

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.2

2020-2021 Bahar Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

Ön Bilgi

- Ders: Pazartesi 10.00-12.50
- Uygulama: Çarşamba 12.00-13.50
- Değerlendirme Kriterleri (Değişebilir):
 - Vize Sınavı: %20
 - Quizler: %15
 - Ödevler: %15
 - Proje: %10
 - Final Sınavı: %40
- Derse Katılım: Uzaktan eğitimde yoklama alınmayacaktır.
 - **Yine de F0 alabilirsiniz!**
- **Bazı haftalarda ders/uygulama saatleri içinde quiz ve hızlı ödevler verileceği için katılımınız önerilir.**

Ön Bilgi

- Kopya Kuralları:
 - Herhangi bir şekilde ödev, quiz, proje veya sınavlarda hazır kaynaklardan / başkalarından kopyalama, ortak çözüm ve hile yapılması durumunda, ilgili tüm taraflar ödevden/sınavdan 0 alırlar.
 - Bu gibi işlemler disiplin yönetmeliği uyarınca değerlendirilecektir.
- Kullanılacak programlama dili:
 - Derste verilen tüm ödev, uygulama, proje, quiz, sınav vb. faaliyetlerde, ANSI standardında **C** programlama dili kullanılmalıdır.
 - Diğer dillerdeki çözümler değerlendirilmeyecektir.
 - C++, C# gibi diller de değerlendirilmeyecektir.
 - İstenen işlevselliği yerine getiren hazır kütüphane kullanılmamalıdır.

Ödev Kuralları

- Bu kurallar ödev değerlendirilmesi sürecini standardize etmek için konulmuş olup dikkat edilmesi önem arz etmektedir. Ödevde özel ek kurallar ödev metninde belirtilebilir.

Genel Kurallar

- Sisteme yanlış ödev yükleyenlerin ödevi geçersiz sayılacaktır.
- Sonradan farklı kanallardan gönderilen ödevler **kabul edilmeyecektir.**
- İnternet ortamındaki bir koda veya teslim edilen ödevlere %50'den fazla benzeyen ödevler kopya olarak değerlendirilecektir. Github gibi ortamlarda ödevlerinizi değerlendirmeler tamamlanana kadar public olarak paylaşmanız kopyaya sebep olabilmektedir.

Ödev Kuralları

Programlama ve Test Ortamı Kriterleri

- Aşağıdaki durumlarda ödev puanınızdan belirtilen miktarda puan eksiltilecektir.
- Aksi belirtilmedikçe ödevinizi C programlama dili ile kodlamanız gerekmektedir. Ödev kontrollerinde sorun yaşanmaması için kodu göndermeden önce, bölüm laboratuvarlarında kullandığımız Dev-C++ (GCC 4.9.2) versiyonunda da test etmeniz gerekmektedir. **(-15 puan)**
- Kodunuzu yazarken ANSI-C standartlarına uygun yazmaya dikkat ediniz. Kodunuzu derlerken herhangi bir özel derleyici modu kullanmayınız (C99 gibi). Aşağıda bir kod örneği verilmiştir: **(-5 puan)**

```
for(int i=0;i<10;i++).. // yanlış kodlama  
int i; for(i=0; i<10; i++).. // doğru kodlama
```

- Genel program geliştirme kuralları takip edilmelidir. Örneğin; Global değişken kullanılmamalı **(-5 puan)**. Dizi, matris gibi büyük değişkenler için statik bellek ayırma yöntemleri kullanılmamalıdır **(-10 puan)**, dinamik bellek ayırma yöntemleri kullanılmalıdır.
- Dosyadan girdi alınmasını gerektiren ödevlerde aksi belirtilmediği sürece dosya yolunu belirlemek için kodunuzda Mutlak yol(Full/Absolute Path) yerine, dinamik yol vermek için Relative Path kullanınız **(-5 puan)**

```
filePath="C:\Users\user\Desktop\folder\input.txt";//yanlış kodlama  
filePath ="input.txt"; // doğru kodlama
```

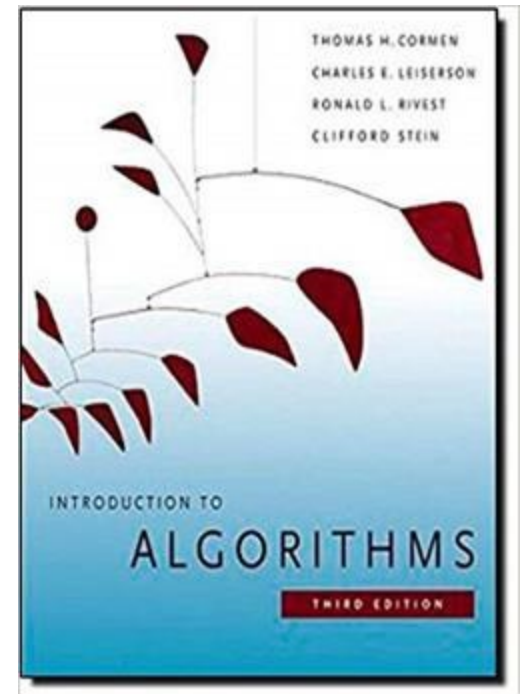
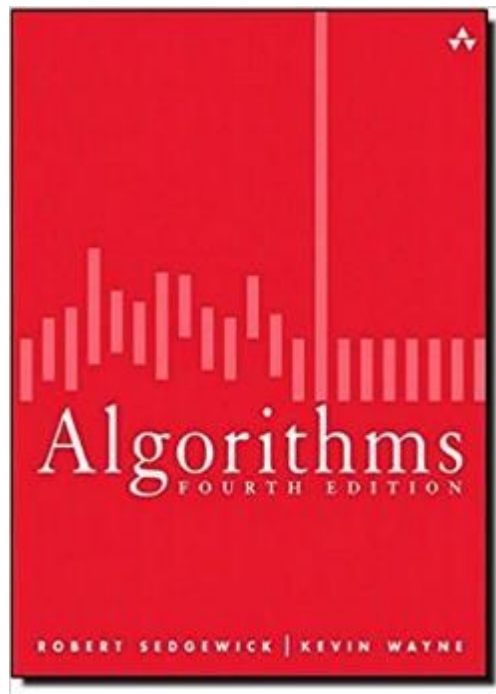
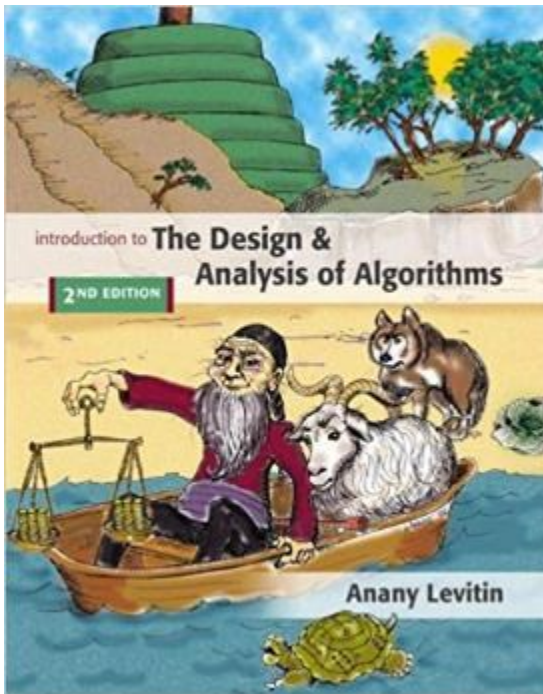
Ödev Kuralları

