

# BLM1011

# BİLGİSAYAR BİLİMLERİNE GİRİŞ

---

GR.2

2023-2024 GÜZ YARIYILI

DR.ÖĞR.ÜYESİ GÖKSEL BİRİCİK

# C Programlama Diline

## Giriş

---

# Uygulama (Program) Geliřtirme

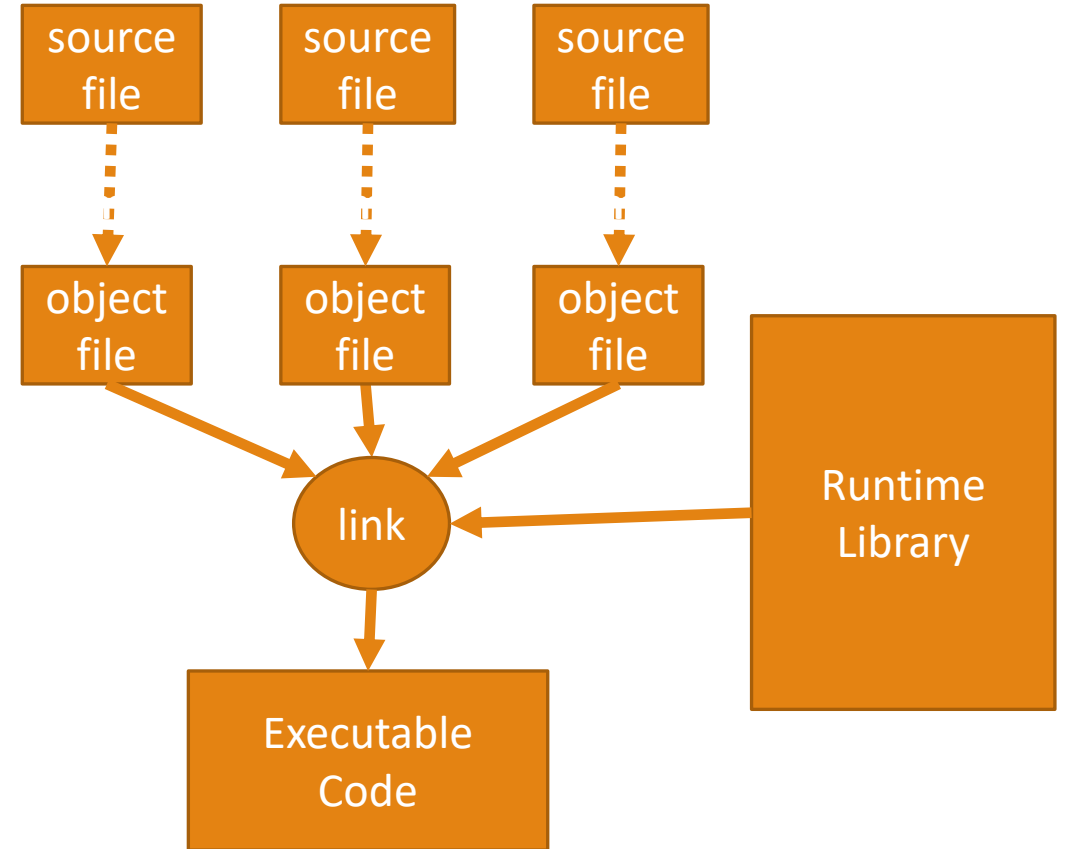
Derleyicinin (Compiler) görevi, kaynak kodunu makine koduna çevirmektir.

Derleyicinin girdisi kaynak kodudur (source code) ve çıktısı ara nesne kodudur (object code).

Bağlayıcı (linker) ise farklı ara nesne kodlarını bir araya getirir ve tek bir dosyada «bağlar»

Bağlayıcı aynı zamanda ihtiyaç duyulan işlevleri çalışma zamanı kütüphanesinden alarak bağlar.

Bağlama (genellikle) otomatik olarak ele alınır.



# Uygulama (Program) Geliştirme

---

C programlama dilinin çok küçük olmasının nedenlerinden biri, pek çok işlemi geniş ve çeşitli çalışma zamanı kütüphanelerine (runtime library) bırakmasıdır.

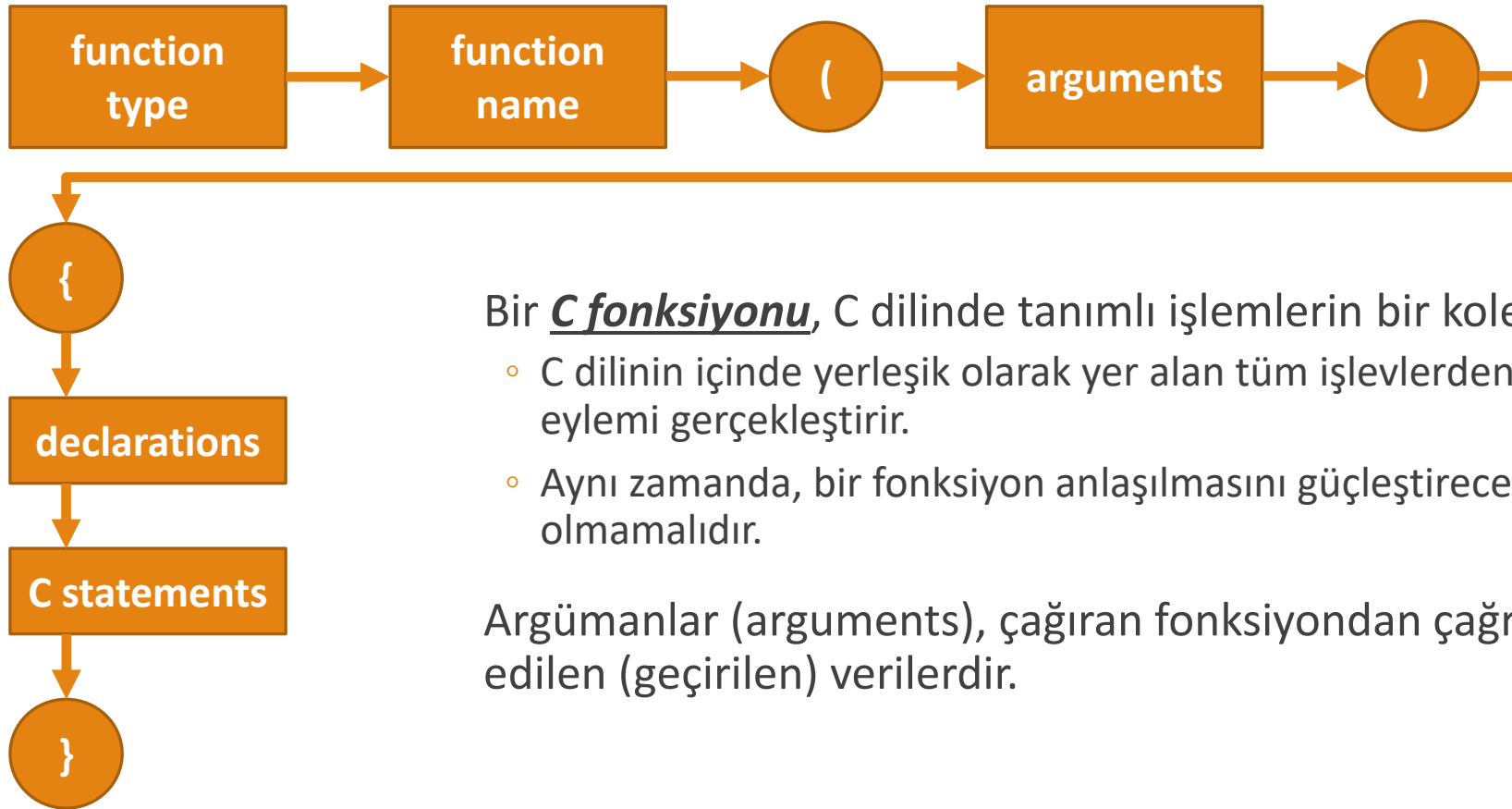
Çalışma zamanı kütüphanesi, bir ara nesne dosyaları koleksiyonudur.

