacaktır. İşte böyle bir durumda, program açıldığında, ilk olarak bu gerekli olan dosyaları kontrol etmek gerekmektedir. Dosyaların kontrolü için;

Dosyanın var olup olmadığını

Dosyanın değiştirilme ve oluşturulma tarihlerinin aynı olup olmadığını kontrol edebiliriz.

Örnek; Programımızın çalışması için 2 dosya gerektiğini bu dosyaların, system.ppx ve hierarchy.ppx isminde olduğunu düşünelim.

Öncelikle kontrol etmemiz gereken, bu dosyaların var olup olmadığıdır.

DosyaVarmı ("system.ppx") ?

DosyaVarmı ("hierarchy.ppx") ?

Daha sonra, dosyaların değiştirilme ve oluşturulma tarihlerinin, doğru olup olmadığını kontrol ederiz.

DosyaOluşturmaTarihi ("system.ppx") =? new Date(14, 10, 2006)

DosyaDeğiştirmeTarihi ("system.ppx") =? new Date(14, 10, 2006)

DosyaOluşturmaTarihi ("hierarchy.ppx") =? new Date(14, 10, 2006)

DosyaDeğiştirmeTarihi ("hierarchy.ppx") =? new Date(14, 10, 2006)

Eğer ayarlar değiştirilmişse, ve ayarları kontrol etmek istiyorsak, bu ayarları

-ya kullanıcının bulamayacağı başka bir yerde tutup karşılaştırabiliriz.
-yada tek tek her bir ayarı kontrolden geçirerek, doğru değerlerin girilip girilmediğini kontrol edebiliriz.

Boş alan ve kodcu tarafından yapılan limitlemelerde ise, yapılacak tek işlem, bir fonksiyon yazarak, verinin, bu kontrolden geçip geçmediğini denemek olacaktır.

```
Function IsimKontrolEt (isim as string) as boolean
```

```
if isim = null then
```

```
return false
```

```
end if
```

```
if isim.length < 2 then
```

```
return false
```

```
end if
```

```
for each karakter in isim.chars
```

```
if NOT isLetter (karakter)
```

```
return false
```

```
end if
```

```
next
```

```
return true
```

```
end function
```

Yukarıdaki fonksiyon girilen verinin bir isim olup olmadığını döndürür. İlk başta eğer, isim olarak gönderilen arguman, null ise, false (yani olumsuz) döndürür. Daha sonra eğer isim argumanının uzunluğu 2 den az ise (tek harfli ise) o zaman tek harfli isim olamayacağından, yine false döndürür. Sonraki işlem, isimdeki herbir karakter için, eğer karakter bir harf değil ise (yani bir sayı ise veya bir işaretse) false döndürür. Eğer bu işlemlerde false dönmüyorsa, sonuç olarak bütün kontrollerden geçmiş demektir ve true döndürür.

Örneğin;

```
IsimKontrolEt ( "Ali" ) => true
```

1

```
IsimKontrolEt ( "x" ) => false
```

2

```
IsimKontrolEt ( "A" ) => false
```

3

```
IsimKontrolEt ( "deneme" ) => true
```

4

```
IsimKontrolEt ( "Ahmet1" ) => false
```

5

```
IsimKontrolEt ( "Bilal%" ) => false
```

6

```
IsimKontrolEt ( null ) => false
```

7

```
IsimKontrolEt ( "" ) => false
```

8

1. ifade, hepsi karakter olduğu için ve uzunluğu 1'den büyük olduğu için doğru döndürür

2. ifade, uzunluğu 1 olduğu için yanlış döndürür
3. ifade, uzunluğu 1 olduğu için yanlış döndürür
4. ifade, uzunluğu 1'den fazla olduğu ve içinde sadece harf olduğu için doğru döndürür (isim olmasa bile!)
5. ifade, içinde sayı olduğu için yanlış döndürür.
6. ifade, içinde % olduğu için yanlış döndürür.
7. ifade, null olduğu için yanlış döndürür.
8. ifade, empty olduğu için yanlış döndürür.

4. ifadedeki, “deneme” isim olmadığı halde bile doğru döndürdü ancak “deneme” isim değildir, bunu nasıl ayırabiliriz? Bu durum için malesef mantıklı bir yöntem bulunmamaktadır. Tek uygulanabilecek yöntem, bütün isimlerin yazılı olduğu bir liste hazırlamak ve girilen ismin bu listede olup olmadığını kontrol etmektir. Ancak bu işlem uzun süreceği, listemizde bulunmayan isimlerden biri girilebileceği için kullanılmaz.

Zaten bir kullanıcı, girdilere, bile bile ! yanlış değerler girmez. Klavyeyi kullanırken başka bir harfe yanlışlıkla basmış olabilir, veya dalgınlıktan en son duyduğu kelimeden bir harf yazmış olabilir. O yüzden kullanıcıları düşman olarak görmek yanlıştır. Ancak kullanıcılara çok temkinli yaklaşmak gerekmektedir. Çünkü, kullanıcılar, programın giriş yapılabilen HER yerinde hata yapabilirler. O yüzden, programın her bölümünü kontrol etmeliyiz.

Tabi birde, kullanıcıların, programı yanlış amaçla kullanması, bilmeden veya anlamadan özellikleri kullanmaya çalışması durumu bulunmaktadır. Bu konu ile ilgili bilgiyi, 8.1 Hatalar bölümünde vermiştik. Kullanıcının programı doğru düzgün kullanabilmesi için gerekli olan bir kullanım kılavuzunun tasarlanamaması gerektiğini belirtmiştik.

7.4.2 Kodcu hataları

Malesef, kullanıcı hatalarının 4 çeşit olmasına karşılık, kodcu hataları çok daha fazla türdedir. Ancak bu durum son derece doğaldır. Bir televizyonu kullanan kişiler, o televizyonun kanal değiştirme - ses açıp kapama gibi işlemlerini kontrol ederler. Televizyon kullanırken yapabilecekleri çok hata bulunmamaktadır. Halbuki, televizyonu üretenler, onun içindeki, binlerce farklı sistemi tasarlarlarken, birçok hata yapabilecektir. Ayrıca unutmayalım ki, kullanıcı hatalarında önlenmesi için çalışanlar, kodculardır.

Peki, bu kodcu hatalarının türleri nelerdir?

Argument Error (Argüman Hatası)

Varsayalım ki, iki nesneyi birbiriyle karşılaştıracakız. Bir sayı ile (integer), kendi belirlediğimiz türde bir nesneyi, aynı türde olmadıkları için karşılaştıramayız.

```
public void Main() {  
    Sınıf1 x = new Sınıf1();  
    int sayı = 34523;  
    int i = sayı.CompareTo(x); // Hata  
}
```

Böyle bir durumda, Sınıf1, bir sayı olarak ifade edilemeyeceğinden dolayı, program hata verecektir.

Unutmayalım ki, iki nesneyi karşılaştıracığımız bir durumda, bu nesnelerin aynı sınıfa ait olmaları gerekmektedir. Argüman hatası, bir metoda gönderilen argüman eğer doğru değilse, oluşur.

Argument Null Error (Argüman Boş Hatası)

Bu hata aynı argüman hatasına benzemektedir. Bir metoda gönderilen argüman eğer NULL ise, birçok durumda, metod için hata oluşur ve Argument null hatası gösterilir.

```
public void Main() {  
    String s = null;  
    String j = String.Trim(s);  
}
```

Burada, 4. satırda, Trim metoduna gönderilen s argümanının değeri NULL'dur. Doğal olarak burada, Argument Null hatası gösterilecektir. Bildiğimiz üzere, Trim fonksiyonu, argüman olarak girilen metnin başındaki ve sonundaki boşlukları silecektir. Ancak, argüman olarak null gönderilmesinden dolayı hiçbir işlem gerçekleşmeyecektir.

Divide By Zero Error (Sıfıra Bölünme Hatası)

Bildiğimiz gibi, matematiksel işlemlerde, bir ifade, 0'a bölünemez. 0'a bölündüğünde, sonuç tanımsız çıkacaktır. İşte aynı işlem eğer programlama da yapılırsa, yani bir ifade 0'a bölünürse, aynı hata oluşacaktır.

$a = (3+4) / 0$

Yukarıdaki ifade de, işlemin ikinci bölümünde (birinci bölümde 3 ile 4 toplanır) toplam, 0'a bölüneceği için, tanımsız sonucu çıkacaktır ve DivideByZero hatası gösterilecektir.