Rapor ve Teslim Kriterleri

- Raporunuzda, aksi belirtilmedikçe kapak sayfasında isim, soyisim, ders adı, grup bilgileri, kaçınıcı ödev olduđu ve mail adresinizin bulunması gerekmektedir. Kapak sayfasına ek olarak “Giriş, Yöntem veya Analiz, Sonuç” vb. bölümlerinin olması beklenmektedir.
- Ödevde sizden akış şeması istenmesi durumunda, anlaşılması kolay olmayan şemalar **değerlendirilmeye alınmayacaktır. Lucidchart, DrawIO, MS Visio** gibi çizim programlarıyla akış diyagramlarını oluşturmanız tavsiye edilmektedir. Fakat elle yapılan okunaklı ve düzgün çizimler de kabul edilir.
- Program ekran çıktılarını ekran görüntüsü alma uygulamaları ile ekranın sadece ilgili kısmının görüntüsünü alarak yapmanız gerekmektedir. Cep telefonu ile çekilen fotoğraflar gibi okunabilirliği zor olan yöntemleri tercih **etmeyiniz. (-20 puan)**
- Aksi belirtilmediği sürece, kod ve rapor dosyalarınız için isim formatı **ÖğrenciNumaranız.c** ve **ÖğrenciNumaranız.pdf** olmalıdır(Örnek 19011001.c ve 19011001.pdf). C dışındaki program dosyaları(.cpp, .h) ve pdf dışındaki rapor dosyaları (.doc, .docx, .rtf vb) **değerlendirilmeyecektir.** Ödevlerinizi .zip, .rar uzantılı sıkıştırılmış ÖğrenciNumaranız.zip şeklinde yükleyiniz. Lütfen sıkıştırılmış klasör dışında, dropbox, drive gibi sistemlere yüklediğiniz dosyaların bağlantılarını **yüklemeyiniz.**
- Son dakikalarda yaşanabilecek aksaklıkların (ağ hızında yavaşlama, bağlantı kopması, bilgisayar kilitlenmesi vb.) yaratabileceği sıkıntıları en aza indirmek için son ödev teslim saatinden mutlaka 10-15 dk önce yükleme işlemini tamamlayınız. Sisteme yüklerken eğer sistemsel bir hata ile karşılaşırsanız **ekran görüntüsü** alınız. Geçerli ve kanıtlanabilir bir mazeret olmadığı sürece e-mail ile gönderilen ödevler **değerlendirilmeyecektir.**

Kaynaklar

- Sınıfta sunulan materyallerden sorumlusunuz.
- Ders kitapları ve diğer ek materyaller sadece bilgilendirme ve yol gösterme amaçlıdır.



Ders Planı (Değişebilir)

Hafta	Tarih	Konular
1	08.03.2021	Giriş, Algoritmik Problem Çözüm Temelleri
2	15.03.2021	Algoritmik Problem Çözüm Temelleri
3	22.03.2021	Algoritma Analizi Temelleri, Büyük-O Notasyonu
4	29.03.2021	Listeler ve Linkli (Bağlantılı) listeler
5	05.04.2021	Kuyruk ve Yığın veri yapıları
6	12.04.2021	Ağaç veri yapısı, İkili ağaçlar, İkili Arama Ağacı
7	19.04.2021	Heap veri yapısı, HeapSort
8	26.04.2021	Vize Sınavı
9	03.05.2021	Öncelikli Kuyruk
10	10.05.2021	Graflar, Minimum Yayılımı Ağaç
11	17.05.2021	Böl ve Yönet algoritmaları
12	24.05.2021	Böl ve Yönet algoritmaları
13	31.05.2021	String arama algoritmaları
14	07.06.2021	Sıralama algoritmaları

VERİ YAPILARI VE ALGORİTMALARA GİRİŞ

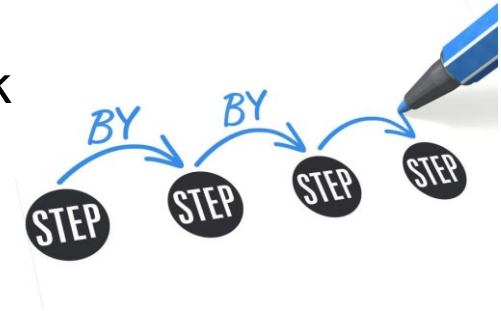
Algoritma'nın Kökeni

- *Ebû Ca'fer Muhammed bin Mûsâ el-Hârizmî*
 - (d. 780, Harezmi - ö. 850, Bağdat)
- *İranlı gökbilimci ve matematikçi, Bağdat'ta yaşamış*
- 825 yılında, Arapça «Al-jabr»
“Hint rakamları ile hesap”
- 1300lerde latinceye çeviri:
“Algoritmi de numero Indorum”



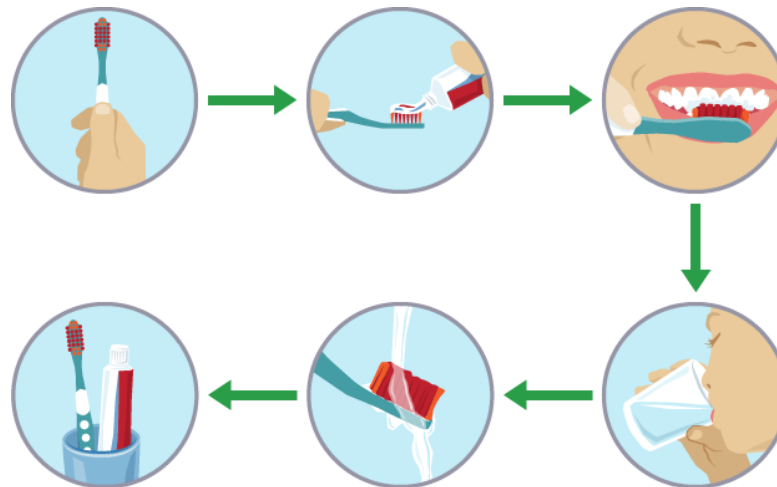
Algoritma Tasarımı

- İşlemsel çözüm gerektiren problemler
- **Algoritma**, bir problemin işlemler ile çözümünde izlenecek yolun **teorik** olarak oluşturulmasıdır.
 - Bir sayı dizisini sıralamak
 - Haritada iki nokta arasındaki en kısa yolu bulmak
 - Ödevi çözmek için zaman planı yapmak
 - Web arama sorgularını yanıtlamak
- Algoritma, bir programlama dilinde (Java, C++, C# gibi) ifade edildiğinde **program** adını alır.



Algoritma

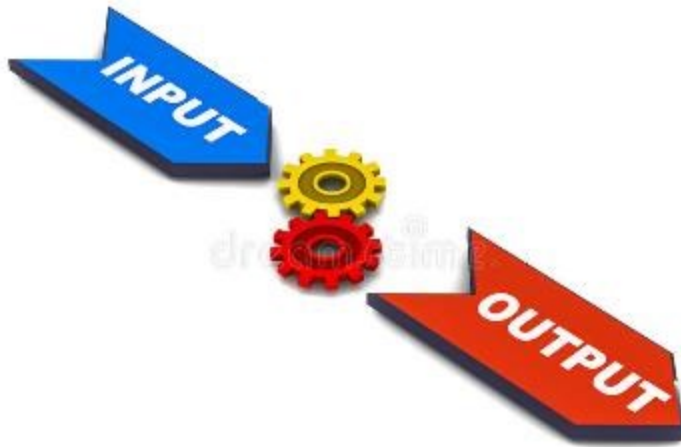
- Algoritma, belirli bir problemi çözmek için art arda uygulanacak **kesin** direktiflerden oluşan **sonlu** bir kümedir.
- Algoritma, mantıklı ve doğru tüm girdiler için **doğrudur** ve doğru çıktıyı sonlu bir zamanda üretir.
- Bunun dışındaki durumlar algoritma değildir.



Algoritma Özellikleri

- **Unambiguous (Anlaşılabilirlik):**

Adımları, giriş ve çıkışları net ve tek bir anlam içermelidir.



- **Input (Giriş):** 0 veya iyi tanımlanmış girdilere sahip olmalı
- **Output (Çıkış):** 1 veya iyi tanımlanmış çıktılarına sahip olmalı

Algoritma Özellikleri

- **Finiteness (Sonlandırılma):**
Belirli bir adımdan sonra sonlandırılabilirmeli
- **Feasibility (Yapılabilirlik):**
Mevcut imkanlarla yapılabilir olmalı
- **Independent (Bağımsızlık):**
Herhangi bir programlama kodundan bağımsız olmalı



Genel Algoritmik Problemler

- Sıralama
- Arama
- Metin (string) işleme
- Graf problemleri
- Kombinatorial problemler
- Geometrik problemler
- Nümerik problemler

Veri

- Veri, algoritmalar tarafından işlenen en temel elemanlardır.
 - Sayısal bilgiler
 - Metinsel bilgiler
 - Resimler
 - Sesler
 - Girdi, çıktı olarak veya ara hesaplamalarda kullanılan diğer bilgiler...
- Algoritmanın işleyeceği verilerin **düzenlenmesi** gerekir.

Veri Nasıl Tutulur?

