2020

RAMAZAN KÖSE

C DİLİ PROGRAMLAMA



İçindekiler

[1.BÖLÜM PROGRAMLAMAYA GİRİŞ 5](#_Toc39411843)

[1.1 PROBLEM ÇÖZME 5](#_Toc39411844)

[1.2 ALGORİTMA 5](#_Toc39411845)

[1.3 ALGORİTMADA OLMASI GEREKEN HUSUSLAR 5](#_Toc39411846)

[ÖRNEK 1.3.1 6](#_Toc39411847)

[ÖRNEK 1.3.2 6](#_Toc39411848)

[ÖRNEK 1.3.3 7](#_Toc39411849)

[2.BÖLÜM C PROGRAMLAMA DİLİNİN GENEL YAPISI 8](#_Toc39411850)

[2.1 TEMEL MADDELER 8](#_Toc39411851)

[1.2 VERİ TÜRLERİ 8](#_Toc39411852)

[2.3 DEĞİŞKENLER 9](#_Toc39411853)

[2.4 OPERATÖRLER 10](#_Toc39411854)

[1. ARİTMETİK OPERATÖRLER 10](#_Toc39411855)

[2. ATAMA PRERATÖRLERİ 10](#_Toc39411856)

[3. KARŞILAŞTIRMA OPERATÖRLERİ 11](#_Toc39411857)

[3.Bölüm PROGRAM DENETİM VE DÖNGÜ YAPISI 11](#_Toc39411858)

[3.1 İF YAPISI 12](#_Toc39411859)

[3.2 İF ELSE YAPISI 12](#_Toc39411860)

[3.3 İF – ELSE İF – ELSE 13](#_Toc39411861)

[ÖRNEK 3.2.1 13](#_Toc39411862)

[3.4 SWİTCH – CASE 14](#_Toc39411863)

[ÖRNEK 3.4.1 14](#_Toc39411864)

[ÖRNEK 3.4.2 16](#_Toc39411865)

[3.5 WHİLE DÖNGÜSÜ 17](#_Toc39411866)

[ÖRNEK 3.5.1 18](#_Toc39411867)

[3.6 DO – WHİLE 18](#_Toc39411868)

[ÖRNEK 3.6.1 18](#_Toc39411869)

[3.7 FOR DÖNGÜSÜ 19](#_Toc39411870)

[ÖRNEK 3.7.1 19](#_Toc39411871)

[4.BÖLÜM DİZİLER 20](#_Toc39411872)

[ÖRNEK 20](#_Toc39411873)

[4.1 İKİ BOYUTLU DİZİLER 21](#_Toc39411874)

[ÖRNEK 4.1.1 22](#_Toc39411875)

[22](#_Toc39411876)

[ÖRNEK 4.1.2 22](#_Toc39411877)

[5.BÖLÜM FONKSİYONLAR 23](#_Toc39411878)

[5.1 FONKSİYONLARIN TANIMLANMASI 24](#_Toc39411879)

[5.2 FONKSİYONLARIN ÇAĞRILMASI 24](#_Toc39411880)

[ÖRNEK 5.2.1 25](#_Toc39411881)

[5.3 FONKSİYONLARA PAREMETRE AKTARMA 25](#_Toc39411882)

[ÖRNEK 5.3.1 25](#_Toc39411883)

[5.4 FONKSİYONLARIN DEĞER DÖNDÜRMESİ 27](#_Toc39411884)

[ÖRNEK 5.4.1 28](#_Toc39411885)

[ÖRNEK 5.4.2 28](#_Toc39411886)

[5.5 YİNELEMELİ (RECURSIVE) FONKSİYONLAR 29](#_Toc39411887)

[ÖRNEK 5.5.1 29](#_Toc39411888)

[6.BÖLÜM POİNTER’LER 30](#_Toc39411889)

[6.1 ADRES KAVRAMI 30](#_Toc39411890)

[ÖRNEK 6.1.1 30](#_Toc39411891)

[6.2 POİNTER TANIMLAMA 31](#_Toc39411892)

[6.3 POİNTER İLE DİZİLERİN İLİŞKİSİ 31](#_Toc39411893)

[ÖRNEK 6.3.2 32](#_Toc39411894)

[7.BÖLÜM DİNAMİK BELLEK YÖNETİMİ 32](#_Toc39411895)

[ÖRNEK 7.1 34](#_Toc39411896)

[ÖRNEK 7.2 35](#_Toc39411897)

[8.BÖLÜM STRİNGLER 36](#_Toc39411898)

[8.1 KATARLARDA PRİNTF() VE SCANF() KULLANIMI 36](#_Toc39411899)

[ÖRNEK 8.1.1 36](#_Toc39411900)

[ÖRNEK 8.1.2 37](#_Toc39411901)

[8.2 GETS() VE PUTS() FONKSİYONLARI 37](#_Toc39411902)

[ÖRNEK 8.2.1 38](#_Toc39411903)

[8.3 STRİNGLERE İLK DEĞER ATAMA 38](#_Toc39411904)

[ÖRNEK 8.3.1 38](#_Toc39411905)

[ÖRNEK 8.3.2 39](#_Toc39411906)

[8.4 STANDART STRİNG FONKSİYONLARI 39](#_Toc39411907)

[8.4.1 \*STRLEN() FONKSİYONU 39](#_Toc39411908)

[ÖRNEK 8.4.1.1 39](#_Toc39411909)

[8.4.2 \*STRCPY() VE \*STRNCPY() FONKSİYONU 40](#_Toc39411910)

[ÖRNEK 8.4.2.1 40](#_Toc39411911)

[ÖRNEK 8.4.2.2 41](#_Toc39411912)

[8.4.3 \*STRCMP() VE \*STRNCMP() FONKSİYONU 41](#_Toc39411913)

[ÖRNEK 8.4.3.1 42](#_Toc39411914)

[8.4.4 \*STRCAT() FONKSİYONU 42](#_Toc39411915)

[ÖRNEK 8.4.4.1 42](#_Toc39411916)

[8.4.5 \*STRSTR() FONKSİYONU 43](#_Toc39411917)

[ÖRNEK 8.4.5.1 44](#_Toc39411918)

[8.4.6 \*ATOİ() VE \*ATOF FONKSİYONLARI 45](#_Toc39411919)

[ÖRNEK 8.4.6.1 45](#_Toc39411920)

[9.BÖLÜM MATEMATİKSEL İŞLEMLER (MATH.H) 46](#_Toc39411921)

[9.1 EXP() FONKSİYONU 46](#_Toc39411922)

[ÖRNEK 9.1.1 46](#_Toc39411923)

[9.2 SQRT() FONKSİYONU 46](#_Toc39411924)

[ÖRNEK 9.2.1 47](#_Toc39411925)

[9.3 POW() FONKSİYONU 47](#_Toc39411926)

[ÖRNEK 9.3.1 48](#_Toc39411927)

[9.4 LOG() FONKSİYONU 48](#_Toc39411928)

[ÖRNEK 9.4.1 49](#_Toc39411929)

[9.5 COS() FONKSİYONU 49](#_Toc39411930)

[ÖRNEK 9.5.1 49](#_Toc39411931)

[9.6 SİN() FONKSİYONU 50](#_Toc39411932)

[ÖRNEK 9.6.1 50](#_Toc39411933)

[9.7 CEİL() VE FLOOR() FONKSİYONLARI 50](#_Toc39411934)

[ÖRNEK 9.7.1 51](#_Toc39411935)

[9.8 FABS() FONKSİYONU 51](#_Toc39411936)

[ÖRNEK 9.8.1 52](#_Toc39411937)

[10.BÖLÜM STRUCTLAR 52](#_Toc39411938)

[10.1 YENİ DEĞİŞKEN VERİ TİPİ OLUŞTURMA 52](#_Toc39411939)

[ÖRNEK 10.1 53](#_Toc39411940)

[ÖRNEK 10.2 54](#_Toc39411941)

[10.2 TYPEDEF KULLANIMI 55](#_Toc39411942)

[ÖRNEK 10.2.1 55](#_Toc39411943)

[10.3 YAPILAR(STRUCT’LAR) 56](#_Toc39411944)

[ÖRNEK 10.3.1 56](#_Toc39411945)

[ÖRNEK 10.3.2 57](#_Toc39411946)

[10.4 İÇ İÇE YAPILAR 58](#_Toc39411947)

[ÖRNEK 10.4.1 58](#_Toc39411948)

[10.5 YAPI DİZİLERİNE POİNTER İLE ERİŞİM 59](#_Toc39411949)

[10.6 YAPILAR VE FONKSİYONLAR 60](#_Toc39411950)

[ÖRNEK 10.6.1 61](#_Toc39411951)

[11.BÖLÜM DOSYALAMA İŞLEMLERİ 62](#_Toc39411952)

[11.1 DOSYA TÜRLERİ 62](#_Toc39411953)

[11.2 DOSYA İŞLEMLERİ 62](#_Toc39411954)

[11.3 BİR DOSYAYI AÇMA-OLUŞTURMA VE DÜZENLEME 62](#_Toc39411955)

[11.4 STANDART GİRİŞ / ÇIKIŞLARDA AÇMA MODLARI 63](#_Toc39411956)

[11.5 BİR DOSYAYI KAPATMA 64](#_Toc39411957)

[ÖRNEK 11.5.1 64](#_Toc39411958)

[ÖRNEK 11.5.2 65](#_Toc39411959)

[11.6 İKİLİ BİR DOSYAYA OKUMA VE YAZMA 65](#_Toc39411960)

[ÖRNEK 11.6.1 66](#_Toc39411961)

[11.7 İKİLİ DOSYADAN OKUMA 67](#_Toc39411962)

[ÖRNEK 11.7.1 67](#_Toc39411963)

[ÖRNEK 11.7.2 68](#_Toc39411964)

# 1.BÖLÜM PROGRAMLAMAYA GİRİŞ

## 1.1 PROBLEM ÇÖZME

Bilgisayar yazılımlarının genel amacı var olan bir problemi çözmektir. Biz problemleri bir yazılımcının düşünme ve çözme mantığı ile aşmaya çalışacağız. Bilgisayar ile problem çözmek 4 oluşmaktadır diyebiliriz. Bunlar;

* Problem belirlenmesi ve anlaşılması
* Algoritmayı oluşturma
* Programı geliştirme
* Programı test etme

## 1.2 ALGORİTMA

Bir problemin çözülebilmesi için yapılan her bir adımın bütünleşmiş halidir. Algoritmaları kullanarak daha kolay bir şekilde programları üretebiliriz. Algoritmaların en muazzam özelliği de her türlü bir alternatif gidiş yolu üreterek sonuca ulaşıldığının garantisini verebilmektir ve bir algoritmada veri girişlerinin değerlendirilmesi ve buna karşılık olarak sonuçların elde edilmesi gerekir.

## 1.3 ALGORİTMADA OLMASI GEREKEN HUSUSLAR

* **Girdi/çıktı**

Girdi, algoritmanın üzerinde işlem yapması için aldığı veridir.