Out Of Memory Error (Hafıza Limiti Aşımı Hatası)

Bu hata, programın, kullanılabilir belleğin dolmasıyla gösterilir. Out of memory, hafıza limiti dışında anlamına gelmektedir. Varsayalım ki, binlerce nesne oluşturuyoruz, veya çok büyük verileri hafızaya alıyoruz. Bunun gibi benzer işlemlerde Out of memory hatası gösterilebilir. O yüzden, hafızayı en uygun şekilde kullanarak, sadece gereken nesneleri oluşturmaları, gereken verileri hafızaya almalı, ve işlevleri bitince, Garbage Collector'u çağırmalıyız.

Overflow Error (Değer Limiti Aşımı Hatası)

Bir değişkene, alabileceği değerden daha fazla bir değer ataması yaparsak, overflow hatası gösterilir. Örneğin, bildiğimiz gibi, bir byte 0 ile 255 arası tamsayı değerlerini tutabilmektedir. Eğer byte olarak tanımladığımız bir değişkene 255 den daha büyük veya 0 dan daha küçük bir değer atarsak, overflow hatası görüntülenecektir.

```
dim i as byte  
i = 260
```

Yukarıdaki ifade de eşitliğin olduğu (2. satırda) satırda, overflow error gösterilecektir.

Null Reference (Referans Boş Hatası)

Bir sınıftan bir nesne türettiğimizi düşünelim. Daha sonra bu nesneyi, NULL'a eşitleyerek, hafızadan kaldırdığımızı düşünelim. Bundan sonra, o nesneyi kullanarak herhangi bir işlem yapmaya kalktığımızda, o nesne artık olmadığı için, null reference hatası gösterilecektir.

```

public class Deneme {
public string x="xxx";
}
public class NullRef {
public void Main() {
Deneme d = new Deneme();
d = null;
string s = d.x; //Hata
}
}

```

Dikkat edersek; ilk başta oluşturulan Deneme sınıfı, d değişkenine yazılıyor ve daha sonra, null'a eşitleniyor. Null'a eşitlenen bir nesne artık yok olduğundan o nesneyi herhangi bir işlemde kullanamayız ve belirtili satırda, null reference hatası gösterilir.

Invalid Cast Error (Yanlış Değiştirme – Dönüştürme Hatası)

Eğer bir türde olan veri, başka bir türe çevrilemiyorsa, invalid cast hatası gösterilir.

```

dim i as integer
i = "deneme"

```

Burada, "deneme" metni, integer olan sayıya çevrilemediği için, invalid cast hatası gösterilir.

Index Out Of Range Error (İndeks Limit Aşımı Hatası)

Varsayalım ki, bir array tanımladık. Bu array'in uzunluğu 5 olsun. Eğer biz, array'in 6. elemanına ulaşmaya çalışırsak (öyleki limit dışında) index out of range hatası görüntülenir

```

dim s(4) as string
s(0) = "deneme"
s(1) = "deneme"
s(8) = "deneme" //hata

```

Buradaki örnekte 3. satırda, array index i olarak belirtilen 8, limit dışında olduğundan dolayı, index out of range hatası görüntülenir.

Stack Over flow Error

Daha önceki bölümlerde programların, hafızada, kendi stacklarına yüklendiğini söylemiştik. Eğer stack a atılan bu program içerisinde, bir veya birden fazla metod, kendini veya başka metodları, birçok kere çağırırsa (programlama diline göre değişen bir limit), stack ın limiti dolacaktır ve daha fazla prosedür çağrılanmayacağı için program stack over flow hatası gösterecektir.

```

public static void recursion() { recursion(); }
public void Main() {
recursion();
}

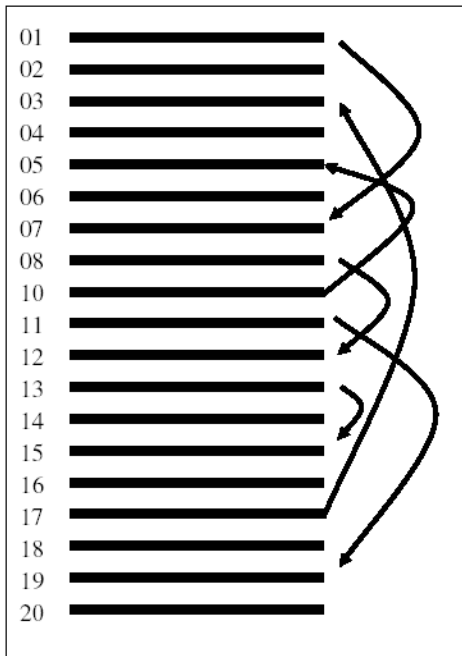
```

7.5 Hataların azaltılması

7.5.1 Kod düzeni

Dikkat edilmesi gereken başka bir nokta ise, kod düzenidir. Nasıl resmi yazışmalarda, dilekçelerde veya benzeri formlarda, yazılan yazılar belirli bir düzende ise, aynı şekilde yazdığımız kodlarda belirli bir düzende olmalıdır.

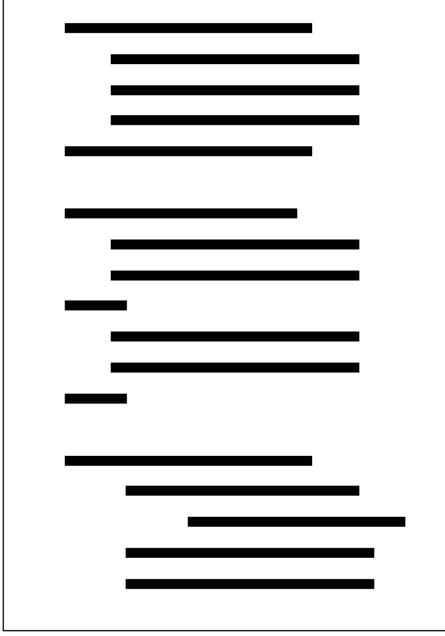
Kodlar genelde, kısa ifadeler şeklindedir. Örneğin, yukarıdaki paragrafın, ilk satırında yani; *“Dikkat edilmesi gereken başka bir nokta ise, kod düzenidir. Nasıl resmi yazışmalarda”* ifadesinde 85 karakter bulunmaktadır. Diyebiliriz ki, bir satırda ortalama 80-90 karakter bulunmaktadır. Halbuki, kodlamada, bir satırın ortalama uzunluğu 30-50 karakterdir. Aslında kodlamada, bir ifade uzunluğu, bir satırı geçebilir, 200,400,500 karakter olabilir. Ancak, bazı ifadeler de vardır ki, sadece 4-5 karakterden hatta bazen 1 karakterden oluşan ifadeler bulunmaktadır. Buradan çıkaracağımız sonuç; X satır normal yazıda, X satır kodda olandan daha fazla karakter bulunduğuudur. Ancak, ifade sayısı daha az olmasına rağmen, kodların anlaşılması daha zordur. Bunun üstüne birde, kodun karışık yazılmasını eklersek, kodumuz iyice anlaşılmaz hale gelecektir.



Figür 7.5.1 A : Spagetti kod

Yukardaki figürü, bir dizi kod satırı olarak düşünelim. Bu kodda, işlemlerin anlaşılması zor olacaktır. Çünkü beşinci bölümde bahsettiğimiz, programsal bloklar, düzgün olarak belirtilmemiştir. Beşinci bölümde her bir blok başlangıcında, bir “sekme” kullanmamız gerektiğini ifade etmiştik.

Yukarıdaki kod, “spagetti kod” olarak nitelendirilir. Yani içinde ne yazıldığı belli olmayan karmaşık bir yapı olduğunu ifade eder. Aşağıdaki kod ise, düzgün koddur ve yukarıdaki kodun olması gerektiği gibidir.



Figür 7.5.1 B : Düzenli kod

Dikkat ederseniz her blok başlangıcında bir satır başı yapılarak içeri girilmiş böylece, blokların birbirlerinden ayrılması kolaylaşmıştır.

