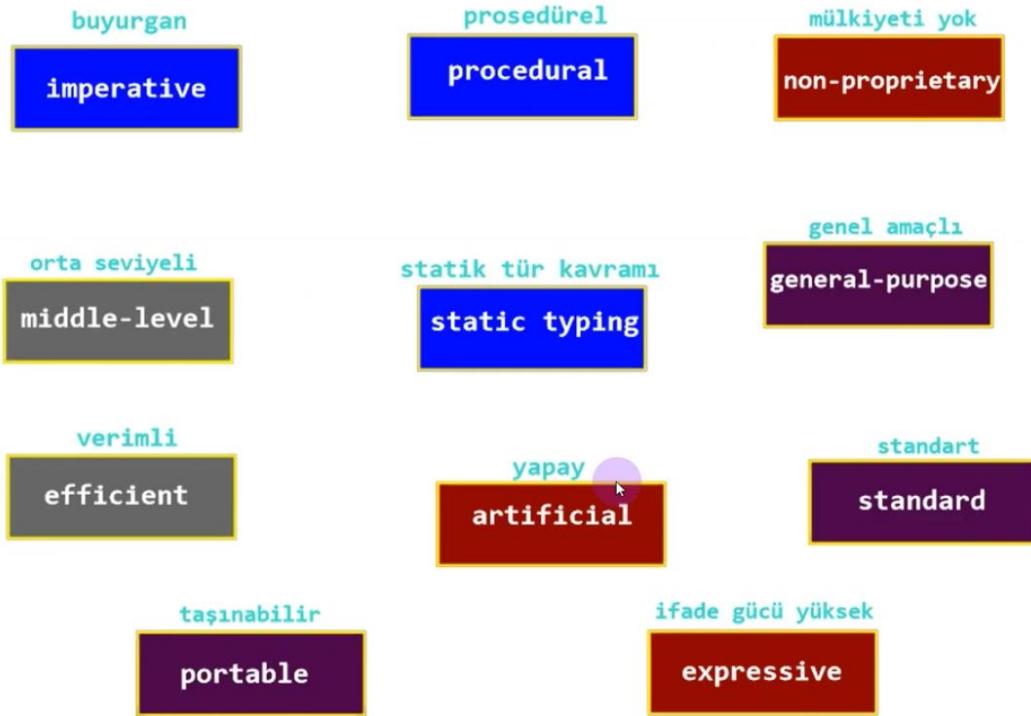


# C Programming Language



Writer : m.said modali

How to definite the c ?:



Compiler : it changes source code → machine language

Dynamic : Python(x = 5)

Static : c (int x = 5;)

C tahmini 1972 civarında icat edildi.

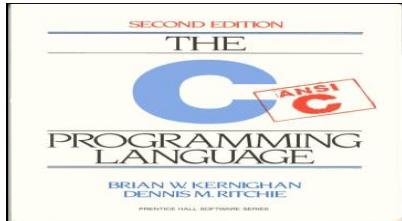


Ken Thompson  
1943 -



Dennis Ritchie  
1941 - 2011

Ve c yi anlatan kitap yazarları : k&r(c programming language)



The files occur from root and extension

Ali.py(root and extension)

Ahmet.h

Necati.c

Said.jpeg

**Compiler optimization** : derleyicinin aynı anlam için kafasına göre ve verimli bir şekilde istenen şeye ulaşmasıdır.

## Mechanism

**Systems :** binary(2),octal(8),decimal(10),hexadecimal (16)

### İşaretsiz 2'lik sayı sistemi sınır değerleri

Byte	Max
1	255
2	65.535
4	4.294.967.295
8	18.446.744.073.709.551.615

### İşaretli 2'lik sayı sistemi sınır değerleri

Byte	Min-Max
1	-128 / 127
2	-32.768 / 32.767
4	-2.147.483.648 / 2.147.483.647
8	-9.223.372.036.854.775.808 9.223.372.036.854.775.807

=====

Source code → translation unit → compiler(tokenizing) → object code (s) → link time → executable code

Tokens :

Keywords : int,if,while

Identifiers:

Case sensitive(a != A) and elements : ffunc() x,y...

Constants:

12,4

Operators: + , - , ?...

String literals : “abc1234(/&%+”

## Delimiters (?)

---

Expression :  $x + 5$

Statement :  $x + 5 ;$

Declaration : `int x ;`

An expression becomes

a-) data type : int or float .....

b-) value category(is it has a address ?):

1-) L value (yes for ex :  $x$ )

2-) R value (no for ex :  $++x$ )

expression	C	C++	I
$++x$	R	L	
$--x$	R	L	
$x, y$	R	L	
$a > 10 ? x : y$	R	L	
$x++$	R	R	
$x--$	R	R	

(differences)

Name lookup : isim arama

Name spaces : isim alanları ve ikiye ayrılır :

Global namespace

1-) Global namespace

Local namespace

2-) Local namespace

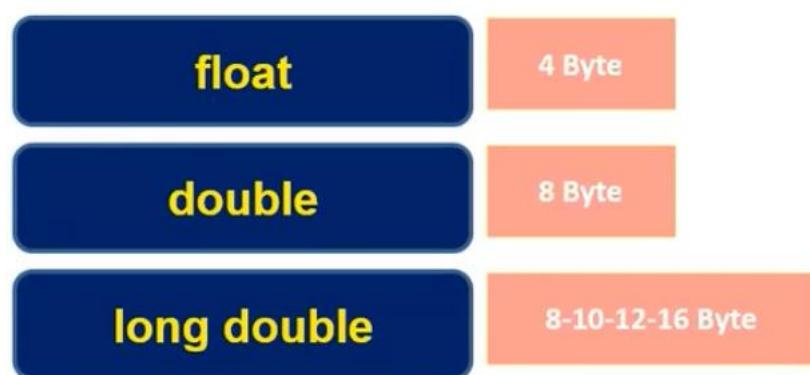
ascii -d								
0 NUL	16 DLE	32	48 0	64 @	80 P	96 `	112 p	
1 SOH	17 DC1	33 !	49 1	65 A	81 Q	97 a	113 q	
2 STX	18 DC2	34 "	50 2	66 B	82 R	98 b	114 r	
3 ETX	19 DC3	35 #	51 3	67 C	83 S	99 c	115 s	
4 EOT	20 DC4	36 \$	52 4	68 D	84 T	100 d	116 t	
5 ENQ	21 NAK	37 %	53 5	69 E	85 U	101 e	117 u	
6 ACK	22 SYN	38 &	54 6	70 F	86 V	102 f	118 v	
7 BEL	23 ETB	39 '	55 7	71 G	87 W	103 g	119 w	
8 BS	24 CAN	40 (	56 8	72 H	88 X	104 h	120 x	
9 HT	25 EM	41 )	57 9	73 I	89 Y	105 i	121 y	
10 LF	26 SUB	42 *	58 :	74 J	90 Z	106 j	122 z	
11 VT	27 ESC	43 +	59 ;	75 K	91 [	107 k	123 {	
12 FF	28 FS	44 ,	60 <	76 L	92 \	108 l	124	
13 CR	29 GS	45 -	61 =	77 M	93 ]	109 m	125 }	
14 SO	30 RS	46 .	62 >	78 N	94 ^	110 n	126 ~	
15 SI	31 US	47 /	63 ?	79 O	95 _	111 o	127 DEL	!!!!!!

Gobalde statement yazılamaz ancak decleration  
yapılabilir.