- Her bir dosyada, çok çeşitli servislerin birinin sunulmasını sağlayan bir işlevin makine komutları yer alır.
- Fonksiyonlar, amaçlarına göre (giriş/çıkış, bellek yönetimi, matematiksel işlevler, string işlemleri vb,) gruplanmışlardır.
- Her bir grup için, «başlık dosyası» (header file) denilen bir kaynak dosyası vardır ve içinde sunulan fonksiyonları kullanabilmeniz için gerekli tanım/bilgiler yer alır.
  - Gelenek olarak, bu dosyalar .h uzantısına sahiptir.

Örneğin, giriş/çıkış çalışma zamanı işlemlerinden birinin adı **printf()**'tir ve uç nokta ekranınızda (terminal) veri gösterebilmenize olanak sağlar. Bu fonksiyonu kullanabilmek için aşağıdaki satırı kaynak kod dosyanıza ekleyerek ilgili kütüphaneyi bağlamak üzere çağırmanız gereklidir.

- `#include <stdio.h>`

# Temel C



Bir **C fonksiyonu**, C dilinde tanımlı işlemlerin bir koleksiyonudur.

- C dilinin içinde yerleşik olarak yer alan tüm işlevlerden daha karmaşık bir eylemi gerçekleştirir.
- Aynı zamanda, bir fonksiyon anlaşılmasını güçleştirecek kadar da karmaşık olmamalıdır.

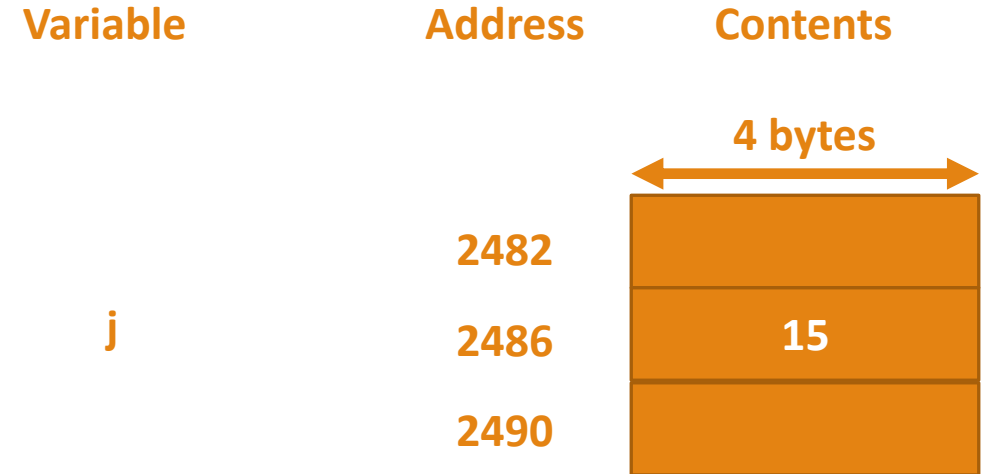
Argümanlar (arguments), çağıran fonksiyondan çağrılan fonksiyona pas edilen (geçirilen) verilerdir.

# Değişkenler ve Sabitler

**İfade:**  $j = 5 + 10$

**Sabit:** hiçbir zaman değişmeyen bir değerdir

**Değişken:** Bilgisayarın belleğindeki bir konum, ya da bir adresi gösterebilmesi ile değişkenliğini kazanır.



# İsimler

---

C programlama dilinde, neredeyse her şeyi isimlendirebilirsiniz.

- Değişkenler, sabitler, fonksiyonlar, hatta programdaki belirli bir konum.

İsimler;

- Harfleri (büyük-küçük), rakamları, alt çizgiyi ( \_ ) içerebilir.
- **Her zaman ya harf ya da alt çizgi ile başlamalıdır !**

C programlama dili büyük-küçük harf duyarlıdır. Büyük ve küçük harflerle yazılan aynı isimler farklı değerlendirilir.

- VaR, var, VAR hepsi farklıdır.

İsimler ayrılmış kelimelerle **aynı olamaz.**

# İsimler

---

## Geçerli İsimler:

- j
- j5
- \_\_system\_name
- sesquipedalian\_name
- UpPeR\_aNd\_LoWeR\_cAsE\_nAmE

## Geçersiz İsimler:

- 5j
- \$name
- int
- bad%#\*@name



# İsimler – Ayrılmış Kelimeler

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile

# İfadeler (Expressions)

---

Operatörler, sayılar ve isimlerin, bir değerin hesaplanmasını sağlayan kombinasyonuna **ifade** adı verilir.

Örnek:

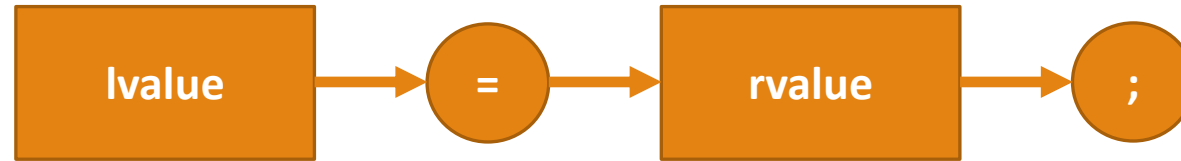
5	Sabit değer
j	Değişken değer
5+j	Bir sabit artı bir değişken
f()	Fonksiyon çağırısı
f() / 4	Sonucu bir sabite bölünen bir fonksiyon çağırısı

# Atama İfadeleri (Assignment Statements)

---

Atamanın solundaki değer (***lvalue***), bellekte değer tutabilecek bir adresi ifade etmelidir.

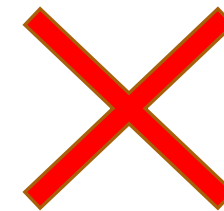
Atama operatörünün sağındaki ifade, sağ değer (***rvalue***) olarak tanımlanır.



answer = num \* num;



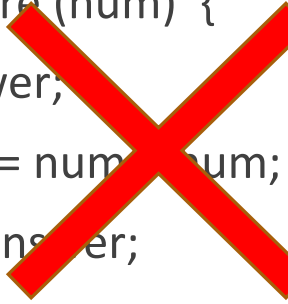
num \* num = answer;



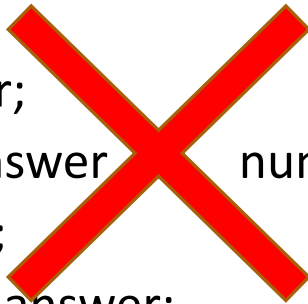
# Kaynak Kodunu Biçimlendirmek

---


```
int square (num) {  
    int answer;  
    answer = num * num;  
    return answer;  
}
```



```
int square (num)  
{ int  
  answer;  
  answer * num  
  return answer;  
}
```



```
int square (num) {  
    int answer;  
    answer = num * num;  
    return answer;  
}
```



# Yorum Satırları (Comments)

---

Yorum, kaynak kod dosyanızda yazdığınız kodun ne yaptığını açıklamak için yazılan metinlerdir.

Yorumlar insanlar içindir. Derleyici tüm yorumları görmezden gelir.

C programlama dilinde yorumlar `/*` ile başlayıp `*/` ile biter.

İççe yorumlar desteklenmez !

Neleri yorumlamalıyız?

- Fonksiyon başlıkları
- Kodda yapılan değişiklikler

```
/* square()
```

```
* Author : P. Margolis
```

```
* Initial coding : 3/87
```

```
* Params : an integer
```

```
* Returns : square of its parameter
```

```
*/
```

# Ana fonksiyon (The main() function)

---

```
int main ( ) {  
    extern int square();  
    int solution;  
    solution = square(5);  
    exit(0);  
}
```

**exit()** fonksiyonu, programın çalışmasının sonlanmasını ve kontrolün işletim sistemine döndürülmesini sağlayan bir çalışma zamanı kütüphanesi fonksiyonudur.

exit() fonksiyonunun değeri 0 ise, programın normal bir şekilde hatasız olarak sonlandığı anlaşılır.