7.5.2 Yorumlar - Açıklamalar

Düzen, bir olayın anlaşılması için bilinmesi gereken en önemli etkidir. Bu nedenle, bir kodun düzenli yazılması çok önemlidir. Bloklarda sekme kullanmak, kodun ne işe yaradığını anlatmayacaktır. İşte bu yüzden, kod içerisinde, kodun ne işlev gerçekleştirdiğini anlatan yorumların yani açıklama yazılarının bulunması gerekmektedir.

Not : Yorumları belirtmek için

- bazı dillerde satır başına // (çift taksim),
- bazı dillerde satır başına ' (tek tırnak),
- bazı dillerde satır başına -- (çift tire)
- bazı dillerde yorum başına /* ve sonuna */ ifadesi eklenir.

Yorum olarak belirtilmiş bir satırda, kod bile olsa, derleyici bu ifadeyi önemsemez!

Yorumlar, ifadeleri ve prosedürleri açıklamak için kullanılmaktadır. Derleyici ve Yorumlayıcı, bu satırları yani yorumları, önemsememektedir. Zaten bir program derlendiğinde içinde yorumlar olmayacaktır. Yorumlar sadece kodcuların görebileceği “kod” içinde bulunacaktır.

Açıklama yazıları, tarafımızdan yazıldığı ve derleyici önemsemediği için bir yazım kuralı bulunmamaktadır. Ancak, açıklamaları bir düzende yazmaz isek, karışık, bizim işimizi kolaylaştırmak yerine zorlaştıracaktır. Bu nedenle, her ne kadar, açıklamaları istediğimiz gibi yazmakta özgür olsak bile, belirli kurallar çerçevesinde yapmak, bizim ve kodu okuyacak olan diğer meslektaşlarımız için daha kolay olacaktır.

Prosedürlerin başına getirilecek olan açıklama metni, prosedürün ismini, yaptığı işlevi, aldığı argümanları, döndürdüğü değeri anlatmalıdır.

```
/* Metod : FaizHesapla
 * Girilen değerlere göre faiz miktarını hesaplayıp
 * döndürür
 * A : Ana para
 * N : Faiz oranı
 * T : Faiz zamanı
 * Return : Faiz miktarı
 */
double FaizHesapla(double A, double N, double T){
    ... kodlar ...
}
```

Yukarıdaki açıklama metninde görüldüğü üzere, metodun ismi, açıklaması ve argümanların açıklaması yer almıştır. Bu şekilde bir açıklama kalıbı kullanmak, o metodu tanıtmak için uygulanacak yöntemlerden en iyisidir.

Bu açıklama metnine bir de, pseudo kod eklemenizi tavsiye ederim. Yani metodun bu belirtilen işlemi nasıl yaptığını anlatan bir pseudo kod ekleyerek, programcılarının, koda bakmadan içinde ne gibi ifadeler olduğunu tahmin edebilmesini sağlayabiliriz.

```
/* Metod : FaizHesapla
 * Girilen değerlere göre faiz miktarını hesaplayıp
 * döndürür
 * A : Ana para
 * N : Faiz oranı
 * T : Faiz zamanı
 * Return : Faiz miktarı
 * -- pseudo
 * return a*n*t/100
 */
```

Sınıfların başına getirilecek olan açıklama metni, sınıfın ismi ve açıklamasını içermelidir.

```
/* Sınıf : Rapor
 * Raporlandırma ile ilgili işlemleri gerçekleştiren
 * sınıftır
 */
```

Sınıflar için çok fazla ayrıntı vermeye gerek bulunmamaktadır. Çünkü, zaten, bir sınıfın, her bir ifadesini – elemanını tek tek açıklamaktayız.

Birçok programsal elemanda, aynı şekilde, açıklama kullanmalıyız. Yani kodlar, önce eleman “Türü”, daha sonra ismi, sonraki satırda açıklaması ve sonra da kodu şeklidende olmalıdır.

```
/* Olay : Silindi
```

* Dosya silindiğinde çağrılacak olan olaydır.

*/

Event Silindi()

Aynı şekilde, kalıp, interface, enum ... benzeri birçok elemanda da aynı şekilde bir açıklama kalıbı kullanmalıyız.

Bir sonraki konu, prosedürlerin iç kısımlarının yorumlanmasıdır. Prosedürlerin içindeki kodların yorumlanmasını karışık yaptığımızda, kod daha karmaşıklaşacaktır. Bu nedenle, bir satırdaki yani bir ifadedeki açıklamayı yaparken ayrı, bir bloktaki veya bölümdeki açıklamayı yaparken ayrı semboller kullanabiliriz.

Örneğin;

```
int Çevre = Çap * Pi
```

ifadesine, bir satırlık olduğu için, `/**/` veya `//` yerine `//-` sembolünü kullanabiliriz.

```
int Çevre = Çap * Pi  //- Çevreyi hesapla
```

Eğer bir dizi aynı – benzer veya ilişkili yada gruplaştırılmış kodun açıklamasını da `/**` sembolüyle yapabiliriz.

```
/** Alanı hesapla  
int alan=0;  
alan = alt_alan + üst_alan;  
alan += yan_alan * 4;
```

Dikkat ederseniz `//-` ifadesi, açıklaması yapılan satırın sağına yazılır. Ancak `/**` ifadesi, takip etmekte olan birkaç satırın (taa ki, boş satır bulununcaya dek) başına yazılır.

```
void sort(int a[]) throws Exception {  
    /** Her bir nesne için  
    for (int i = 1; i < a.length; i++) {  
        int j = i;  
        int B = a[i];  
        while ((j > 0) && (a[j-1] > B)) {  
            if (stopRequested){ //- eğer dur emri geldiyse  
                return;  
            }  
            a[j] = a[j-1];  
            j--;  
            pause(i,j);  
        }  
        a[j] = B;  
        pause(i,j);  
    }  
}
```

7.5.3 Syntax hataları

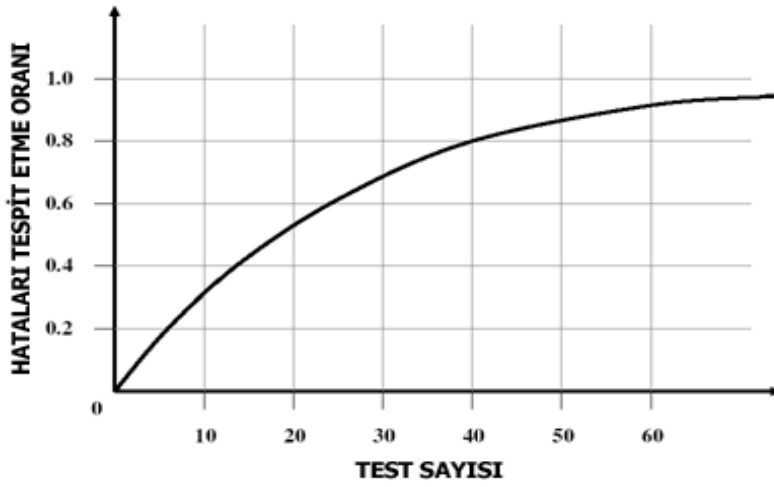
Daha önceki bölümlerde Syntax'ın ne olduğunu öğrenmiştik. Syntax yani yazım hataları, kodu yazmakta olduğunuz dilin, desteklemediği bir formatta ifade yazdığımızda oluşan hatalara denir. Yani eğer bir ifade yanlış yazılmışsa (metinsel olarak), bu bir syntax hatasıdır.

Syntax hatalarını düzeltmek çok kolay olacaktır, çünkü;

- Syntax hataları gözle görünür, rahat tespit edilebilir
- Syntax hatalarını derleyici zaten otomatik olarak göstermekte ve hatta bazı derleyiciler otomatik olarak düzeltmektedir.

7.5.4 Testler

Test, tasarlarken veya kodlarken önleyemediğimiz hataları engellemek için tek yöntemdir. Test sayısı arttıkça hata oranı – sayısı azalacaktır. Ne kadar çok ve birbirinden farklı test yaparsak, program içinde hata olma oranı o kadar azalacaktır. Bir hatanın saptanma oranı da aynı şekilde, test yapılarak artacaktır.



Figür 7.5.4 A : Hata tespit oranı

Yukarıdaki grafikte görülebileceği gib, test sayısı arttıkça, bir hatanın saptanma oranı, “azalır” şekilde artmaktadır.