Çıktı, algoritmanın girdiye karşılık verdiği sonuçtur.

* **Açıklık**

Her bir adım açık, anlaşılır şekilde olmalıdır. Birden fazla tabiri olabilecek ifadelerin kullanılmamasına dikkat edilmelidir.

* **Sonluluk**

Her algoritma mutlaka sonlu sayıda işlemi olmak zorundadır ve bu işlemlerin de bir sonlu süreye sahip olması gerekmektedir.

* **Etkinlik**

Yazacağımız algoritmalarımız gereksiz yere tekrarlanan adımlardan arındırılmış ve gerektiği zamanlarda başka algoritmalar içerisinde de kullanılabilir olmalıdır.



### ÖRNEK 1.3.1

Girilen bir sayının karesini alıp ekrana yazan bir programın algoritmasını yazalım.

**Girdi :** Karesi alınacak sayı ( **X** )

**Çıktı :**  Girilen sayının karesi (**Sonuç**)

|  |
| --- |
| 1. Başla 2. x değerini gir. 3. Sonuç = x \* x 4. Sonuç değerini ekrana yaz. 5. Bitir. |



### ÖRNEK 1.3.2

Girilen iki sayıyı karşılaştırıp büyük olan sayıyı bulan algoritmayı yazalım.

**Girdi :** Birinci sayı (**X**), ikinci sayı (**Y**)

**Çıktı :** Büyük sayı

|  |
| --- |
| 1. Başla 2. X değerini gir. 3. Y değerini gir. 4. Eğer x, y’ den büyük ise adım 6’ ya git. 5. Eğer y, x’ ten büyük ise adım 7’ ye git. 6. X değerini ekrana yaz ve adım 8’ e git. 7. Y değerini ekrana yaz ve adım 8’ e git. 8. Bitir. |



### ÖRNEK 1.3.3

Bir sınıftaki öğrencilerin bir dersten aldıkları notların ortalamasını ekrana yazan algoritmayı oluşturalım

**Girdi :** Öğrencilerin notlarını ( ,,,….. ,), öğrenci sayısı (**X**), indis (**İ**), notların toplamı (**toplam**),

**Çıktı :** Ortalama (**ort**)

|  |
| --- |
| 1. Başla 2. X değerini gir. 3. İ değerine ilk değer olarak 1değerini ver. 4. Öğrencinin notunu (n) gir. 5. toplam = toplam + n yap. 6. İ’yi 1 arttır. 7. Eğer İ > X ise Adım 9’ a git. 8. Adım 4’e git. 9. ort = toplam / x 10. ort değerini ekrana yaz. 11. Bitir. |

# 2.BÖLÜM C PROGRAMLAMA DİLİNİN GENEL YAPISI

## 2.1 TEMEL MADDELER

* C Programlama Dili genel amaçlı orta seviyeli ve yapısal bir programlama dilidir.
* C Programlama Dili, hemen her alanda kullanılmaktadır.
* C taşınabilir (portable) bir dildir.
* C, güçlü ve esnek bir dildir.
* C, güçlü ve esnek bir dildir. C ile işletim sistemi veya derleyici yazabilir, kelime işlemciler oluşturabilir veya grafik çizebiliriz.
* C++, Java, JavaScript, JavaApplet, PHP, C# gibi diller C dilinden esinlenmiştir.

## VERİ TÜRLERİ



C dilindeki veri türlerine baktığımız zaman her programlama dilinde olduğu gibi C dilinde de birkaç önemli veri türleri vardır. Bunlar ;

* Char
* İnt
* Float
* Double

veri tipleridir.

**Char :**

* Karakter veri tipi sadece bir karakteri saklamak için kullanılır.
* Bu veri tipi bellekte 1 byte yer ayırır ve sadece 1 harf saklar.
* “char” anahtar kelimesi ile değişken tanımlaması yapılır.

**İnt :**

* Tam sayı tipindeki sayısal değerleri saklamak için kullanılır.
* “İnt” anahtar kelimesi ile değişken tanımlaması yapılır.
* Bu veri tipi bellekte 2, 4 ya da 8 byte boyutunda yer ayırır. Bu boyut işlemciye göre değişir.

**Float :**

* Float veri tipi ondalıklı değerleri saklamak için kullanılır.
* Bellekte 4 byte yer kaplar
* Float veri tipinde ondalık değer olarak en fazla 6 basamak kullanılır.(10,123456)

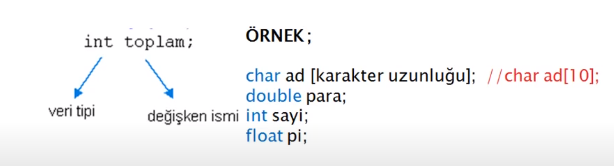
**Double :**

* Double veri tipinin float veri tipinden en önemli farklarından birisi ondalık değer olarak 15 basamak kullanılabilir olması(10,123456789012345)

Not : float veri tipi 4 byte, double veri tipi ise bellekte 8 byte yer ayırır.

## 2.3 DEĞİŞKENLER

Değişkenler bilgisayarın geçici belleğinde bilginin saklandığı gözlere verilen sembolik adlardır. Bir C programında, bir değişken tanımlandığında bu değişken için bellekte bir yer ayırır. Her değişkenin tuttuğu değerin nasıl bir veri olduğunu gösteren bir veri tipi vardır.

Aşağıdaki resimde kullanım şekilleri sembolize edilmiştir.

## 2.4 OPERATÖRLER

### 1. ARİTMETİK OPERATÖRLER

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Açıklama | Örnek | Anlamı |
| + | **Toplama** | **X+Y** | **X ve Y’nin toplamı** |
| - | **Çıkarma** | **X-Y** | **X ve Y’nin farkı** |
| \* | **Çarpma** | **X\*Y** | **X ve Y’nin çarpımı** |
| / | **Bölme** | **X/Y** | **X ve Y’nin oranı** |
| % | **Mod** | **X%Y** | **X ve Y’nin mod’u** |

### 2. ATAMA PRERATÖRLERİ

|  |  |  |  |
| --- | --- | --- | --- |
| OPERATÖR | AÇIKLAMA | ÖRNEK | ANLAMI |
| = | **Atama** | **X = 7;** | **X = 7;** |
| += | **Ekleyerek atama** | **X + = 3;** | **X = X +3;** |
| -= | **Eksilterek atama** | **X - = 5;** | **X = X-5;** |
| \*= | **Çarparak atama** | **X \* = 4;** | **X = X \* 4;** |
| /= | **Bölerek atama** | **X / = 2;** | **X = X / 2;** |
| %= | **Bölüp, kalanını atama** | **X % = 9;** | **X = X % 9;** |
| ++ | **Bir arttırma** | **X++; VEYA ++X;** | **X = X + 1;** |
| -- | **Bir azaltma** | **X--; VEYA --X;** | **X = X – 1;** |

### 3. KARŞILAŞTIRMA OPERATÖRLERİ

|  |  |  |  |
| --- | --- | --- | --- |
| OPERATÖR | AÇIKLAMA | ÖRNEK | ANLAMI |
| > | Büyüktür | X > Y | X, Y’den büyük mü ? |
| < | Küçüktür | X < Y | X, Y’den küçük mü ? |
| = = | Eşittir | X = = Y | X, Y’ye eşit mi ? |
| > = | Büyük eşittir | X > = Y | X, Y’den büyük yada eşit mi ? |
| < = | Küçük eşittir | X < = Y | X, Y’den küçük yada eşit mi ? |
| ! = | Eşit değil | X ! = Y | X, Y’den farklı mı ? |
| & & | Mantıksal VE | X > 2 & & X < Y | X, Y’den büyük VE y den küçük mü ? |
| | | | Mantıksal VEYA | X > 2 | | X < Y | X, Y’den büyük VEYA y den küçük mü ? |

# 3.Bölüm PROGRAM DENETİM VE DÖNGÜ YAPISI

C’de yazdığımız kodlar yukarıdan aşağıya doğru, işleme konur. Ancak bazı yapılar ile bu işlemlerin sırasını değiştirebiliriz. Bunlar;

* İF
* İF ELSE
* İF - ELSE İF – ELSE

Yapılarıdır.

## 3.1 İF YAPISI

 C Programlama dilinde program akışını kontrol etmekte en çok kullanılan yapı “if” yapısıdır.

**Kullanım**

İf (test edilecek ifade)

Eğer ifade doğru ise bu kısım çalışır;

Veya

İf (test edilecek ifade) eğer ifade doğru ise bu kısım çalışır;

Eğer ifade doğru ise ve bu kısımda birden fazla satır bulunuyorsa o zaman süslü parantezler {} kullanmamız gerekir.

İf (test edilecek ifade )

{

İfade doğru ise çalışır.

**…**

**..**

**.**

}

## [3.2 İF ELSE YAPISI](#_3.2__İF)

Peki yazdığımız koşul yanlış ise ne yapacağız ? şimdi de ona bakalım.

Böyle bir durumda “else” yapısı devreye giriyor. Aşağıdaki örneği inceleyelim



**Kullanım**

İf (test edilecek ifade )

{

//eğer ifade doğru ise bu kısım çalışır;

}

Else

{

// eğer ifade yanlış ise bu kısım çalışır;

}

## 3.3 İF – ELSE İF – ELSE

 Bu yapı kendisinden önce bir “if” ifadesi var ise kullanılır. “if” yapısı içindeki şart doğru değilse bu seferde “else if ” yapısı kullanılır ve koşulun doğruluğu kontrol edilir.

### ÖRNEK 3.2.1

#include <stdio.h>

int main ()

{

int sayi;

printf("Bir sayi giriniz.");

scanf("%d",&sayi);

if(sayi>0){

printf("0'dan büyüktür.\n ");

}

else if (sayi<0){

printf("0'dan küçüktür. \n ");

else{

printf(" 0'dir. ");

}

return 0;

}

## 3.4 SWİTCH – CASE

 Bu yapının başlıca amacı bir değişkenin değerine göre programa yön vermektir.

**Kullanım :**

switch( degisken ) {

case sabit1:

komut(lar)

[break]

case sabit2:

komut(lar)

[break]

case sabitN:

komut(lar)

[break]

default:

komut(lar);

}



### ÖRNEK 3.4.1

Dışardan girilen (1-12 ) aralığındaki sayıya karşılık gelen ay ismini yazan programı yazalım.

