

1. Giriş

Algoritmanın sahip olması gereken Özellikler

1. Giriş - akış bilgisi
2. Sonluluk
3. Kesinlik → anlaşırlık
4. Etkinlik → tekrar tekrar kullanılabilirlik
5. Başarım ve Performans → verimlilik : az enerji çok iş gibi

→ Düz yazı
BAŞLA
BITIR
-metin

→ Pseudo Kod (sözdə)
daha resmi

→ Akış Şeması

○	BASLA - BITİR
	Aritmetik işlemler
↙ ↘	Disaridan bilgi girişi
↗ ↖	Çıktı
↙ ↘ ↗ ↖	Koşul

- ↳ Matematiksel işlemler
- ↳ Karşılaştırma işlemleri < > <= >=
- ↳ Mantıksal işlemler || && !

- Tanımlayıcı → değişkenleri anlandırmak için kullanılan kelimeler
- Değişken
- Atama
- Sayı
- Döngü

- IDE → Yazılım geliştirme ortamı
- Derleyici → Kaynak kodu okur. Mantıksal - yazışsal hataları kullanıcıya gösterir.
Kaynak kodu makine koduna çevirir.
- Yorumlayıcı

- Syntax error → yazım hatası
- Runtime error → çalışma zamanı hatası
- Logic error → mantıksal hata — en tehlikeli olanı —

2. C Diline Giriş

- Donanım & işletim sisteminden bağımsızdır.

Kaçış Karakterleri :

\n	yeni satır	newline
\r	satırbaşı	carriage return
\t	tab	tab
\v	dikey tab	vertical tab
\b	backspace	backspace
\a	uyarı sesi	alert
\"	tırnak işaretİ	

C dili elementleri :

Tanımlayıcılar - identifiers -

A-Z a-z _ 0-9
 bunlarla başlayabilir
 bunları içerebilir

Anahat sözcükler - keywords - → bu kelimeler tanımlayıcı olarak kullanılamaz.

Veri türleri - data types - → bellekte ayrılanın boyutunun belirlenmesi.

%s	c	↳ char	1	
%d	d	↳ int	4	short : yarıya indirir
%f	f	↳ float	4	long : ikiye katlar
%lf	lf	↳ double	8	unsigned : saklanacak değerin sıfır ve sıfırdan büyük olması sağlanır
		↳ void		

Değişkenler - variables -

i++ → i'yi işlemde kullandıkten sonra 1 arttır.
 ++i → i'yi 1 artır, sonra işlemede kullan.

Sabitler - constants -

printf ("% ... %d ... ", sayı1); scanf ("%d", &sayı2);

Operatör Önceliği

1. () [] → üye erişimi
2. * (pointer) & (adres) type() sizeof()
3. */ /
4. + -
5. < <= >= >
6. == !=
7. &&
8. ||
9. ?:
10. = += -= *= /= *=
11. , (gökku ifade ayrımı)



Extra

Not

|| : Koşulun başı doğrusa 2. kısım kontrol edilmez.

&& : Koşulun başı yanlışsa 2. kısım kontrol edilmez.

x=4;
 if (x>3) || (y/0 == 8));
 ↴ y/0 için error vermez
 çünkü ilk şart doğru olduğundan 2. kısma bakma! Ama x=0 olsaydı hata verirdi.

3. Koşul + Ternary

if () {

_____;
 _____; } burada tek ifade kullanılabilecek { } kullanmasın da olur.
 _____; ama kullanmak iş ortamı ve profesyonellik açısından garanti olur.
 }

else → en yakın if'e bağlanır

switch-case → char ve int kullanılabilsin!

→ şartı sağlayan case'le sıra, break görene kadar, break yoksa blok bitene kadar oku.

```
sayi = 2;  
switch (sayi) {  
    case 1 :  
        ==
```

case 2 : ← şartı sağlayan case' e sıradır.

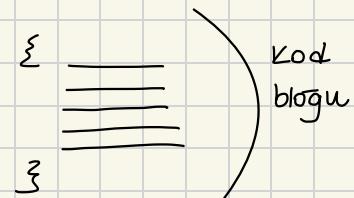
case 3 :
 ==
} ← break olmadığı için blok bitene
 kadar kodları okudu.

- sadece sağlanan case okunsun diye her case sonuna "break;" koy!
- switch case bloğunun sonuna default eklemek zorunluluk değildir ama güvenlik için önemlidir.

```
sayi = 2;  
switch (sayi) {  
    case 1 :  
        ==  
    case 2 :  
        ==  
        break;  
    case 3 :  
        ==  
        break;  
    default :  
        break;  
}
```

blokten akitildi

↓ bloğun altındaki kodlar okunmaya devam ediliyor.
sadece case 2 okundu



islevsellik

if - else



switch - case



char - int kısıtlaması
 $x > 100$ gibi karşılaştırmaların
olmaması

performans



okunabilirlik



Ternary (? :) → (koşul) ? (deyim 1) : (deyim 2)

4. Döngüler

break; döngüden çıkar.

continue; continue ve blok bitisi arası kodları galistirmeden döngünün başına döner.