!!!!!! it is ascii table →

Local namespace

## Data types and its storages:



## **işaretsiz tamsayı sınır değerleri**

<b>Byte</b>	<b>Sınır Değerleri</b>
<b>1</b>	<b>0 - 255</b>
<b>2</b>	<b>0 – 65.535</b>
<b>4</b>	<b>0 – 4.294.967.285</b>
<b>8</b>	<b>0 – 18.446.744.073.709.551.615</b>

## **işareti tamsayı sınır değerleri**

<b>Byte</b>	<b>Sınır Değerleri</b>
<b>1</b>	<b>-128 / 127</b>
<b>2</b>	<b>-32.768 / 32.767</b>
<b>4</b>	<b>-2.147.483.648 – 2.147.483.647</b>
<b>8</b>	<b>-9.223.372.036.854.775.808 9.223.372.036.854.775.807</b>

**Storage duration :**

**Storage classes :**

**1-) static storage class: (int x ; or static int x ;)**

Objects with static storage duration live for the lifetime of the program. First value always has 0

These are global variables and defined in local namespace as a static with (keyword static)

**2-) automatic storage class :({int x;})**

Objects with automatic storage duration live for the lifetime of the block in which they execute.

These are defined in local namespace as simple. First value is always !garbage value

### 3-) dynamic storage class :

Objects with allocated storage duration live until they are destroyed by a call to `free()`.

!!!!!!!!!!!!!!

Variables has a automatic storage duration (so, in local variables without static keyword) can create ub(while garbage value becomes running.) (undefined behaviour).

!!!!!!!!!!!!!!

Behaviours : defined behaviours, undefined behaviours, unspecified behaviours (sub : implementation behaviours : derleyiciye bağlı).

### Name lookup rules : (and scope)

1-) isim bir kez aranır ve bulunduğu arama işlemi biter.

!! ismin static duration olması scopunu asla değiştirmez.

!! aynı isim farklı scope'lara sahip değişkene verilebilir.(yani scope'ları aynı ise 2 kez bildirim yapılmaz.)

## Functions:

➔ Control statements : if , while , for , do while , switch , go to , break , return and continue ....

- Return yalnız halde kullanılabilir return ;
- !!!UB : geri dönüş değeri üreten bir fonksiyon return ifadesiz sonlanorsa undefined behaviour olur .for ex(int func(int a, int b))
- C ve c++ dillerinde nested function kavramı yoktur.
- Eğer 0 ise logic yanlış , eğer 1 ise logic doğrudur.
- Başarı bilgisi veren fonksiyonlarda 0 doru diğerleri yanlış
- Call by value f(x);
- Call by reference f(&x);

- 3rd library : it is not library , writing by others.

c de olmayanlar : nested functions , function overloading , default argument.

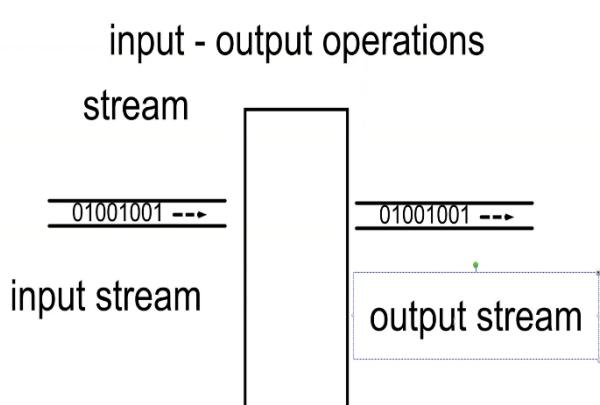
\*\*static variableslar sabit ifadesi almak zorunda.

- Ub : işaretli int türünde taşma olması ub
- Unsigned'da taşıma olmaz mod alınır .
- C'de character constant type is int for ex 'A' = 65

??? Neden ascii'de büyük harflerden sonra 6 karakter ve onu takip eden küçük karakterler var ? : 6 karakter olmasının sebebi her karakterin büyüğüyle küçüğü arasında 32 fark var bu da bitsel olarak maliyet düşüren bir uygulama .

Standard input-output :

Standard input output streams



(formatted- unformatted input-output functions)

# Printf && scanf &&getchar&&putchar...()

Note : scanf : line-buffered function(satır tamponlu fonksiyon ), girişin tanımlanması için \n olması lazım buffer : bellek eğer atanamamışsa bellek dolu kalır diğer scanf e atar. Aynı bufferi getchar da kullanır.

## Operators in c

Operatörler kaç tane operant almasına göre kategorize olur : **Unary(unique),binary(couple),ternary(multiple)**

So, ++x → unary operatör

A + B → binary operatör

A > B ? a : b → ternary operatör

(operator ):

Operator priority:

```

1   ()  [ ]  .  ->                      soldan sağa
-----
2   + - ! ~ sizeof (type) & * ++ --          sağdan sola
-----
3   * / %
-----
4   +
-----
5   >>  <<
-----
6   >  >=  <  <=
-----
7   ==  != 
-----
8   &
-----
9   ^
-----
10  |
-----
11  &&
-----
12  ||
-----
13  ?  :
-----                                     sağdan sola
14  =  +=  -=  *=  /=  %=  &=  ^=  |=  >>=  <<=  sağdan sola
-----
15  ,
-----
```

I

**opertor önceliği :** hangi operatorun neyin operandı olucagina karar verilmesidir ve bunlar aynı grupta ise associativity kavramına göre önceliği sıralanır.(left associative and right associative)

**Ub!! Bir sayının sıfıra bölümü**

**!! float sayıları == ile sınıma yapma.**

**! DEMORGAN KURALLARI GEÇERLİDİR :** !(A && B) = !A || !B

**Short Circuit Behaviour(kısa devre davranışları) :** sadece && and || operatörleri için geçerlidir.

**Sequence point :** side effectlerin uygulanış noktası :

;

&&

||

,

? :

Control statements 'lar sequence pointlerdir.

Not :

$++x \&\& expr2 \rightarrow ++x$  den sonra x artık yeni değerinde kullanılabilir.

!!!!!!UB:

`y = ++x + x; and x = ++x;`

//===== UB : side effectli bir nesneyi sequence pointden önce kullanmak.

```
int x = 10;
int y = (x = 7) + x; //ub
```

(2,5) == 5 çünkü virgül operatorunun ürettiği değer sağ operanttır.

5,6 => 5.6 değil bu 6 dır.

```
0 control statements
1 expression statement
2 null statement
3 compound statement
4
5 control statement
6
7 if statement
8
9 while statement
10 do while statement
11 for statement
12
13 switch statement
14 goto statement
15
16 return statement
17 break statement
18 continue statement
19
20
```

\*getchar ve scanf aynı bufferi kullanır...

function's name	standard or not	the return value?	(Kullanıcı entered?) fine buffered?	formatted?	in or out?	library
*printf()	standard	printf("2") → 1	X	✓	out	<stdio.h>
*scanf()	standard	Scan success numbers	✓	✓	in	<stdio.h>
*getchar()	standard	Ascii code of taken value	✓	X	in	<stdio.h>
- getch() ↳ with no echo ↳ visibility	no standard	Ascii code of the taken value	X	X	in	<conio.h>
- getch() ↳ with echo	no standard	Ascii code of the taken value	X	X	in	<conio.h>
*putchar()	standard	Ascii code of the taken value	X	X	out	<stdio.h>
Test functions		ok standard in-out functions.				
is...()	standard	0 or non-zero	X	X		

Test fonksiyonlarının doğru değeri (lojik kontrolller hariçtir) kesinlikle 1 döndürmek zorunda değildir ama kesinlikle doğru ise non-zero döndürür.bu hatayı gidermek için for ex:

if (!!isprime(a) == !!isprime(b)) gibi değilinin değilini kullanılarak lojik değer elde edilir.

Ternary kullanım örneği : (çoklu ife alternatif;)

```
x = a == 2 ? 2 :  
      a == 4 ? 5 :  
      a == 1 ? 0 :  
      a == 9 ? 9 : -1;
```

## LOOP STATEMENTS : FOR, WHILE , DO WHILE

CONTROL STATEMENTS : break , continue;

Maximum munch rule : boşluk olmadığında en büyük anlamlı tokeni seçmesidir.

using do while :

```
do {  
    statement_1;  
    statement_2;  
    statement_3;  
    statement_4;  
} while (exp);
```

Infinite loops : while (1) and for (;;).