- Uint (32 bit):

1111573057

- Ham Veri:

01000010010000010100001001000001

- BCD Kodlama:

0100 0010 0100 0001 0100 0010 0100 0001

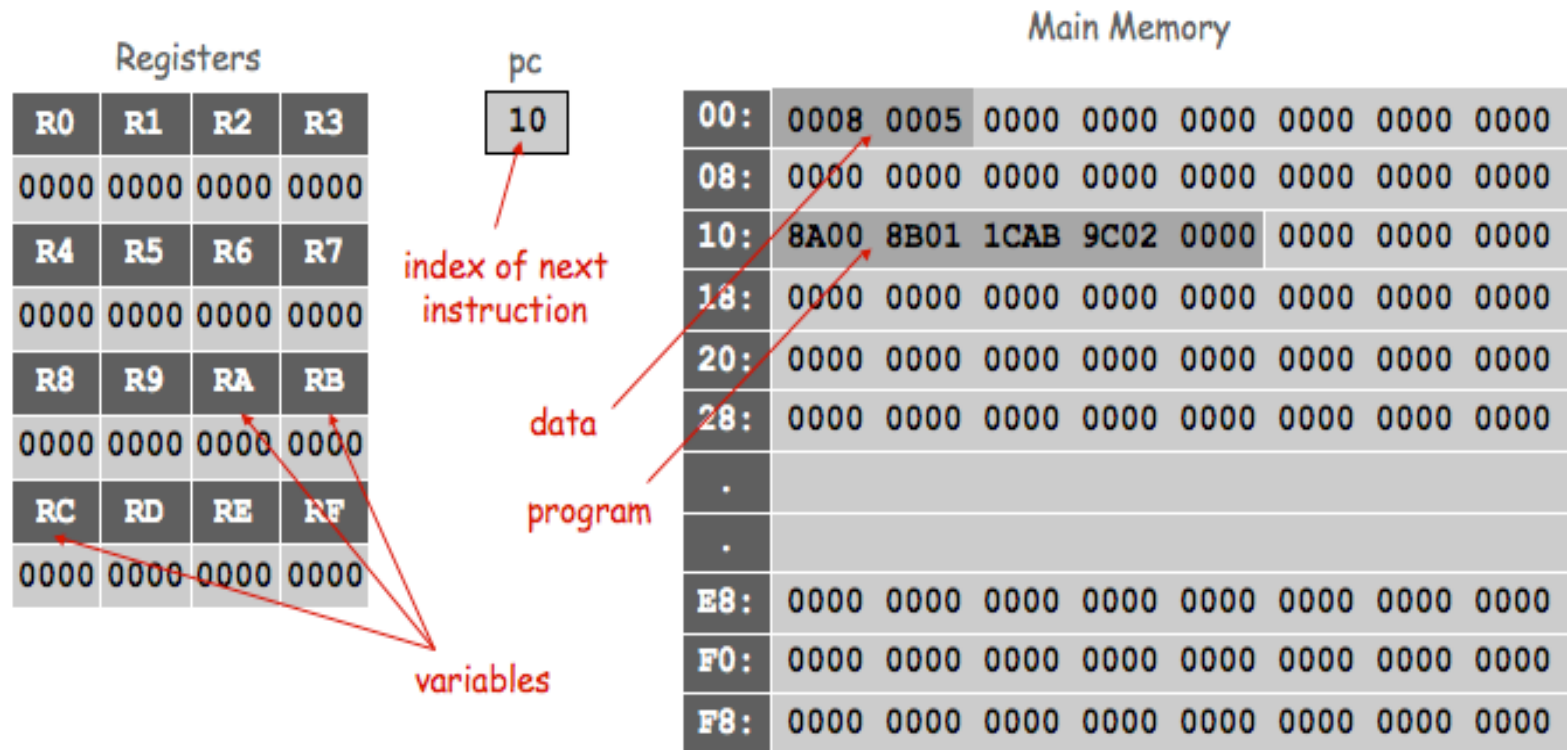
- HEX:

42 41 42 41

- ASCII:

B A B A

Veri Nasıl Tutulur?



Veri Yapıları

- Algoritmaların; erişme, değiştirme gibi çeşitli işlemlerinde kullanmak üzere bilginin saklandığı oluşumlara veri yapıları adı verilir.
 - Listeler
 - Diziler
 - Linkli listeler
 - Stack
 - Queue
 - Hash tabloları
 - Ağaçlar
 - Heap, Öncelikli kuyruk
 - Küme ve sözlük
- Verinin/bilginin bellekte tutulma şekli ve düzeni

Neden Veri Yapıları Gerekli?

- Veri yapıları, verinin etkin olarak saklanması ve işlenebilmesi için gereklidir.
- Doğru yerde kullanıldığında en basit veri yapısı dahi oldukça etkili olabilir.
- $n \times n$ boyutunda bir matris
 - $n=1,000$ gibi küçük bir değer olsa bile
 - 1. ve n . sütun ile diyagonal haricindeki değerlerin 0 olması durumunda
 - Tüm matris için bellekte yer ayrılması
ya da
 - Sadece sıfırdan farklı değerlerin tutulması!!

ALGORİTMAYI İFADE ETMEK

Algoritma Hazırlama Süreci

- Tasarım (design)
- Doğruluğu ispat (validation)
- Analiz (analysis)
- Uygulama (implementation)
- Test

Algoritmayı İfade Etmek

- Sözde-Kod (Pseudo-Code)
 - Yarı programlama dili, yarı konuşma dili

Toplam Hesapla (dizi, toplam,)

Girdi: n sayıdan oluşan dizi

Çıktı: dizi elemanlarının toplam sonucu

for i:=1 to n do

 toplam:= toplam + dizi[i]

end for



Algoritmayı İfade Etmek-Sözde Kod

- Hata mesajları gibi (şu aşamada) gereksiz detayları vermeden *ne* yapılması gerektiğini tanımlarız.
- Veri elemanlarını tanımlamaya gerek yoktur.
 - Algoritmada ilk kullandığımız yerde otomatik tanımlanır.
 - Tipi içerik ile belirlenir.
- Bir veri yapısı, tanımlanmalıdır. (Adı ve içeriği)
- Başlıkta algoritma adı, parametreleri, ön ve son koşulları ile geri dönüş değer(ler)i tanımlanır.
- İfadeler alt numara kullanımı ile sıralanır.
- Değişkenler akıllıca isimlendirilmelidir. Tek karakter ya da jenerik isimler (toplam, ortalama, satır, dosya, ...) verilmemelidir.

Algoritmayı İfade Etmek-Sözde Kod

- Yapısal programlama modelini ilk önerdiğinde, Edsger Dijkstra herhangi bir algoritmanın sadece üç programlama yapısını kullanarak yazılabileceğini belirtti: **dizilim**, **seçim** ve **döngü**.
- Bir **dizilim**, bir algoritma içindeki yürütme yolunu değiştirmeyen bir veya daha fazla ifadedir.
- Bir **seçim** ifadesi bir koşulu değerlendirir ve sıfır veya daha fazla alternatif yürütür.
 - Değerlendirme sonuçları hangi alternatiflerin işletileceğini belirler.
- Bir **döngü** ifadesi bir kod bloğunu tekrarlar.

Algoritmayı İfade Etmek-Sözde Kod

Algoritma başlığı

Amacı

Koşullar

Çıktı

İfade No 1

İfade No 1.2.1

İfade No 1.4

Sıradaki ifade

Algorithm sample (pageNumber)

This algorithm reads a file and prints a report.