Sıfırdan farklı değerler, programın normal dışı sonlanması anlamını taşır.

main() fonksiyondan exit() fonksiyonunu çağırmak ile return ifadesini işletmek tamamen aynı etkiyi yaratır.

# printf() ve scanf() fonksiyonları

---

printf() fonksiyonu herhangi sayıda argüman alabilir.

İlk argümana biçimlendirme karakter dizgesi (**format string**) adı verilir. Çift tırnak içinde yazılır ve hem metin, hem de biçimlendirme belirteçlerini içerebilir.

scanf() fonksiyonu printf'in tersi işi yapar. Terminale veri yazmak yerine, klavyeden girilen verileri okumak için kullanılır.

İlk argümanı yine format string'dir.

```
int main ( ) {
```

```
    int num;
```

```
    scanf("%d", &num);
```

```
    printf("number : %d\n", num);
```

```
    return 0;
```

```
}
```

# Önişlemci (Preprocessor)

---

Önişlemci, program derlendiğinde otomatik olarak çalışır.

Tüm önişlemci komutları diyez (#) işareti ile başlar. #, satırdaki boşluk harici ilk karakter olmalıdır.

C ifadeleri noktalı virgül ( ; ) ile sonlanır. Önişlemci komutları ise satırbaşı ile sonlanır, ( ; ) **noktalı virgül kullanılmaz.**

Önişlemci ile

- Makro işleme
- Koşullu derleme
- Hazır bulunan (built-in) makrolar ile hata ayıklama

işlemleri de yapılabilir.



# Önişlemci (Preprocessor)

---

**define** özelliği:

Sabit bir değere isim ataması yapmak için kullanılır

**#define NOTHING 0**

Sabitler için büyük harfleri kullanmak bir gelenektir.

Sabitleri isimlendirmenin iki faydası vardır:

- Anlam ifade etmeyen sayısal değer yerine anlamlı bir ifade kullanmış oluruz.
- Programda değişiklik yapmayı kolaylaştırır.

Tanımlarınızı **DEĞİŞKEN OLARAK KULLANAMAZSINIZ.**

**NOTHING = j + 5;    X**

# Önişlemci (Preprocessor)

---

**include** özelliği:

#include yönlendirmesi, derleyicinin o anda derlemekte olduğu dosyadan farklı bir kaynaktan içeriğin de okunmasını sağlar.

İki formu vardır:

- **#include <filename>**
- Önişlemci, işletim sistemi tarafından belirlenen özel bir yere bakar. Burası, tüm sistem çalışma zamanı kütüphanelerinin durduğu konumdur.
- **#include "filename"**
- Önişlemci, kütüphaneyi kaynak kodunun bulunduğu klasörden yükler. Bulamazsa, sistem öntanımlı alana da bakar.

# Hello, World!

---

```
#include <stdio.h>
```

Standart giriş çıkı kütüphanesini ekle

```
int main ( void ) {
```

Programımızın başlangıç noktası

```
    printf("Hello, World!\n");
```

Ekrana ifadeyi yazdır

```
    return 0;
```

Çağırın programa 0 değerini döndür

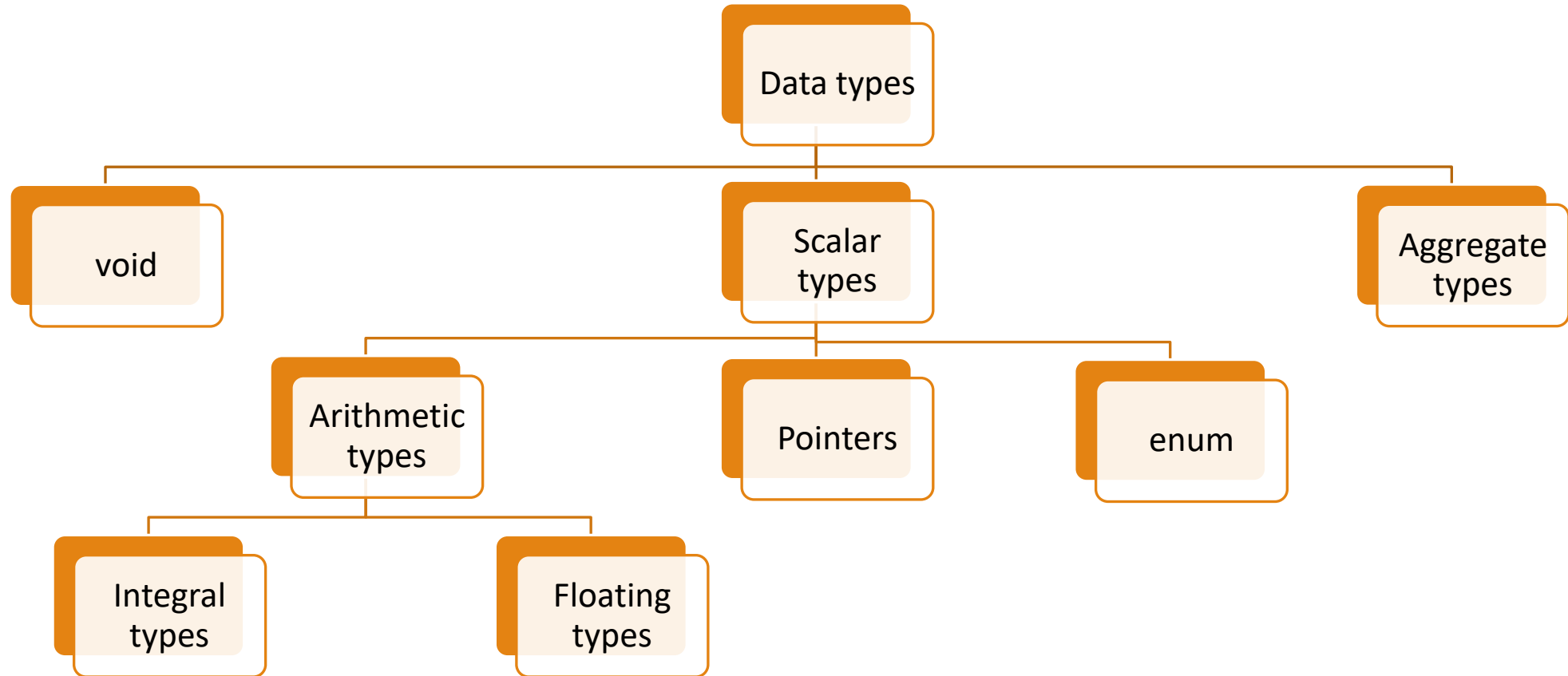
- 0: başarı 😊

```
}
```

Programımızın sonu

# Veri Tipleri

---



# Veri Tipleri

---

Skalar veri tipleri için 9 ayrılmış kelime vardır.

Basit tipler:

- char, int, float, double, enum

Nitelendiriciler: (Qualifiers)

- long, short, signed, unsigned

Tamsayı j değişkenini tanımlamak:

- int j;

Aynı tipte birden fazla değişkeni tanımlamak:

- int i, j, k;

**Tüm değişken tanımlamaları, herhangi bir işletim ifadesinden önce yapılmalıdır! (ANSI)**

# Çeşitli Tamsayı Tipleri

ANSI standardını tanımladığı tek gereksinim, bir byte'ın **en az 8 bit uzunlukta** olmasıdır.

Bu durumda int **en az 16 bit olmalıdır** ve bilgisayarın «**doğal**» boyutlandırmasını taşımalıdır.

- Doğal, CPU'nun tek bir komutta işleyebildiği bit sayısını ifade eder.