## Function Declarations

Separate compilation model in C : relationship between compiler & linker.

```
IN C
```

```
int foo() // perhaps it is and perhaps it is not  
int func(void) // parametre almıyor
```

```
IN C++
```

```
int foo()           //// almıyor....  
int func(void)     /// almıyor.....
```

Bir modül genelde : main.c ve main.h gibi bir source ve bir header fileden oluşur.

Bir başlık dosyasında neler bulunur?

- a) önişlemci komutları  
macro'lar (önişlemci define komutu ile önişlemci programına tanıtılan isimler)
- b) fonksiyon bildirimleri
- c) user-defined types tür (type) bildirimleri
- c) tür eş isim (type alias) bildirimler



New topic : preprocessor (pp directives)

```
pp directives

# (null directive)
#include
#define
#undef
#if
#else
#elif
#endif
#ifndef
#define
#line
#error
#pragma
```

#include(copy-past):

#include <> = if standard

#include "" = if users defined

!önislemci blok kavramına sahip değil.(scopu aşağısı olan her yer...)

#define directives:

1 – (object-like macros )) #define MAX 100 + \  
200

MAX = 100 + 200 (directively copy-past)

2 – (function-like macros) )

#define max2(a,b) ((a) > (b) ? (a) : (b))  
! max2(a++,b++) gibi ifadelerden kaçın

Not : ortada hem bir makro hem bir fonksiyon varsa normal çağrımda makro, parantezli çağrımda fonksiyon çağrılmış olur . ...

```
int square(int x)
{
    printf("harbi fonksiyon\n");
    return x * x;
}

#define square(a) ((a) * (a))

int main()
{
    int ival;
    printf("bir tam sayı girin: ");
    scanf("%d", &ival);

    int y = (square)(ival);
}
```

Gibi.

!!!!!!!

Preprocessors her zaman önce çalışır name lookup yok.(her türlü çalışır.)

!!!!!!!

```
#define    isleap_year(x)          ((x) % 4 == 0 ? ((x) % 100 != 0 || (x) % 400 == 0) : 0)
int (isleap_year)(int x)
{
    printf("çalisti");
    return x * x;
}

int main()
{
    int y;

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

    (isleap_year)(y);

}
```

```
#define    isleap_year(x)          ((x) % 4 == 0 ? ((x) % 100 != 0 || (x) % 400 == 0) : 0)
int (isleap_year)(int x)
{
    printf("çalisti");
    return x * x;
}

int main()
{
    int y;

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

    isleap_year(y);

}
```

## Pp operators :

önişlemci programın kendi operatörleri var

preprocessor operator

I

# operatörü  
## operatörü  
defined operatörü

## # stringification operator

### Using :

```
#define     iprint(x)           printf(#x " = %d",x)

int main()
{
    int a = 20, b = 10;
    iprint(a + b);
    iprint(a * b);

}
```

## ## token – pasting operator using:

```
#define creat_max_function(type)      type get_max_##type(type a,type b) \
{\
return ((a) > (b) ? (a) : (b));\
}\

creat_max_function(int);
creat_max_function(double);

int main(void)
{
    int a = 10, b = 20;
    double x = 20., y = 40.;
    printf("int a + b = %d", get_max_int(a, b));
    printf("double x + y = %f", get_max_double(x, y));
}
```

## Others :

```
1 #include <stdio.h>
2
3
4 #define SAID ..... 200
5
6
7 #if SAID
8 int x;
9 int y;
10
11
12
13 #else
14 int a;
15 int b;
16
17
18
19
20#endif
```

Not1 : if lojik kısmında sadece makro ve lojik ifadeleri olur

Not2 : double sayıların aritmetiği yok yani

#if SAID == 200.5 gibi bir ifade kullanılamaz.

Not3 : tanımlı olmayan makrolar lojik değerde 0 kabul edilir preprocessor tarafından...

```
1  #include <stdio.h>
2
3
4  #define USD      0
5  #define EUR      1
6  #define GBP      2
7  #define JPY      3
8  #define CURRENCY -1
9
10 #if  CURRENCY == USD
11     typedef double dollar;
12
13 #elif CURRENCY == EUR
14     typedef double euro;
15 #elif CURRENCY == GBP
16     typedef double jerfgoktep;
17 #else
18     typedef double o_none;
19
20
21
22
23
24
25
26 #endif
```

Multiple Inclusion Guard: başlık dosyaları korumak.

1-)

```
#ifndef NUTILITY_INCLUDED
#define NUTILITY_INCLUDED
//codes.....
//.....
#endif
```

2-) not standard.

```
Object (Global Scope)
1  #pragma once
2
```

Using #undef :

!!!!ub

İki defa aynı makroyu define etmek ub

!!!!1

```
#include <stdio.h>
#undef PHILIPS
#define PHILIPS
```

→ PROTECTING UB

#using :#error

```
1
2
3 #ifndef __cplusplus
4
5 #error IT MUST BE C++ COMPILER....
6
7 #endif
```

Predefined symbolic constants :

predefined symbolic constant  
öntanımlı sembolik sabit

=====

```
__LINE__      10
__FILE__      "main.c"
__DATE__      "TARİH"
__TIME__      "saat"
__func__
```

```
__STDC__
```



\_\_STDC\_\_ = \_\_cplusplus

\*\*\*\*\*

## SCOPES

- 1-) file scope : **in all file**
- 2-) block scope : **including block**
- 3-) function prototype scope : **for parameters**
- 4-) function scope : **everywhere in function (for : goto)**

## Goto:

Jumps :

- 1-) near jump or local jump(goto)
- 2-) long jump

**Label(etiket)** : for ex **out.**

```
ornek_goto.txt  a X mervanh    nutility.h    notlar.txt"    main.c"
#define SUCCESS 0
#define ERROR 1

int init_device(void)
{
    if (allocate_memory() != SUCCESS)
        goto out1;
    if (setup_interrupts() != SUCCESS)
        goto out2; I
    if (setup_registers() != SUCCESS)
        goto out3;
    // more logic ...
    return SUCCESS;
out3:
    teardown_interrupts();
out2:
    free_memory();
out1:
    return ERROR;
}
```

- Goto genelde nested loops lardan çıkışmayı sağlar...

## Switch-case(alternatively else-if ladder

```
switch (integer expr) {  
    case 1:  
        statement1;  
    case 2:  
        statement2; I  
  
    case 5:  
        statement3;  
        statement4;  
  
    case 7:  
        statement5;  
  
}
```

```
switch (x)  
{  
default :  
    printf("gecersiz...");  
    break;  
case 1 :  
    printf("pazartesi\n");  
    break;  
case 2 :  
    printf("sali\n");  
    break;  
case 3 :  
    printf("carşamba\n");  
    break;  
case 4 :  
    printf("persembe\n");  
    break;  
case 5 :  
    printf("cuma\n");  
    break;  
case 6 :  
    printf("cumartesi\n");  
    break;  
case 7 :  
    printf("pazar\n");  
    break;  
}  
return 0;
```

## Type Conversion:

a + b gibi bir işlemde rank önemlidir.

Rank(mertebe) :

Long double

Double

Float

Long long int

Long int

int

.

. (converting to int always)

1-) implicit converison

**1 ➔ long long int**

    Unsigned long long int   == unsigned long long

**2 ➔ aynı**

    Aynı               == aynı

**3 ➔ unsigned int /4byte**

    Signed long / 4 byte === unsigned long

    Unsigned int / 4byte

    Signed long / 8 byte

Atamada dönüşüm = atamanın olacağı türe her zaman dönüştürülür.

```
int x;  
double y = 3.14;  
x = y;
```

burada 3.94 inte dönüştürülerek yuvarlama olmadan 3 değerini alır. Veri kaybı olur.

!lub : standard – input outputta inti doubleye doubleyi inte çevirtmek.