7.6 Hataların yakalanması

Bir öğretmen, yazılı yaparken, dikkatini, en çok (potansiyel olarak) kopya çekebileceğini düşündüğü kişiler üzerine yoğunlaştırır. Ola ki, gözünden kaçarda bu öğrenciler kopya çeker diye... Hatta lisede, tarih dersinden kopya çekerken, arkadaşlarıyla birlikte, öğretmen bizi yakalayamasın diye, bizden şüphelenmesini engellemek için, ona güven vermiştik. Böylece, öğretmen, “bu öğrenciler zaten kopya çekmez” şeklinde düşünmüştü ve bizimle ilgilenmemiştir. Sonuç olarak, çok daha az zorlanarak kopya çekebilmiştik. Tabi burada önemli olan öğretmenin güvenini kazanmaktır. Bunun için, örneğin, başka bir derse ait notları, sıranızın altına koyabilirsiniz. Öğretmen bu kağıdı yakaladığında kopya zannedecek, ancak incelediğinde başka bir derse ait not olduğunu anlayacaktır. Hatta bu sırada, “hocam, bu başka bir derse ait not, benim kopya ile ne işim olur” gibi bir ifade kullanırsanız, öğretmen, sizden şüphelenmeyecektir. Hatta, sizi yanlış yere suçladığı için, kendi kendine (içinden), “yanlış yaptım, bu öğrenci kopya çekmez” diyecektir.

Böyle komik bir olaydan sonra, konumuza geri dönelim. Bir iş yaparken, hatanın oluşabileceği, veya hata yapabileceğimizi düşündüğümüz yerleri izleme altına alırız.

Tedbir. Bu sözcüğün hayatımızdaki önemi çok büyüktür. Peki ne gibi durumlarda tedbir alırız?

Bir futbol maçı esnasında, hakemin en önemli görevlerinden biri, saha içerisinde en uygun noktalarda bulunmaktır. En uygun noktalarda bulunmalı ki, birisi sakıncalı bir hareket yaparsa veya bir oyuncuya zarar verirse bu hareketi tespit edebilsin. Yani hakemin önem verdiği, HATA yapılabilir noktalardır. Eğer bir durumda, bir kod bloğunda, bir dizi veya bir satır kodda hata oluşabilecekse, bizde o bloğu kontrol altına almalıyız, göz önünde tutmalıyız.

İşte bir kod bloğunu veya bir dizi işlemi izlemek istiyorsak, buraya Try – Catch ifadelerini kullanırız. (Try : Dene, Catch : Yakala)

```
Metod metod1{  
    İşlem 1  
    İşlem 2  
    İşlem 3  
    İşlem 4  
    İşlem 5  
    İşlem 6  
}
```

Varsayalım ki, yukarıdaki gibi bir metodumuz ve bir dizi işlemimiz bulunuyor. Ancak, gerçekleştirilecek olan 4. İşlem’de, bir hata oluşabileceğini düşünüyoruz.

Try – Catch ifadeleri aşağıda gösterildiği şekilde kullanılır.

```
İşlem 1  
İşlem 2  
İşlem 3
```

```

Try{
    İşlem 4
}
Catch{
    HATA OLUŞTUĞUNDA YAPILACAK OLAN İŞLEMLERİN OLDUĞU
    BLOK
}
İşlem 5
İşlem 6

```

ÇEVİRME	<p>Hatırlatma_1'i hatırlayalım, amacımız, gerçek dünyada metinsel olarak ifade ettiğimiz bir işlemi bilgisayar içine programsal kod olarak aktarabilmek. Gerçek dünyada, “dikkat et”, “burasını dikkatli yap”, “aman dikkat” diye ifade ettiğimiz metinleri bilgisayar içinde, programlarken, Try-Catch ifadelerini kullanılırız.</p> <pre> Try{ Dosyayı yüklerken dikkat et! } Catch{ Dosya yükleme hatası oluştu } </pre>
----------------	--

Çeviri 10 : Try Catch

SONUÇ

Prof. Dr. Ahmet Mete IŞIKARA'nın söylediği gibi, “depremle yaşamayı öğrenmeliyiz”. Aynı şekilde eğer program yazmayı düşünüyorsak, hatalarla yaşamayı, onlarla mücadele etmeyi ve sorunları çözmeyi öğrenmeliyiz.

Bir sistemi test edebilmek için, en basit yöntem olarak, flow chart kullanmanızı tavsiye ederim. Binlerce satır kodun olduğu bir dosya içinde nerede hata olduğunu bulmanız çok zordur. Ancak, görsel olarak ifade edilebilen bir diagramda hatanın nerede olduğunu bulmak daha kolay olacaktır. Çünkü insan gözü, bir sistemi bütün olarak görürse daha iyi kavrar. Ancak siz bir sayfa koda bakarak sadece 50-60 satır belkide daha az satır kodu görebilirsiniz. Halbuki duvarınıza asılı olan bir diagramda, yaptığınız projelerin bütün ayrıntılarını görebilirsiniz.

İki türlü test yapmamız gerekmektedir: (1) Sistemi doğru mu tasarladık? (2) Tasarladığımız sistemi, belirtilen programlama dilinde düzgün mü ifade ettik (doğru mu yazdık?) İlk testi geçmeden asla ikinci kademeye gelmemeliyiz.

Yedinci bölümde kullanılan terimler

hata : error

anlaşma : agreement

yama : patch

çalışma zamanı : runtime

evre : phase

8 Sistemler

İçerik

- 6.1 Veri tabanları
- 6.2 Veri tabanı yönetim sistemi
- 6.3 Tablolar / Kolonlar / Kayıtlar
- 6.4 Entity – Relation yapısı

Amaçlar

Altıncı bölüm,

- v,
- İleri değişken işlemlerini, programsal anlatmaktadır.

Anahtar sözcükler

yeni, oluşturucu, özel, genel, sabit, dosya, yorum, doğru, yanlış

8.1 Mimari & Performans

Programlamada en önemli nokta, yapılacak olan işlemi, en iyi şekilde ifade edebilecek, en az hata içerebilecek, en hızlı çalışabilecek şekilde tasarlamaktır. Bu yüzden, bütün profesyoneller, programlarını tasarlamaya, öncelikle, basit ve amatör bir yöntem ile başlarlar. Yani bir projeyi tasarlamaya en önemli kısımlarından, EN BASİT şekilde başlarlar. Kurulan algoritmalar ve mimari, ne kadar amatörce ise, o kadar basit olacağı için, sistem bütün ayrıntılarıyla anlaşılmış olunur.

Daha sonra bu basit sistem, yavaş yavaş değiştirilerek, daha profesyonel, kompleks ve büyük sistemlere dönüşür. Bir sonraki adım, değiştirilen bölümlerin, diğer bölümlerle uygunlaştırılmasıdır. En son adım ise, yazılmış olan kodun, anlaşılır olmasından çok, performans bakımından daha iyi olmasına önem gösterilmesidir.

Bu konuda bir ilkeden bahsedelim: **SMILE**. Tabiki smile'in buradaki anlamı gülümsemek değildir. SMILE: Simple **m**akes **i**t **l**ots **e**asier (bir işi basitçe tasarlamak onu **çok** daha kolaylaştırır) anlamına gelmektedir. Yani bir işi tasarlarken basitten başlayıp, basit olarak tasarladıktan sonra birleştirdiğinizde, sistemi komple bir kerede tasarladığınız süreden çok çok çok daha az olacaktır anlamına gelir.

Daha açıkcası, bir programı öncelikle amatörce, basit ve sadece önemli kısımlarını tasarlamaya başlarız. Daha sonra, sistemleri geliştiririz ve daha profesyonel hale getiririz. En sonunda da, bu yazdığımız kodun bir kopyasını (yedeğini) alır, orjinalinde, program içi performansı arttırmak üzere gerekli değişiklikleri yaparız. Bu işlemi yapmadan önce kopyasını almamız gerekmektedir çünkü, performans arttırmak için yapacak olduğunuz değişiklikler, kodu karıştıracak ve kodun anlaşılmasını zorlaştıracaktır. Bir programın kodunun anlaşılması ne kadar zor ise, o program, performans bakımından o kadar yüksektir. Örneğin, tane tane yazılan ve kolay anlaşılması için çok sayıda prosedürlere ayrılan bir kodun çalışma zamanı, tek prosedürde çalışan ancak karmaşık görünen bir koda göre daha uzun sürecektir (daha yavaş çalışacaktır).