Type	Size (in bytes)	Value Range
int	4	$-2^{31}$ to $2^{31} - 1$
short int	2	$-2^{15}$ to $2^{15} - 1$
long int	4	$-2^{31}$ to $2^{31} - 1$
unsigned short int	2	0 to $2^{16} - 1$
unsigned long int	4	0 to $2^{32} - 1$
signed char	1	$-2^7$ to $2^7 - 1$
unsigned char	1	0 to $2^8 - 1$

# Çeşitli Tamsayı Tipleri

Tamsayılar:

- Onluk (decimal)
- Sekizlik (octal)
- Onaltılık (hexadecimal)
- İkilik (binary)

olabilir. u, U, l, L soneklerini alabilirler.

85     /\* decimal \*/

0213   /\* octal \*/

0x4b   /\* hexadecimal \*/

30     /\* int \*/

30u    /\* unsigned int \*/

30l    /\* long \*/

30ul   /\* unsigned long \*/

Decimal	Octal	Hexadecimal
3	003	0x3
8	010	0x8
15	017	0xF
16	020	0x10
21	025	0x15
-87	-0127	-0x57
255	0377	0xFF

# Kesirli Sayı Tipleri (Floating Point Types)

---

Kesirli sayıları tutabilecek bir değişken tanımı için:

- **float**
- **double**

Double, iki kat hassasiyet anlamını taşır.

- float'tan yaklaşık iki kat yüksek hassasiyeti temsil edebilir.
- **float** genellikle **4 byte**, **double** ise **8 byte** yer kaplar.

Sınırları için <limits.h> incelenebilir.

**Ondalık noktalı:**

0.356

5.0

0.000001

.7

7.

**Bilimsel notasyon:**

3e2

5E-5



# İlk Değer Atamaları

---

Değişkeni tanımlamak, bellekte değişken için yeterli alanın ayrılmasını sağlar.

Bu alanda herhangi bir içerik / ilk değer saklanmasını sağlamaz.

**Bu değişkene bir atama yapmadan değerini okursanız, tahmin edilemeyecek sonuçlarla karşılaşabilirsiniz.**

Değişkene ilk değer ataması için tanımdan hemen sonra değer ataması yapılabilir.

```
char ch = 'A';
```

Ya da;

```
char ch;
```

```
ch = 'A';
```

# Tipleri Bir Arada Kullanmak

---

Gizli dönüşüm (Implicit conversion)

İşaretli ve işaretsiz tiplerin bir arada kullanımı

Tamsayı ve kayan noktalı sayıların bir arada kullanımı

Açık dönüşüm (Explicit conversion)

**long double**

**double**

**float**

**unsigned long int**

**long int**

**unsigned**

**int**

# Gizli dönüşüm (Implicit conversion)

---

Derleyici bir **ifade** ile karşılaştığında, onu **alt ifadelere** böler.

Her alt ifade bir **operatör** ve operatöre **bağlı** bir veya daha fazla nesneden **-operand-** oluşur.

**Örnek:**  $-3 / 4 + 2.5$  ifadesinde  $-$ ,  $/$ ,  $+$  operatörleri bulunmaktadır.

Her operatörün operand tipi anlaşması için farklı kuralları vardır. Genellikle ikili operatörler her iki operandın da aynı tipte olmasını bekler.

- Tipler farklı ise, derleyici bir operandı diğeriyle anlayacak tipe dönüştürür.
- Bu dönüşüm için, önceki slayttaki hiyerarşiye başvurur.

**Örnek:**  $1 + 2.5$  ifadesinde bir ***int***, bir ***float*** operand mevcut. *int*, *float*'a yükseltilerek işlem gerçekleşir.

# İşaretli ve işaretsiz tiplerin bir arada kullanımı

---

İşaretli ve işaretsiz tamsayı tipleri arasındaki tek fark yorumlanma şekilleridir.

- İkisi de bellekte aynı miktarda yer kaplar.

11101010

- 2's complement: -22
- unsigned char: 234

10u - 15 = ?

- -5
- 4,294,967,291

# Tamsayı ve kayan noktalı sayıların bir arada kullanımı

---

## Görünmez dönüşümler:

`int i;`

`float f;`

`i + f;    // i float'a dönüşerek işlenir`

`i + f + 2.5    // hem i hem f double'a dönüşerek işlenir`

## Hassasiyet kaybı (loss of precision):

`i = 2.5;    // i'nin değeri 2 olur.`

`i = 999999999999.888888    // overflow`

# Açık dönüşüm (Explicit conversion) – Cast

---

Açık dönüşüme –casting- denir ve bir cast yapısı çağrılarak oluşturulur.

```
int j=2, k=3;
```

```
float f;
```

```
f = k / j;    // f=0.0
```

```
f = (float) k / j;    // f=0.6666...
```

Bir ifadeyi cast etmek için, hedef veri tipini parantez içinde ifadenin önüne ekleriz.

# Enumeration Veri Tipi

---

Enumeration tipi ile bir değişken tanımlayıp bu değişkenin alabileceği sabit isim değerlerini belirleyebilirsiniz.

Varsayılan değerler 0'dan başlayıp her enumerasyon ad eklemesinde bir artarak devam eder.

Varsayılan değerleri, başka değerler atayarak ezebilirsiniz.

```
enum COLOR { red, blue, green, yellow } color;  
enum INTENSITY { bright, medium, dark } intensity;
```

```
color = yellow;           // OK
```

```
color = bright;           // tip çatışması
```

```
intensity = bright;       // OK
```

```
intensity = blue;         // tip çatışması
```

```
color = 1;                 // tip çatışması
```

```
color = green + blue;     // yanıltıcı kullanım
```

# void Veri Tipi

---

**void** veri tipinin iki önemli amacı vardır.

Birincisi, bir fonksiyonun herhangi bir değer almadığını veya döndürmediğini belirtir.

```
void func(int a, int b);
```

```
int rand(void);
```

İkincisi, jenerik işaretçi (generic pointer) tanımlamamızı sağlar

- İkinci dönemde.



# typedef

---

**typedef** anahtar kelimesi ile veri tiplerine kendi belirlediğiniz isimleri verebilirsiniz.

Anlamsal olarak, değişken ismi veri tipi için eş anlamlı hale gelir.

Geleneksel olarak, typedef tanımları büyük harflerle yazılır.

```
typedef long int INT32;
```

```
long int j;
```

```
INT32 j;
```

# İfadeler (Expressions)

---

## Sabit ifadeler

- 5
- $5 + 6 * 13 / 3.0$

## Tamsayı ifadeler (int j,k)

- j
- $j / k * 3$
- $k - 'a'$
- $3 + (\text{int}) 5.0$

## Kesirli sayı ifadeler (double x,y)

- $x / y * 5$
- $3 + (\text{float}) 4$

## İşaretçi ifadeler (int \* p)

- p
- $p + 1$
- "abc"

# Öncelik ve Birleşirlik (Precedence & Associativity)

---

Tüm operatörlerin **öncelik** ve **birleşirlik** özellikleri vardır.

- Her ikisi de operandların operatöre nasıl bağlandığını etkiler.


Görülme sırasından bağımsız olarak, daha yüksek önceliğe sahip operatörlerin operandları, daha düşük önceliğe sahip operatörlerden önce bağlanarak işlem görür.