**Çözüm**

#include <stdio.h>

#include <stdlib.h>

int main(){

int ay;

printf("Ay degerini giriniz(1-12)= ");

scanf("%d",&ay); // Kullanıcıdan değer girmesi beklenir

switch(ay){

case 1: printf("Girdiniginiz deger \"Ocak\" ayidir"); // Ekrana " yazdırmak için \" yazmak lazım!!!

break;

case 2: printf("Girdiniginiz deger \"Subat\" ayidir");

break;

case 3: printf("Girdiniginiz deger \"Mart\" ayidir");

break;

case 4: printf("Girdiniginiz deger \"Nisan\" ayidir");

break;

case 5: printf("Girdiniginiz deger \"Mayis\" ayidir");

break;

case 6: printf("Girdiniginiz deger \"Haziran\" ayidir");

break;

case 7: printf("Girdiniginiz deger \"Temmuz\" ayidir");

break;

case 8: printf("Girdiniginiz deger \"Agustos\" ayidir");

break;

case 9: printf("Girdiniginiz deger \"Eylul\" ayidir");

break;

case 10: printf("Girdiniğini deger \"Ekim\" ayidir");

break;

case 11: printf("Girdiniğini deger \"Kasim\" ayidir");

break;

case 12: printf("Girdiniğini deger \"Aralik\" ayidir");

break;

default: printf("Hatali islem sectiniz!!!"); //1-12 aralığından farklı bir değer girerse ekrana default yazdırılır

break;

}

printf("\n");

system("pause"); // Kullanıcı herhangi bir tuşa basana kadar programı beklemeye alır

return 0;

}



### ÖRNEK 3.4.2

Switch – case yapısı kullanarak basit bir hesap makinası programını yazalım.

**ÇÖZÜM**

**#include <stdio.h>**

**int main(){**

**int x,y,tercih;**

**printf("\_\_\_ MENU \_\_\_\n");**

**printf("[1] Toplama\n");**

**printf("[2] Cikarma\n");**

**printf("[3] Carpma\n");**

**printf("[4] Bolme\n");**

**printf("Tercih Ettiginiz Islem Numarasi: ");**

**scanf("%d",&tercih);**

**switch (tercih){**

**case 1:**

**printf("Islem Yapilacak Sayilari Giriniz: ");**

**scanf("%d %d",&x,&y);**

**printf("Sonuc: %d",x+y);**

**break;**

**case 2:**

**printf("Islem Yapilacak Sayilari Giriniz: ");**

**scanf("%d %d",&x,&y);**

**printf("Sonuc: %d",x-y);**

**break;**

**case 3:**

**printf("Islem Yapilacak Sayilari Giriniz: ");**

**scanf("%d %d",&x,&y);**

**printf("Sonuc: %d",x\*y);**

**break;**

**case 4:**

**printf("Islem Yapilacak Sayilari Giriniz: ");**

**scanf("%d %d",&x,&y);**

**printf("Sonuc: %d",x/y);**

**break;**

**default:**

**printf("Gecerli Bir Numara Girmediniz!");**

**}**

**return 0;**

**}**

## 3.5 WHİLE DÖNGÜSÜ

 While döngüsünün kullanımı aşağıdaki gibidir.

**Kullanım**

while( koşul )

{

komut1;

komut2;

komut3;

**….**

}

While döngüsünde koşul doğru olduğu sürece döngü dönmeye devam eder.



### ÖRNEK 3.5.1

#include<stdio.h>

int main( void )

{

int i = 0;

while( i++ < 10 ) {

printf("%2d: Merhaba Dünya\n",i);

}

return 0;

}

## 3.6 DO – WHİLE

Yaptığı iş, while ile hemen hemen aynıdır; verilen işi, döngü koşulu bozulana kadar sürdürür. Ancak while'a göre önemli bir farkı vardır.

while döngülerinde, döngü içersindeki işlem yapılmadan önce, sunulan koşul kontrol edilir. Şayet koşul sağlanmıyorsa, o while döngüsünün hiç çalışmama ihtimali de bulunmaktadır. do while döngülerindeyse, durum böyle değildir. İlk çalışmada koşul kontrolü yapılmaz. Dolayısıyla, her ne şartta olursa olsun, döngünüz -en azından bir kere- çalışacaktır.

### ÖRNEK 3.6.1

#include<stdio.h>

int main( void )

{

int i = 0;

do

{

printf("%2d: Merhaba Dünya\n",++i);

}

while( i < 10 );

return 0;

}

## 3.7 FOR DÖNGÜSÜ

For döngüsünün kullanım şekli aşağıdaki gibidir.

**Kullanım**

for( ilk\_deger\_atama; koşul; artırma değeri ){

komut(lar)

}

İlk değer atamaları, döngüde kullanılacak sayaç değişkenine atama yapılan kısımdır. Koşul kısmında döngünün hangi koşulda çalışacağı mantıksal operatörler ile gösterilir. Artırma değeri ise döngünün sayaç değerininin ne kadar artacağını ya da azalacağını belirtmek için kullanılır.

### ÖRNEK 3.7.1

#include<stdio.h>

int main( void )

{

int i;

for( i = 0 ; i < 10; i++ ) {

printf("%2d: Merhaba Dünya\n",(i+1));

}

return 0;

}

# 4.BÖLÜM DİZİLER

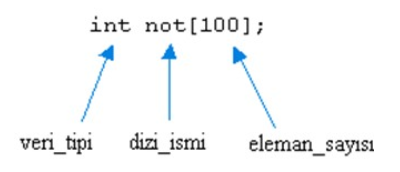
Bellekte sıralı bir şekilde bulunan ve aynı türden bilgilerin saklandığı veri yapısına dizi(array) denir.

Değişkenler aynı anda tek bir değer tutabilen temel değişkenler ve birden fazla değer saklayabilen bileşik değişkenler olmak üzere ikiye ayrılır.

Temel değişkenler bellekte tek bir hücreyi tanımlayıp, içlerinde tek bir değeri tutabilirler.

Diziler ise ardarda sıralanmış bellek hücreleridirler. Diziler bu bağlamda bileşik değişkenlerdir ve bellekte aynı anda birden fazla değerin saklamasını mümkün kılarlar.

**Kullanım**

Veri\_tipi Dizi\_ismi[eleman\_sayısı];

Bellek görüntüsü de aşağıdaki gibidir.



### ÖRNEK 4.1

Kullanıcıdan alınan 10 elemanlı bir sayı dizisinin elemanlarının toplamını bulan programı yazalım.

**ÇÖZÜM**

#include <stdio.h>

main()

{

int i;int toplam=0;

int a[10];

for(i=0;i<10;i++)

{

printf("dizi elemanlarini gir:");

scanf("%d",&a[i]); printf("\n");

toplam = toplam + a[i];

}

printf("toplam= %d \n",toplam);

getch();

}

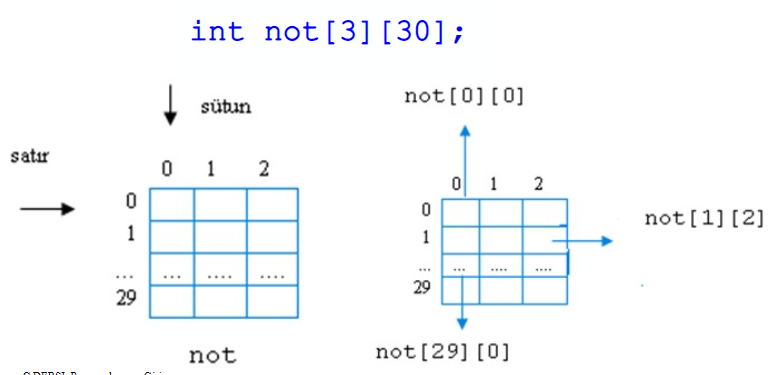
## 4.1 İKİ BOYUTLU DİZİLER

Çok boyutlu dizilerin tanımlanması da mantıken tek boyutlu diziler ile aynıdır. Çok boyutlu diziler, matris mantığıyla çalışırlar. Bu nedenle iki adet farklı indeks kullanılır. Birinci indeks satır elemanlarını tutarken ikinci indeks sütun elemanlarını tutacaktır.

**Kullanım**

(veri tipi) (dizinin\_adı [] [] ) veya (veri tipi) (dizinin\_adı [] [] ) = {};

Şeklinde tanımlanır.



### ÖRNEK 4.1.1

int sayilar [] [] ;

int sayilar [3] [3] = {{1,4,7},{2,5,8},{3,6,9}};

Böyle bir durumda sayi [3] [3] dizisi oluşmuş olur.

sayi[0] [0] = 1

sayi[0] [1] = 4

sayi[0] [2] = 7

sayi[1] [0] = 2

sayi[1] [1] = 5

sayi[1] [2] = 8

sayi[2] [0] = 3

sayi[2] [1] = 6

sayi[2] [2] = 9

### 

### ÖRNEK 4.1.2

Kullanıcının, 30 bir sınıftaki her öğrenci için 3’er sınıv notu gireceği ve her sınav için sınıf ortalamasının ekranda gösterileceği bir program yazınız.

**ÇÖZÜM**

#include <stdio.h>

int main(void)

{ int not[30][3];

int i,j,toplam;

double orta;

/\* Kullanicidan notlarin alinmasi\*/

for(i=0;i<30;++i)

{ printf("%d. ogrenci notlari:",i+1);

for(j=0;j<3;++j)

scanf("%d", &not[i][j]);

}

/\*Her sinavin ortalamasinin bulunmasi\*/

for(j=0;j<3;++j)

{ toplam=0;

for(i=0;i<30;++i)

toplam+=not[i][j];

orta=toplam/30.0;

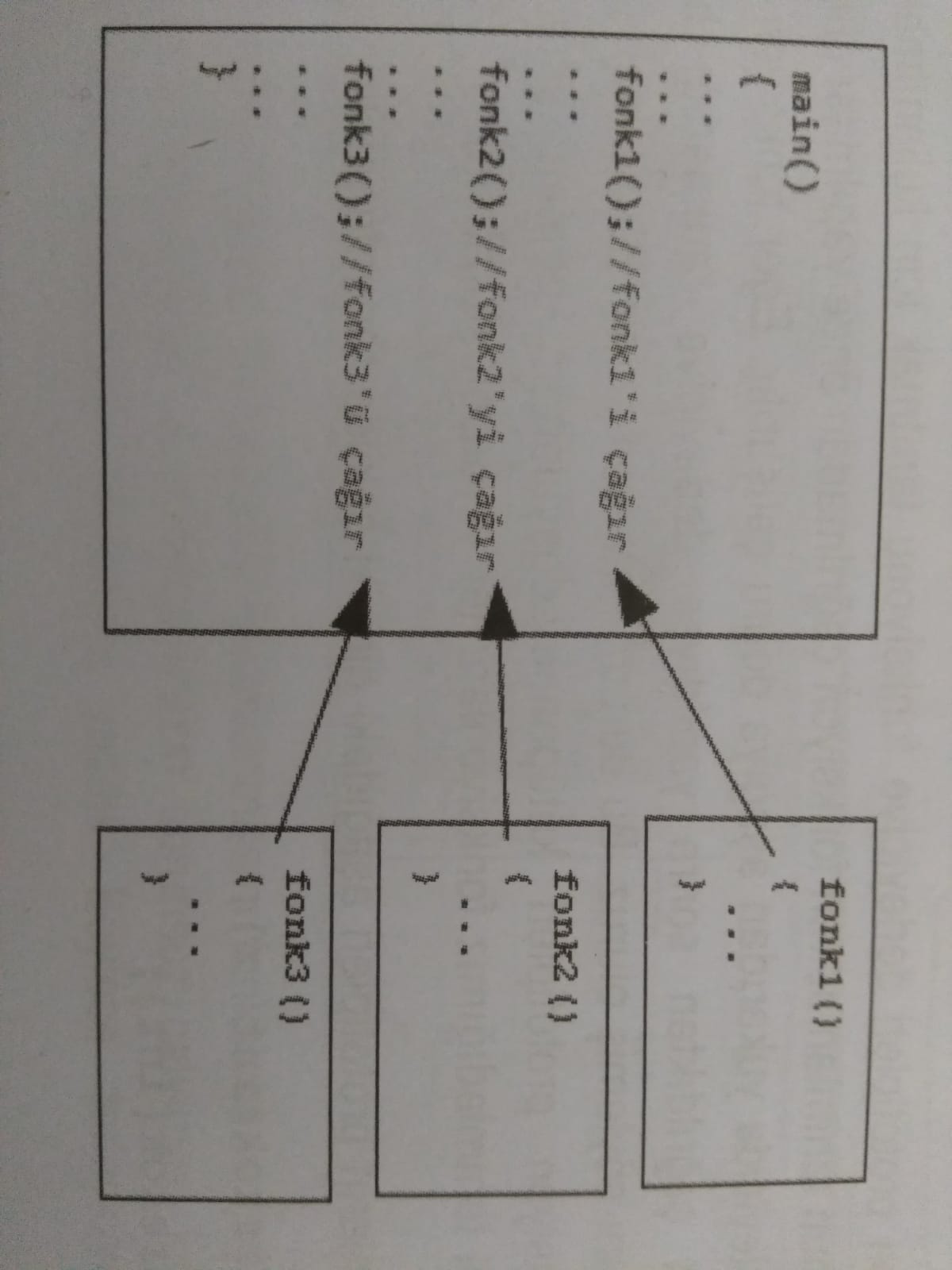
printf("%d. sinav ortalamasi: %5.2f\n",j+1,orta);