break ve continue, kendine en yakın for - while - case 'e bağlanır.

```

for ( i , i , ) {
    dekam   sayac
    koşulu  güncelleme kodu
}

```

döngüdeki deyimler

do {

döngüdeki deyimler

} while (koşul);

}

5. Diziler

Tanım:

tip ad [boyut veya eleman sayısı]

int sayılar [5] = { 0, 1, 2, 3, 4 }
 ↓
 0. dizi
 elemanı 1. dizi
 ↓ elemanı
 dizinin
 1. elemanı

int a [] = { 100, 200, 300 }
 ↓

kendisi 3 elemanlı olduğunu hesaplar

int sayılar [20] = { 0 } ;

tüm dizi elemanlarına

0 atanır.

int sayılar [20] = { 1, 2, 3 } ;

dizinin ilk 3 elemanına 1, 2, 3

değerleri atanır. Diğer elemanlara 0 atanır.

Arama Teknikleri :

1. Doğrusal Arama (Linear Search) : Dizinin ilk elemanından başlanarak dizinin her elemanı, aranan değer ile karşılaştırılır.

```

for( int i=0 ; i<100 ; i++ ) {
    aranacak_dizi[i] = i ;
}
scanf( "%d" , &aranan_bilgi );

for( int i=0 ; i<100 ; i++ ) {
    if ( aranan_bilgi == aranacak_dizi[i] ) {
        bulunan_indis = i ;
        break ;
}
}

```

linear search örneği

2. ikili Arama (Binary Search) : Aranan değerin, dizinin orta değerine eşit olup olmadığını bakılır. Eşit değilse orta değerden fazla veya az olmasına göre 2 yarından birinde aynı kontrolleri sağlar.

```

while( bilTahmin != aranacakSayi) {
    if ( bilTahmin < aranacakSayi) {
        altSinir = bilTahmin +1;
    } else if ( bilTahmin > aranacakSayi) {
        ustSinir = bilTahmin - 1;
    }
    bilTahmin = (altSinir + ustSinir) / 2 ;
}
printf("sayi bulundu!");

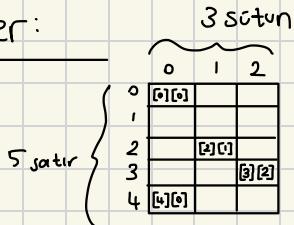
```

Binary Search Örneği

Gök Boyutlu Diziler:

```
int y[5][3]
```

satır sütun



```
int y[5][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};
```

0	1	2
0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

Matris Çarpımı Kod Örneği:

3x4

4x3

i	0,0	0,1	0,2	0,3

j	0,0	0,1	0,2
0,0	0,1	0,2	0,3
1,0	1,1	1,2	
2,0	2,1	2,2	
3,0	3,1	3,2	

0	1	2
1		
2		

3x3

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

```
int main () {
```

```
int matris1 [3][4] = { 1,2,3,4, 5,6,7,8, 9,10,11,12};
```

```
int matris2 [4][3] = { 1,2,3, 4,5,6, 7,8,9, 10,11,12};
```

```
int matris3 [3][3] = {0};
```

```
for (int i=0 ; i<3 ; i++) {
```

```
    for ( int j=0 ; j<3 ; j++) {
```

```
        for ( int k=0 ; k<4 ; k++) {
```

```
            matris3[i][j] += matris1[i][k] * matris2[k][j];
```

```
}
```

```
}
```

```
for (int i=0 ; i<3 ; i++) {
```

```
    for ( int j=0 ; j<3 ; j++) {
```

```
        printf("%d ", matris[i][j]);
```

```
}
```

```
return 0;
```

6. Karakter Dizileri

Sonlandırıcı karakter : \0 → dizinin bittiği yeri gösterir.

```
char dizi [7] = {'d','e','n','e','m','e','\0'};
```

```
char dizi [7] = {"deneme"}; → sonlandırıcı karakter derleyici tarafından konur.
```

gets(); → klavyeden karakter dizisi almak için kullanılır.

gets(dizi); → içini dolduracağı dizi adı

yeni
kütüphane
include
etmeye
gerek
yok.

strlen(dizi); → 6 değerini döndürür.

\0 değeri sayılmaz.

```

strcat( dizi1 , dizi2 ); // dizi1 dibine dizi2 'yi ekler.
strcpy ( dizi1 , dizi2 ); // dizi1 iini temizleyip dizi2 içeriğini yapıştırır.

strcmp ( dizi1 , dizi2 ); // 2 dizi de aynıysa 0 , farklıysa 1 döndürür.

strrev( dizi1 ); // diziyi tersine çevirir.

```

```

char dizi [ ] = "Merhaba";
printf ("%s", dizi);

```

char dizi [7] → bellekte 7 byte'lik yer ayırır.
 "deneme" + '\0'

char dizi [6] = { "deneme" }; → bu durumda '\0' karakterine yer kalmayacak.

Bellek taşıması
(undefined behavior)

'\0' karakteri , bellekte
ayırılan yerin sınırlarını aşarak
dizinin dışına yazılabilir.

printf ("%s", dizi);

'\0' karakteri bulana kadar okumaya
devam eder.
→ denemeXG&13*AB
gibi bir çıktı alabilirsin.