Aynı önceliğe sahip operatörlerde, birleşirlik (associativity, bazen de binding) ile operatörler ile operandların gruplanması belirlenir.


$$2 + 3 * 4$$

$$3 * 4 + 2$$

# Öncelik ve Birleşirlik (Precedence & Associativity)

Operatör Sınıfı	Sınıftaki operatörler	Associativity	Precedence
primary	() [] -> .	Soldan sağa	<div>En Yüksek</div> 
unary	cast operator sizeof & (address of) * (dereference) - + ~ ++ -- !	Sağdan sola	
multiplicative	* / %	Soldan sağa	
additive	+ -	Soldan sağa	
shift	<< >>	Soldan sağa	
relational	< <= > >=	Soldan sağa	
equality	== !=	Soldan sağa	

# Öncelik ve Birleşirlik (Precedence & Associativity)

Operatör Sınıfı	Sınıftaki operatörler	Associativity	Precedence
bitwise AND	&	Soldan sağa	 EN DÜŞÜK
bitwise exclusive OR	^	Soldan sağa	
bitwise inclusive OR		Soldan sağa	
logical AND	&&	Soldan sağa	
logical OR		Soldan sağa	
conditional	? :	Sağdan sola	
assignment	= += -= *= /= %= >>= <<= &= ^=	Sağdan sola	
comma	,	Soldan sağa	

# Parantezler

---

Derleyici parantez içindeki ifadeleri önce gruplar. Özellikle belirli bir önceliklendirme yapmak için parantezleri kullanabiliriz.

$$(2 - 3) * 4$$

$$2 - (3 * 4)$$

En içteki parantez en öncelikli olarak ele alınır. Örnekte  $(3+1)$  ve  $(8-4)$  aynı derinliğe sahiptir, bu nedenle istenen sırada işlenebilirler.

$$1 + ((3+1) / (8 - 4) - 5)$$

$$1 + (4 / (8 - 4) - 5)$$

$$1 + (4 / 4 - 5)$$

$$1 + (1 - 5)$$

$$1 + -4$$

$$-3$$

# İkili (Binary) Aritmetik Operatörler

Operatör	Sembol	Form	İşlem
Çarpma	*	$x * y$	x çarpı y
Bölme	/	$x / y$	x bölü y
Kalan	%	$x \% y$	x'in y'ye bölümünden kalan
Toplama	+	$x + y$	x artı y
Çıkarma	-	$x - y$	x eksi y

# Kalan Operatörü

---

Diğer aritmetik operatörler hem tamsayı hem de kesirli sayı tipinde operand alırken, kalan operatörü **sadece tamsayı** operand kabul eder!

Eğer operandlardan biri negatif ise, kalan negatif ya da pozitif olabilir, gerçekleştirmeye göre değişebilir.

ANSI standardı, kalan ve bölme operatörleri arasında aşağıdaki ilişkinin bulunmasını gerektirir.

Tüm a ve b tamsayı değerleri için,

$$a = a \% b + (a / b) * b$$



# Aritmetik Atama Operatörleri

Operatör	Sembol	Form	İşlem
Atama	=	$a = b$	$b$ değerini $a'$ 'ya koy
Toplayarak atama	+=	$a += b$	$a+b$ değerini $a'$ 'ya koy
Çıkararak atama	-=	$a -= b$	$a-b$ değerini $a'$ 'ya koy
Çarparak atama	*=	$a *= b$	$a*b$ değerini $a'$ 'ya koy
Bölerek atama	/=	$a /= b$	$a/b$ değerini $a'$ 'ya koy
Kalanı atama	%=	$a \% = b$	$a\%b$ değerini $a'$ 'ya koy

# Aritmetik Atama Operatörleri

---

**int m = 3, n = 4;**

**float x = 2.5, y = 1.0;**

**m += n + x - y**

**m = (m + ((n+x) - y)) // 8**

**m /= x \* n + y**

**m = (m / ((x\*n) + y)) // 0**

**n %= y + m**

**n = (n % (y + m)) // hata**

**x += y -= m**

**x = (x + (y = (y - m))) // 0.5**

# Arttırma / Azaltma Operatörleri

Operatör	Sembol	Form	İşlem
Sonra artım	<b>++</b>	<b>a++</b>	a değerini al, sonra 1 arttır
Sonra azaltım	<b>--</b>	<b>a--</b>	a değerini al, sonra 1 azalt
Önce artım	<b>++</b>	<b>++a</b>	a'yı 1 arttır, sonra değerini al
Önce azaltım	<b>--</b>	<b>--b</b>	a'yı 1 azalt, sonra değerini al

# Arttırma / Azaltma Operatörleri

---

```
main () {  
    int j=5, k=5;  
    printf("j: %d\t k : %d\n", j++, k--);  
    printf("j: %d\t k : %d\n", j, k);  
    return 0;  
}
```

j: 5      k : 5

j: 6      k : 4

```
main () {  
    int j=5, k=5;  
    printf("j: %d\t k : %d\n", ++j, --k);  
    printf("j: %d\t k : %d\n", j, k);  
    return 0;  
}
```

j: 6      k : 4

j: 6      k : 4

# Arttırma / Azaltma Operatörleri

---

`int j = 0, m = 1, n = -1;`

`m++ --j`

$(m++) - (--j)$  (2)

`m += ++j * 2`

$m = ( m + ((++j) * 2 )$  (3)

`m++ * m++`

$(m++) * (m++)$  (gerçeklemeye bağlı olarak değişir)

# Virgül Operatörü

---

Tek ifadeye izin verilen bir yerde iki veya daha fazla ayrık ifadenin çalıştırılabilmesini sağlar.

Örnek:

```
for (j = 0, k = 100; k - j > 0; j++, k-- )
```

Sonuç, en sağdaki operandın değeridir.

# İlişkisel Operatörler

Operatör	Sembol	Form	Sonuç
Büyüktür	>	$a > b$	$a > b$ ise 1, değilse 0
Küçüktür	<	$a < b$	$a < b$ ise 1, değilse 0
Büyüktür veya eşittir	>=	$a \geq b$	$a \geq b$ ise 1, değilse 0
Küçüktür veya eşittir	<=	$a \leq b$	$a \leq b$ ise 1, değilse 0
Eşittir	==	$a == b$	a ve b eşitse 1, değilse 0
Eşit değildir	!=	$a != b$	a ve b eşitse 0, değilse 1

# İlişkisel Operatörler

---

**int j=0, m=1, n=-1;**

**float x=2.5, y=0.0;**

<code>j &gt; m</code>	<code>j &gt; m</code>	(0)
-----------------------	-----------------------	-----

<code>m/n &lt; x</code>	<code>(m / n) &lt; x</code>	(1)
-------------------------	-----------------------------	-----

<code>j &lt;= m &gt;= n</code>	<code>( (j &lt;= m) &gt;= n)</code>	(1)
--------------------------------	-------------------------------------	-----

<code>++j == m != y * 2</code>	<code>((++j) == m) != (y * 2)</code>	(1)
--------------------------------	--------------------------------------	-----



# Mantıksal Operatörler

Operatör	Sembol	Form	Sonuç
mantıksal AND	<b>&amp;&amp;</b>	<b>a &amp;&amp; b</b>	ikisi birden sıfır değilse 1, aksi hallerde 0
mantıksal OR	<b>  </b>	<b>a    b</b>	ikisinden biri bile sıfır değilse 1, aksi halde 0
mantıksal Değil	<b>!</b>	<b>!a</b>	a 0 ise 1, değilse 0

# Mantıksal Operatörler

---

**int j=0, m=1, n=-1;**

**float x=2.5, y=0.0;**

j && m	(j) && (m)	(0)
--------	------------	-----

j < m && n < m	(j < m) && (n < m)	(1)
----------------	--------------------	-----

x * 5 && 5    m / n	((x * 5) && 5)    (m / n)	(1)
---------------------	---------------------------	-----

!x    !n    m + n	((!x)    !n)    (m + n)	(0)
-------------------	-------------------------	-----

# Bit Manipölasyon Operatörleri

Operatör	Sembol	Form	Sonuç
sağa shift	>>	$x \gg y$	x sağa y bit shift edilir
sola shift	<<	$x \ll y$	x sola y bit shift edilir
bitwise AND	&	$x \& y$	x y ile bitwise ANDlenir
bitwise inclusive OR		$x   y$	x y ile bitwise Orlanır
bitwise exclusive OR (XOR)	^	$x \wedge y$	x y ile bitwise XORlanır
bitwise complement	~	$\sim x$	x'in bitwise complement'i alınır