Bir web sayfasında, Visual basic programlama dili için, hızlandırıcı işlemlerin ve ipuçlarının olduğu bir döküman bulmuştum. Bu dökümanda, aynı prosedür içinde ifade edilen bir işlemin, başka bir prosedür içinden çağrılmasından %75 daha kısa süreceği belirtiliyordu.

Örnek olarak;

```
method a(){
    İşlem 1
    İşlem 2
    İşlem 3
    b();
}

method b(){
    İşlem 4
    İşlem 5
}
```


.. şeklinde, iki metod kullanmak yerine,

```
Method a(){  
    İşlem 1  
    İşlem 2  
    İşlem 3  
    İşlem 4  
    İşlem 5  
}
```

bu iki metod içindeki işlemler tek bir metod içine yazılabilir. Tabi bu işlemi her durumda uygulamayazsınız. Çünkü bazen bir metod iki veya daha fazla metod tarafından kullanılmakta olabilir. Böyle bir durumda, eğer ikinci metodu kaldırırsanız, “b” metodunu kullanan diğer metodlarda hata oluşacaktır.

Programlama için performans çok önemlidir. Ancak, performans sadece ve sadece geliştirme evresi sonrasında düşünülmelidir. Eğer programı tasarlarken, bir sistemin, daha yüksek performansta olması için tasarlıyorsanız, o sistemin anlaşılması çok zor olacaktır. Bunun yerine bütün sistem tasarlandıktan sonra, performansı arttırmak için ek bir çalışma yapmalısınız. Unutmayın ki, kullanıcılar için önemli olan programın ne kadar profesyonel tasarlandığı değil, çalışma hızıdır. Kullanıcılar için sadece ön yüzüyle ilgilendiği için, kullanıcılara, hızlı çalışabilen bir program sunmalıyız.

Performans arttırmak için bilmeniz gereken iki önemli nokta bulunmaktadır. Bunlardan birincisi, dili çok iyi tanımak, ikincisi, dillerin ortak yapısını bilmektir.

Dili iyi tanıyarak; bir işlemin daha kısa sürede nasıl yapılacağını, veya daha az kod kullanarak yapılacağını tespit edebilirsiniz. Bunu bir konuşma dili bilmeye benzetebiliriz. Eğer (örnek) Fransızca’yı çok iyi biliyorsak, derdimizi daha iyi anlatabiliriz kendimizi daha iyi ifade edebiliriz.

“Dillerin ortak yapısı” ile ifade edilen, şimdiye kadar anlatılmış olan konulardır. Bu konularda yeterli başarıyı gösterdiğiniz takdirde, performansı yüksek program yazmış olacaksınız.

Tasarlamadan (planlamadan) yapılan geliştirme, sadece “zaman kaybı”dır. Bazıları bu zaman kaybınında önemli olduğunu söylerler. İşte ileriye göremeyenler bu süreyi, DENEYİM diye adlandırırlar. Bu evre, deneyim değil boşa harcanan bir süredir. Deneyim, daha fazla proje tasarımı yapmak olabilir. Bir projeyi yanlış tasarlamak deneyim olabilir. Ancak bir projeyi tasarlamadan başlamak ve bir noktada tıkanmak, veya herşeyi karıştırmak DENEYİM değildir. Bu süre sadece, “bir musibet bin nasihattan iyidir” sözündeki “musibete” tekamül etmektedir. Sonuç olarak, yanlışta doğrudan tasarlasak bir sistemi tasarlamamız gerekmektedir. Tasarım üzerinde değişikliklerin yapılması çok kısa zaman alacaktır ve bir maliyeti olmayacaktır. Ancak, belirli bir noktaya kadar inşa ettiğiniz bir binayı düşünün. Bu noktadan sonra, binanın bir kısmını yanlış tasarladığınızı farkettiğinizi ve bir bölümünü yıkıp tekrar yaptığınızı varsayalım. Plana harcanak 10 saat mi, yoksa hatalardan kayıp edilecek 10 hafta ve milyarlar mı daha önemlidir.

Birçoğumuz plan ve tasarımı çok fazla önemsemez. Ancak dünyanın en düzgün işlerini teslim eden ve sanayi devi olan japonlara baktığımızda, bu başarılarını plan yaparak kazandıklarını görebiliriz.

8.2 Tasarım

Programlamanın en kötü yanı, arka tarafta yazdığınız sınıfların, metodların, kurmuş olduğunuz algoritmaların, mantıkların, tasarladığınız mimarilerin, kullanıcı tarafından anlaşılamamasıdır veya bilinmemesidir. Kullanıcı sadece ve sadece, kullanıcı arayüzü ile muhatap olduğu için, sizin arkada (program içinde) ne gibi zorluklar çekerek o projeyi tasarladığınızı anlayamaz. Kullanıcı için tek anlaşılır nokta, gördüğü tasarımıdır. Bu yüzden kod yazmada profesyonel olduğunuz kadar, tasarımda da profesyonel olmanız gerekmektedir.

Tasarılma için gerçek hayattan; bir kitabın, bir gazetenin tasarlanması, bir anket formunun, bir web sitesinin tasarlanması v.b. gibi örnekler verebiliriz.

Tasarım, sadece görünendir. Bir tabloda (ünlü bir ressamın yaptığı bir resim tablosu) önemli olan renklerin ne kadar çok kullanıldığı, hangi renkler karıştırılarak elde edildiği, hangi tip fırçaların kullanıldığı değil, bütün resmin birbiriyle olan uyumu önemlidir. Zaten tabloya bakan, o resmin tasarlanırken ne gibi zahmetlere katlanıldığını anlayamaz. Ancak önemli olan bu zahmetler veya çıkan görüntüdeki renkler – boyalar değildir. Önemli olan ressamın, ne anlam ifade etmek istediğidir. Yani resmin yapılmasından daha önemli olan, o resmin ne anlattığının ressam tarafından belirlenip karar verilmesidir.

Programlamaya girişte tasarımın önemi çok büyük değildir. Herkes “önce kodlamayı (zor olanı) yapalım da, gerisini (tasarımı) çözeriz” düşüncesine sahiptir. Ancak tasarımı inanılmaz derecede anlaşılmaz ve dağınık olan programların kullanımı da zor olacaktır. Kodlaması ne kadar mükemmel olursa olsun, tasarımı yeterince kaliteli olmayan bir program kullanıcıya itici gelecektir. Bunu bir derginin içeriği ve kapağı olarak düşünebiliriz. Ambalaj veya kapak, insanların ilk baktığı noktadır. Tabiki sadece kapak yeterli değildir ancak mükemmel bir içeriğe sahip olsada kapağı hoş olmayan bir dergi çok satmayacaktır.

Bu nedenle, tasarım büyük önem taşımaktadır. Dikkat edersek, artık hızla gelişen teknoloji, sistemler ve mimariler sayesinde, programlara birçok ek özellik eklenmektedir. Bu eklenen özelliklerin anlaşılması için, profesyonel ve mantıklı bir şekilde tasarlanmalıdır ki, kullanıcı bir daha programı öğrenmek veya anlamak için ekstra vakit kaybetmesin.

8.3 İsimlendirme

Programlamada en çok sevdiğim, değişkenlerin ve sınıfların (hatta özelliklerin - metodların - olayların), tamamen kendimizin belirlediği şekilde isimlendirilebilmesidir. Kurallara uyduğu sürece, bu elemanlara istediğimiz isimleri verebiliriz. Tabi ki, eğer konu ile alakasız - ilgisiz isimler kullandığımız takdirde, daha sonra kodu tekrar açıp, incelediğimizde, hangi ismin ne anlama geldiğini anlamakta zorlanırız. Üniversitedeki hocalarımdan biri, yıllar önce yazdığı bir programda, **Tom, Jerry, Bugs, Bunny** gibi çizgi film kahramanlarının isimlerini değişken isimleri olarak kullanmış. Tabiki, değişken - sınıf veya herhangi bir ismin, bu gibi isimler olmasında hiçbir sakınca yoktur. Aynı şekilde hocamında dediği gibi program çalışmaktaydı. Ancak, 2-3 yıl sonra, programı tekrar açıp incelediğinde, hangi değişkenin - hangi sınıfın ne amaçla kullanıldığını, hatırlamakta zorlanmış.