Pre pageNumber passed by reference

Post Report Printed

 pageNumber contains number of pages in report

Return Number of lines printed

1 loop (not end of file)

1 read file

2 if (full page)

1 increment page number

2 write page heading

3 end if

4 write report line

5 increment line count

2 end loop

3 return line count

end sample

**Döngü
ifadesi**

**Karar
ifadesi**

Algoritmayı İfade Etmek-Sözde Kod

- **Atama** → genellikle $:=$ (Pascal dilindeki gibi) ya da \leftarrow
 - $\text{Max} := 1000$
 - $\text{Sum} \leftarrow \text{Sum} + \text{newItem}$
- **Eşitlik kontrolü** → $=$
 - $a = b$
- **Seçim**
 - if (condition) action1 end if
 - if (condition) action1 else action2 end if
- **Döngü**
 - while (condition) do statements end while
 - do statements while (condition)
 - repeat statements until (condition)
 - for start to end step increment do statements end for
- **İndis**
 - $\text{arr}[i]$
 - $\text{avg}[i+3]$

Örnek: Bir dizinin toplamı

Algoritma ToplamHesapla (dizi)

Verilen dizinin elemanlarının toplamını hesaplar

Girdi: n elemanlı tamsayı dizisi

Çıktı dizinin elemanlarının toplam değeri

1 toplam := 0

2 for i := 1 to n do

1 toplam := toplam + dizi[i]

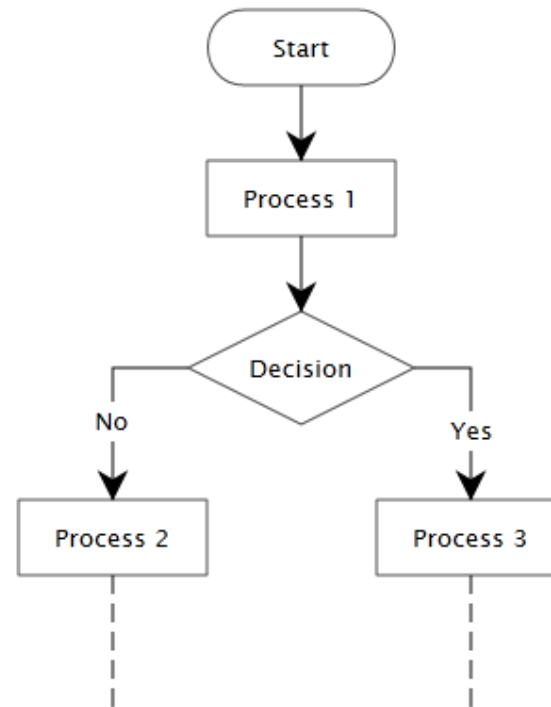
3 endfor

4 return toplam

end ToplamHesapla

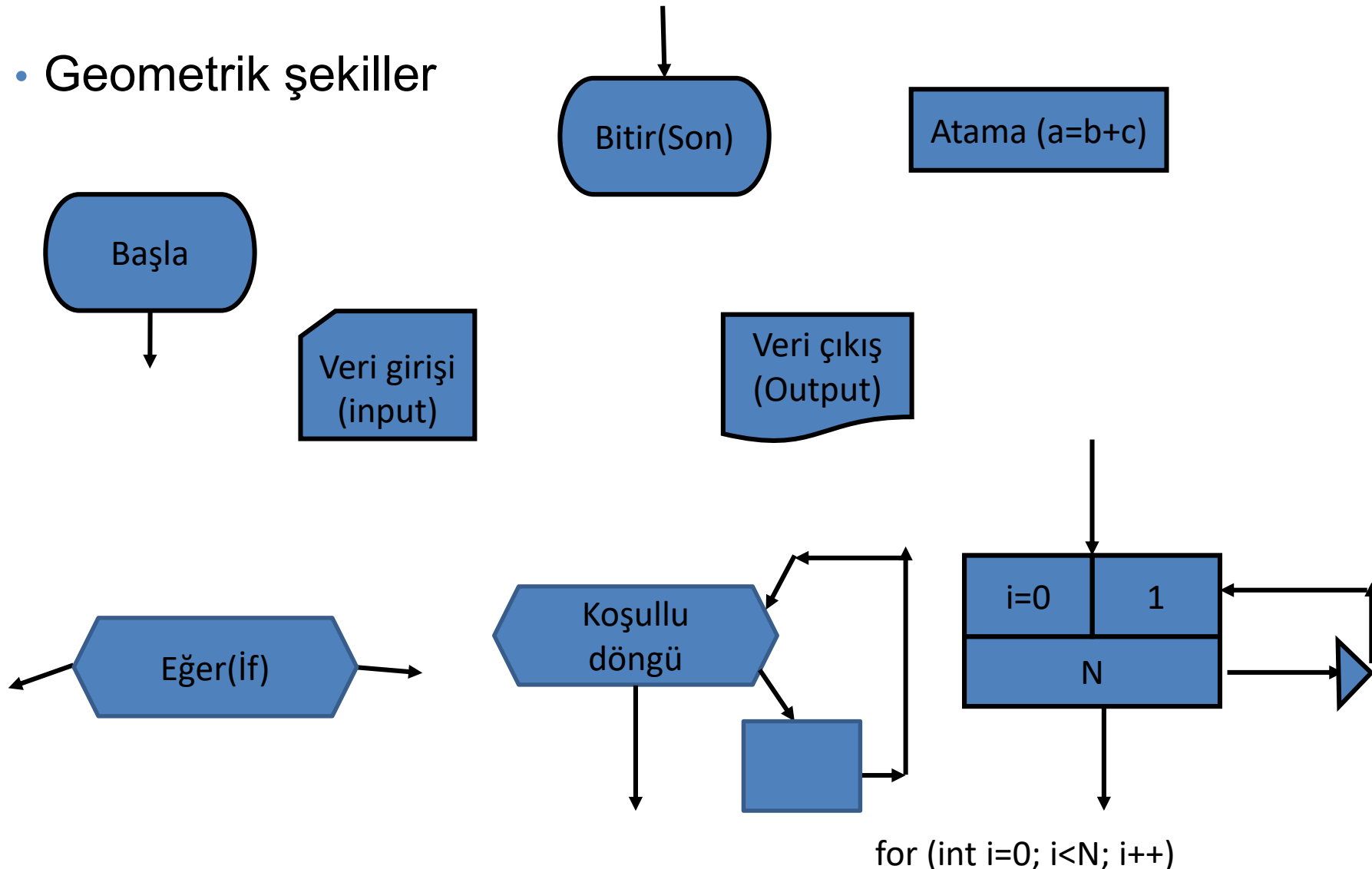
Algoritmayı İfade Etmek-Akış Diyagramı

- Diyagram elemanları
- Bağlantılar



Algoritmayı İfade Etmek-Akış Diyagramı

- Geometrik şekiller



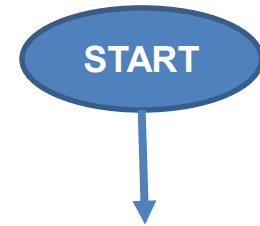
Akış Diyagramı Bileşenleri– I

- Başla - Bitir (Start - Stop)
- Okuma (Read / Inputs)
- Bağlantılar (Connections)
 - Oklar, **çemberler** (arrows, **circles**)
- İfadeler/İşlemler (Statements/Process)
- Kararlar (Decisions)
 - if, case/switch, ...
- Döngüler (Loops)
 - for, while, do while ...

Akış Diyagramı Bileşenleri– II

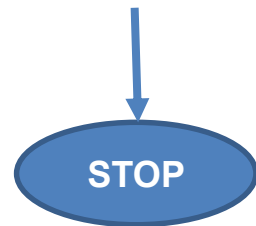
- BAŞLA / START

- Algoritmanın başlangıcını gösterir
- Tüm akış diyagramlarının bir başlangıcı olmalıdır.



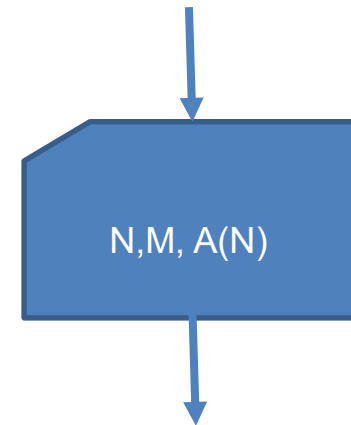
- BİTİR / STOP

- Algoritmanın sonlandığı yeri gösterir
- Tüm akış diyagramlarının bir sonu olmalıdır.