# Bit Manipölasyon Operatörleri

İfade	Sol Operandın Binary Karşılığı	Sonucun Binary Karşılığı	Sonuç Değeri
$5 \ll 1$	00000000 00000101	00000000 00001010	10
$255 \gg 3$	00000000 11111111	00000000 00011111	31
$8 \ll 10$	00000000 00001000	00100000 00000000	$2^{13}$
$1 \ll 15$	00000000 00000001	10000000 00000000	$-2^{15}$

İfade	Sol Operandın Binary Karşılığı	Sonucun Binary Karşılığı	Sonuç Değeri
$-5 \gg 2$	11111111 11111011	00111111 11111110	$2^{13} - 1$
$-5 \gg 2$	11111111 11111111	11111111 11111110	-2

# Bit Manipölasyon Operatörleri

İfade	Onaltılık değeri	Binary Karşılığı
9430	0x24D6	00100100 11010110
5722	0x165A	00010110 01011010
9430 & 5722	0x0452	00000100 01010010

İfade	Onaltılık değeri	Binary Karşılığı
9430	0x24D6	00100100 11010110
5722	0x165A	00010110 01011010
9430   5722	0x36DE	00110110 11011110

# Bit Manipölasyon Operatörleri

---

İfade	Onaltılık değeri	Binary Karşılığı
9430	0x24D6	00100100 11010110
5722	0x165A	00010110 01011010
9430 ^ 5722	0x328C	00110010 10001100

İfade	Onaltılık değeri	Binary Karşılığı
9430	0x24D6	00100100 11010110
~9430	0xDB29	11011011 00101001

# Bitwise Atama Operatörleri

---

Operatör	Sembol	Form	Sonuç
sağa-shift-atama	$\gg=$	$a \gg= b$	$a \gg b$ değerini a'ya ata.
sola-shift-atama	$\ll=$	$a \ll= b$	$a \ll b$ değerini a'ya ata.
AND-atama	$\&=$	$a \&= b$	$a \& b$ değerini a'ya ata.
OR-atama	$ =$	$a  = b$	$a   b$ değerini a'ya ata.
XOR-atama	$\wedge=$	$a \wedge= b$	$a \wedge b$ değerini a'ya ata.

# cast ve sizeof Operatörleri

---

Cast operatörü bir değeri farklı bir tipe dönüştürmenizi sağlar.

En bilindik kullanımı, tamsayıyı kayan nokta sayıya çevirerek kesme (truncation) olmamasını sağlamaktır.

- $3/2 \rightarrow (\text{float}) 3/2$

**sizeof** operatörü iki tür operand alabilir: Bir ifade veya bir veri tipi

- **ifade function, void, bit field OLAMAZ!**

**sizeof** operatörü, operandın bellekte kapladığı byte miktarını sayısal olarak döndürür.

- `sizeof(3+5)`  $\rightarrow$  int'in boyutunu döndürür
- `sizeof(short)`



# Koşullu operatör ( ? : )

Operatör	Sembol	Form	İşlem
Koşullu operatör	? :	$a ? b : c$	eğer a sıfırdan farklı ise sonuç b, değilse sonuç c

Tek üçlü operatördür.

if..else yapısının kısa halidir.

**if (x<y)**

**z = x;**

**else**

**z = y;**

**$z = ( (x < y) ? x : y );$**

# Bellek Operatörleri

Operatör	Sembol	Form	İşlem
address of	<b>&amp;</b>	<b>&amp;x</b>	x'in adresini al.
dereference	<b>*</b>	<b>*a</b>	a adresindeki nesnenin değerini al.
array elements	<b>[]</b>	<b>x[5]</b>	5 indisindeki elemanın değerini al.
dot	<b>.</b>	<b>x.y</b>	x struct'ındaki y elemanının değerini al.
right-arrow	<b>-&gt;</b>	<b>p -&gt; y</b>	p ile gösterilen struct'taki y elemanının değerini al.

# Akış Kontrolü

---

## Koşullu Dallanma

if, iç içe if

switch

## Döngüler

for

while

do..while

# if..else İfadesi

## Örnek1 :

if (x)

statement1; // x sıfırdan farklı ise işletilir

statement2; // her zaman işletilir

## Örnek2:

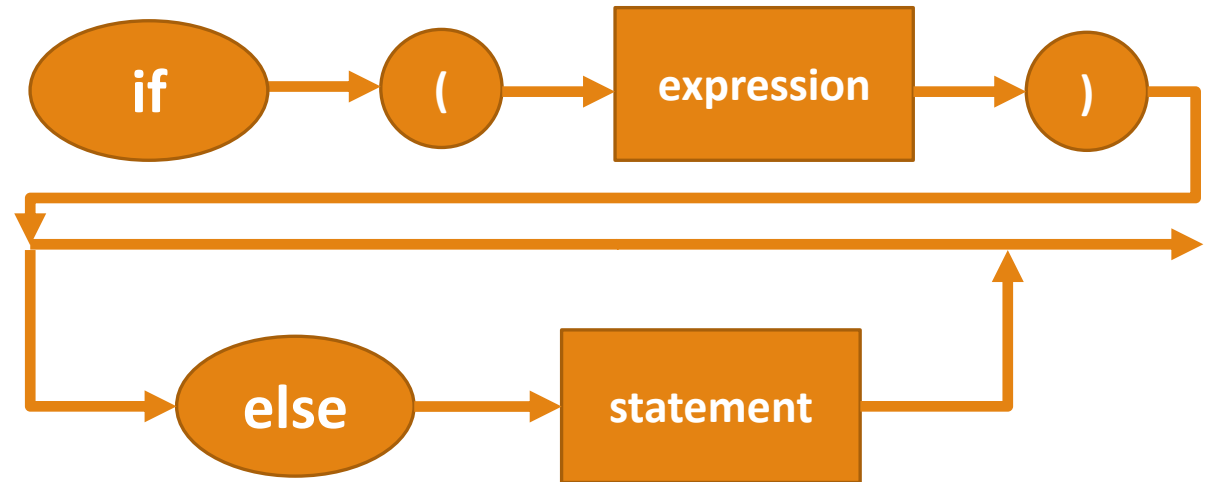
if (x)

statement1; // x sıfırdan farklı ise işletilir

else

statement2; // x sıfır ise işletilir

statement3; // her zaman işletilir



# İç içe if ifadeleri

---

Bir ***else*** ifadesi hemen yeni bir ***if*** ifadesi ile devam ediyorsa,

Genellikle aynı satırda yazılırlar.

Genel olarak ***else if*** ifadesi olarak bilinir.

İç içe if ifadeleri, her bir else ifadesinin sağındaki if ile eşleşmesi problemine yol açar.