İşte bizde eğer yazıyor olduğumuz program ile alakalı isimler kullanırsak, daha sonra hatırlamakta veya anlamakta zorlanmayız. Ayrıca başkalarının, yazmış olduğumuz kodu, görebileceğini ve inceleyebileceğini düşünmeliyiz.

8.3.1 Genel Kurallar

- İsimler, 255 karakterden uzun olmamalıdır. Genelde, çok uzun olmayan ifadeler kullanılmalıdır
- İsimler arasında boşluklar bulunamaz. Boşluk kullanmak yerine iki ifadeyi bitişik yazıp, her ifadenin ilk harfini büyük yazınız. Örneğin, “Hesap yap” metodunun ismini HesapYap şeklinde kullanabilirsiniz.
- İsimler, harflerle veya “_” alt çizgi ile başlamalıdır (bazı diller sadece harf ile başlamayı desteklemektedir).
- İsimler, harflerden, sayılardan ve alt çizgilerden meydana gelmektedir.
- Dil tarafından kullanılan anahtar sözcüklerin isimleri, isim olarak kullanılmamalıdır (bazı dillerde, anahtar sözcükler [] işaretleri arasına alınarak isim olarak kullanılabilir).

Bunların haricinde;

- Türkçe karakterler kullanılmaması önerilir. Çünkü bazı dillerde bu karakterler desteklenmemektedir
- *İsimlendirmede, Türkçe karakter kullananlar ve kullanmayanlarda, farklılık olacağı için (Oğrenci ve Öğrenci gibi) bütün isimleri ve kodu İngilizce yazmanızı tavsiye ederim. Tabiki böylece, kodunu uluslar arası anlaşılabilir.*
- Sınıf, metod, özellik, olay isimleri, aynen o nesnenin ismi olarak seçilmelidir. Ancak sınıf değişkenlerinin başına ek ifadeler getirilmelidir.
- İsimler, daima büyük harfle başlamalı ve sonraki harfler küçük olmalıdır. Sadece, iki kelimenin birleşmesi durumunda, kelimelerin ilk harfleri büyük olmalıdır.

8.3.2 Sınıfların İsimlendirilmesi

Sınıfların isimlendirilmesinde, temsil ettiği nesnenin ismini seçmeniz en uygun olanıdır. Yani, programlama ortamına aktaracak olduğunuz nesnenin ismi, yazıyor olduğunuz

sınıfın ismi olarak seçilmelidir. Dilbilgisinde “isim” olarak ifade edilen sözcükleri, sınıf ismi olarak seçebiliriz.

Class : Otomobil
Class : Siparis
Class : Urun
Class : Ogrenci

Class : Car
Class : Order
Class : Product
Class : Student

Aynı durum, Kalıp(Structure)’lar içinde geçerlidir.

8.3.3 Metodların İsimlendirilmesi

Metodlar, genelde;

- bir fiil
- bir nesne ve bir fiil

şeklinde ifade edilmelidir. Yani, yapılan işin ismini, veya yapılan işin ismi ile ayrıntısını belirtmeliyiz.

Hesapla (fiil)
Sipariş Ver (nesne, iş + fiil)

Calculate (fiil)
MakeOrder (fiil + nesne : ingilizce de, emir cümlesi dizilişi, fiil + nesne şeklindedir.)

Bu şekilde, hangi işlemin yapıldığı daha iyi anlaşılacaktır. Burada dikkat edilmesi gereken bir nokta bulunmaktadır. İsimlendireceğiniz metod eğer tek bir ifade ise, o ifadenin KESİNLİKLE bir fiil olmasına dikkat ediniz. Çünkü özellik isimlerinde benzer bir kelime kullanılmış olabilir.

Daha açıkcası, yukarıdaki Hesapla ifadesine, -mek, -mak ekleri getirdiğimizde (hesaplamak) bir eylem belirtiyorsa uygun isimlendirme yapmışız demektir. İngilizcede ise, sonuna değil başına “to” mastarını getirdiğimizde eğer bir fiil oluyorsa o zaman uygun isimlendirme yapmışız demektir.

8.3.4 Özelliklerin İsimlendirilmesi

Özelliklerin isimlendirilmesi, aynı, sınıfların isimlendirilmesi gibi, genelde tek bir ifade halinde, özelliğin ismi şeklinde yapılır. Yukarıdaki bölümde, metodların isimlendirilmesinde, tek ifadelere, -mek, -mak ekleri getiriyorduk ve anlamlı bir kelime çıkmasını istiyorduk. Özelliklerde ise tam tersi yapılır. Yani, özellik ismi sonuna, -mek –mak eklerini getirdiğimizde anlamlı bir ifade olmaması gerekmektedir Kİ, metod isimleriyle, özellik isimleri karışmasınlar. Örneğin; Hesapla metodunun olduğu bir sınıf Hesapla özelliği ekleyemeyiz. Onun yerine sonuna –mek, -mak getirdiğimizde, anlamlı bir ifade olmayan

Hesap kelimesini kullanmalıyız (hesapmak X). İngilizcesinde ise, (to) Calculate değil, bu fiilin isim hali olan Calculation ifadesini kullanabiliriz.

Özelliklerin isimlendirilmesinde, dilbilgisindeki, isimler veya sıfatlar kullanılır.

8.3.5 Olayların İsimlendirilmesi

Olayların isimlendirilmesi, dah önceden gördüğümüz isimlendirmelerden farklıdır. Bir olay, o olayın, gerçekleşmesinin akabinde, veya olay gerçekleşiyor iken, çağrıldığı için, isimlendirilmesinde de, o fiilin, geçmişte yapıldığını veya şuanda yapılıyor olduğunu belirten bir ifade kullanılması doğru olur. Örneğin, Sil (delete) adında bir metodumuz bulunuyor ve bu metod çağrıldıktan sonra, bir dosya siliniyor ve bir olay gerçekleşiyor. Bu olayı, Silindi (deleted) şeklinde adlandırmamız en doğrusu olacaktır.

Aynı şekilde, Kopyala (Copy) adında bir metodumuz bulunduğunu ve bu metod çağrıldığında bir dosyanın kopyalanmaya başlandığını ve bu esnada, bir olayın çağrıldığını düşünelim. Bu olayı “Kopyalanıyor (Copying)” olarak adlandırmak en doğrusu olacaktır.

8.3.6 Sınıf içindeki değişkenlerin İsimlendirilmesi

Sınıf içinde kullanılan değişkenler için, özellik isimlendirme kuralları uygulanır ancak, sınıf içinde kullanılan değişkenlerin isimlerinin başına “_” alt çizgi getirilir. Bazı dillerde “_” alt çizginin ilk karakter olarak desteklenmediğini belirtmiştim. Böyle durumlarda, bütün ismi, küçük harflerle yazabilirsiniz.

_Numara
_Number

numara
number

8.3.7 Sabit değerlerin İsimlendirilmesi

Sabit değerlerin (constant) isimlerini, tamamen büyük yazabilirsiniz. Bu durum, bir değişkenin sabit mi, değişken mi olduğunu anlatacaktır.

PI Pi sayısı
YERÇEKİMİ
GRAVITY

8.3.8 Enumların İsimlendirilmesi

Enumlar genelde bir tür – tip belirttiği için, verilen standart (özellik ismi kurallarına göre) isimden sonra, Tür – Tip veya ingilizcede Type eki kullanılabilir.

BiletTipi

TicketType

8.3.9 Aduzaylarının İsimlendirilmesi

Aynı işlevi yapan, sınıfları, enumlari, kalıpları... bir Aduzayına atayabilirsiniz. Aduzayı bir tür gruptur. Sınıfları ve benzer elemanları gruplandırmak amacıyla kullanılır.

Bir aduzayını yani bir grubu adlandırırken, metod ismi kurallarını kullanmalıyız. O gruptaki elemanların yaptığı ortak işin ismini grup ismi olarak seçmeliyiz.

Örneğin; raporlar işlemler yapan, rapor oluşturan ve görüntüleyen sınıflarımız olduğunu ve bu sınıfları ortak bir aduzayına attığımızı düşünelim. Bu aduzayını isimlendirirken, “Raporlandırma” ifadesi en uygunu olacaktır.