Eğer diziyi { "....." } şeklinde başlatırsan C derleyicisi otomatik olarak dizi sonuna '\0' karakterini ekler.

```

dizi[0] = 'd'
dizi[1] = 'e'
dizi[2] = 'n'
dizi[3] = 'e'
dizi[4] = 'm'
dizi[5] = 'e'
dizi[6] = '\0' derleyici bunu otomatik ekler!

```

AMA

diziyi sonradan dolduruyorsan (döngüyle) > derleyici '\0' ekmez!
 diziyi boyutla tanımlayıp iñin boş bıraklığında
 diziyi sonradan doldururken '\0' eklemek senin sorumluluğundadır.

```

int main () {
    char dizi [7];
    for ( int i=0 ; i<6 ; i++ ) {
        dizi [i] = 'A' + i ;
    }
    dizi [6] = '\0';
    printf ("%s", dizi);
    return 0;
}

```

char dizi [5][4] = { "ABC", "DEF", "GHI", "JKL", "MNO" };

	0	1	2	3
0	'A'	'B'	'C'	'\0'
1	'D'	'E'	'F'	'\0'
2	'G'	'H'	'I'	'\0'
3	'J'	'K'	'L'	'\0'
4	'M'	'N'	'O'	'\0'

'\0' karakteri
satır sınırlarına
otomatik olarak
eklenir. Yukarıdaki
AMA geçerli

```

for ( int i=0 ; i<5 ; i++ )
    printf ("%s", dizi[i]), // satırları yazdırır.
}

```

GİKTİ:

'ABC'
'DEF'
'GHI'
'JKL'
'MNO'

NULL ('\0') karakterine yer ayırmayı
hattırla !

7. Fonksiyonlar

• Böl ve ele geçir (divide & conquer)

Fonksiyonun
dönüşüregi
deger

fonksiyonun prototipi

fonksiyonun imzası

int fonk (int a , int b)

parametre Lisesi

fonksiyonun yerel değişkenidirler.
Sadece fonksiyonun içinde tanımlılar.

Fonksiyonun Tipi : int

Fonksiyonun Adı : fonk

Fonksiyonun Aldığı Parametreler : a, b

Fonksiyonun Döndürdüğü Deger : 0

3
 _____;
 return 0; → döndürülen int deger.

Fonksiyonun bildiriminde

fonksiyona girdi olarak

kullanılan değişkenlere

"parametre" denir

fonksiyon, main içinde çağırılırken gönderilen değerlerine "argüman" denir.



```
void main () {  
    int sayi1 = 2;  
    int sayi2 = 5;  
    fonk (sayi1, sayi2);  
    argüman  
}
```

3

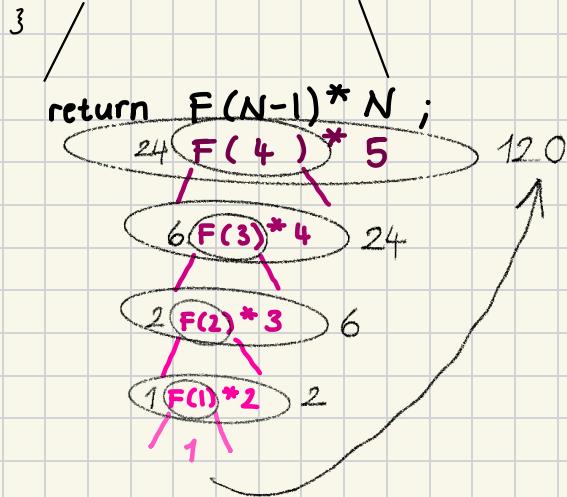
char dizisinin elemanları kopyalanmaz, metnin adresi kullanılır. Aslında char dizileri, dizinin ilk elementinin adresini tutan bir pointerdir.



RECURSIVE Özrnelemeli Fonksiyon → Kendi kendini çağırın fonksiyon.

Faktöriyel Örneği ($N = 5$ olarak kullanıcıdan aldığımizi varsayıyalım)

```
int F(int N){  
    if (N == 0 || N == 1) // 0!=1 1!=1  
        return 1;  
    else  
        return F(N-1)*N;  
}
```



$N=1$ olana kadar fonksiyon devamlı kendini çağrıdı.
Ne zaman ki $N=1$ çıktı yukarıdaki if içine girip $F(1)$ değeri için 1 döndürdü. Sonra fonksiyonu yukarı doğru kapatmaya başladık ve SONUÇ 120 çıktı.