Akış Diyagramı Bileşenleri– III

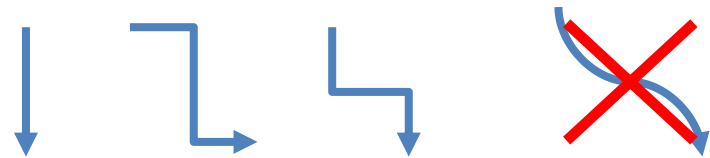
- Girdi
 - Kullanıcıdan alınan değişkenleri gösterir.
 - Her değişken virgül ile ayrılarak yazılır.
- Dizi okuma
 - 1.seçenek: for döngüsü içinde girdi kullanın
 - 2.seçenek: $A(N)$ notasyonunu kullanın
 - **$A(N)$ 'den önce N 'i okumalısınız!**



Akış Diyagramı Bileşenleri– IV

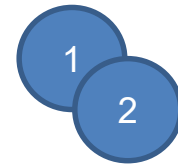
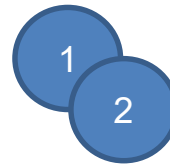
- Oklar

- Akıştaki iki elemanı birbirine bağlar
- Akışın yönünü gösterir
- **Eğri büğrü değil, köşeli çizilir.**



- Çemberler

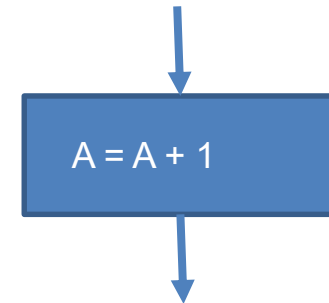
- Bağlantıları basitleştirmek / okunabilir kılmak için
 - İç içe döngüler, iç içe kontroller vs
- **Numaralandırmalara dikkat edin!**



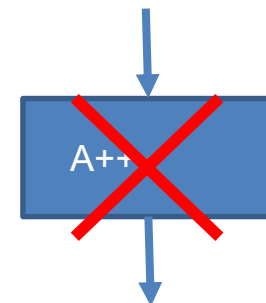
Akış Diyagramı Bileşenleri– V

- İfadeler / İşlemler

- Aşağıdaki ifadelerden en az birini içerir:
 - Aritmetik işlem, atama, ...

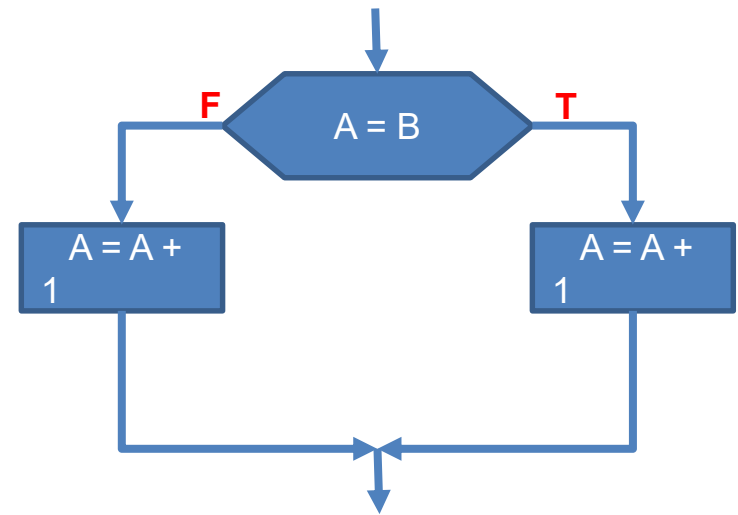


- Algoritmalar programlama dillerinden bağımsız tasarlanmalıdır!
 - Dile özel operatör KULLANILMAMALIDIR
 - Dile özel değişken KULLANILMAMALIDIR



Akış Diyagramı Bileşenleri– VI

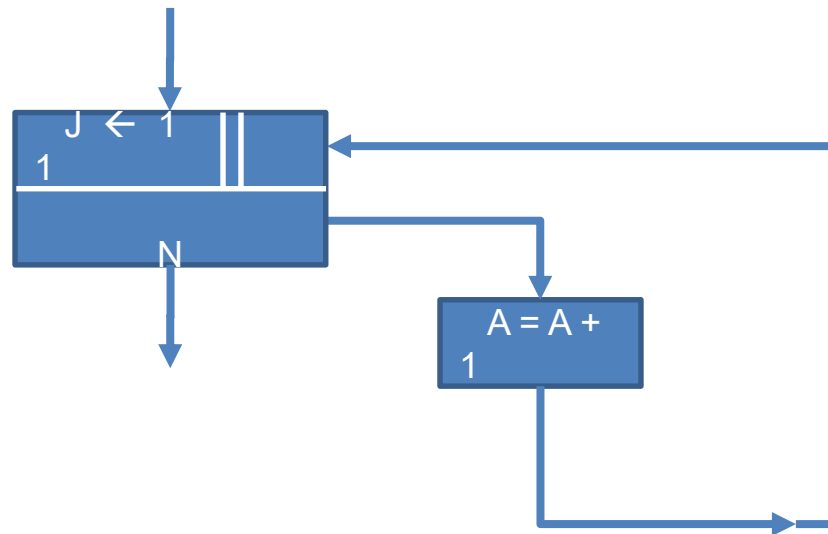
- IF İfadesi
 - Her zaman Doğru (**T**)rue ve Yanlış (**F**)alse kollarını belirtin.
 - Her if ifadesinde MUTLAKA doğru (T) dalı olmalıdır.
 - Yanlış (F) dalı isteğe bağlıdır.
- Algoritmadaki tüm IF ifadelerinde T/F dallanmalarını aynı tarafta kullanmak tercih sebebidir.



Akış Diyagramı Bileşenleri– VIII

- **FOR döngüleri**

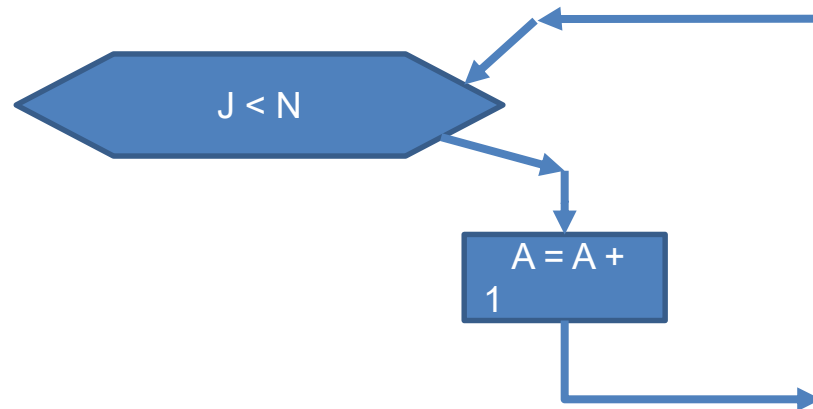
- Belirli sayıda iterasyona sahip döngüler için kullanılır.
- Üç parçadan oluşur
 - İlk koşul (başlangıç)
 - Son koşul (durma)
 - Adım (+/-)



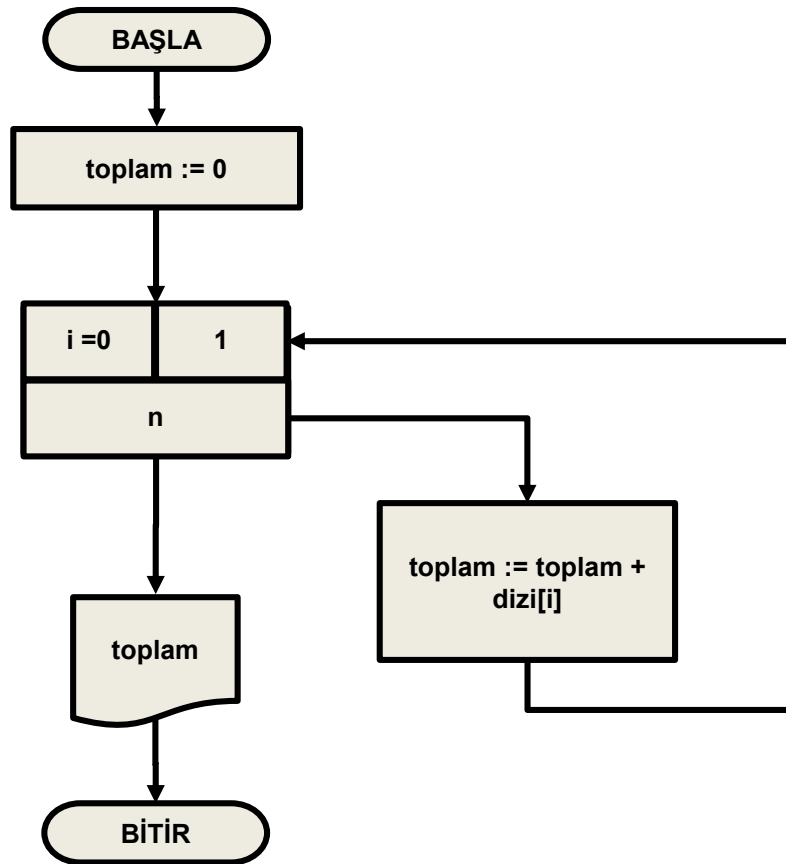
Akış Diyagramı Bileşenleri– IX

- **While döngüleri**

- Bilinmeyen sayıda iterasyona sahip döngüler için kullanılır.
 - Bir şartın sağlandığı sürece işletilmesi esasına dayanır