}

return (0); }

# 5.BÖLÜM FONKSİYONLAR

Fonksiyonlar, modülerlik sağlar. Sayının asallığını test eden bir fonksiyon yazıp, bunun yanlış olduğunu fark edersek, bütün programı değiştirmemiz gerekmez. Yanlış fonksiyonu düzeltmemiz programın doğru çalışmasını sağlayacaktır. Üstelik yazdımız fonksiyonlara ait kodu, başka programlara taşımamız da oldukça basittir.

 Programlamada fonksiyon kullanmanın sebeblerinden biride kodların tekrar etmesini engellemektir. Örneğin; karekök hesabu yapar bir fonksiyon tasarlarsak programımızda her karekök hesabı yapıldığında karekök işlemlerini yazmaktan kurtuluruz.

## FONKSİYONLARIN TANIMLANMASI

**Kullanım**

(Tür) fonksiyon\_adı(paremetreler)

{

komutlar;

}

Burada tür, fonksiyonun içindeki işlemlerin sonucunda bize dönecek değerin türüdür. Döndürmek istediğimiz değer tam sayı (int), karekter(char) vb. olabilir. Örneğin;

int toplam(); fonksiyonunun bize döndürdüğü değer bir tam sayı olacaktır.

Eğer fonksiyonumuz değer döndürmüyorsa tanımlamadaki tür kısmına void yazılır.

**Kullanım**

void ekranayaz()

{

Printf("Derğer döndürmeyen fonksiyon. ");

}

Yukarıda örnek, değer döndürmeyen bir fonksiyondur. Bu fonksiyon çağrıldığında ekrana çıktı yazılır ve sonra fonksiyonun işlemi birmiş olur.

## 5.2 FONKSİYONLARIN ÇAĞRILMASI

 Bütün C Programları çalışmaya main fonksiyonundan başlar. main fonksiyonu, programın başladığı nokta olması dışında diğer fonksiyonlardan hiçbir farkı yoktur. main fonksiyonunun işlevi bitince program sonlanır. main fonksiyonu dışındaki fonksiyonlar, main içinde gerekli oldukları yerlerde işlemlerini yapmaları için çağırılırlar.

### ÖRNEK 5.2.1

Klavyeden girilen sayının karesini ekrana yazdıran programı bir fonksiyon yardımı ile yazalım.

#include <stdio.h>

void kareal()

{

int x;

printf(“karesi alınacak sayiyi giriniz:”);

scanf (“%d”,&x);

printf(“sayininin karesi = %d \n”,x\*x);

}

int main()

{

kareal();

return 0;

}

## 5.3 FONKSİYONLARA PAREMETRE AKTARMA

Fonksiyonu tanımlarken, fonksiyona nasıl bir değerin gönderileceğini belirtiriz. Gönderilecek değerin hangi değişken tipinde olduğunu ve değişken adını yazarız. Fonksiyonu tanımlarken, yazdığımız bu değişkenlere 'parametre' (parameter) denir. Argüman (argument) ise, parametrelere değer atamasında kullandığımız değerlerdir.



### ÖRNEK 5.3.1

Aşağıdaki örnekte evin duvarlarını dışarıdan kullanıcının girdiği karakter olarak belirleyip evi çizen programı inceleyelim.

**ÇÖZÜM**

/\* Ev sekli cizen program \*/

#include<stdio.h>

// Evin catisini cizen fonksiyon.

void catiyi\_ciz( void )

{

printf( " /\\ \n" );

printf( " / \\ \n" );

printf( " / \\ \n" );

printf( " / \\\n" );

printf( "----------\n" );

}

// Evin katini cizen fonksiyon.

// sol ve sag degiskenleri fonksiyon

// parametreleridir.

void kat\_ciz( char sol, char sag )

{

printf( "%c %c\n", sol, sag );

printf( "%c %c\n", sol, sag );

printf( "%c %c\n", sol, sag );

printf( "----------\n" );

}

// Programin calismasini saglayan

// ana fonksiyon.

int main( void )

{

char sol\_duvar, sag\_duvar;

printf( "Kullanılacak karakterler> " );

scanf( "%c",&sol\_duvar );

scanf( "%c",&sag\_duvar );

catiyi\_ciz( );

// sol\_duvar ve sag\_duvar, fonksiyona

// giden argumanlardir.

kat\_ciz( sol\_duvar, sag\_duvar );

kat\_ciz( sol\_duvar, sag\_duvar );

kat\_ciz( sol\_duvar, sag\_duvar );

return 0;

}

## 5.4 FONKSİYONLARIN DEĞER DÖNDÜRMESİ

Değer döndüren bir fonksiyon çağrıldığında doğal olarak bir dönüş olacaktır. Fonksiyonlarda bu dönüş return ifadesi yardımı ile yapılır. İlk olarak oluşturacağımız bir fonksiyon yardımıyla,o fonksiyon içindeki bir değişkenin değerini ekrana yazdıralım.

### ÖRNEK 5.4.1

#include <stdio.h>

int deneme()

{

int x = 10;

return x;

}

int main()

{

printf("x = %d",deneme());

return 0;

}



### ÖRNEK 5.4.2

Kullanıcıdan iki sayı isteyen ve bu sayıları toplayıp döndüren fonksiyon yardımıyla, girilen iki sayının toplamını ekranda gösteren programı yazalım.

**ÇÖZÜM**

#include <stdio.h>

int topla()

{

int x,y;

printf("iki sayi girin: ");

scanf("%d%d",&x,&y);

return x+y;

}

int main()

{

printf("sayilarin toplami(x+y)= %d",topla());

return 0;

}

## 5.5 YİNELEMELİ (RECURSIVE) FONKSİYONLAR

**Yinelemeli fonksiyonlar,** kendi kendini çağıran fonksiyonlardır.

Örneği inceleyerek daha iyi görebiliriz.

### ÖRNEK 5.5.1

Dışarıdan girilen bir sayının faktöriyelini recursive fonksiyon kullanarak yazalım.

#include <stdio.h>

int faktoriyel(int);

int main()

{

int sayi;

printf("faktöriyeli alınacak sayiyi girin: ");

scanf("%d",&sayi);

printf("sayinin faktöriyeli= %d \n ",faktoriyel(sayi));

return 0;

}

int faktoriyel(int a)

{

int x;

if(a>1)

x=a\*faktoriyel(a-1);

else

x=1;

return x;

}

# 6.BÖLÜM POİNTER’LER

Pointerler, içerisinde bir hafıza adresi olan değişkenlerdir. Burada adres, bellekteki bir verinin yerini gösterir. Bellekte yer kaplayan her nesnenin bir adresi vardır. Değişkenler bir verinin değerini içerirken göstericiler bir veriyi tutan değişkenlerin adresini içerir. Normalde bir değişken veriyi doğrudan belirtirken göstericiler veriyi dolaylı yoldan belirtir.

|  |  |
| --- | --- |
| Char | 1 byte |
| İnt | 2 byte |
| double | 4 byte |

## 6.1 ADRES KAVRAMI

* Adres, bir değişken için bellekte ayrılan yerdir.
* Adres bilgileri hexadecimal sayı olarak gösterilir.

Adres operatörü = &

### ÖRNEK 6.1.1

Girilen 3 değişkenin adreslerini ekrana yazdıran programı yazınız.

**ÇÖZÜM**

#include <stdio.h>

#include <conio.h>

int a=5;

double b=6.8;

char c='K';

void main(){

printf("a değişkeninin değeri = %d\n",a);

printf("b değişkeninin değeri = %f\n",b);

printf("c değişkeninin değeri = %c\n",c);

printf("a değişkeninin adresi = %x\n",&a);

printf("b değişkeninin adresi = %x\n",&b);

printf("c değişkeninin adresi = %x\n",&c);

getch();

}

## 6.2 POİNTER TANIMLAMA

* Pointer tipinde değişkenin tanımlanmasında önce değişkenin ne tip bir değeri gösterdiğini tanımlamak gerekiyor.
* Pointer tipi değişkenler her zaman “\*” işareti ile gösterilir:
  + Veritipi \*d\_ adı;
* Veri tipi, d\_ adi adli pointer’ in hafızada gösterdiği adreste ne tip bir değişken olduğunu belirtir. Teknik olarak herhangi bir pointer hafızada herhangi tipte bir değişkeni gösterebilir. Ama pointer’lar ile yapılan işlemlerde gösterdiği veri tipi göz önüne alınarak hesap yapıldığından veri tipi önem kazanır.

## 6.3 POİNTER İLE DİZİLERİN İLİŞKİSİ

 C’ de, diziler ve pointerler iç içedir ve çoğu yerde birbirlerinin yerine de kullanılabilirler. Dizi ismini sabit bir gösterici olarak da düşünebiliriz. Göstericiler, dizilerin birçok işleminde kullanılırlar.

**Kullanım**

int \*p,x[5];

p = x; // x dizisinin başlangıç adresi p’ye atandı.

veya

p = &x[0]; // x dizisinin başlangıç adresi p’ye atandı

Aşağıdaki örneği inceleyerek daha göreceğiz.