- Buna avare else (dangling else) problemi adı verilir.
- Bir else her zaman en yakın önceki if ile ilişkilendirilir.

```
if(a<b)
    if(a<c)
        return a;
    else
        return c;
else if (b<c)
    return b;
else
    return c;
```

```
#include <stdio.h>
```

```
int main( void ) {
```

```
    int a, b, c;
```

```
    printf("please enter first number : ");    scanf("%d", &a);
```

```
    printf("please enter second number : ");    scanf("%d", &b);
```

```
    printf("please enter third number : ");    scanf("%d", &c);
```

```
    if (a < c) {
```

```
        if (a < b) {
```

```
            printf("min number is %d\n", a);
```

```
        }else {
```

```
            printf("min number is %d\n", b);
```

```
        }
```

```
    } else {
```

```
        if (c < b) {
```

```
            printf("min number is %d\n", c);
```

```
        } else {
```

```
            printf("min number is %d\n", b);
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

# switch İfadesi

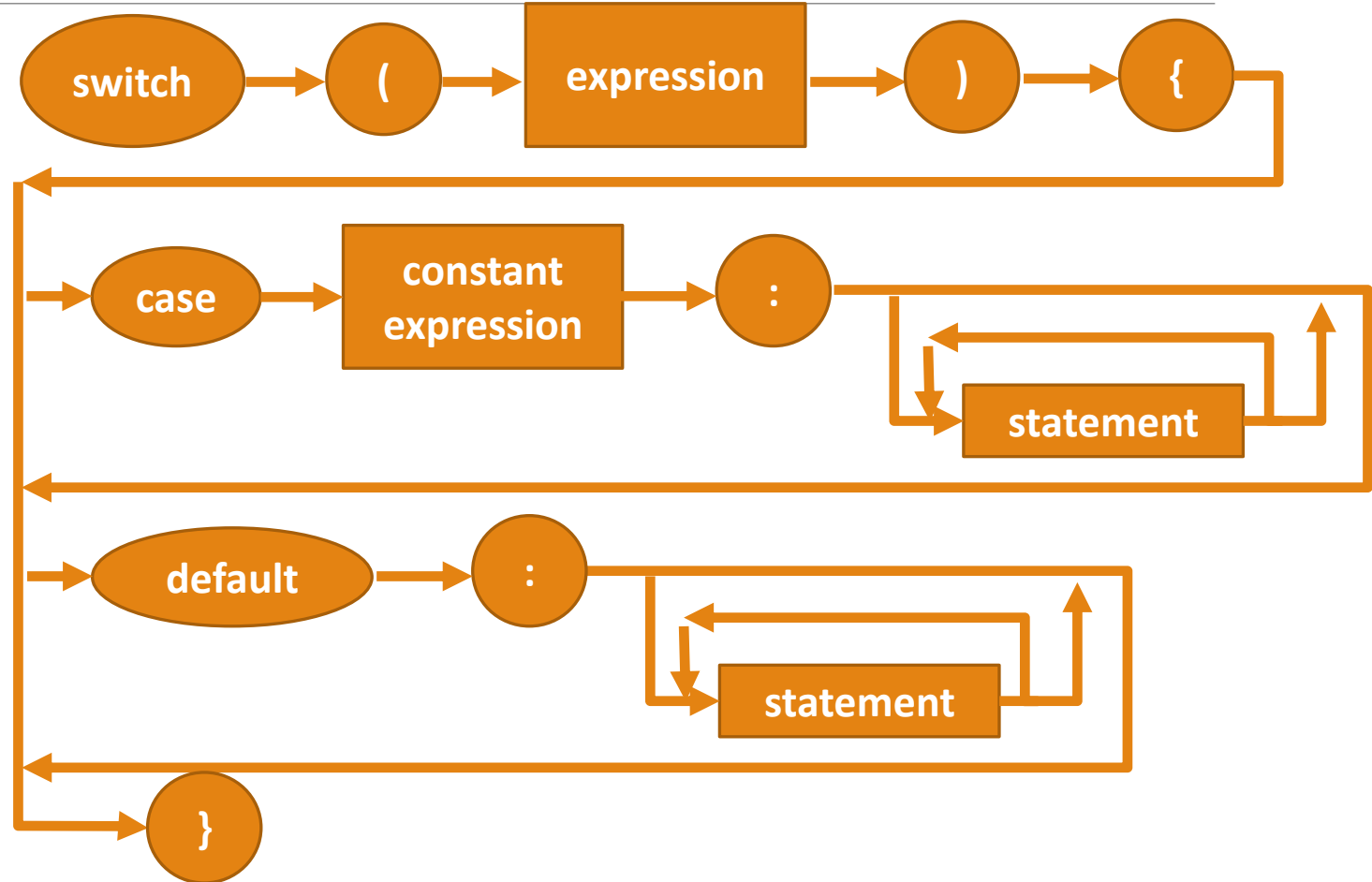
**Switch** ifadesi eğer **case** etiketlerinden biri ile eşleşirse çalışır.

Program akışı, eşleşen etiketın sonrasında devam eder.

Eğer hiçbir **case** ile eşleşmezse, **default** etiketinden çalışmaya devam eder. Bunun için **default** etiketinin tanımlanmış olması gerekir!

İki farklı case etiketi aynı değeri taşıyamaz.

default durum son durum olmak zorunda değildir ama en sona yazmak iyi bir stildir.



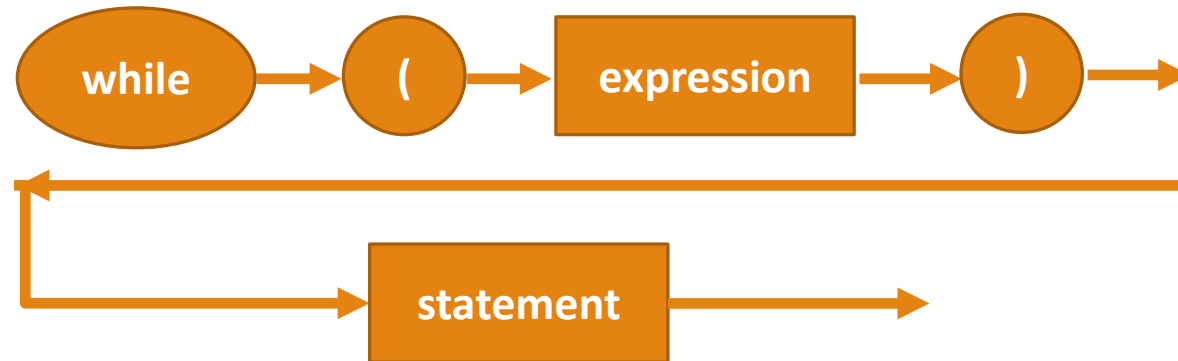
# while İfadesi

---

İlk olarak ifade değerlendirilir. Sıfırdan farklı ise, ifadeler çalıştırılır.

İfadeler tamamlandıktan sonra, program kontrolü while ifadesinin başına döner, ifade tekrar değerlendirilir.

İfade sıfıra eşit olana kadar döngü tekrarlanır.





```
#include <stdio.h>
```

```
#include <math.h>
```

```
int main( void ) {
```

```
    int i=1;
```

```
    float x, th, old, new, e;
```

```
    printf("please enter a number : ");
```

```
    scanf("%f", &x);
```

```
    printf("please enter threshold: ");
```

```
    scanf("%f", &e);
```

```
    old = 1;
```

```
    new = (old + x/old) / 2;
```

```
    th = new - old;
```

```
    while(th > e) {
```

```
        printf("old : %f - new : %f - th : %f - i : %d\n", old, new, th, i);
```