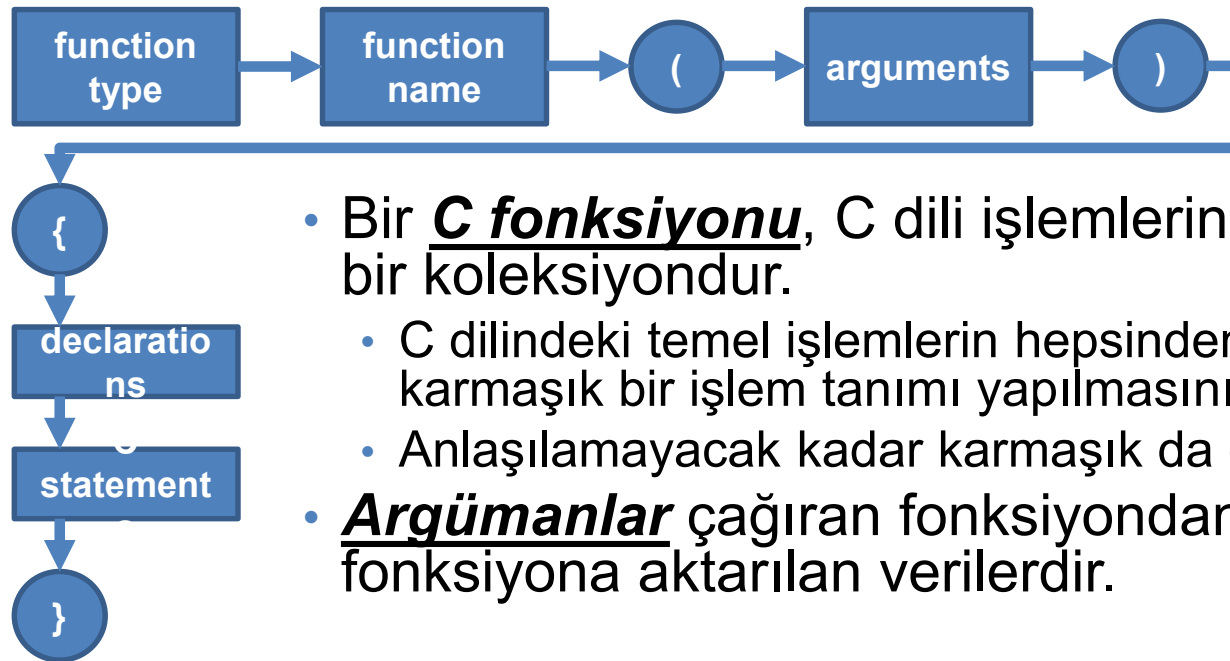


Örnek: Bir dizinin toplamı



C – HATIRLAMA NOTLARI

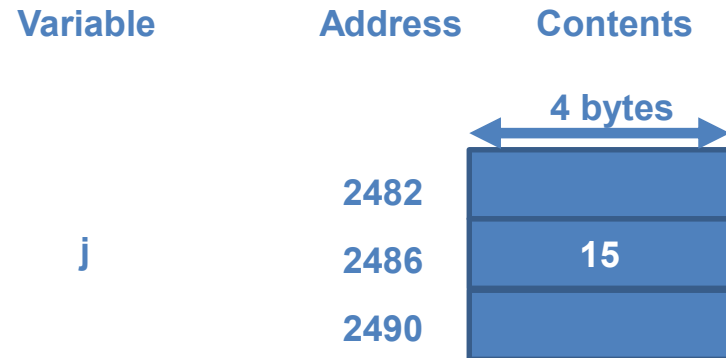
C Temelleri



- Bir **C fonksiyonu**, C dili işlemlerinden oluşan bir koleksiyondur.
 - C dilindeki temel işlemlerin hepsinden daha karmaşık bir işlem tanımı yapılmasını sağlar.
 - Anlaşılamayacak kadar karmaşık da olmamalıdır.
- **Argümanlar** çağıran fonksiyondan çağrılan fonksiyona aktarılan verilerdir.

Değişkenler & Sabitler & İfadeler

- İfade: $j = 5 + 10;$
- **Sabit** hiç değişmeyen bir değerdir.
 - 5
 - $5+6*13/3.0$
- **Değişken** bellekteki bir konumun, **adresin** gösterimidir.
 - `int j,k;`
 - j
 - $j/k*5$
 - k-'a'
 - $3+(int)5.0$
 - `double x,y;`
 - $x/y*7$
 - $3+(float)4$
 - `int *p //pointer`
 - p
 - p+1
 - "abcd"



İsimlendirme

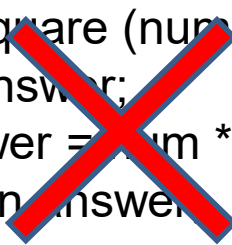
- C dilinde her şeye bir isim verilebilir.
 - Değişkenler, sabitler, fonksiyonlar, programdaki bir konum...
- İsimlerde harfler, rakamlar, alt çizgi _ bulunabilir.
 - Harf veya alt çizgi ile başlamalıdır.
 - j
 - j5
 - __system_name
 - sesquipedalial_name
 - UpPeR_aNd_LoWeR_cAsE_nAmE
 - 5j
 - \$name
 - int
 - bad%#@name
- C dili büyük/küçük harf duyarlıdır.
- Ayrılmış anahtar kelimeler isim olarak kullanılamaz.

İsimlendirmede Kullanılamayanlar

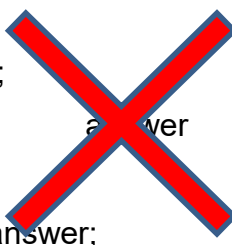
- auto
- double
- int
- struct
- break
- else
- long
- switch
- case
- enum
- register
- typedef
- char
- extern
- return
- union
- const
- float
- short
- unsigned
- continue
- for
- signed
- void
- default
- goto
- sizeof
- volatile
- do
- if
- static
- while

Kaynak Kodun Biçimlendirilmesi

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



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



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



main() Fonksiyonu

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

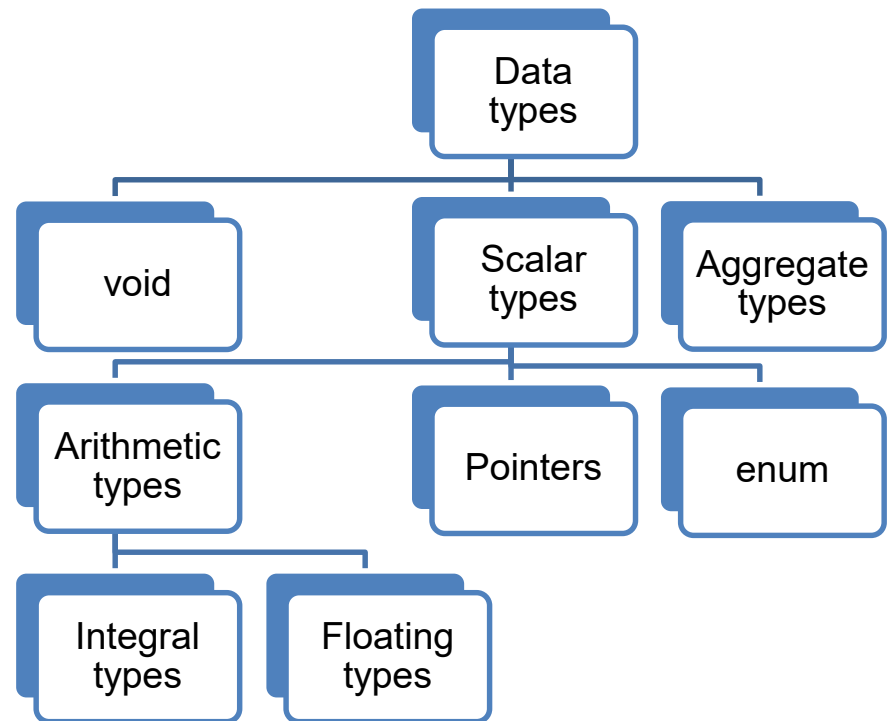
- **exit()** fonksiyonu programın sonlanarak işletim sistemine geri dönülmesini sağlayan bir çalışma zamanı kütüphanesi yordamıdır.
 - Argüman 0 ise, program hatasız şekilde normal olarak sonlanır.
 - Sıfır harici argümanlar programın normal dışı şekilde sonlandığını gösterir.
- main() fonksiyondan exit() çağırmak, **return** ifadesini çağırmakla aynıdır.