Vineleme varsa → recursive bellekte çok yer tutar
Tekrar varsa → döngü

8. Esnek Argümanlı Fonksiyonlar (Variadic Functions)

#include <stdarg.h>

<https://www.notion.so/C-small-notes-4c9fff5867604c32ade0f373bd4de37f?pvs=4>

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

```
int Topla ( int miktar , .. ) {
```

```
    va_list args ; // va_list tipinde, "args" adında, değişken argümanları tutan bir liste oluşturur  
    int geciciToplam = 0 ,  
        va_start ( args , miktar ); // miktar → değişken argümanlardan önceki son sabit parametredir. "args" listesi, değişken argümanları  
        // tutacak, start ile bu değişken argümanların "miktar" değişkeninden hemen sonra başladığını bildiriyoruz.  
    for ( int i = 0 ; i < miktar ; i ++ ) {  
        geciciToplam += va_arg ( args , int ); // argüman listesinden, bir sonraki değeri int türünde al.  
    }  
    // va_arg → argüman listesindeki (args) sıradaki elemanı otomatik olarak okur.  
    // argüman listesinden, bir sonraki değişkeni "int" tipinde al.  
    va_end ( args ); // Kullanımı bittiğten sonra argüman listesini sonlandırır ve temizler.  
    return geciciToplam ;  
}
```

```
int main () {
```

```
    int sonuc1 = Topla ( 3 , 10 , 20 , 30 ); // 3 argüman : 10 + 20 + 30  
    int sonuc2 = Topla ( 5 , 1 , 2 , 3 , 4 , 5 ); // 5 argüman : 1 + 2 + 3 + 4 + 5  
  
    printf ( " Sonuc1 : %d \n " , sonuc1 );  
    printf ( " Sonuc2 : %d " , sonuc2 );  
    return 0 ;
```

"va_arg" makrosu, argümanı hangi türde okuması gerektiğini belirtmeni ister.
Bunun nedeni, C dilinin değişken argümanlarının türlerini otomatik olarak anlamamasıdır. Argümanlar bir nevi "ham" veri olarak aktarılır ve tür bilgisi taşınmaz.

MAİN'E PARAMETRE AKTARIMI

Parametreli main fonksiyonunu eğer programı komut satırından çalıştırıcasan ve kullanıcıdan veri almak gerekiyorsa kullanırsın.
Girilen her şey string olarak algılanır. Eğer tür değiştirmek isterse, sonrasında atoi(), atol(), atof() fonksiyonlarını kullanabilirsin.

- atoi(): ASCII to integer
- atol(): ASCII to long integer
- atof(): ASCII to double

→ int argc : Komut satırından girilen argüman sayısını tutar.

ilk argüman daima programın adı (veya yolu) olduğu için bu değer her zaman en az 1 olacaktır.

→ char *argv[] : Komut satırından girilen argümanları tutan bir dizi pointer'ıdır.

argv[0] : program adını icerir.

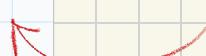
argv[1], argv[2], ... : Komut satırından girilen argümanlardır.

```
int main( int argc , char *argv[] ) {  
    if (argc < 2) {  
        printf("En az 2 sayı girmelisiniz!");  
        return 1;  
    }  
    int toplam = 0 ;  
    for ( int i = 0 ; i < argc ; i ++ ) {  
        toplam += atoi ( argv [ i ] );  
    }  
    printf ( "Toplam : %d \n " , toplam );  
    return 0 ;
```

dinamik olarak

terminal üzerinden kullanıcıdan

sınırsız sayı alıp toplam fonksiyonu.



Eğer zaten ağaçın dosyanın dizin yolundaysan direkt ağaçın dosyanın adını yazabilirsin.

```
gcc program.c -o topla.exe  
topla.exe 3 5
```

Toplam: 8

Ama akasız bir dizindeyse dosyanın tam yolunu yazabilirsin. ("cd" kullanarak önce o dizine gidip ilk yolu da kullanabilirsin.)

C://Desktop/projeler/program.c → dizin örneği

9. Dinamik Bellek

#include <stdlib.h>

Değişkenlerin 4 temel özelliği vardır.

- adı
- tipi
- değeri (igeriği)
- adresi



→ sayı

→ int

→ 7

→ 4 baytlik bir adres

Fazla bellek ↗

Fazla maliyet

Yavaş program

Statik Dizi : dizinin boyutu ve eleman sayısı programın çalışması esnasında değişmez. → char dizi[50]

Dinamik Dizi : program çalışırken dizinin boyut ve eleman sayıları bazı koşullarla değiştirilebilir.

dinamik dizilerde bellek bölgesi → işletim sisteminden istenir:

→ Kullanılır.

→ boşaltılır (opsiyonel)

Dinamik Bellek Yönetimi

- i. malloc - memory allocation -
- ii. realloc - re allocation -
- iii. calloc - clean allocation -
- iv. free - FREE ↴

char kelime[20]; // statik

int sayı=20;

char *kelime2 = (char *) malloc(sayı * sizeof(char)); // dinamik - heap memory -

char kelime[20]=0, // içini sıfırladık

for (int i=0; i<sayı; i++) { // içini sıfırladık
 kelime2[i]=0;

char * kelime3 = (char *) calloc(eleman, sayısı); // içi tamamen 0 olan bir dizi verir.

dinamik diziyle işin bittiye ve bellekte yer tutsun istemiyorsan silebilirsin. Program verimini artırır.

free(kelime3);

bütün değiştirmen gerekiyor :

kelime2 = (char *) realloc(kelime2, 25);

ikisi de birbirine.

kelime2 = (char *) realloc(kelime2, 25 * sizeof(char));

denktir.

malloc - calloc - realloc → geriye bellek adresi döndürür.

free(); kaymasan program bittiğinden sonra bir süre daha bellekte tutulup sonra silinir. Ama free kaymak daha profesyonelcedir 😊

10. Pointer

pointer → bir değişkenin adresini tutan bir değişkendir.