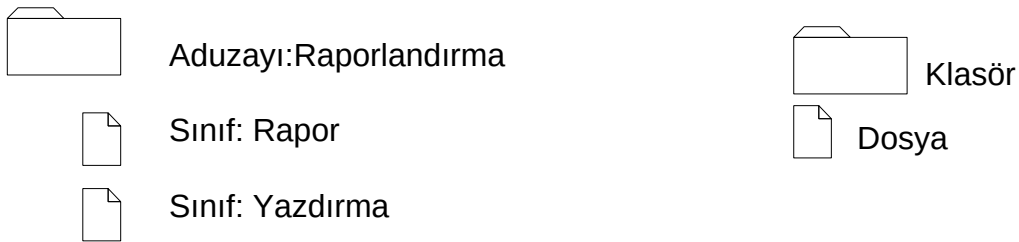
Aynı işlem için aşağıda birkaç örnek verilmiştir.

Ad uzayı	Raporlandırma
Sınıf	Rapor
Metod	Raporlandır

Klasörler ve dosyalar

Yazdığımız sınıfları, enumlari.... dosyalar içine kayıt ederiz. Dosyaları da klasörler içine yerleştiririz. Daha sonra bir kodu görüntülememiz gerektiğinde, kodun bulunduğu dosyanın bulunduğu klasöre gireriz, kodun bulunduğu dosyayı açarız ve görüntüleriz. Burada dikkat edilmesi gereken, hangi kodun hangi dosya içinde olduğunu, ve o dosyanın da hangi klasör içinde olduğunu nasıl hatırlayacağımızdır.

Bunun için, en kolay yöntem, dosya isimleriyle, sınıf – kalıp – interface isimlerini aynı, klasör isimleriyle de, aduzaylarının isimlerini aynı yapmaktır.



Figür 8.3 A : Dosya ve klasörler

SONUÇ

Şimdiye kadar öğrendiklerimizin birleşimi, neredeyse bütün programlama dünyasında gerekli olacakların temelini önümüze sunacaktır. Bu sayede, bir dilin öğrenilmesi, programın yazılması, sistemin kurulması bizim için çok kolay hale gelecektir. Şuan itibarıyla, bir program yazmamız olasıdır. Öğrendiklerimizi kullanmanın ve otomatik pilota bağlamanın zamanı gelmiştir. Bu kitaptaki bilgileri öğrendikten sonra yapabilecekleriniz hayal gücünüzle sınırlıdır. Yani yapabildiklerinizi limitlemek ve bu limitleri aşmak sizin elinizdedir.

Tecrübe ile ilgili daha önce söylediklerimi hatırlarsanız; özet olarak diyebiliriz ki; **tecrübe bir süreç içinde öğrenilen ve işe yarar farklı bilgiler topluluğudur.** Çok daha fazla farklı ve işe yarar bilgiyi, size anlatan bir kaynak sahibi olmak isterseniz, bu serinin ikinci kitabını tavsiye ederim. Ancak direkt olarak ikinci kitaptan başlamamanız gerekir. Bu kitabı tamamen bitirdikten sonra ikinci kitabı da okursanız, çok daha hızlı bir şekilde, daha fazla bilgi sahibi olursunuz ve profesyonelliğiniz daha kısa sürede artar.

İkinci kitap konuları içinde;

<ul style="list-style-type: none">• UML• Veritabanları dosya yapısı• Yazılım mühendisliği• OODB• Web / Mobil uygulamaları• Sistemler ve teknolojiler• İleri planlama	<ul style="list-style-type: none">• Bilgisayar Bilimleri• İleri OOP• İnternet Programlama• İleri Veritabanı işlemleri• Bilgisayar hukuku & Haklar• Güvenlik• İleri programlama dilleri
--	--

bulunmaktadır.

Günlük hayatta, ulaşım esnasında veya boş geçen bütün vakitlerinizde kullanılmakta olan programların nasıl tasarlanabileceğini düşünmenizi, “ben olsam bu programı nasıl yapardım” sorunu gördüğünüz her program için sormanızı ve bu soruya cevap bulmaya çalışmanızı tavsiye ederim.

Kitabın ilk bölümünde; belirttiğim yanlışlıkları ve farklılıkları (programlamanın ve algoritmanın öğretilmesi ile alakalı) kitabı okumayı tamamladıktan sonra daha iyi anlayabilirsiniz. Bireysel olarak ricam; bu yanlışlıkları ve farklılıkları unutmayıp bu bilgiler ışığında programlama yapmanızdır.

Bütün programcı adaylarına – artık uzmanlarına; yazılım hayatlarında başarılar dilerim. Kendinizi geliştirmeyi ve ilerletmeyi sakın ihmal etmeyin, sakın hiç bir seviye ile yetinmeyin... **Seviyeler sizi değil, siz seviyeleri belirleyin..**

Saygılarımla
Emre YAZICI

Ekler

Ek 1 - Pseudo kod standartları:

Pseudo kodları tamamen kendimizin belirleyebileceğinden bahsetmiştim. Ancak eğer bir ekiple çalışıyorsak veya yaptığımız çalışmayı daha sonra birisine göstereceksek, böyle bir durumda, ISO standartları gibi standartlar kullanmamız daha mantıklı olacaktır. Çünkü, eğer bir proje, projenin anahtarı(hangi ifadenin ne anlama geldiğini belirten bir tür anahtar) olmazsa, o projeyi sadece siz çözebilirsiniz demektir ve diğer kişilerin işine yaramayacaktır.

1. GENEL İFADELER

Girdi: READ(oku), OBTAIN(elde et), GET (al)

Programa, dışarı bir kaynaktan veya başka bir programdan bir girdi olacağını belirtmek için yukarıdaki ifadelerden herhangi birini kullanabilirsiniz. READ, genelde kullanıcının yazdığı bir metni okumak veya kullanıcıdan direkt olarak bir girdi almak için kullanılır. OBTAIN ve GET, herhangi bir şekilde, bir başka kaynaktaki veriyi elde etmek için kullanılır.

Örneğin;

READ kullanıcı ismi	//kullanıcı ismini oku
READ işlem kodu	//işlem kodunu oku
GET hafıza boyutu	//hafıza boyutunu bul (al getir)
GET yıl	//bulduğumuz yılı bul (al getir)

Çıktı: PRINT (yazdır), DISPLAY (göüntüle) , SHOW (göster)

Programdan, kullanıcıya veya programlamacıya, bir tür çıktı göstermek veya mesaj vermek için yukarıdaki ifadelerden biri kullanılır. PRINT genelde, kullanıcı tarafından görülmeyecek, programlamacıya, program ile ilgili (test aşamasında) bilgi verecek durumlar için kullanılır. DISPLAY ve SHOW, kullanıcıya bir mesaj göstermek veya bir işlem – hesaplama sonucunu göstermek için kullanılır.

Örneğin;

PRINT kontrol işlemi yapılıyor	//programlamacıya, kontrol işleminin yapıldığını göster
PRINT hata	//hata mesajı göster
DISPLAY hoşgeldiniz	//kullanıcıya hoşgeldiniz mesaj göster
DISPLAY	

Ayarla – Tanımla: SET (ayarla - ata), INIT (tanımla)

Programda bir değişken tanımlamak (ilk olarak değişkeni oluşturmak) için, INIT ifadesi kullanılmaktadır. Tanımlanmış bir değişkene, değer atamak için ise SET ifadesi kullanılır.

Örneğin;

INIT kullanıcıismi	//kullanıcıismi adında bir değişken tanımla
INIT firmakodu	//firma kodu adında bir değişken tanımla

INIT firmakodu AS INTEGER	//integer (tamsayı) formatında, firmakodu adında bir değişken tanımla
SET kullanıcıismi="Ahmet"	//kullanıcıismi değişkeninin değerini Ahmet olarak atama yap

Çağır – Çalıştır: CALL (çağır)

Bir prosedürü, veya başka bir fonksiyonu çağırmak, o fonksiyon veya prosedür içindeki işlemleri çalıştırmak için, CALL ifadesi kullanılır.