Önişlemci - Preprocessor

- Ön işlemci program derlendiğinde otomatik çalışır.
- Tüm önişlemci tanımları `#` ile başlar, satırdaki ilk (boşluk harici) karakter `#` olmalıdır.
 - Normal C ifadeleri `;` ile biter ama önişlemci ifadeleri enter (yeni satır) ile sonlanır.
 - Ön işlemci ile başka yapılabilecekler:
 - Macro işleme
 - Koşula dayalı derleme
 - Macrolar ile hata ayıklama (debug)
- `#define` ile sabite isim atanabilir `#define ZERO 0`
 - Değişken olarak kullanamazsınız.
- `#include` ile derleme aşamasında başka bir kaynak dosyasını okuyarak eklemeniz mümkündür.
 - `#include <filename>` ile işletim sistemde belirli klasördeki kaynaklar okunur.
 - `#include «filename»` ile önce dosyanın klasörü, orada yoksa işletim sistemde belirli klasördeki kaynaklar okunur.

C Veri Tipleri

- 9 scalar veri tipi
- Basit tipler:
 - char, int, float, double, enum
- Niteleyiciler:
 - long, short, signed, unsigned
- Tanımlama:
 - int i;
 - int i, j;
 - int i, j, k=0;
- ANSI C: Tüm tanımlamalar herhangi bir işletilebilir ifadeden önce yapılmalıdır!



Integer Tanımları

- ANSI Standardının tek gereksinimi bir byte'ın en az 8 bit olmasıdır.
- int en az 16 bit olmalıdır ve bilgisayarın «doğal» boyutunu temsil etmelidir.
 - doğal: CPU'nun tek bir komutta işleyebildiği bit uzunluğu

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 $2^8 - 1$


Tip dönüşümleri

- Tamsayı-kesirli, işaretli-işaretsiz tipler arasında dönüşümler olasıdır.
- Implicit: $-3 / 4 + 2.5$
 - İşlem öncelik sırasını gözeterek, derleyici terimleri birbiri ile uyumlu olacak şekilde dönüştürür.
- Explicit:
 - `int j=2; k=3; float f;`
 - `f = k / j;`
 - `f = (float) k/j;`


void Veri Tipi | typedef | sizeof

- void veri tipinin iki önemli kullanımı vardır.
 - Fonksiyonun değer döndürmediğini belirtir: `void f(int a);`
 - Jenerik pointer tanımı sağlar.
- typedef anahtar kelimesi ile veri tiplerine kendi belirlediğimiz isimleri atayabiliriz.
 - `typedef long int int32;`
 - `int32 i, j, k;`
- sizeof operatörü ile terimin bellekte kaç bayt yer kapladığını hesaplayabiliriz.
 - `sizeof(3+5) // sizeof integer`
 - `sizeof(short) // sizeof short`

İşlem Öncelikleri

Operatör Sınıfı	Sınıftaki Operatörler	Birleşirlik	Öncelik
Öncül	() [] -> .	Soldan Sağa	En Yüksek 
Birli	cast operator sizeof & (address of) * (dereference) - + ~ ++ -- !	Sağdan Sola	
Çarpım	* / %	Soldan Sağa	
Toplama	+ -	Soldan Sağa	
Kaydırma	<< >>	Soldan Sağa	
İlişkisel	< <= > >=	Soldan Sağa	
Eşitlik	== !=	Soldan Sağa	

İşlem Öncelikleri

Operatör Sınıfı	Sınıftaki Operatörler	Birleşirlik	Öncelik
Bit AND	&	Soldan Sağa	
Bit exclusive OR	^	Soldan Sağa	
Bit Inclusive OR		Soldan Sağa	
Mantıksal AND	&&	Soldan Sağa	
Mantıksal OR		Soldan Sağa	
Koşullu	? :	Sağdan Sola	
Atama	= += -= *= /= %= >>= <<= &= ^=	Sağdan Sola	
Virgöl	,	Soldan Sağa	En Düşük

Artan (Kalan) operatörü

- Diğer tüm aritmetik operatörler tamsayı ve gerçel sayı terimleri kabul ederken, kalan operatörü sadece tamsayı terim kabul eder.
- Herhangi bir terim negatif ise, kalan –gerçeklemeye bağlı olarak- pozitif veya negatif olabilir.
- ANSI standardına göre, kalan ve bölme operatörleri arasında şu ilişki olmalıdır:
- Herhangi iki tamsayı a ve b için,
 - $a = a \% b + (a / b) * b$

Aritmetik Atama Operatörleri

- **int m = 3, n = 4;**
- **float x = 2.5, y = 1.0;**

- $m += n + x - y$ $m = (m + ((n+x) - y))$

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

- $n \% = y + m$ $n = (n \% (y + m))$

- $x += y -= m$ $x = (x + (y = (y - m)))$

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++ (gerçeklemeye göre değişir. 2?3?)

Virgül Operatörü

- Tek bir ifadeye izin verilen yerlerde birden fazla ifadenin yerine getirilmesini sağlar.
- Örnek: `for (j = 0, k = 100; k - j > 0; j++, k--)`
- Sonuç (dönüş değeri) en sağdaki terimin değeridir.

İlişkisel Operatörler

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

float x=2.5, y=0.0;

j > m

j > m (0)

m/n < x

(m / n) < x (1)

j <= m >= n

((j <= m) >= n) (1)

++j == m != y * 2

((++j) == m) != (y * 2) (1)

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 Operatörleri

İfade	Sol terim: İkili	Sonuç: İkili	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 terim: İkili	Sonuç: İkili	Sonuç Değeri
$-5 \gg 2$	11111111 11111011	00111111 11111110	$2^{13} - 1$
$-5 \gg 2$	11111111 11111111	11111111 11111110	-2

? koşullu operatörü

- Üç terimli tek operatördür.
- if..else dallanması için kısayoldur.

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

if (x<y)

z = x;

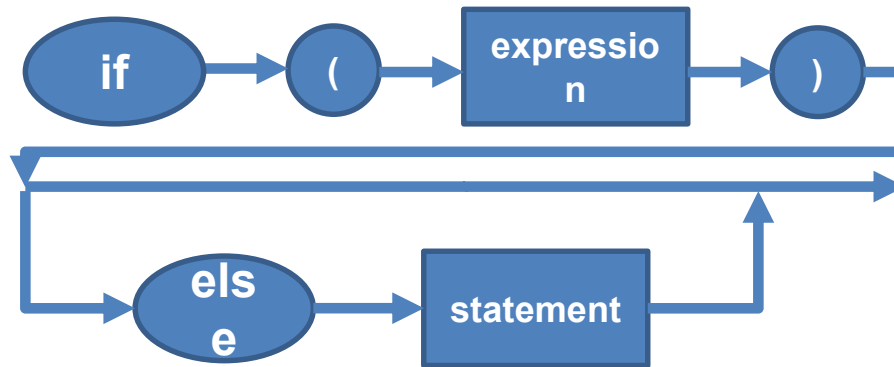
else

z = y;

Bellek Operatörleri

Operatör	Sembol	Örnek	İşlem
Adresi	&	&x	x'in adresini al
Dereference	*	*a	a adresindeki varlığın değerini al
Dizi elemanı	[]	x[5]	Dizinin 5 elemanının değerini al
Nokta	.	x.y	X struct'ındaki y elemanının değerini al
Sağ-Ok	->	p -> y	p ile gösterilen adresteki struct'ın y elemanının değerini al.