!!!!!!mülakat sorusu !!!!!!

? int : double; → değeri neolursa olsun double cinsinden olucaktır

.2-) explicit conversion : (int)x;

ival = (int)dval; → veri kaybının olacağını bilsende yaz.

## Arrays In C

İnt a[x olmaz çünkü burası constant olması gereklidir.] ama vla kullanılarak dinamik hale getirilebilir.

\*\*\*\*\*

İnt a[100] → tüm değerler sıfırla başlar.

İnt main()

{

İnt b[100] → garbage value

Static int c[100] → hepsi 0.

}

\*\*\*\*\*

C dilinde parametreler dizi olamaz

Return value dizi olamaz

\*\*\*\*\*

A[4] x gibi kullanılabilir → ++a[4];

\*\*\*\*\*

Array decay or array to pointer conversion

&A[0] == a;

\*\*\*\*\*

Sizeof operator :

\*\* Bir compile time operatördür yani değeri derleme zamanında constant olarak derleyici tarafından bilinir.

\*\* bir türün storage ihtiyacının derleyiciden derleyiciye göre değişen değerini hesaplar return değeri unsigned türündendir.

```
int main(void)
{
    printf("%zu\n", sizeof(char));
    printf("%zu\n", sizeof(short int));
    printf("%zu\n", sizeof(int));
    printf("%zu\n", sizeof(long int));
    printf("%zu\n", sizeof(long long int));
    printf("%zu\n", sizeof(float));
    printf("%zu\n", sizeof(double));
    printf("%zu\n", sizeof(long double));
    printf("%zu\n", sizeof 4);
}
```

```
printf("%zu\n", sizeof c);
printf("%zu\n", sizeof +c);
```

C is char 1,4 çünkü int altı türler işleme sokulduğunda integral promotion olur inte yükseltilir .

\*\*\*\*\*

### Unevaluated context

```
//unevaluated context
|
int main()
{
    int x = 10;

    printf("%zu\n", sizeof(++x));
    printf("%d\n", x);

}
```

X yine 10 çünkü sizeof işlem kodu üretmez.

```
*****  
int main()  
{  
    int a[] = { 2, 5, 7, 1, 34, 6, 9, 44, 678, 123, 45 };  
  
    for (int i = 0; i < sizeof(a) / sizeof(a[0]); )  
  
}  
//-----
```

Sizeof'un clacis kullanımı.

```
*****
```

## YAZI İŞLEMLERİ :

Yazılar '\0'(null) karakteri ile sonlanır.

Bir yazı eğer a[100]; şeklinde bildirilirse max 99 (biri null) min 0 karakter tutabilir.

```
#include <stdio.h>  
#include "nutility.h"  
  
char str[100];  
  
int main()  
{  
  
    str[0] = 'C';  
    str[1] = 'A';  
    str[2] = 'N';  
  
    for (int i = 0; str[i] != '\0'; ++i)  
        putchar(str[i]);  
}  
//-----
```

!!!! this's not ub

```
int main(void)  
{  
  
    char a[] = "furkan";// otomatik olarak sonuna null eklenir yani dizinin boyutu 7  
    char b[6] = "furkan"; // null eklenemedi... boyut 6  
    char c[7] = "furkan"; //null eklenebildi boyut 7...  
}
```

Scanf ve printf null a duyarlıdır ve olması gerek.

Puts(str(yazı)) → ekrana yazıyı yazar ve \n ekler.

## İNLİNE EXPANSİON NEDİR?:

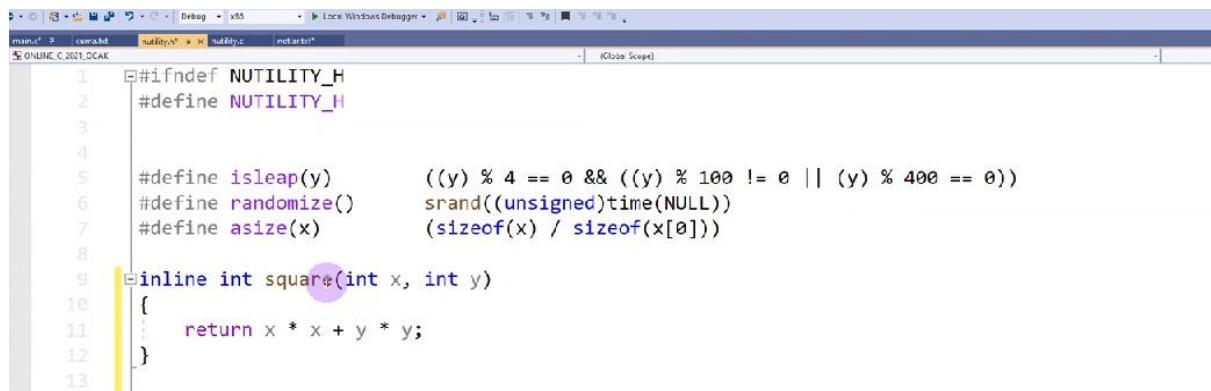
Normalde bir header başlık dosyasında fonksiyon tanımı olmaz çünkü diğer main.c dosyalarında da include edilirse linker hata verir aynı fonksiyon birden fazla yerde tanımlı diye.

Ama inline keywordunu koyup tanımını yapsak hata vermez.

Derleyici ne zaman inline expansion yapar ?

- 1-) derleyicinin fonksiyon tanımını mutlaka görmesi lazım.
- 2-) verimli olup olmıcagi konusunda karar vermesi
- 3-) optimizasyon switchleri ayarları bunu kullandırmaya müsaitse yapabilir..
- 4-) kod çok kompleks olmaması gerekiyor adeta function-like macro gibi olması lazım.

!! işlevi : fonksiyonu çağrırmak yerine derlenmiş kodunu oraya koyar derleyici...(maliyet azaltma )



The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays a C file named 'main.c'. The code includes several preprocessor directives and a single inline function definition:

```
#ifndef NUTILITY_H
#define NUTILITY_H

#define isleap(y)      ((y) % 4 == 0 && ((y) % 100 != 0 || (y) % 400 == 0))
#define randomize()   srand((unsigned)time(NULL))
#define asize(x)       (sizeof(x) / sizeof(x[0]))

inline int square(int x, int y)
{
    return x * x + y * y;
}
```

The word 'square' is highlighted in yellow, indicating it is the target of an inline expansion. The rest of the code is shown in standard purple and black syntax highlighting.

## POİNTERS(adres belirten nesneler):

\*\* pointerlar bir adres tutar ve bildirimi şu şekilde yapılır  
T \* ptr = &x burada x 'in adresi ptr isimli türden türe göre  
size'si değişmeyen (derleyiciye göre 4-8byte)ptr'de  
tutulur.

Array decay or array to pointer conversion rule:

a = &a[0](sizeof ve diğerı hariç)

pointer operators :

- 1 ) [] subscript/ index operator
- 2 ) -> member selection / arrow operator
- 3 ) & address of operator
- 4 ) \* dereferencing / indirection operator...

\*Operatorü adres tutan değişkenle kullanılır ve anlamı

\*ptr ptr neyin adresini tutuyorsa onun içindeki  
neseneye erişim anlamına gelir...

Adresin printf'de formatı %p dir.

```
1 #include <stdio.h>
2
3
4 void swap(int* x, int* y)
5 {
6     int temp = *x;
7     *x = *y;
8     *y = temp;
9 }
10
11
12
13
14 int main()
15 {
16
17     int x = 1;
18     int y = 3;
19     swap(&x, &y);
20     printf("x = %d", x);
21     printf("y = %d", y);
22 }
```

```
#include <stdio.h>

int main()
{
    int x = 2;
    double y = 5.;
    int* ptr = &x;
    double* d_ptr = &y;
    printf(" address of x : %p", ptr);
    printf("sizeof(ptr) = %zu", sizeof(ptr));
    printf("sizeof(d_ptr) = %zu", sizeof d_ptr);
}
```

\*\*\*\* İnt , double gibi küçük verilerle iş yaptığımızda pointera ihtiyaç duymayız ama büyük verilerde örneğin büyük bir matrix için fonksiyon kullanırsak pointer kullanmamız gereklidir çünkü fonksiyonun içinde return ifadesinde matixlerimizi ayrı ayrı kopyalamamız bu da belleği gereksiz doldurmamıza neden olur.