* → yönlendirme operatörü. Bu değişkenin veri değil, bir adres tutucuştur.

char *diz → diz, tek bir karakterin adresini tutar.

int *sayi → sayı, bir tamsayının adresini tutar.

& → address of : sağında yazan değişkenin adresini ifade eder.

* → value at : sağında yazan değişkenin tuttuğu değeri ifade eder.

int sayı = 7;

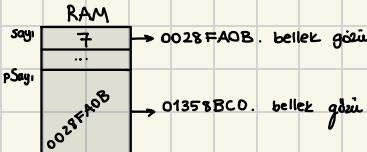
sayı → 7

int *pSayı;

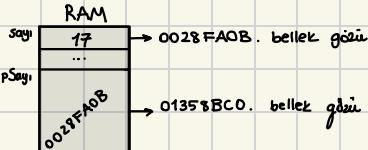
& sayı → 0028FA0B

pSayı → 0028FA0B

* pSayı → 7



* pSayı = 17;

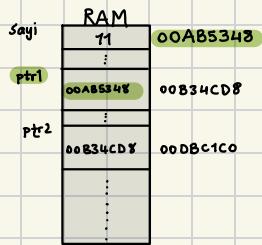


int *p;

p = NULL; // p, bellekte hiçbir adresi göstermiyor.

*p = 19; // hata alırız. Bellekteki "0" bölgisi degistirilemez!

Pointer Zinciri



int sayı = 11;

int *ptr1 = & sayı;

ptr1'in veri tipi → int * → yani int adresi tutuyor.

int **ptr2 = & ptr1;

ptr2'nin veri tipi → int ** → yani int adresi tutanın adresini tutuyor.

* ptr1
*(00AB534B)

11

Fonksiyona Pointer Parametre Aktarmak

Fonksiyona parametre olarak pointer gönderirsen "Pass by Value" (kopýalanarak olan) değil "Pass by Reference" ile işlem yapılır ve yapılan işlemler gerek parametreyi değiştirir. Yani pointer'in gösterdiği adresdeki değeri.

void F1(int n){

n=19;

void F2(int*a){

*a=17; → a pointerının tuttuğu adresdeki değere 17 ata.

void main(){

int sayı = 1;

int *pSayı = & sayı;

F1(sayı); → Fonksiyon bittiğten sonra sayı değeri korunmaz ve yeniden 1 olur. Pass by Value

F2(pSayı); → Fonksiyon bittiğten sonra sayı değeri korunur ve sayı değeri kalıcı olarak 17 olur. Pass by Reference

3

Fonksiyon İsaretçisi

int (*F)(int, int); → F, 2 int alan, 1 int döndüren bir fonksiyon pointer'dır.

F = & Topla; → Topla, 2 int isteyen bir fonksiyon olmak üzere

Artık F, 2 int alan, 1 int döndüren "Topla" fonksiyonunun pointer'i.

Topla(x,y); → iki şekilde de fonksiyonları main içinde çağırabilirsin.

F(x,y);

0. adres işletim sistemi tarafından

Kullanılır ve erişime izin verilmez.

void main(){

int dizi[] = {10, 20, 30, 40};

int *p = & dizi[0];

int **p2 = & p;

int ***p3 = & p2;

printf("%d", *p);

printf("%d", **p2);

*(0x12345678)

10

// * (value at) operatörü +,- 'ye göre

Önceliklidir!!

printf("%d", *(**p3+2));
* (*(0x3456789A)+2)
* (*(0x23456789)+2)
* (0x12345678+2)
*(0x1234568F)

30

offset

Bir fonksiyona parametre olarak fonksiyon aktaracaksan kullanabilirsin.

+

Fonksiyonları dinamik olarak çağırırmak

```

#include <stdio.h>
int Topla ( int a , int b ) {
    return a + b ;
}
int Corp ( int a , int b ) {
    return a * b ;
}

int main () {
    char islem;
    int (*Fonk) ( int , int );
    printf (" Islem turunu secin : ");
    scanf (" %c ", &islem);
    if ( islem == '+' )
        Fonk = Topla;
    else if ( islem == '*' )
        Fonk = Corp;
    printf (" Sonuc : %d ", Fonk ( 3 , 4 )), 
}

```



Void Pointer : Herhangi bir veri tipine sahip olmayan işaretçiler.

Genel işaretçi olarak da adlandırılırlar

→ Mesela 2 int alan ve hiçbir şey içermeyen bir fonksiyonun pointerini oluşturacaksın

```

void (*F)( int , int );
F = &fonksiyon;

```

STRUCT POINTER

Belleği verimli kullanmak

Fonksiyonlar arasında verileri daha etkili kullanmak.

1. Bellek Tasarrufu

· Büyüük bir struct doğrudan kopyalanırsa bellekte fazladan yer kaplar.