### ÖRNEK 6.3.2

#include<stdio.h>

int main( void )

{

int i;

// dizi'yi tanimliyoruz.

int dizi[ 6 ] = { 4, 8, 15, 16, 23, 42 };

// ptr adinda bir pointer tanimliyoruz.

int \*ptr;

// ptr'nin dizi'yi isaret etmesini soyluyoruz.

ptr = dizi;

// ptr'in degerini artirip, dizi'nin butun

// elemanlarini yazdiriyoruz.

for( i = 0; i < 6; i++ )

printf( "%d\n", \*( ptr + i ) );

return 0;

}

# 7.BÖLÜM DİNAMİK BELLEK YÖNETİMİ

Dinamik bellek yönetiminde, dizilerin boyutları önceden belirlenmez. Program akışında dizi boyutunu ayarlarız ve gereken bellek miktarı, program çalışırken tahsis edilir. Dinamik bellek tahsisi için calloc(  ) ve malloc(  ) olmak üzere iki önemli fonksiyonumuz vardır. Bellekte yer ayrılmasını bu fonksiyonlarla sağlarız. Her iki fonksiyon da stdlib kütüphanesinde bulunur. Bu yüzden fonksiyonlardan herhangi birini kullanacağınız zaman, programın başına #include<stdlib.h> yazılması gerekir.

Calloc() fonksiyonu aşağıdaki gibi kullanılır.



**Kullanım**

işaretçi \_adı = calloc ( eleman\_ sayısı, her \_elemanın \_boyutu );

calloc (  ) fonksiyonu eleman sayısını, eleman boyutuyla çarparak hafızada gereken bellek alanını ayırır. Dinamik oluşturduğunuz dizi içersindeki her elemana, otomatik olarak ilk değer 0 atanır.

malloc (  ) fonksiyonu, calloc (  ) gibi dinamik bellek ayrımı için kullanılır. calloc (  ) fonksiyonundan farklı olarak ilk değer ataması yapmaz. Kullanımıysa aşağıdaki gibidir:



**Kullanım**

isaretci\_adi = malloc( eleman\_sayisi \* her\_elemanin\_boyutu );

Gereksiz tüketilen bellekten kaçınmak gerekmektedir. Bunun için fazla bir şey yapmamız gerekmez; calloc(  ) fonksiyonuyla tahsis ettiğimiz alanı, işimiz bittikten sonra free(  ) fonksiyonuyla boşaltmamız yeterlidir.

### ÖRNEK 7.1

#include<stdio.h>

#include<stdlib.h>

int main( void )

{

// Dinamik bir dizi yaratmak icin

// pointer kullaniriz.

int \*dizi;

// Dizimizin kac elemanli olacagini

// eleman\_sayisi isimli degiskende

// tutuyoruz.

int eleman\_sayisi;

int i;

// Kullanicidan eleman sayisini girmesini

// istiyoruz.

printf( "Eleman sayısını giriniz> ");

scanf( "%d", &eleman\_sayisi );

// calloc( ) fonksiyonuyla dinamik olarak

// dizimizi istedigimiz boyutta yaratiyoruz.

dizi = calloc( eleman\_sayisi, sizeof( int ) );

// Ornek olmasi acisindan dizinin elemanlarini

// ekrana yazdiriliyor. Dizilerde yapabildiginiz

// her seyi hicbir fark olmaksizin yapabilirsiniz.

for( i = 0; i < eleman\_sayisi; i++ )

printf( "%d\n", dizi[i] );

// Dinamik olan diziyi kullandiktan ve isinizi

// tamamladiktan sonra free fonksiyonunu kullanip

// hafizadan temizlemelisiniz.

free( dizi );

return 0;

}

 Önceki örnekte calloc(  ) ile yazdığımız programın aynısını şimdi de malloc(  ) fonksiyonunu kullanarak yazalım:

### ÖRNEK 7.2

#include<stdio.h>

#include<stdlib.h>

int main( void )

{

// Dinamik bir dizi yaratmak icin

// pointer kullaniriz.

int \*dizi;

// Dizimizin kac elemanli olacagini

// eleman\_sayisi isimli degiskende

// tutuyoruz.

int eleman\_sayisi;

int i;

printf( "Eleman sayısını giriniz> ");

scanf( "%d", &eleman\_sayisi );

// malloc( ) fonksiyonuyla dinamik olarak

// dizimizi istedigimiz boyutta yaratiyoruz.

dizi = malloc( eleman\_sayisi \* sizeof( int ) );

for( i = 0; i < eleman\_sayisi; i++ )

printf( "%d\n", dizi[i] );

// Dinamik olan diziyi kullandiktan ve isinizi

// tamamladiktan sonra free fonksiyonunu kullanip

// hafizadan temizlemelisiniz.

free( dizi );

return 0;

}

# 8.BÖLÜM STRİNGLER

String dediğimiz şey de aslında bir dizidir. Değişken tipi char yani karakter olan diziler 'string' olarak isimlendirilirler.

Stringleri, şimdiye kadar gördüğümüz dizilerden ayıran, onları farklı kılan özellikleri yoktur. Örneğin bir tam sayı ( int ) dizisinde, tam sayıları saklarken; bir karakter dizisinde -yani katarda- karakterleri ( char ) saklarız. Bunun dışında bir fark yoktur. Ancak sık kullanılmalarına paralel olarak, katarlara ayrı bir önem vermemiz gerekir. Yaptığımız işlemler bilimsel ve hesaplama ağırlıklı değilse, hangi dili kullanırsak kullanalım, en çok içli dışlı olacağımız dizi tipi, karakter dizileridir. İsimler, adresler, kullanıcı adları, telefonlar vs... sözle ifade edilebilecek her şey için karakter dizilerini kullanırız. String’ler işte bu yüzden önemlidir.

## KATARLARDA PRİNTF() VE SCANF() KULLANIMI

Dizilerde elemanlara değer atama ya da onlardan değer okuma adım adım yapılan bir işlemdi. Genellikle bir döngü içerisinde, her dizi elemanı için scanf(  ) veya printf(  ) fonksiyonunu çağırmamız gerekiyordu. String’ler için böyle bir mecburiyet bulunmuyor. Tek bir kelimeyi, tek bir scanf(  ) fonksiyonuyla okutabilir ve elemanlara otomatik değer atayabiliriz. Yani "Merhaba" şeklinde bir girdi gelirse, 3.dizi elemanı 'r' olurken; 6.dizi elemanı 'b' olur. Önceki dizilerde gördüğümüzün aksine, eleman atamaları kendiliğinden gerçekleşir. Aşağıdaki örneği inceleyelim:

### ÖRNEK 8.1.1

#include<stdio.h>

int main( void )

{

char isim[30];

printf( "İsim giriniz> ");

scanf( "%s", isim );

printf( "Girdiğiniz isim: %s\n", isim );

return 0;

}

 String’lerin, esasında bir dizi olduğundan bahsetmiştim. Şimdi bunun uygulamasını yapalım. String’e değer atamak için yine aynı kodu kullanırken; string’den değer okumak için kodumuzu biraz değiştirelim:

### ÖRNEK 8.1.2

#include<stdio.h>

int main( void )

{

char isim[30];

int i;

printf( "İsim giriniz> ");

scanf( "%s", isim );

printf( "Girdiğiniz isim: ");

for( i = 0; isim[i]!='\0'; i++ )

printf( "%c", isim[i] );

printf("\n");

return 0;

}

### 8.2 GETS() VE PUTS() FONKSİYONLARI

Gets(  ) isminden anlayacağımız ( get string ) gibi string’e değer atamak için kullanılır. puts(  ) ( put string ) ise, bir string’in içeriğini ekrana yazdırmaya yarar. gets(  ) atayacağı değerin ayrımını yapabilmek için '\n' aramaktadır. Yani klavyeden Enter'a basılana kadar girilen her şeyi, tek bir string’e atayacak. puts(  ) fonksiyonuysa, printf(  ) ile benzer çalışır.

Boş karakter ( NULL Character ) yani '\0' ulaşana kadar string’i yazdırır; printf(  ) fonksiyonundan farklı olarak sonuna '\n' koyarak bir alt satıra geçer.

### ÖRNEK 8.2.1

#include<stdio.h>

int main( void )

{

char cumle[40];

printf( "Cümle giriniz> ");

gets( cumle );

printf( "Girdiğiniz cümle:\n" );

puts( cumle );

return 0;

}

### 8.3 STRİNGLERE İLK DEĞER ATAMA

Bir string tanımı yaptığımız anda, string’in bütün elemanları otomatik olarak '\0' ile doldurulur. Yani string’in bütün elemanlarına boş karakter (NULL Character) atanır. İstersek, string’i yaratırken içine farklı değerler atayabiliriz. String’ler de ilk değer ataması iki şekilde yapılır.

Birinci yöntemle değer ataması yapacağız, istediğimiz kelimeyi bir bütün olarak yazacağız:

### ÖRNEK 8.3.1

#include <stdio.h>

int main( void )

{

// Her iki stringede ilk deger

// atamasi yapiliyor. Ancak

// isim stringinde, boyut

// belirtilmezken, soyad stringinde

// boyutu ayrica belirtiyoruz.

char isim[] = "RAMAZAN";

char soyad[5] = "KOSE";

printf( "%s %s\n", isim, soyad );

return 0;

}

İkinci yöntemdeyse, kelime bütün olarak yazılmaz. Bunun yerine harf harf yazılır ve sonlandırmak için en sonuna boş karakter ( NULL ) eklenir:

### ÖRNEK 8.3.2

#include<stdio.h>

int main( void )

{

char isim[] = { 'R', 'A', 'M', 'A',

'Z', 'A', 'N', '\0' };

char soyad[5] = { 'K', 'O', 'S', 'E', '\0' };

printf( "%s %s\n", isim, soyad );

return 0;

}

## 8.4 STANDART STRİNG FONKSİYONLARI

Stringler’le daha kolay çalışabilmek için, bazı hazır kütüphane fonksiyonlarından bahsedeceğim. Bu fonkisyonlar, string kütüphanesinde bulunuyor. Bu yüzden, programınızın başına, #include<string.h> eklemeniz gerekiyor.

### 8.4.1 \*STRLEN() FONKSİYONU

 String’lerin boyutunu saptamak için, boş karakter ( NULL Character ) işaretinin yani "\0" simgesinin konumuna bakılır. Her seferinde arama yapmamıza gerek kalmasın diye strlen(  ) fonksiyonu geliştirilmiştir. strlen(  ) kendisine argüman olarak gönderilen bir string’in boyutunu geri döndürür. Aşağıdaki gibi kullanılmaktadır:

### ÖRNEK 8.4.1.1

#include<stdio.h>

#include<string.h>

int main( void )

{

printf( "String Uzunluğu: %d\n", strlen("Merhaba") );

return 0; }

### 8.4.2 \*STRCPY() VE \*STRNCPY() FONKSİYONU

Bir string’i, bir başka string’e kopyalamak için *strcpy(  )* fonksiyonunu kullanırız. String’ler aynı boyutta olmak zorunda değildir. Ancak kopya olacak string, kendisine gelecek kelimeyi alacak boyuta sahip olmalıdır. Fonksiyon prototipi aşağıdaki gibidir, geriye pointer döner.