\*\*\*\* bir fonksiyonda tuple tarzı veri alımı

Func(a, b, &c, &d ,&e)

\*\*\* fonksiyonlara doğrudan dizi gönderebiliriz fakat adresini gönderebiliriz.(not call by value , it is call by reference)

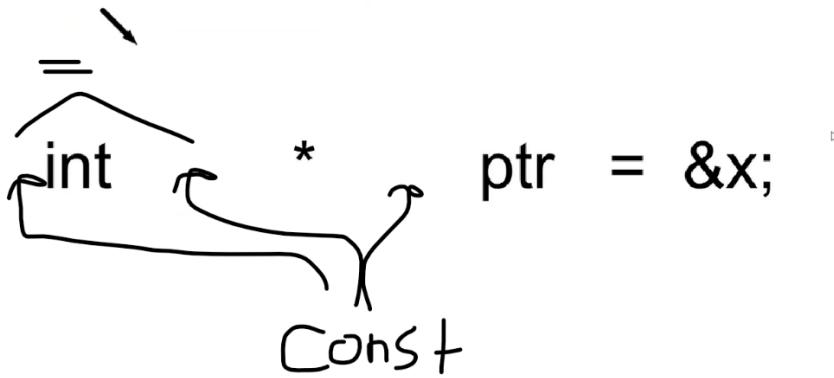
The Const keyword :

Const anahtar sözcüğü değişkenin initialize edildikten sonra bidaha atama yapılmasını ve bunun sentaks hatasını vermesini saglayan bir keyworddur.

!!!!ub:

Const ile bildirilen değişkenin adresini bir pointerda tutup daha sonra değiştirmek . derleyici anlayamaz bunu.

Pointer değişkenler ve const keyword ‘u.



Const en solda olursa anlamı gelen nesne asla değiştirilmeyecek.

Sadece reading yapılcak writing yok.(syntax error)

(low level const)

Const '\*'ın sağında olursa ptry'nin tuttuğu adres değiştirilmicek.

(top level const)

Her iki yere yazılırsa hem o ptr hem de tuttuğu nesne yalnız okunulabilir.

(top/low level const)

Fonksiyonları yazarken bunlar kilit taşıdır. Yani :

Void (int\* x) : verdiğin parametreyi değiştircem.(mutator,setter)

Void (const int\* x) = değiştirmeyeceğim.(getter ,(reading))

## POINTER ARITMETİCS :

Pointer aritmetiklerinde yapılabilecek işlemler

1-) bir tam sayı ile bir adresi toplamak

2-) bir adres ile tam sayıyı toplamak

3-) bir adresden bir tam sayı çıkarmak

4-) syntax error : bir tam sayıdan bir adres çıkarmak

\*\* örneğin bir adres ile 1 i toplamak adresin değerini bir arttırmaz diğer nesneye geçmeyi saglar. (char ise 1 er 1 er int ise 4 er 4 er.)

The screenshot shows the Microsoft Visual Studio Debug window. The code in the editor is:

```
1 #include <stdio.h>
2
3
4
5
6 int main(void)
7 {
8     char a[100] = { 0 };
9
10    printf("&a[0] = %p\n", a);
11    printf("&a[1] = %p\n", &a[0] + 1);
12    printf("&a[2] = %p\n", &a[0] + 2);
13 }
```

The output window displays the memory dump:

```
&a[0] = 0000009AEC37F730
&a[1] = 0000009AEC37F731
&a[2] = 0000009AEC37F732
```

Below the dump, the command line shows the execution results:

```
C:\Users\said_\OneDrive\Masaüstü\working on c\Project1\x64\Debug\Project1.exe (process 23196) exited with code 0.
Press any key to close this window . . .
```

The screenshot shows the Microsoft Visual Studio Debug window. The code in the editor is:

```
1 #include <stdio.h>
2
3
4
5
6 int main(void)
7 {
8     char a[5] = { 0,1,2,3,4 };
9     char* c_ptr = a;
10
11    printf("&a[0] = %d\n", *a);
12    printf("&a[1] = %d\n", *(a[0] + 1));
13    printf("&a[2] = %d\n", *(a[0] + 2));
14 }
```

The output window displays the memory dump:

```
&a[0] = 0
&a[1] = 1
&a[2] = 2
```

Below the dump, the command line shows the execution results:

```
C:\Users\said_\OneDrive\Masaüstü\working on c\Project1\x64\Debug\Project1.exe (process 29704) exited with code 0.
Press any key to close this window . . .
```

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>
#include "nutility.h"

int main()
{
    int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int* ptr = a;

    for (int i = 0; i < 10; ++i) {
        printf("%d %d %d %d\n", a[i], *(i + a), *(a + i), *ptr);
        ++ptr;
    }
}
```

a[x]	*(a + x)
x[a]	*(x + a)

Yukarıdakinden hareketle  $a[i] = i[a]$  aynı şey.

\*\* bir dizinin yüksek indeksli elemanın adresiyle daha küçük indeksli adresinin elemanını çıkarırsak pozitif bir tam sayı elde ederiz.

Void func(cons int\* ptr) == void func(const int a[])

\*\* c++ 'da yaygın olarak dizilerin bittiği adres kullanılabilir to c

- A[5] → a\_end = a + 5;

\*\* adresler karşılaştırılabilir.

syntax errors :

`&x++` ==> on the right r value expression

`&++x` ==> on the right r value expression

`++&x` ==> on the right r value expression

invalid and commonly used pointer idioms :

`*ptr++`, `*++ptr`, `++*ptr`

The screenshot shows a Microsoft Visual Studio IDE window. The top part displays the code editor with a dark theme, showing a C program. The bottom part shows the 'Microsoft Visual Studio Debug' output window.

**Code Editor (Project1.cpp):**

```
1 #include <stdio.h>
2 #include "nutility.h"
3
4
5
6 int main(void)
7 {
8     int a[10] = { 1,100,200,300,400 };
9     int* ptr = a;
10
11
12     printf("%d\n", *ptr++);
13     printf("%d\n", *++ptr);
14     printf("%d", ++*ptr);
15 }
```

**Output Window:**

```
1
200
201
C:\Users\said_\OneDrive\Masaüstü\working on c\Please\Project1.exe (process 12696) exited with code 0.
Press any key to close this window . . .
```

```
<string.h>
size_t strlen(const char *p);
char *strchr(const char *p, int c);
char * strrchr(const char *p, int c);
char * strstr(const char *psource, const char *psearch);
char * strpbrk(const char *psource, const char *pchar);

char * strcpy(char *pdest, const char *psource);
```

*const*

### TYPEDEF

```
Typedef int* IPTR;
```

```
IPTR ptr1, ptr2;
```

```
ptr1 of type : int*
```

```
ptr2 of type : int*
```

```
*****
```

```
but, #define IPTR int*
```

```
IPTR(int* )ptr1, ptr2;
```

```
ptr1 of type : int*
```

```
ptr2 of type : int;
```

```
*****
```

cogu typedef isimleri standard olarak vardır(include stddef.h)

`size_t` : derleyiciden derleyiciye portability için değeri değişebilir ve `sizeof()` un geri dönüş değeri ve genelde `len` değeri için kullanılır.

`sizeof(x) ==> size t türünden değer dönderir.`

tuzakkkkkkkkkkkkkkkkkkk

`&x = r value`

**Ptr = l value;**

A ifadesi r valuedir.

(++(\*ptr++))

!!! → pointers are comparable ( $\&\text{ptr}[2] > \text{ptr}[1]$ ) → true and 1.

`** ptr2 - ptr1 = ptrdiff_t` türündendir

Adres farkı indiyi verir.

Automatic storage class example :

1-)Local variables.

Static storage class example :

1-)global variable,

2-)static variable in local,

3-)string literals.

\*\*\*\* huge error and ub :

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

int* get_int(void)
{
    int x;           I

    printf("bir tam sayı giriniz: ");
    scanf("%d", &x);

    return &x; //ub
}

int main()
{
    int* ptr;
    //

    ptr = get_int();
    printf("girilen sayı: %d\n", *ptr);
}
```

issues found

\*\* const int x = x in türü const intdir.