if...else ifadesi



Ex1 :

```
if (x)
    statement1;
    statement2;
```

// x !=0 ise işletilir
// her zaman işletilir

Ex2:

```
if (x)
    statement1;
else
    statement2;
    statement3;
```

// x !=0 ise işletilir

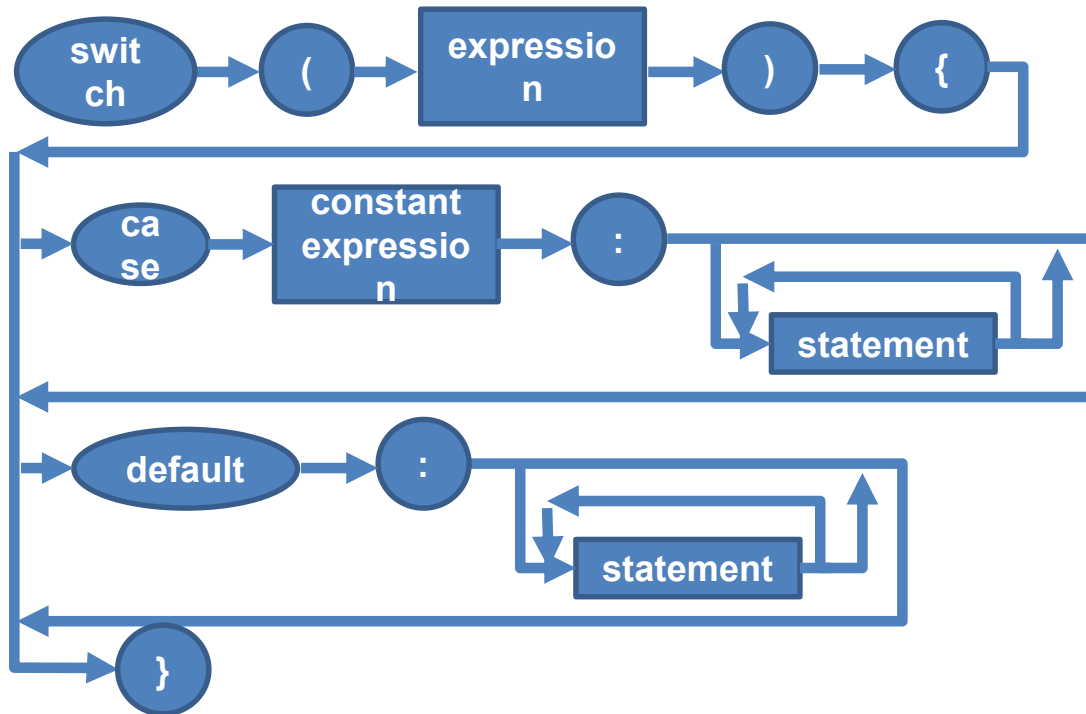
// x ==0 ise işletilir
// her zaman işletilir

İççe if ifadeleri

- Düzgün kodlanmazsa «Dangling else» problemi yaşanabilir!
 - Bir else, her zaman ona en yakın öncül if ile ilişkilendirilir.

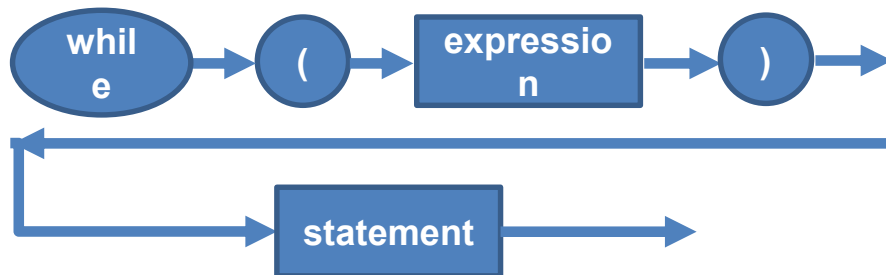
```
if(a<b)
    if(a<c)
        return a;
    else
        return c;
else if (b<c)
    return b;
else
    return c;
```


switch ifadesi



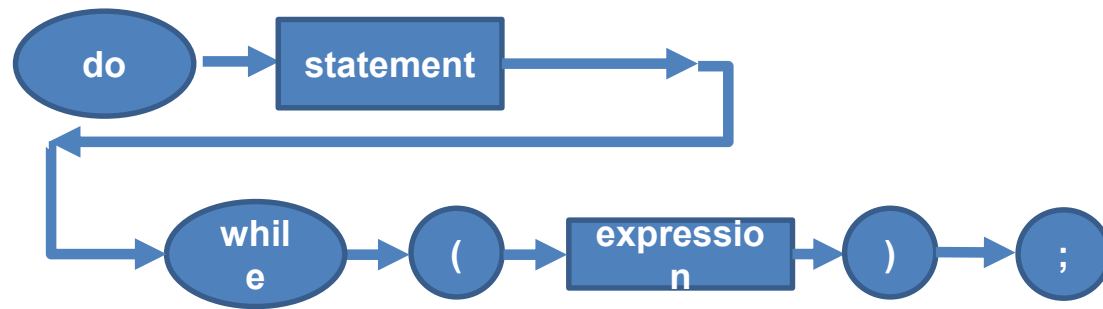
- **switch** ifadesi şu şekilde işletilir:
 - karşılaştırma koşulu case etiketlerinden biri ile eşleşirse, oradaki ifadeler işletilir.
 - Hiçbir koşul ile eşleşmeze (eğer var ise) default etiketi ifadeleri işletilir.
- İki ayrı durum aynı etiketi taşıyamaz.
- default son koşul olmak zorunda değildir ama sona koymak iyi bir pratiktir.

while ifadesi



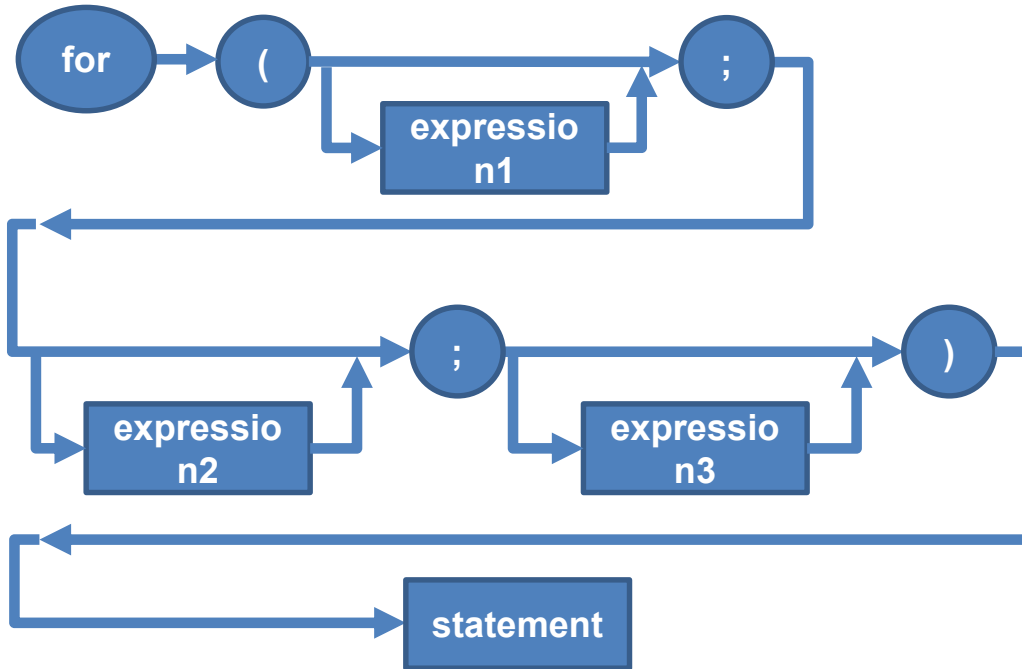
- Önce koşul işletilir. Sıfırdan farklı bir değerse, bloktaki ifadeler işletilir.
- İfade işletildikten sonra koşul yine kontrol edilir.
- Koşul 0 olana kadar ifade bloğu işletilmeye devam edilir.

do...while ifadesi



- Normal while döngüsünden tek farkı, koşul testinin ifade bloğunun sonunda işletilmesidir.
 - Programda ifade bloğu en azından bir kere mutlaka işletilmiş olur!

for ifadesi



- **expression1** önce işletilir.
- Sonra **expression2** işletilir.
 - İadenin koşullu kısmıdır. .
 - **expression2 yanlış** ise, program for bloğunun dışına çıkar.
 - **expression2 doğru** ise, **statement** işletilir.
- **statement** işletildikten sonra, **expression3** işletilir.
- Döngü bloğu, **expression2** testine geri döner.

break & continue & goto

- **break**

- **switch** ifadesinde bahsettik.
- Döngüde kullanıldığında, programın döngüyü takip eden satırdan işletilmesine devam edilmesini sağlar.

- **continue**

- Döngünün başına normalden önce dönmeyi sağlar.
- Bir sebepten dolayı döngünün kalanını işletmek istemiyorsanız kullanabilirsiniz.
- Kullanılması pek hoşnutlukla karşılanmaz.

- **goto**

- İlkel / Öncül dillerde kullanımı bir zorunluluktur.
- C'de bunu kullanıyorsanız, ...

Gelecek Ders

- Algoritmik Problem Çözüm Temelleri
- Algoritma Tasarımı ve Analizi Örnekleri