**Kullanım**

char \*strcpy( char[ ], char[ ] );

### ÖRNEK 8.4.2.1

#include<stdio.h>

#include<string.h>

int main( void )

{

char kaynak[40]="Merhaba Dünya";

char kopya[30] = "";

strcpy( kopya, kaynak );

printf( "%s\n", kopya );

return 0;

}

Yukardaki örneği strncpy(  ) fonksiyonuyla tekrar edelim:

### ÖRNEK 8.4.2.2

#include<stdio.h>

#include<string.h>

int main( void )

{

char kaynak[40]="Merhaba Dünya";

char kopya[30] = "";

strncpy( kopya, kaynak, 9 );

printf( "%s\n", kopya );

return 0;

}

### 8.4.3 \*STRCMP() VE \*STRNCMP() FONKSİYONU

strcmp(  ) fonksiyonu, kendisine verilen iki string’i birbiriyle karşılaştırır. Stringler birbirine eşitse, geriye 0 döner. Eğer ilk string alfabetik olarak ikinciden büyükse, geriye pozitif değer döndürür. Eğer alfabetik sırada ikinci string birinciden büyükse, geriye negatif değer döner. Bu dediklerimi, daha iyi anlaşılması için bir tabloda göstereyim:

|  |  |
| --- | --- |
| Dönen Değer | Açıklama |
| <0 | String1, string2’den küçüktür. |
| 0 | String1 ve string2 birbirine eşittir |
| >0 | String1, string2’den büyüktür. |

strncmp(  ) için de aynı kurallar geçerlidir. Tek fark, karşılaştırılacak karakter sayısını girmemizdir. strcmp(  ) fonksiyonunda iki string, null karakter işareti çıkana kadar karşılaştırılır. Fakat strncmp(  ) fonksiyonunda, başlangıçtan itibaren kaç karakterin karşılaştırılacağına biz karar veririz.

Her iki fonksiyonu da kapsayan aşağıdaki örneği inceleyelim:

### ÖRNEK 8.4.3.1

#include<stdio.h>

#include<string.h>

int main( void )

{

int sonuc;

char ilk\_string[40]="Maymun";

char ikinci\_string[40]="Maytap";

sonuc = strcmp( ilk\_string, ikinci\_string );

printf( "%d\n", sonuc );

sonuc = strncmp( ilk\_string, ikinci\_string, 3 );

printf( "%d\n", sonuc );

return 0;

}

### 8.4.4 \*STRCAT() FONKSİYONU

 strcat(  ) kendisine verilen stringleri tamamen birleştirir.

### ÖRNEK 8.4.4.1

#include<stdio.h>

#include<string.h>

int main( void )

{

char ad[30], soyad[20];

char isim\_soyad[50];

printf( "Ad ve soyadınızı giriniz> " );

scanf( "%s%s", ad, soyad );

// isim\_soyad <-- ad

strcat( isim\_soyad, ad );

// isim\_soyad <-- ad + " "

strcat( isim\_soyad, " " );

// isim\_soyad <-- ad + " " + soyad

strcat( isim\_soyad, soyad );

printf( "Tam isim: %s\n", isim\_soyad );

return 0;

}

### \*STRSTR() FONKSİYONU

Bir string içinde, bir başka string’i aradığımız durumlarda, strstr(  ) fonksiyonunu kullanırız. strstr(  ) fonksiyonu, bir string içinde aradığımız bir string’i bulduğu takdirde bunun bellekteki adresini geriye döndürür. Yani dönen değer çeşidi bir pointer'dır. Eğer herhangi bir eşleşme olmazsa geriye bir sonuç dönmez ve pointer null olarak kalır. İnsanlar için hafıza adreslerinin veya pointer değerlerinin pek bir anlamı olmuyor. Bir string içinde arama yapıyorsanız, aradığınız yapının string’in neresinde olduğunu tespit etmek için aşağıdaki kodu kullanabiliriz:

### ÖRNEK 8.4.5.1

/\* strstr( ) fonksiyon ornegi \*/

#include<stdio.h>

#include<string.h>

int main( void )

{

char adres[] = "Esentepe Caddesi Mecidiyekoy Istanbul";

char \*ptr;

// 'adres' katari icinde, 'koy' kelimesini

// ariyoruz. Bu amacla strstr( ) fonksiyonunu

// kullaniyoruz. Fonksiyon buyuk-kucuk harf

// duyarlidir. Eger birden fazla eslesme varsa,

// ilk adres degeri doner. Hic eslesme olmazsa,

// pointer degeri NULL olur.

ptr = strstr( adres, "koy" );

if( ptr != NULL )

printf( "Baslangic notkasi: %d\n", ptr - adres );

else

printf( "Eslesme bulunamadi.\n" );

return 0;

}

### 8.4.6 \*ATOİ() VE \*ATOF FONKSİYONLARI

Verilen string’i, sayıya çevirmek gerekebilir. Eğer elimizdeki metni, bir tam sayıya ( int ) çevireceksek, atoi(  ) fonksiyonunu kullanmamız gerekir. Şayet dönüşüm sonunda elde etmek istediğimiz değişken tipi, virgüllü sayı ise ( float ), atof(  ) fonksiyonunu kullanırız. Her iki fonksiyon (stdlib.h) kütüphanesi içindedir.

### ÖRNEK 8.4.6.1

#include<stdio.h>

#include<stdlib.h>

int main( void )

{

char kok\_iki[] = "1.414213";

char pi[] = "3.14";

char tam\_bir\_sayi[] = "156";

char hayatin\_anlami[] = "42 is the answer";

printf( "%d\n", atoi( tam\_bir\_sayi ) );

printf( "%d\n", atoi( hayatin\_anlami ) );

printf( "%f\n", atof( kok\_iki ) );

printf( "%f\n", atof( pi ) );

return 0;

}

# 9.BÖLÜM MATEMATİKSEL İŞLEMLER (MATH.H)

## 9.1 EXP() FONKSİYONU

Matematikten bildiğimiz e sayısının**[e=2.71828182845904523536]**üssünü almaya yarayan **math.h** kütüphanesi içerisindeki bir fonksiyondur..

### ÖRNEK 9.1.1

#include<stdio.h>

#include<math.h>

int main()

{ int sayi;

double x;

printf("sayi girin");

scanf("%d",&sayi);

x=exp(sayi);

printf("e uzeri %d= %.2f",sayi,x);

getchar();getchar();

return 0;

}

## 9.2 SQRT() FONKSİYONU

Karekök alma işlemini yapan fonksiyonumuz sqrt() fonksiyonudur..Burada önemli olan karekök içerisinde negatif sayı olmaması gerektiğidir. Bunun için (do while) kalıbıyla örneğimizde önlem aldık.



### ÖRNEK 9.2.1

#include<stdio.h>

#include<math.h>

int main()

{

int sayi;

double x;

do {

printf("karekokunu almak istediginiz sayiyi girin");

scanf("%d",&sayi);

}while(sayi<0);

x=sqrt(sayi);

printf(" %d 'sayisinin karekoku = %.2f ",sayi,x);

getchar();getchar();

return 0;

}

## 9.3 POW() FONKSİYONU

Power yani üs anlamına gelen bu fonksiyon verdiğimiz bir sayının yine verdiğimiz diğer sayı kadar üssünü almaktadır... mesela x ve y sayılarını verip x üzeri y yi hesaplattıran bir c programı yazalım..

### ÖRNEK 9.3.1

#include<stdio.h>

#include<math.h>

int main()

{

int x,y;

int sonuc;

printf("X uzeri y icin x ve y degerlerini giriniz:");

scanf("%d%d",&x,&y);

sonuc=pow(x,y);

printf("%d uzeri %d = %d",x,y,sonuc);

getchar();getchar();

return 0;

}

## 9.4 LOG() FONKSİYONU

Bu fonksiyon verdiğiniz sayının doğal logaritmasını döndürür, şöyle ki; verdiğiniz sayıyı ln şeklinde hesaplar mesela e sayısı verirseniz size lne yi hesaplayıp 1 diyecektir. Örneğimizle devam edelim..



### ÖRNEK 9.4.1

#include<stdio.h>

#include<math.h>

int main()

{ int x;

double sonuc;

printf("Bir sayi giriniz:");

scanf("%d",&x);

sonuc=log(x);

printf("logaritma %d = %d",x,sonuc);

getchar();getchar();

return 0;

}

## 9.5 COS() FONKSİYONU

Kosinüsü hesaplatan fonksiyondur.

### ÖRNEK 9.5.1

#include<stdio.h>

#include<math.h>

int main()

{

int x;

double sonuc;

printf("Bir sayi giriniz:");

scanf("%d",&x);

sonuc=cos(x);

printf("Cosinus%d = %d",x,sonuc);

getchar();getchar();

return 0;

}

## 9.6 SİN() FONKSİYONU

Sinüste yine kosinüs fonksiyonu ile aynı şekilde kullanılır.

### ÖRNEK 9.6.1

#include<stdio.h>

#include<math.h>

int main()

{

int x;

double sonuc;

printf("Bir sayi giriniz:");

scanf("%d",&x);

sonuc=sin(x);

printf("Sinus%d = %d",x,sonuc);

getchar();getchar();

return 0;

}

## 9.7 CEİL() VE FLOOR() FONKSİYONLARI

Verdiğimiz bir ondalık sayıyı ceil bir üste floor ise bir aşağıya yuvarlar.Örneğin 3.4 girdiysek bu sayıyı ceil 4’e floor ise 3’e yuvarlar.

### ÖRNEK 9.7.1

#include<stdio.h>

#include<math.h>

int main()

{

double x;

printf("Bir ondalik sayi giriniz (ondalik olmasi icin araya Nokta koyunuz):");

scanf("%lf",&x);

printf("Floor icin = %f \n",floor(x));

printf("Ceil icin = %f",ceil(x));

getchar();getchar(); getchar();getchar();

return 0;

}

## 9.8 FABS() FONKSİYONU

Bu fonksiyon mutlak değer anlamına gelir ve negatif aldığı sayıyı pozitif pozitifleri ise yine pozitif döndürür. Burada önemli olan alınacak olan sayının double olmasıdır yoksa fonksiyon çalışmaz.

### ÖRNEK 9.8.1

#include<stdio.h>

#include<math.h>

int main()

{

double x;

printf("Bir sayi giriniz :");

scanf("%lf",&x);

printf("mutlak deger icin = %.2f ",fabs(x));

getchar();getchar();

return 0;

}

# 10.BÖLÜM STRUCTLAR

## 10.1 YENİ DEĞİŞKEN VERİ TİPİ OLUŞTURMA

Tam sayıları, karakterleri, virgüllü sayıları, stringleri gördük. Ancak kullanabileceğimiz değişken tipleri bunlarla sınırlı değildir. Kendi değişken tiplerimizi, yaratabiliriz. Örneğin boolean diye yeni bir tip yaratarak, bunun alabileceği değerleri true ve false olarak belirleyebiliriz; üçüncü bir ihtimal olmaz. Ya da mevsimler diye bir değişken tipi belirleyip, alabileceği değerleri aylar olarak kısıtlayabiliriz. İşte bu tarz işlemleri yapmak için enum kullanılır. enum kelimesi, enumerator yani 'sayıcı', 'numaralandırmacı'dan gelmektedir.