Örneğin;

CALL dosyayısıl	//dosyayısıl adındaki prosedürü çağır - çalıştır
CALL veritabanınıaç	//veritabanını aç adındaki prosedürü çağır

2. KOŞUL İFADELERİ

Tekil Koşul: IF

Koşul ifadelerini daha önce görmüştük, eğer bir duruma bağlı işlem yapılıyorsa, koşul ifadeleri kullanıyoruz

Örneğin;

```
IF koşul THEN
    //eğer koşul doğru ise bu bloktaki işlemler yapılacak
ELSE
    //eğer koşul doğru değilse bu bloktaki işlemler yapılacak
ENDIF
```

```
IF koşul THEN
    //eğer koşul doğru ise bu bloktaki işlemler yapılacak
ENDIF
```

Çoğul Koşul: CASE (durum)

Bir dizi koşulu ifade etmek için CASE ifadesi kullanılır

CASE değer OF

3 : Yapılacak işlem (yukarıda belirtilen değer 3 ise, bu bloktaki işlemi yap)
5 : Yapılacak işlem (yukarıda belirtilen değer 5 ise, bu bloktaki işlemi yap)

ENDCASE

3. DÖNGÜ İFADELERİ

Olduğu Sürece: WHILE

WHILE koşul

//koşul doğru olduğu sürece yapılacak olan işlemler

ENDWHILE

X'den, Y'ye kadar: FOR

FOR değişken = başlangıç_değeri TO bitiş_değeri
//işlemler

ENDFOR

Yukarıdaki ifade daha önce gördüğümüz basit döngüleri tanımlamak için kullanılır.

Olana dek: REPEAT (tekrar et)

REPEAT UNTIL koşul //koşul doğru olana dek devam et
//işlemler

ENDREPEAT

Bu ifadeler haricinde, bilmeniz gereken, kalıpsal – bloksal ifadelerde, bir başlangıç ve bir bitiş olması gerektiği ve genelde bu anahatar sözcüklerin, büyük harflerle ifade edildiğidir.

Ek 2 - Backus Naur Form (BNF)

BNF aslında direkt olarak programlama ile alakalı değildir. Ancak bu kitabın en önemli özelliği olan birden fazla dilin aynı kitapta öğrenilebilmesi için gerekli olan bir syntax tanımlama formatıdır. Yani, bir dilde yazılacak olan program, hangi yapıda olmalıdır, nasıl dizilmelidir, önce hangi ifade kullanılmalıdır... Nasıl bir cümlede, önce, özne ile başlayıp sonra tümleçle devam edip, fiil ile bitiriyorsak, aynı şekilde de programlama dillerinde bir formatı bulunmaktadır.

BNF aslında programlama ile birlikte birçok konuda karşınıza çıkacaktır. O yüzden bilmeniz de fayda bulunmaktadır. Örneğin İngilizce öğrenmede... Dört yıl boyunca İngiltere’de kaldım. Ancak İngilizceyi kolay öğrenebilmeyi, İngilizceyi öğrendikten sonra keşfettim. Aslında üstünde çok durmadım ama birden karşıma, İngilizceyi çok kolay öğrenmek için bir yol çıktı. Bildiğiniz gibi bir dili öğrenmek demek, o dilin **grammerini** öğrenmek ve kelime **ezberlemek** üzere ikiye ayrılır. Grammarı öğrendikten sonrası kelime ezberlemedir. Grammarı kolay öğrenebilmek ve algoritmasını anlayabilmek için BNF bilmek gerekmektedir (benim düşüncem).

BNF, bir kalıbın ne gibi bileşenlerden oluştuğunu ifade etmek amacıyla kullanılır. BNF’nin belirli standartları yoktur. Sadece BNF ile belirteceğiniz bir veriyi görüntülemeyi önce kendiniz bir standart belirler ve bu standardı tanımlarsınız.

Örneğin, Türkçe cümleler, sırayla: özne, tümleç, yüklemden meydana gelir ifadesini BNF ile tanımlamak için;

<cümle> ::= <özne> <tümleç> <yüklem>
(ilk olarak özne gelir, sonrasında tümleç, sonrasında yüklem)

tanımı kullanılabilir. Yukarıdaki “::=” işareti, soldaki tanımın açıklandığı anlamına gelmektedir. “<” ve “>” işaretleri arasında kalan bölüm ise, tanımlanmakta olan terimin ismidir. Bunun haricinde, yine özneyi de alt tanım olarak ifade edebilirsiniz. Yine hatırladığımız üzere, “özne” ya bir zamirdir (ben, sen, o,) yada isim tamlamasıdır (Ahmet’in kedisi, Mustafa’nın arkadaşı....).

<özne> ::= <zamir> | <isim tamlaması>

Yukarıdaki ifade de kullanılan “|” işareti, “veya” anlamına gelmektedir. İşte programlama dillerinin syntax’larında aynen bu şekilde tanımlanır.

BNF Kuralları:

1. Yorum - açıklama yazısı için, ! işareti kullanılır
2. Bir metni, direkt olarak ifade etmek için, tek veya çift tırnak kullanılır.
3. Bir terimi ifade etmek için, terimi, “küçüktür” ve “büyüktür” işaretleri arasında yazarız.
4. Bir terimin tanımını yazmak için “::=” işaretleri kullanılır

Referanslar

- [e1] Stephen Hawking, <http://www.farmakim.com.tr/bilimsel5-ed-sh.html> [01.03.2008]
- [e2] Algorithm, <http://www.einfochips.com/download/glossary.pdf> [01.03.2008]
- [e3] Algoritma Tarihi, <http://ansiklopedi.turkcebilgi.com/Algoritma> [01.03.2008]
- [e4] Program, <http://wordnet.princeton.edu/perl/webwn?s=program> [01.03.2008]
- [e5] Software, <http://en.wikipedia.org/wiki/Software> [01.03.2008]
- [e6] Plan, <http://www.seslisozluk.com/?word=plan> [01.03.2008]
- [e7] Programming, <http://www.maccaws.org/kit/glossary/> [01.03.2008]
- [e8] Hesap Makinesi, http://tr.wikipedia.org/wiki/Hesap_makinesi [01.03.2008]
- [e9] Sistem, <http://tr.wikipedia.org/wiki/Sistem> [01.03.2008]
- [e10] Tümevarım, <http://tr.wikipedia.org/wiki/Tümevarım> [01.03.2008]
Tümdengelim, <http://tr.wikipedia.org/wiki/Tümdengelim> [01.03.2008]
- [e11] Analyses, www.ohlone.edu/org/capac/docs/blooms-tax2.html [01.03.2008]
- [e12] Anlama, www.tdk.gov.tr [01.03.2008]
- [e13] Sıkıştırma algoritması örneği, <http://www.gzip.org/algorithm.txt> [01.03.2008]
- [e14] Charles Babbage, http://tr.wikipedia.org/wiki/Charles_Babbage [01.03.2008]
- [e15] Expression / Statement, www.seslisozluk.com [01.03.2008]
- [e16] http://www.arastirma.org/MOC/index.php?secim=makale&m_id=1538 [01.03.2008]
- [e17] <http://tr.wikipedia.org/wiki/Cpu> [01.03.2008]
- [e18] <http://tr.wikipedia.org/wiki/Mikroişlemci> [01.03.2008]
- [e19] <http://www.tiobe.com/tpci.htm> [01.03.2008]
- [e20] <http://www.developer.com/lang/article.php/3390001> [01.03.2008]
- [e21] <http://www.cs.berkeley.edu/~flab/languages.html> [01.03.2008]
- [e22] http://tr.wikipedia.org/wiki/Veri_tabanı
- [e23] <http://tdk.gov.tr>
- [e24] <http://tr.wikipedia.org/wiki/Varlık>
- [e25] <http://www.cs.usyd.edu.au/~comp4302/ann1-6s.pdf>
- [e26] <http://tr.wikipedia.org/wiki/Otomobil>
- [e27] <http://tr.wikipedia.org/wiki/Varlık>, Felsefe Terimleri Sözlüğü, Bedia Akarsa, İnkılap Yayınevi
- [e28] <http://ogrenci.hacettepe.edu.tr/~b0145575/baglantilar/zeka.html>

Yararlanılan kaynaklar

Dave Mount, Algorithms, Spring 1998, Lecture Notes, FreeTechBook (2)
Software Engineering, 6th Edition, Ian SOMMERVILLE
A short introduction to operating systems, Mark Burgess, October 3, 2001
University of Essex, Introduction to programming Slide notes
Prof Dr. Haluk ERKUT, (Analiz, tasarım ve uygulamalı) Sistem Yönetimi, 2.Baskı 2000
Architecture Principles of Operation. IBM, Chapter 17. 2007-04-14.
http://www.bim.gazi.edu.tr/bilgisayar_birimleri.doc
Güçlü Hafıza, Ahmet Yıldız, 2007