· Ancak bir struct pointer kullanıldığında yalnızca yapı adresi taşınır, böylece daha az bellek kullanılır.

```

struct Student {
    char name[50];
    int age;
};

void printStudent ( struct Student * s ) {
    printf (" Name: %s , Age: %d ", s->name , s->age );
}

int main () {
    struct Student s = { "Vera" , 20 };
    printStudent (&s); // Böylece tüm yapıyı değil sadece adresi gönderiyoruz
    return 0;
}

```

```

printf (" Name: %s , Age: %d ", s->name , s->age );
(*s).name , (*s).age

```

2. Fonksiyon Performansı

· Bir struct, fonksiyona parametre olarak geçirildiğinde, tüm üyeleri kopyalanır.

· Fonksiyon içinde struct üyelerine yapmak istediğiniz her işlem de bu kopyalar üzerine yapılır. (Call by Value)

3. Dinamik Bellek Kullanımı

· Pointer'lar sayesinde bir struct, programın çalışma zamanında (runtime) dinamik olarak oluşturulabilir ve boyutlandırılabilir.

```

#include <stdio.h>
#include <stdlib.h>

typedef struct {
    char name [50];
    int age;
} Ogrenci;

int main () {
    Ogrenci * s = (Ogrenci *) malloc ( sizeof ( Ogrenci ) );
    if ( s == NULL ) {
        printf (" Bellek tahsis edilemedi ");
        return 1;
    }
}

```

```

    s->age = 21;
    strcpy ((s->name), "Vera");
    printf ("Name: %s, Age: %d", s->name, s->age);
    free (s);
    return 0;
}

```

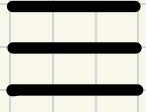
11. STRUCT

```

struct bilgiler {
    char isim[50];
    int yas;
};

typedef struct bilgiler Ogrenci;

```



```

typedef struct {
    char isim[50];
    int yas;
} Ogrenci;

```

```

int main () {
    struct bilgiler ahmet; // 2 türde oluşturulabilirsin.
    Ogrenci mehmet;
}

```

strcpy (ahmet.isim, "Ahmet");
ahmet.yas = 23;

* mehmet = ahmet; **struct kopyalamak mümkün**. ahmet içeriği bozulmadan, mehmet içine kopyalandı.

* mehmet == ahmet; **DİYEMESEN** Karşılaştırmayı üyeler üzerinden yapabilirsin.

* mehmet.isim == ahmet.isim; **şeklinde karşılaştırma yapabilirsin.**

* struct tipinden bir değişken, fonksiyona parametre olarak atanabilir.

```

void Yazdir (Ogrenci ogr) {
    printf ("Isim: %s, Yas: %d", ogr.isim, ogr.yas);
}

int main () {
    Ogrenci kisi0 = {"Vera", 20};
    Yazdir (kisi0);
}

```

Mesela, buraları
pointer kullanırsın
daha verimli bir kod
olurdu.

* bir fonksiyon, geri dönüş değeri olarak struct tipinde bir veri türü geriye döndürebilir.

```

Ogrenci ogrenci_olustur (char isim[], int yas) {
    Ogrenci kisi1;
    ...
    return kisi1;
}

```

* struct tipi içerisinde dizi tipinde üyeler tanımlamak mümkün.

```

typedef struct {
    char isim [50];
    int vineNotu [2];
}

```

X Dizileri struct tipinde tanımlamak mümkün.

```
struct OgrenciNot {  
    char isim [50];  
    int vize [2];  
    int final;  
}  
  
int main () {  
    struct OgrenciNot ogr [100] // struct OgrenciNot tipine sahip , ogr adında 100 elemanlık bir dizi oluşturuldu.  
    strcpy (ogr [0]. isim), "Vera");  
    ogr [0]. vize [0] = 85;  
    ogr [0]. vize [1] = 92;  
    ogr [0]. final = 100;  
    return 0;  
}
```

X struct içinde struct tipinde bir üye bulunması mümkün değildir.

- Kaynak kodun tekrar kullanılabilirliğini artırmak.
- Kaynak kodun okunabilirliğini artırmak + sadeleştirmek.

```
typedef struct {  
    int vize;  
    int final;  
    int quiz [2];  
} Notlar;  
  
typedef struct {  
    char tel [50];  
    char mail [50];  
} Iletisim;
```

```
typedef struct {  
    int no;  
    char isim [100];  
    Notlar Not;  
    Iletisim OgrIletisim;  
} Ogr;
```

```
int main () {  
    Ogr ogr;  
    ogr.no = 1;  
    strcpy (ogr. isim, "Vera");  
    ogr.OgrNot. vize = 78;  
    ogr.OgrNot. final = 98;  
    ogr.OgrNot. quiz [0] = 56;  
    ogr.OgrNot. quiz [1] = 73;  
    strcpy (ogr.OgrIletisim. tel, "0555 555 5555");  
    strcpy (ogr.OgrIletisim. mail, "ellewoods@gmail.com");  
}
```

TYPEDEF

veri türlerine takma ad takmamızı sağlar.

```
typedef float ondalikli;  
void main () {  
    ondalikli not = 92.05;  
}
```

Structlarda
daha çok
kullanıyoruz.