*boolean* diye tabir ettiğimiz değişken tipini oluşturalım:

### ÖRNEK 10.1

#include<stdio.h>

int main( void )

{

// Değişken tipinin nasıl olacaini tanimliyoruz

enum boolean {

false = 0,

true = 1

};

// Simdi de 'dogru\_mu' adinda bir degisken tanimliyoruz

enum boolean dogru\_mu;

// Tanimladigimiz 'dogru\_mu' degiskenine

// deger atayip, bir alt satirda da kontrol yapıyoruz.

dogru\_mu = true ;

if( dogru\_mu == true )

printf( "Dogru\n" );

else

printf("yanlis\n");

return 0;

}

Programınız derlenirken, karşılık girilmeyen değerlere sırayla değer atanmaktadır. İlla ki bizim bir eşitlik oluşturmamız gerekmez. Mesela üç ana rengi ( Kırmızı, Sarı ve Mavi )alabilecek ana\_renkler veri tipini oluşturalım:

### ÖRNEK 10.2

#include<stdio.h>

int main( void )

{

// Degisken tipinin nasil olacagini tanimliyoruz

enum ana\_renkler {

Kirmizi,

Mavi,

Sari

};

// Degiskeni tanimliyoruz.

enum ana\_renkler piksel;

// Degisken degerini Mavi olarak belirliyoruz.

// Dilersek Sari ve Kirmizi da girebiliriz.

piksel = Mavi;

// Degisken degeri karsilastiriliyor.

if( piksel == Kirmizi )

printf( "Kırmızı piksel\n" );

else if( piksel == Mavi )

printf( "Mavi piksel\n" );

else

printf( "Sarı piksel\n" );

return 0;

}

## 10.2 TYPEDEF KULLANIMI

Her seferinde “enum” yazmaktan “typedef” kurtaracaktır.

Typedef kullanımı aşağıdaki şekildedir.

**Kullanım**

typedef veri\_tipi\_eski\_adi veri\_tipi\_yeni\_adi

Kullanacağımız “typedef ” ile herhangi bir değişken tipini, bir başka isimle adlandırabiliriz. Örneğin “typedef int tam\_sayi;” komutuyla, değişken tanımlarken int yerine “tam\_sayi” da yazabiliriz. Bunun enum için kullanımına bakalım:

### ÖRNEK 10.2.1

#include<stdio.h>

int main( void )

{ // Yeni veri tipini olusturuyoruz

// Ayrica yeni veri tipinden, bir degisken tanimliyoruz.

enum boolean {

false = 0,

true = 1

};

// Alttaki komut sayesinde, boolean veri tipini tek adimda yaratabiliyoruz.

typedef enum boolean bool;

bool dogru\_mu;

dogru\_mu = true;

if( dogru\_mu == true )

printf( "Doğru\n" );

return 0;

}

## 10.3 YAPILAR(STRUCT’LAR)

Yapılar ( structures ); tam sayı, karakter vb. veri tiplerini gruplayıp, tek bir çatı altında toplar. Bu gruplandırma içinde aynı ya da farklı veri tipinden dilediğimiz sayıda eleman olabilir.



### ÖRNEK 10.3.1

#include<stdio.h>

int main( void )

{

struct {

int yil;

int ay;

int gun;

} dogum\_gunu;

printf( "Dogum gununuzu " );

printf( "GG-AA-YYYY olarak giriniz> ");

scanf( "%d-%d-%d", &dogum\_gunu.gun,&dogum\_gunu.ay,&dogum\_gunu.yil );

printf( "Dogum gununuz: " );

printf( "%d/%d/%d\n", dogum\_gunu.gun,dogum\_gunu.ay,dogum\_gunu.yil );

return 0;

}

Şimdi de aynı anda bizim ve iki kardeşimizin doğum gününü soran programı yazalım.

### ÖRNEK 10.3.2

#include<stdio.h>

int main( void )

{

struct {

int yil;

int ay;

int gun;

} siz, kiz\_kardes, erkek\_kardes;

printf( "Doğum gününüzü giriniz> ");

scanf( "%d-%d-%d", &siz.gun,&siz.ay,&siz.yil );

printf( "Kız kardeşiniz> " );

scanf( "%d-%d-%d", &kiz\_kardes.gun,&kiz\_kardes.ay,&kiz\_kardes.yil );

printf( "Erkek kardeşiniz> " );

scanf( "%d-%d-%d", &erkek\_kardes.gun,&erkek\_kardes.ay,&erkek\_kardes.yil );

printf("\n\n ----------------------------------------------------------------------------\n\n");

printf( "Doğum gününüz: ");

printf( "%d-%d-%d \n\n", siz.gun,siz.ay,siz.yil );

printf( "Kız kardeşinizin dogum günü> " );

printf( "%d-%d-%d\n\n", kiz\_kardes.gun,kiz\_kardes.ay,kiz\_kardes.yil );

printf( "Erkek kardeşiniz> " );

printf( "%d-%d-%d \n\n", erkek\_kardes.gun,erkek\_kardes.ay,erkek\_kardes.yil );

return 0;

}

## 10.4 İÇ İÇE YAPILAR

 Bir yapı içersine tıpkı bir değişken koyar gibi, bir başka yapı da koyulabilir. Örneğin kullanıcı bilgisi alan bir programda, isim, boy ve doğum tarihi bilgilerini aynı yapı altına toplayabiliriz. Ancak doğum tarihi bilgilerini daha alt bir yapı içersinde tutmak yararlı olabilir. Bunu koda dökersek şöyle olur:

### ÖRNEK 10.4.1

#include<stdio.h>

int main( void )

{

struct {

char isim[40];

int boy;

struct {

int yil;

int ay;

int gun;

} dogum\_bilgileri;

} kisi;

printf( "Adınız: " );

scanf( "%s", kisi.isim );

printf( "Boyunuz: " );

scanf( "%d", &kisi.boy );

printf( "Doğum tarihi: ");

scanf("%d-%d-%d", &kisi.dogum\_bilgileri.gun,&kisi.dogum\_bilgileri.ay,&kisi.dogum\_bilgileri.yil );

printf( "Girilen bilgiler:\n" );

printf( "İsim: %s\n", kisi.isim );

printf( "Boy: %d\n", kisi.boy );

printf("Doğumtarihi:%d/%d/%d\n",

kisi.dogum\_bilgileri.gun,kisi.dogum\_bilgileri.ay,kisi.dogum\_bilgileri.yil );

return 0;

}

## 10.5 YAPI DİZİLERİNE POİNTER İLE ERİŞİM

Dizilerden bahsettiğimiz bir yerde pointer'lardan bahsetmemek mümkün değil. Bir yapı dizisinin başlangıç adresini pointer'a atadığımız takdirde, elemanlara bu işaretçi üzerinde de ulaşabiliriz.

ÖRNEK 10.5.1

#include<stdio.h>

int main( void )

{

int i;

// Dogum tarihi tutmak icin

// 'dogum\_tarihi' adinda bir yapi

// olusturuyoruz

struct dogum\_tarihi {

int gun;

int ay;

int yil;

};

// Kisiye ait bilgileri tutmak

// icin 'sahis\_bilgileri' adinda

// bir yapi kuruluyor.

struct sahis\_bilgileri {

char isim[40];

int boy;

// Yapi icinde bir baska yapiyi

// kullanmak mumkundur. dogum\_tarihi

// yapisindan 'tarih' adinda bir

// degisken tanimlaniyor.

struct dogum\_tarihi tarih;

};

struct sahis\_bilgileri \*ptr;

// Dizi elemanlarina ilk deger atamasi yapiyoruz. Dilerseniz

// klavyeden deger girmeyi tercih edebilirsiniz.

struct sahis\_bilgileri kisi[3] = { "Ali", 170, { 17, 2, 1976 },

"Veli", 178, { 14, 4, 1980 },

"Cenk", 176, { 4, 11, 1983 } };

// Yapi dizisi yazdiriliyor:

for( i = 0, ptr = &kisi[0]; ptr <= &kisi[2]; ptr++, i++ ) {

printf( "Kayıt no.: %d\n", ( i + 1 ) );

printf( "Ad: %s\n", ptr->isim );

printf( "Boy: %d\n", ptr->boy );

printf( "Doğum Tarihi: %d/%d/%d\n\n", ptr->tarih.gun,

ptr->tarih.ay,

ptr->tarih.yil );

}

return 0;

}

Pointer'ın tanımlamasını yaparken, '*sahis\_bilgileri*' şablonundan türetilen değişkenlerin işaret edileceğini bildirmemiz gerekiyor. Yazmış olduğumuz "*struct sahis\_bilgileri \*ptr;*" kodu bundan kaynaklanmaktadır. for döngüsünde ise ,  *kisi* isimli yapı dizisinin ilk elemanının adresini, *ptr* işaretçisine atadık. Her seferinde de, *ptr* değeri bir adres bloğu kadar artmaktadır. Döngünün devamı, adresin son dizi elemanından küçük olmasına bağlıdır. Kullandığımız  **- >**  operatörüyse, pointer ile dizi elemanlarını göstermemizi sağlar.

## 10.6 YAPILAR VE FONKSİYONLAR

*enum* ile yarattığımız değişken tiplerini, fonksiyonlarda kullanmak için global olarak tanımlıyorduk. Yapıları, fonksiyonlarda kullanılmak için izlenecek yöntem aynıdır; yine global tanımlanması gerekir. Bir örnekle yapıların fonksiyonlarla kullanımını görelim:



### ÖRNEK 10.6.1

#include<stdio.h>

#include<string.h>

struct sahis\_bilgileri {

char isim[40];

int boy;

};

struct sahis\_bilgileri bilgileri\_al( void );

void bilgileri\_goster( struct sahis\_bilgileri );

int main( void )

{

struct sahis\_bilgileri kisi;

kisi = bilgileri\_al( );

bilgileri\_goster( kisi );

return 0;

}

struct sahis\_bilgileri bilgileri\_al( void )

{

struct sahis\_bilgileri sahis;

printf( "İsim> " );

gets( sahis.isim );

printf( "Boy> " );

scanf( "%d", &sahis.boy );

return sahis;

}

void bilgileri\_goster( struct sahis\_bilgileri sahis )

{

printf( "Ad: %s\n", sahis.isim );

printf( "Boy: %d\n", sahis.boy );

}

# 11.BÖLÜM DOSYALAMA İŞLEMLERİ

## 11.1 DOSYA TÜRLERİ

**1. Metin dosyaları**

Metin dosyaları normal .txt dosyalarıdır. Not Defteri gibi herhangi bir basit metin düzenleyiciyi kullanarak kolayca metin dosyaları oluşturabiliriz. Bu dosyaları açtığımızda, dosyadaki tüm içeriği düz metin olarak görürüz. İçeriği kolayca düzenleyebilir veya silebiliriz. Bakım yapmak için az çaba harcarlar, kolayca okunurlar ve en az güvenlik sağlarlar ve daha büyük depolama alanı alırlar.