! Const t\* türünden bir nesneyi t\* yapma.

NULL POİNTER (not null character '\0', is a  
macro)

pointer 3 şekilde geçerlidir :

- 1 ➔ dizinin son adresi
- 2 Null pointer
- 3 Var olan nesnenin adresini tutan pointer.

\*\*Stdio da vardır ve lojik ifadelerce değerlendirilebilir.

`**int* ptr = 0; == int* ptr = NULL;`

`**if (ptr) null değilse çalışır.`

\*\*Global veya local de tanımlanan static pointerlar hayatı NULL pointer ile başlar.

Null pointer kullanım alanları :

- 1-) bazı fonksiyonların geri dönüş değeri pointer ise başarısızlık durumunda null pointer döndürmesi (bakmak gereklidir)
- 2-) bazı fonksiyonlara null göndermek
- 3-) dinamik bellek yönetiminde.

`Ptr = NULL;`

`*ptr = 2 !ub`

## STRING LITERALS

“mehmet” = a anlamına gelir.

“said” ➔ 5 elemanlı ve sonunda null character olan bir dizi.

```
char* p = "kaya";
```

```
*p = 'M';
```

Bir string literalini değiştirme girişimi ub'dır.

\*\*\* c de const char\* türünün char\* a dönüşümü uyarı iken c++da syntax error. Ve c de string literaller char\* c++da const char\* dir.

```
int main()
{
    char* ptr = "ali"; //////////| it is false
    const char* ptr = "ali"; //it is true
}
```

\*\*String literalleri static ömürlüdür programın başından sonuna kadar bellekte kalır.

\*\*İlk adres + n = n. index;

\*\*n.Adres – ilk adres : n.index;

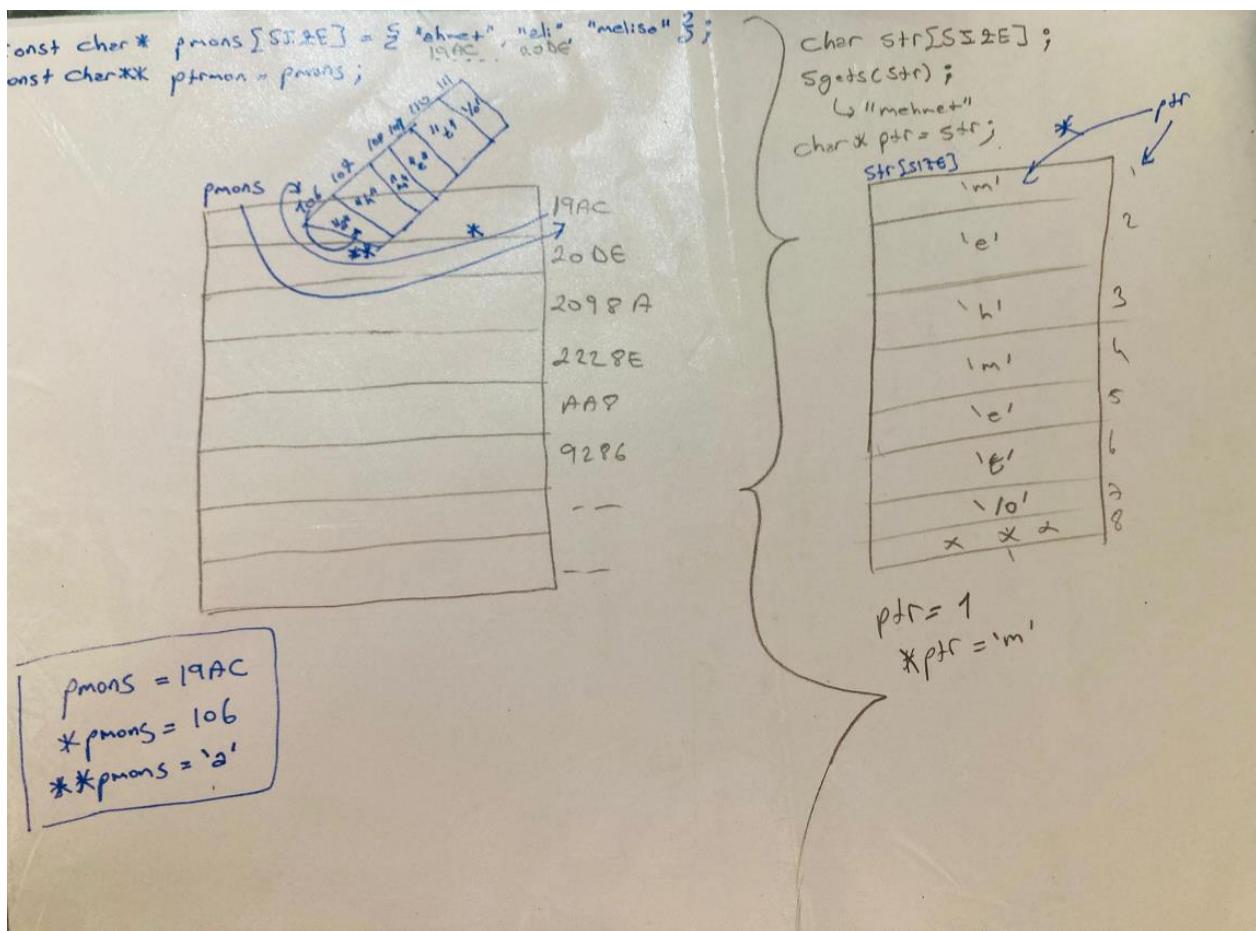
\*\*aynı string literalleri aynı adreste olmak zorunda değil (unspecified behaviour);

## POİNTER ARRAYS

```

int main()
{
    const char* const ptr[] = {
        "ocak",
        "subat",
        "mart",
        "nisan",
        "mayis",
        "haziran",
        "temmuz",
        "agustos",
        "eylul",
        "ekim",
        "kasim",
        "aralik",
    };
    for (size_t i = 0; i < sizeof(ptr) / sizeof(ptr[0]); ++i)
        printf("%s\n", ptr[i]);
}

```



## POİNTER TO POİNTER...

\*\*Her nesnenin adresi türünün sonuna yıldız eklenmesiyle oluşur

```
190
191 int main(void)
192 {
193
194     int x = 2;
195     int* ptr1 = &x;
196     int** ptr2 = &ptr1;
197     int*** ptr3 = &ptr2;
198     printf("%d", ***ptr3);
199 // |output ==> 2
200 }
```

int\* bir tür int\* türden bir ürünü taşıyamaz!!!!

Const char\*const\*const ptr;

⇒ Okuyana bilgi vermek amacıyla const kullanılabilir

## VOİD POİNTERS

- Void is a type.
- An expression can be void but not any object

Void pointers carry only address for generic unlike others.so you can not \* vptr = any;

```
void gswap(void* p1, void* p2, size_t size)
{
    char* cp1 = (char*)p1;
    char* cp2 = (char*)p2;

    while (size--)
    {
        char ctemp = *cp1;
        *cp1++ = *cp2;
        *cp2++ = ctemp;
    }
}
```

Restrict keyword in function declaration : search!

\*mem fonksiyonları byte byte işlem yapar.