```
        i++;
```

```
        old = new;
```

```
        new = (old + x/old) / 2;
```

```
        th = fabs(new - old);
```

```
    }
```

```
    printf("Square root of %f is %f at iteration %d\n", x, new, i);
```

```
    return 0;
```

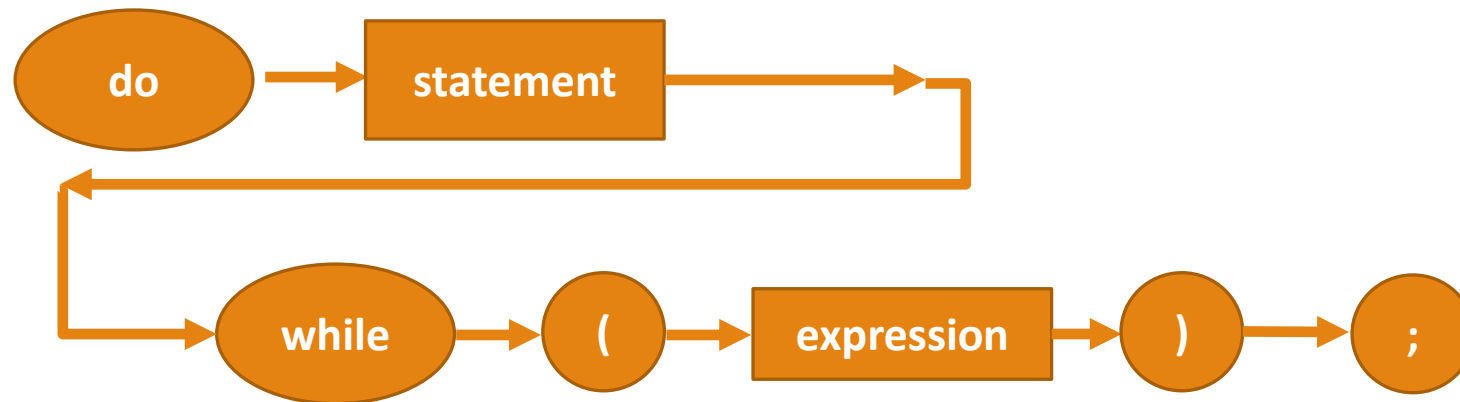
```
}
```

# do..while İfadesi

---

Tek fark, ifade kontrolünün başta değil sonda yapılmasıdır.

Döngü ifadeleri en azından bir kere mutlaka çalışır.



# for İfadesi

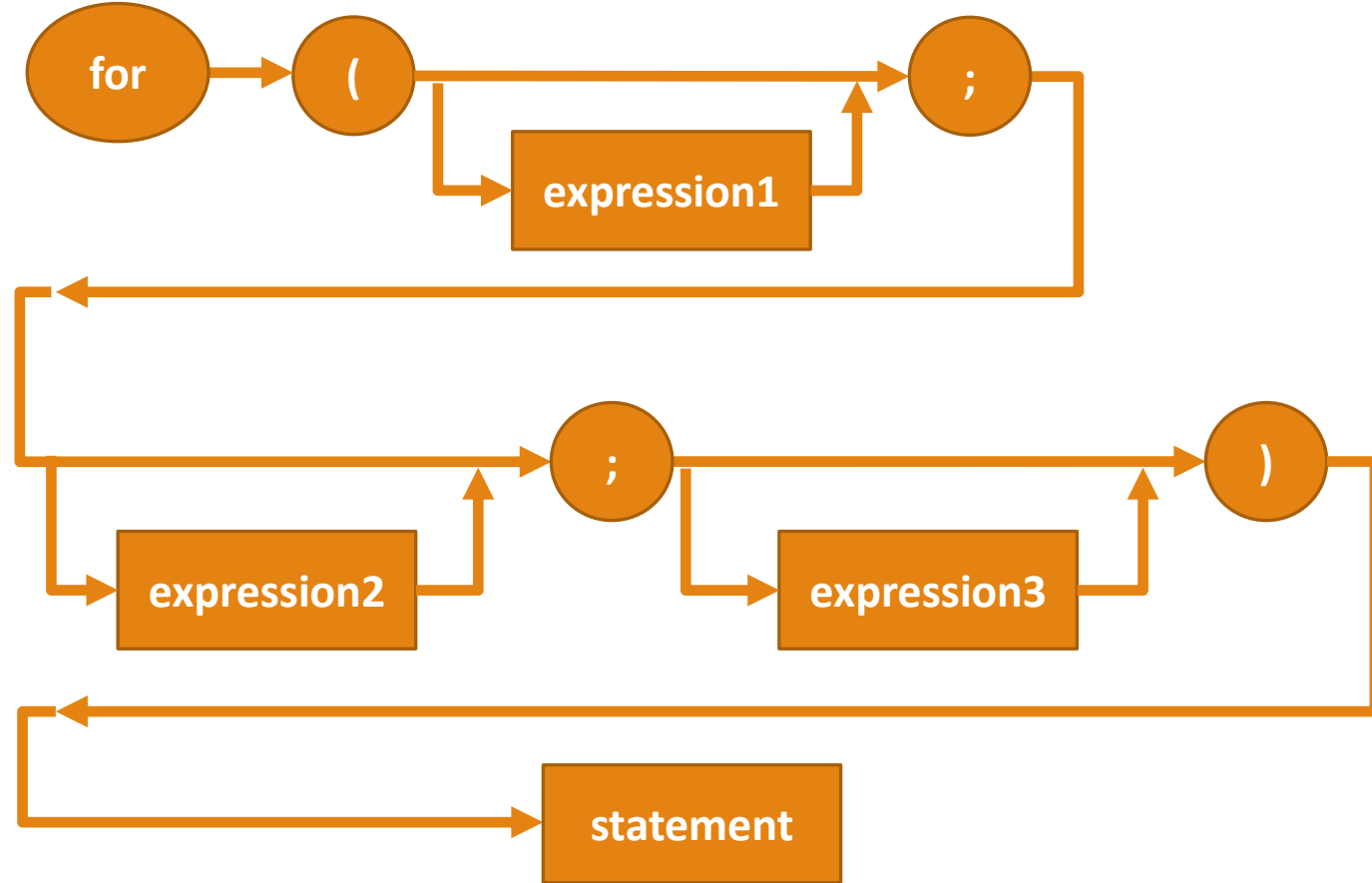
Önce ifade1 işletilir.

Sonra ifade2 işletilir.

- Burası ifadenin koşullu kısmıdır.
- ifade2 yanlış ise program kontrolü döngünün dışına çıkar.
- ifade2 doğru ise ifade (döngü bloğu) işletilir.

Blok işletildikten sonra, ifade 3 işletilir.

ifade2 kontrolüne geri dönülür.



```
#include <stdio.h>
```

```
int main( void ) {
```

```
    int i, n, mul=1;
```

```
    printf("please enter a number : ");
```

```
    scanf("%d", &n);
```

```
    for(i = 1; i <= n; i++) {
```

```
        mul = mul * i;
```

```
    }
```

```
    printf("mul : %d\n",mul);
```

```
    return 0;
```

```
}
```

```
#include <stdio.h>
```

```
int main( void ) {
```

```
    int i, n;
```

```
    float tmp, avg=0;
```

```
    printf("please enter number of elements : ");
```

```
    scanf("%d", &n);
```

```
    for(i = 1; i <= n; i++) {
```

```
        printf("enter a number :");
```

```
        scanf("%f", &tmp);
```

```
        avg += tmp;
```

```
    }
```

```
    avg /= n;
```

```
    printf("average of the elements is %f\n", avg);
```

```
    return 0;
```

```
}
```

# NULL ifadeler

---

for döngüsünde ifadelerden birini, ya da döngü gövdesini atlamak mümkündür.

```
for(c = getchar(); isspace(c); c = getchar());
```

## DİKKAT

test ifadesinden sonra noktalı virgül koyarsanız, derleyiciye «koşul doğru ise NULL ifadeyi işlet» demiş olursunuz.

```
if ( j == 1);
```

```
    j = 0;
```

# İç içe Döngüler

---

Döngüleri istenen derinlikte iç içe barındırabilirsiniz.

İçteki döngüler önce bitmelidir, dıştaki döngü ancak o zaman bir adım ilerleyebilir.

Kontrol ve döngüleri iç içe kullanmak da olasıdır.

```
for( j = 1; j <= 10; j++) {    //dış döngü
    printf("%5d|", j);
    for( k=1; k <=10; k++) {
        printf("%5d", j*k); // iç döngü
    }
    printf("\n");
}
```

# break, continue, goto

---

## **break:**

switch'te tartıştık.

Döngüde kullanıldığında, program kontrolünün döngüden sonraki ifadeye geçmesini sağlar.

## **continue:**

Döngü bloğunun geri kalanının işletilmeden tekrar döngü kontrolüne dönülmesini sağlar.

Bir sebepten geri kalan döngü kod bloğunun işletilmemesi istendiğinde faydalıdır.

Kullanmanız tavsiye edilmez.

## **goto:**

Eski dillerde kullanımı gereklilikti.

Lütfen asla ve asla kullanmayınız.