**2. Binary dosyalar**

Binary dosyalar çoğunlukla bilgisayarımızdaki .bin dosyalarıdır. Verileri düz metin olarak saklamak yerine, ikili biçimde (0 ve 1’ler) depolarlar. Daha yüksek miktarda veri tutabilirler, kolayca okunamazlar ve metin dosyalarından daha iyi güvenlik sağlarlar.

## 11.2 DOSYA İŞLEMLERİ

C’de, dosyalar üzerinde dört ana işlem gerçekleştirebiliriz:

* Yeni bir dosya oluşturma.
* Mevcut bir doyayı açmak.
* Bir dosyayı kapatmak.
* Bir dosyadan bilgi okuma ve yazma.

**Dosyalarla çalışmak**

Dosyalarla çalışırken, bir tür dosya göstericisini bildirmemiz gerekir. Bu bildirim dosya ile program arasındaki iletişim için gereklidir.

FILE \*fptr;

## 11.3 BİR DOSYAYI AÇMA-OLUŞTURMA VE DÜZENLEME

Bir dosyayı açmak stdio.h başlık dosyasında tanımlanan fopen() fonksiyonu kullanılarak gerçekleştirilir.

Bir dosyayı standart G / Ç’de açmak için kullanılan sözdizimi şöyledir:

ptr = fopen("fileopen","mode");

**Örneğin :**

fopen("E:\\cprogram\\newprogram.txt","w");

fopen("E:\\cprogram\\oldprogram.bin","rb");

Yazma modu, dosyanın içeriğini oluşturmamıza ve düzenlememize (üzerine yazmanıza) izin verir.

Okuma modu sadece dosyayı okumamıza izin verir, dosyaya yazamayız.

## 11.4 STANDART GİRİŞ / ÇIKIŞLARDA AÇMA MODLARI

|  |  |  |
| --- | --- | --- |
| MOD | MOD’UN ANLAMI | DOYANIN DURUMU |
| r | Okumak için açar. | Dosya yoksa fopen (), NULL değerini döndürür. |
| rb | İkili modda okumak için açar. | Dosya yoksa fopen (), NULL değerini döndürür. |
| w | Yazmak için açar | Dosya varsa, içeriğinin üzerine yazılır. Dosya yoksa, oluşturulur. |
| wb | İkili modda yazmak için açar. | Dosya varsa, içeriğinin üzerine yazılır. Dosya yoksa, oluşturulur. |
| a | Eklemek için açar. Veriler dosyanın sonuna eklenir. | Dosya yoksa oluşturulur. |
| ab | İkili modda eklemek için açar. Veriler dosyanın sonuna eklenir. | Dosya yoksa oluşturulur. |
| r+ | Hem okuma hem de yazma için açar. | Dosya yoksa fopen (), NULL değerini döndürür. |
| rb+ | + Hem okuma hem de yazma için ikili modda açar. | Dosya yoksa fopen (), NULL değerini döndürür. |
| w+ | Hem okuma hem de yazma için açar. | Dosya varsa, içeriğinin üzerine yazılır. Dosya yoksa, oluşturulur. |
| wb+ | + Hem okuma hem de yazma için ikili modda açar. | Dosya varsa, içeriğinin üzerine yazılır. Dosya yoksa, oluşturulur. |
| a+ | Hem okumak hem de eklemek için açar. | Dosya yoksa, oluşturulur. |
| ab+ | Hem okumak hem de eklemek için ikili modda açın. | Dosya yoksa, oluşturulur. |

## 11.5 BİR DOSYAYI KAPATMA

Dosya (hem metin hem de ikili) okuma / yazma işleminden sonra kapatılmalıdır. Bir dosyayı kapatmak, fclose() işlevi kullanılarak gerçekleştirilir.

fclose(fptr);

Burada, fptr, kapatılacak dosyayla ilişkili bir dosya işaretçisidir.

**Metin dosyasına okuma ve yazma**

Bir metin dosyasına okumak ve yazmak için, fprintf () ve fscanf () işlevlerini kullanırız.

Bunlar sadece printf () ve scanf () dosya sürümleridir. Tek fark, fprint () ve fscanf () öğesinin FILE yapısında bir işaretçi beklemesidir.

### ÖRNEK 11.5.1

Bir metin dosyasına yazma:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int num;

FILE \*fptr;

fptr = fopen("D:\\program.txt","w");

if(fptr == NULL)

{

printf("Hata!");

exit(1);

}

printf("Enter num: ");

scanf("%d",&num);

fprintf(fptr,"%d",num);

fclose(fptr);

return 0;

}

Bu program kullanıcıdan bir numara alır ve program.txt dosyasında saklar.

### ÖRNEK 11.5.2

Bir metin dosyasından okumak:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int num;

FILE \*fptr;

if ((fptr = fopen("D:\\program.txt","r")) == NULL){

printf("Error! opening file");

exit(1);

}

fscanf(fptr,"%d", &num);

printf("Value of n=%d", num);

fclose(fptr);

return 0;

}

Bu program program.txt dosyasında bulunan tam sayıyı okur ve ekrana yazdırır.

**Fgetchar ()**, **fputc ()** vs. gibi diğer fonksiyonlar da benzer şekilde kullanılabilir.

## 11.6 İKİLİ BİR DOSYAYA OKUMA VE YAZMA

Fread () ve fwrite (), ikili dosyalar biçiminde sırasıyla diskteki bir dosyadan okumak ve yazmak için kullanılır.

**İkili dosyaya yazma**

İkili bir dosyaya yazmak için fwrite () işlevini kullanmanız gerekir. İşlevler dört argüman alır:

1. **Diske yazılacak verinin adresi**
2. **Diske yazılacak verilerin boyutu**
3. **Veri sayısı**
4. **Yazmak istediğiniz dosyadaki işaretçi**

fwrite(addressData, sizeData, numbersData, pointerToFile);

### ÖRNEK 11.6.1

fwrite () kullanarak ikili bir dosyaya yazmak :

#include <stdio.h>

#include <stdlib.h>

struct threeNum

{

int n1, n2, n3;

};

int main()

{

int n;

struct threeNum num;

FILE \*fptr;

if ((fptr = fopen("D:\\program.bin","wb")) == NULL){

printf("Error! opening file");

exit(1);

}

for(n = 1; n < 5; ++n)

{

num.n1 = n;

num.n2 = 5\*n;

num.n3 = 5\*n + 1;

fwrite(&num, sizeof(struct threeNum), 1, fptr);

}

fclose(fptr);

return 0;

}

Bu programda, C sürücüsünde program.bin adlı yeni bir dosya oluşturuyoruz. n1, n2 ve n3 olmak üzere üç sayı içeren bir threeNum yapısını ilan ediyoruz ve bunu ana fonksiyonda num olarak tanımlıyoruz. Şimdi, for döngüsü içinde, değeri fwrite () kullanarak dosyaya kaydederiz.

İlk parametre num adresini alır ve ikinci parametre üç sayı yapısının boyutunu alır. Yalnızca bir num örneği eklediğimiz için, üçüncü parametre 1’dir. Ve, son parametre \* fptr verileri sakladığımız dosyayı gösterir. ve dosyayı kapattık.

## 11.7 İKİLİ DOSYADAN OKUMA

**Fread ()** işlevi, yukarıdaki **fwrite ()** işlevine benzer 4 argüman alır.

fread(addressData, sizeData, numbersData, pointerToFile);

### ÖRNEK 11.7.1

**fread () kullanarak ikili dosyadan okuma:**

#include <stdio.h>

#include <stdlib.h>

struct threeNum

{

int n1, n2, n3;

};

int main()

{

int n;

struct threeNum num;

FILE \*fptr;

if ((fptr = fopen("D:\\program.bin","rb")) == NULL){

printf("Error! opening file");

exit(1);

}

for(n = 1; n < 5; ++n)

{

fread(&num, sizeof(struct threeNum), 1, fptr);

printf("\n n1: %d\t\n n2: %d\t\n n3: %d\n", num.n1, num.n2,

num.n3); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

fclose(fptr);

return 0; }

Bu programda, program.bin dosyasını okuruz ve kayıtları birer birer döngüden geçiririz Basit bir ifadeyle, \* fptr ile gösterilen dosyadan üç sayı büyüklüğünde bir üç sayı kaydederiz. Örnek 3’e eklediğimiz kayıtları alırız.

**fseek () kullanarak veri alma**

Bir dosyanın içinde çok sayıda kayda sahipsek ve belirli bir pozisyondaki bir kayda erişmemiz gerekiyorsa, kaydı almak için önce tüm kayıtlardan geçmemiz gerekir. Bu, çok fazla bellek ve çalışma zamanını boşa harcar. Gerekli verileri elde etmenin daha kolay bir yolu fseek () kullanılarak elde edilebilir. Adından da anlaşılacağı gibi, fseek () imleci dosyadaki belirtilen kayda göre arar.

fseek(FILE \* stream, long int offset, int whence);

İlk parametre dosyanın işaretçisidir. İkinci parametre, bulunacak kaydın konumudur ve üçüncü parametre, ofsetin başladığı konumu belirtir.

### ÖRNEK 11.7.2

**fseek() kullanarak ikili dosyadan okuma:**

#include <stdio.h>

#include <stdlib.h>

struct threeNum

{

int n1, n2, n3;

};

int main()

{

int n;

struct threeNum num;

FILE \*fptr;

if ((fptr = fopen("D:\\program.bin","rb")) == NULL){

printf("Error! opening file");

exit(1);

}

fseek(fptr, -sizeof(struct threeNum), SEEK\_END);

for(n = 1; n < 5; ++n)

{

fread(&num, sizeof(struct threeNum), 1, fptr);

printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2,

num.n3);

fseek(fptr, -2\*sizeof(struct threeNum), SEEK\_CUR);

}

fclose(fptr);

return 0;

}

Bu program, program.bin dosyasındaki kayıtları ters sırayla (en sondan önceye) okumaya başlayacak ve yazdıracaktır.

KAYNAKÇA

1. <http://burakisci.com>
2. <http://cdersleri.com>
3. <http://www.cagataycebi.com>
4. <https://slideplayer.biz.tr>
5. <https://sanalkurs.net>
6. KODLAB C++ KİTABI (Muhammed MASTAR-Süha ERİŞ)
7. PROGLAMLAMAYA GİRİŞ VE ALGORİTMALAR KİTABI (Doç. Dr. Soner ÇELİKKOL)
8. ALGORİTMA GELİŞTİRME VE PROGRAMLAMAYA GİRİŞ KİTABI (Doç. Dr. Fahri VATANSEVER)