UNION

değişkenler aynı bellek bölgelerini ortaklaşa kullanırlar.
en büyük alan kaplayacak türün boyutu kadar bellekte yer tutar.

```
typedef union {
    int x;
    char y [10];
} Birlik;
```

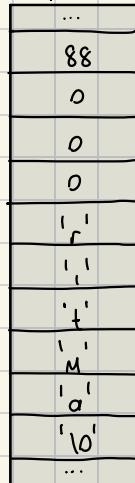
```
void main () {
    Birlik c;
    strcpy (c.y, "Algoritma");
    c.x = 88;
}
```

iki veya daha çok değişkenden biri tutulacağa kullanılır.

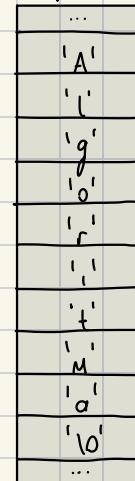
Mesela, kullanıcıdan eposta ve telefon numarasından sadece birini girmesini istediğimiz bir durumda hem tel no hem de eposta için bellekte boş yere yer tutulmuştur.

Bellekte 14 değil
10 byte yer tutulur.

RAM



RAM



Birlik c

12. DOSYALAR I

Değişken ve diziler içinde depolanan veriler bellekte tutulur ve geçicidirler. Program sonlanınca bellekten kaybolurlar.

Dosyalar büyük miktarda veriyi kalıcı olarak tutmak için kullanılırlar.

Dosyalar, ikincil depolama cihazlarında, özellikle de disk depolama cihazlarında tutulurlar.

Her dosya, dosya sonu belirteci (end of file EOF) ile sonlanır.

1. Dosya aç
2. Dosya yaz
3. Dosya Kapat

1. Dosya aç.

```
FILE *fptr;
```

```
fptr = fopen ("dosya.txt", " ");
```

dosya yolu ve adı, eğer yol

dosyanın modu - açılma amacı

girmeden sade oluşturacagın

dosyanın adını girersen bulunacağı

dizinde o dosyayı oluşturur.

"C:\\Users\\verad\\OneDrive\\Masaüstü\\c-dersleri\\data.txt"

① yerine ② şeklinde yazmalısın!

3. Dosya kapat :

```
fclose (fptr);
```

```
if (fptr == NULL) {
    printf("dosya açılamadı!");
    exit(1);
} else
    printf ("dosya açıldı");
```

→ dosyanın açılıp açılmadığının
kontrolü

exit() → <stdlib.h>

"r" : read only : • sadece okunur

- dosya mevcut değilse hata verir.
- dosya işaretçisi başta olur.
- dosya varsa okur.
- mevcut icerik korunur.
- yazma işlemi yapılamaz.

"w" : write only : • sadece yazma

- eğer dosya zaten varsa içeriği tamamen silinir.
- eğer dosya yoksa yeni bir dosya oluşturulur.
- dosya işaretçisi başta olur.
- mevcut icerik silinir.

"a" : append : • ekleme

- dosya yoksa oluşturulur.
- dosya varsa, dosya içeriği ile birlikte açılır.
- mevcut icerik korunur.
- dosya işaretçisi sondadır.
- yazılılan veriler dosya sonuna eklenir.

"r+" : okuma + yazma

- dosya mevcut değilse
hata verir
dosya işaretçisi başta olur.
mevcut icerik korunur.

"wt" : yazma + okuma

- eger dosya zaten varsa içeriği tamamen silinir.
eger dosya yoksa yeni bir dosya oluşturulur.
dosya işaretçisi başta olur.
mevcut icerik silinir.

"at" : okuma + yazma

- dosya yoksa oluşturulur.
dosya varsa, dosya içeriği ile birlikte açılır.
mevcut icerik korunur.
dosya işaretçisi sondadır.
yazılan veriler dosya sonuna eklenir.

Mod	Amaq	Dosya Mevcut mu?	icerik	Dosya işaretçisi
r	sadece okuma	Gerekli	Korunur	Baş
w	sadece yazma	istege bağlı	Silinir	Baş
a	sadece ekleme	istege bağlı	Korunur	Son
r+	okuma ve yazma	Gerekli	Korunur	Baş
wt	okuma ve yazma	istege bağlı	Silinir	Baş
a+	ekleme ve okuma	istege bağlı	orunur	Son

2. Dosya yaz :

fputc() : dosyaya karakter yazma

fputc('A', fptr);

Yazılacak karakter. karakterin yazılacağı dosyanın pointeri.

Tam sayılarak verilir.

- != EOF

```
fputc('A', fptr); // 'A' karakterini dosyaya yazar.
fputc('\n', fptr); // yeni satır ekler.
```

```
char name;
for (name = 'A'; name <= 'Z'; name++) {
    putc(name, fptr);
    putc('\n', fptr);
}
```

3 // 'A' dan 'Z' ye harflerin yazımı.

fputs();

• string (karakter dizisini) dosyaya yazma

fputs("Merhaba Dünya! \n", fptr);

yazılacak string stringin yazılacağı dosyanın pointeri

- Başarılıysa yazılan karakter sayısı döner.