It is hard example :

```
void greverse(void* vptr, size_t size ,size_t one_el)
{
    char* cptr = vptr;
    for (size_t i = 0; i < size / 2; ++i)
    {
        gswap(cptr + i * one_el, cptr + (size - i - 1) * one_el, one_el);
    }
}
```

BYTE BYTE BYTE.... ONA GÖRE HESAP ET!!!

The screenshot shows a code editor window with several tabs at the top: actions.txt, nutility.c, nutility.h, notlar.txt\*, and main.c\*. The main.c\* tab is active. The code in main.c\* is as follows:

```
4 #include <string.h>
5 #include "nutility.h"
6
7 int x = 10;
8 double dval = 2.3;
9 char str[10];
10
11 int main()
12 {
13     void* a[] = { &x, &dval, str };
14 }
```

The code uses syntax highlighting with green for comments, red for strings, blue for keywords, and purple for identifiers. It also features code folding, indicated by square brackets [ ] around sections of code.

## FUNCTION POINTERS

&func == func (like array decay)

# pointers

object pointers

function pointers

```
int foo(int x, int y);
double fx(double x, double y);

int main()
{
    // type of foo : int (*) (int, int)
    // type of fx  : double (*) (double, double)
```

```
int fx(int x);

int main()
{
    int (*fp) (int) = fx // or int (*fp) (int) = &fx;
}
```

`fp() == (*fp)();`

Tüm nesne pointerlarının sizeof u aynı

Tüm fonksiyon pointerlarının sizeof u aynı fakat ikisinin ki de aynı olamayabilir.

```
void f1(void (*fptr) (void))
{
    printf("f1 called and f1 calls other functions...\n");
    fptr();
}

void f2(void)
{
    printf("f2 called...\n");
}

void f3(void)
{
    printf("f3 called...\n");
}

int main()
{
    f1(f2);
    f1(f3);
    return 0;
}
```

it is called as callback generally ==> ^

```
void print_chars(const char* p, int (*fp) (int))
{
    printf("\nhere are the characters the function name : %s\n\n", p);

    for (int i = 0; i < 128; ++i)
        if (fp(i))
            printf("%c", i);
}

int main(void)
{
    print_chars("isupper", isupper);
    print_chars("islower", islower);
    print_chars("isalpha", isalpha);
    print_chars("ispunct", ispunct);
    return 0;
}
```

Qsort using =>

```
for instance using qsort function

int my_compare(const void* vptr1, const void* vptr2)
{
    const int* cptr1 = (const int*)vptr1;
    const int* cptr2 = (const int*)vptr2;

    if (*cptr1 > *cptr2)
        return 1;
    else if (*cptr1 < *cptr2)
        return -1;
    return 0;
}

int main(void)
{
    int a[7] = {2456789, 31234567};
    print_array(a, 7);
    qsort(a, 7, sizeof(a[0]), my_compare);
    print_array(a, 7);

    return 0;
}

attention ! ==> my compare function applies int because you know otherwise, you find bug... ub!!!!
```

## TYPEDEF (TYPE ALIAS) 1 -)

Makros and typedef are different from each other.

Preprocessor knows makro before compiler.

Compiler knows typedef after preprocessor.

Example:

```
Typedef int* IPTR;  
#define IPTR2 int*  
IPTR x1, x2; => both of them are int*  
IPTR2 x1, x2 ===> x1 = int* but x2 = int;
```

Rule of Naming objects for typedef:

- 1-) define a object with type
- 2-) add typedef in front of it.
- 3-) change the object name

Example :

```
Int a[10]
```

```
Typedef int a[10];
```

```
Typedef int INTA10[10];
```

Perfect!

```
int main()
{
    typedef int (*FPTR)(int, int);

    FPTR fp;
    //int (*fp)(int, int)
}
```

```
typedef int word;

int main(void)
{
    word x = 2;
}
```

Some typedef declarations for portability :

```
size_t
time_t
clock_t
ptrdiff_t
fpos_t
int16_t
uint16_t
bool|
```

What do you use that ? there is typedef declarations.

```
void f1(int (*fp)(const void*, const void*));
void f2(int (*fpx)(const void*, const void*), int (*fpy)(const void*, const void*));
int(*f3(int (*fpx)(const void*, const void*), int (*fpy)(const void*, const void*))(const void *, const void *))
```

```
I
```

```
int main()
{
    int (*fp)(const void*, const void*) = strcmp;
    //fp nin adresiyle ilk degerini alan fptr degişkenini tanımlayın
    //pointer to function pointer
    int (**fptr)(const void*, const void*) = &fp;
    //elamanları fp gibi olan 10 elemanlı bir dizi tanımlayın fa
    int (*fa[10])(const void*, const void*);
```

```
typedef int (*FPTR)(const void*, const void*);

//void f1(int (*fp)(const void*, const void*));
void f1(FPTR);

//void f2(int (*fpx)(const void*, const void*), int (*fpy)(const void*, const void*));
void f2(FPTR, FPTR);

//int(*f3(int (*fpx)(const void*, const void*), int (*fpy)(const void*, const void*))(const void*, const void*));
FPTR f3(FPTR, FPTR);
```

Typedef rule :

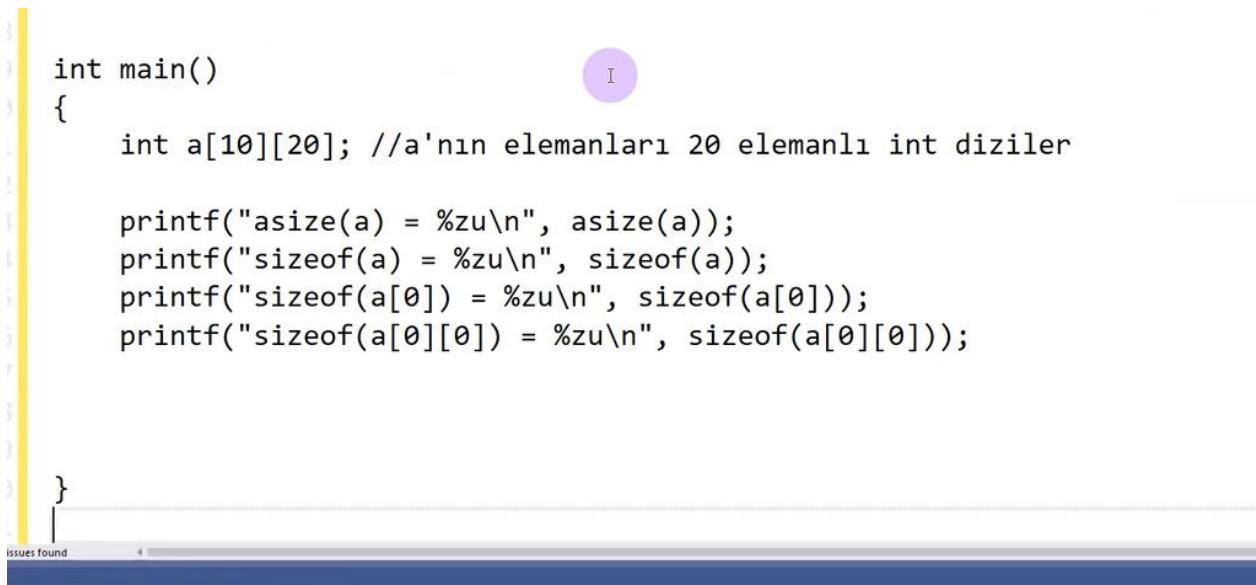
- 1 - ) değişkeni tanımla
- 2-) başına typedef koy
- 3-) değişkenin ismini değiştir.

!! önemli

Eğer bir dizi üzerinde işlem yapılıyorsa unutma ki dizinin elemanı int ise kendisi int\*, eğer elemanı int\* ise kendisi int\*\*!!