- Başarısızsa EOF döner.

```
char dizi [100];
gets(dizi); // Kullanicidan veri alma
fputs(dizi, fptr); // alınan veriyi dosyaya yazdırma
```

```
if (fputs(dizi, fptr) == EOF) {
    printf("Yazım hatası oluştu!");
}
```

} dosyaya yazılmışlığı kontrolü

```
fprintf();
```

- dosyaya yazmamızı sağlar.

```
int yas = 19;
fprintf(fptr, "Benim yasim : %d", yas);
```

yazılacak string
dosyanın pointeri
icinе
yazılacak

```
fwrite();
```

- binary moda dosyaya veri yazmak.

```
FILE *file = fopen("data.dat", "wb"); // ikili moda yazma
int numbers[] = {1, 2, 3, 4, 5};
```

```
fwrite(numbers, sizeof(int), 5, file);
```

yazılacak
verilerin başlangıç
adresini belirten pointer
yazılacak
öğenin boyutu
kaç elementi
yazılacak
dosya pointeri

```
fclose(file);
```

```
fread();
```

- binary moda dosyadan veri okuma

```
FILE *file = fopen("data.dat", "rb");
int readNumbers[5];
fread(readNumbers, sizeof(int), 5, file);
for (int i=0; i<5; i++)
    printf("%d", readNumbers[i]);
fclose(file);
```

```
char data[50];
```

```
gets(data); // kullanıcıdan veri girişi
```

```
fprintf(fptr, "%s", data);
```

```
fgetc(): dosyadan karakter okuma
```

```
fgetc(fptr);
```

karakterin okunacağı
dosyanın pointeri

- Dosya sonuna ulaşırsa EOF değerini döndürür.

- Okunan değerleri tam sayı olarak döndürür.

- \neq EOF

```
fgets();
```

- dosyadan string metin okuma

```
char buffer[50];
```

```
fgets(buffer, sizeof(buffer), fptr);
```

okunan karakterlerin saklanacağı dizi
en fazla okunacak karakter sayısı (n-1) karakter okunur, sonunda '\0' eklenir.

Verinin okunacağı kaynak
- stdin → klavye girişi
- fptr → dosya pointeri

- Başarısızsa NULL döner

- \neq NULL

```
printf("%c", fgetc(fptr)); // A yazar konsola
```

```
char buffer[50];
```

```
FILE *file = fopen("example.txt", "r");
```

```
while (fgets(buffer, sizeof(buffer), fptr) != NULL) {
    printf("%s", buffer);
}
```

dosya içinde
VERA
yazıyor.

buffer = "VERA"

Eğer GÖK satırıysa
dosyanın, döngüde yozdurmakisin.

feof()

- dosyanın sonundan gelinip gelinmediğinin kontrolü.

feof(fptr);

kontrol edilecek dosyanın pointeri

- 0 dönerse → dosyanın sonuna ulaşılmasını gösterir
- 1 dönerse → dosyanın sonu

fscanf()

- dosyadan veri okuma

char name[50];

int age;

fscanf(fptr, "%s %d", name, &age);

okuma

yapılacak dosyanın pointeri

printf("%s, %d", name, age);

} dosyada
Vera 19
yazdığını desinelim

bağıla denk geldiğinde scanf'de olduğu gibi, okunur, her kelime iain ayrı değişken tanımlanır gereklidir.

rewind();

- dosya imleci dön.

fseek()

SEEK_SET : dosya başını bas alır.

SEEK_CUR : mevcut bulunan yeri bas alır.

SEEK_END : dosya sonunu bas alır.

fseek(fptr, 10, SEEK_SET);

hangi	Kaç	dosya imlecinin
dosya ise	Karakter	başlangıç noktasını
onun	hareket	
pointeri	edileceksin	
(- olursa		
geriye		
gider)		

fseek(fptr, -8, SEEK_END); // imleç, dosya sonundan 8 karakter geri gider.

ftell()

- imlecin konumunu döndürür.

ftell(fptr);

} dosya pointeri

dosya kopyalama

```
FILE *fptr, *fptrcopy;
fptr = fopen("data.txt", "r");
fptrcopy = fopen("datacopy.txt", "w");
while (!feof(fptr)) {
    fputc(fgetc(fptr), fptrcopy);
}
```

while (!feof(fptr)) {

printf("%c", fgetc(fptr)),

}

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

int main ( int argc , char *argv[] ) {
```

```
    int sayi = atoi ( argv [2] );
```

```
    for ( int i=0 ; i<sayi ; i++ ) {
        if ( argv [1][i] == 'z' ) { —————→ ( argv [1] + i ) == 'z' → dersen 1.'nin adresine i
            printf ("z");
            break;
        } else {
            printf ("%c", argv [1][i]); —————→ ( argv [1] + i ) → 1.'nin adresine i ekler ve onu
        }
    }  
}
```

1 karakter yazdırılacağı için.

ekleyip onu 'z' ile kıyaslar.

yordur.