## Multi – dimesional arrays :

int[10][20] : elemanları 20 şer elemanlı dizi olan 10 elemanlı dizi.

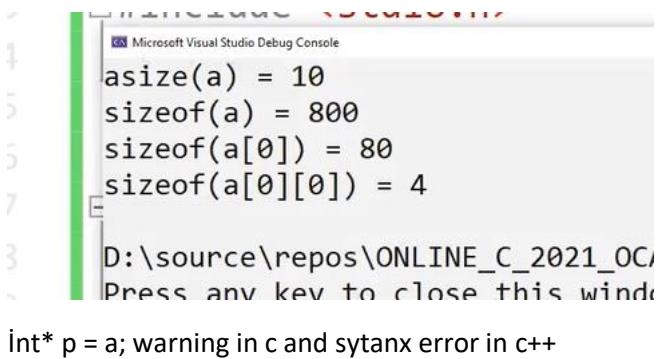


```
int main()
{
    int a[10][20]; //a'nın elemanları 20 elemanlı int diziler

    printf("asize(a) = %zu\n", asize(a));
    printf("sizeof(a) = %zu\n", sizeof(a));
    printf("sizeof(a[0]) = %zu\n", sizeof(a[0]));
    printf("sizeof(a[0][0]) = %zu\n", sizeof(a[0][0]));

}
```

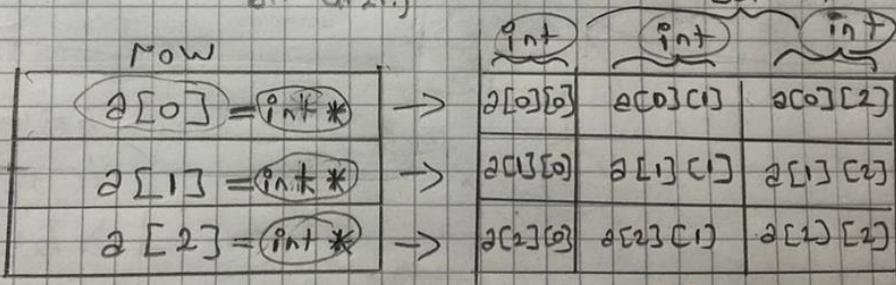
Output :



```
Microsoft Visual Studio Debug Console
asize(a) = 10
sizeof(a) = 800
sizeof(a[0]) = 80
sizeof(a[0][0]) = 4
D:\source\repos\ONLINE_C_2021_OCA\Debug> Press any key to close this window
int* p = a; warning in c and syntax error in c++
```

## Multi-Dimensional Arrays

int a[3][3]; (a, elementleri int[3] denenden oluşan 3 elementli bir dizisi.)



$\text{int} ** \text{ptr} = \&a[0] \rightarrow = \downarrow$

$\text{int} * \text{ptr} = \&a[0][0]$

good choice!  $\leftarrow = \rightarrow$

```
int main(void)
{
    int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    int* ptr = a[0];

    for (int i = 0; i < 6; ++i)
        printf("%d", *(ptr + i));
    return 0;
}
```

```
int main(void)
{
    int row, col;
    printf("enter row : ");
    scanf("%d", &row);

    printf("enter col : ");
    scanf("%d", &col);

    int** ptr = (int**)malloc(sizeof(int*) * row);
    if (!ptr)
    {
        printf("not available...\n");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < row; ++i)
    {
        ptr[i] = (int*)malloc(sizeof(int) * col);
        if (!ptr[i])
        {
            printf("not available...\n");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < row; ++i)
        for (int j = 0; j < col; ++j)
        {
            scanf("%d", &ptr[i][j]);
        }

    for (int i = 0; i < row; ++i)
        for (int j = 0; j < col; ++j)
        {
            printf("%d", ptr[i][j]);
        }

    for (int i = 0; i < row; ++i)
        free(ptr[i]);

    free(ptr);

    return 0;
}
```

Designated initializer :

The screenshot shows a Microsoft Visual Studio interface. On the left, there is a code editor window containing a C program. The code defines a 2D integer array 'a' of size 10x5. It initializes the first row with {3, 3, 3, 3} and the second row with {7, 7, 7, 8}. Subsequent rows are initialized with {2, 5, 8, 3, 6}. The program then prints the entire matrix row by row using nested loops and the printf function.

```
int main()
{
    int a[10][5] = {
        [3]= {3, 3, 3, 3},
        [7] = {7, 7, 7, 8},
        [9] = {2, 5, 8, 3, 6},
    };

    for (int i = 0; i < 10; ++i) {
        for (int k = 0; k < 5; ++k) {
            printf("%d ", a[i][k]);
        }
        printf("\n");
    }
}
```

The output window on the right is titled "Microsoft Visual Studio Debug Console". It displays the printed matrix:

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
3	3	3	3	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
7	7	7	8	0
0	0	0	0	0
2	5	8	3	6

Below the output, the path "D:\source\repos\ONLINE\_C\_" is visible, followed by the message "Press any key to close th".

Dizilerde ilk değer verilirken boyut yazılmayabilir ama sütün sayısı mutlak surette yazılmalıdır.

For ins : a[][4] → geçerli.

```
int a[][] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int b[3][] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int c[][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Burada a'nın elemanlarının türü gösteriliyor :

```
7 int main()
8 {
9     int a[5][4];
10    //int[4]
11 }
12 //-----
13 //-----
```

TypeDef ile bildirim :

```
typedef int INTA10[10];
```

```
//INTA10  int[10]
```

```
int main()
{
    //int a[20][10];
    INTA10 a[20];
}
//-----
//-----
```

Sırasıyla nesnenin türleri.

```
int main()
{
    int a[10][20][40]; //int [10][20][40];
    //int [20][40];
    //int [40];
}
```

Multi dimensional using :

```
int main()
{
    int a[2][2][2] =
    {
        {{1, 1 }, {2, 2 } },
        {{3, 3 }, {4, 4 } },
    };
    int n = 8;
    int* p = a[0][0];
    while (n--)
        printf("%d ", *p++);
}
```

Sending a multi-dimensional array to a function :

```
void set_random_20(int(*p)[20], size_t size)
{
    for (size_t i = 0; i < size; ++i) {
        for (int k = 0; k < 20; ++k) {
            p[i][k] = rand() % 10;
        }
    }
}
```

By generically :

```
void print_matrix(const int* p, size_t row, size_t col)
{
    for (size_t i = 0; i < row; ++i) {
        for (size_t k = 0; k < col; ++k) {
            printf("%d", p[i * col + k])
        }
    }
}
```

Normalde charda isim tutmak için char\* names[] ile yapıyorduk :

```
int main(void)
{
    char names[][20] = {
        "ali",
        "mehmet",
        "melisa",
        "yeliz",
        "sudenaz",
        "ayse",
        "melahat",
        "emre",
        "yaren",
        "enes",
        "abdulmuptalib",
        "sezar",
        "muhittin",
        "turabi",
        "hayati",
        "ummuugulsum",
        "feriha",
        "fatma zehra",
        "pakize",
        "papatya",
        "meltem",
        "muhammed",
        "said",
        "serhat",
        "zaim"
    };

    for (int i = 0; i < sizeof(names) / sizeof(names[0]); ++i)
        printf("%s\n", names[i]);
    return 0;
}
```

ama burda yapılan multi-d ile bunun yapılmasının nedeni dizinin artık pointer adreslerini tutmamasıdır yani artık diziye tek tek yazılıyor isimler change edilebilir ub olmaz.

```
int foo(int(*p)[20], int size);  
int foo(int p[][20], int size);
```

Fonksiyon parametreleri için iki notasyon da kullanılabilir.

## End Processes

There are two ways:

With Normal termination → exit(0 or non0)

Exit guarantees buf flashing so, if there is openfile , is done(writing finish)

With abnormal termination → abort(zero or nonzero)

Not guaranteed

- Atexit(function1)

- Atexit(function2) → before every exit (included return 0) call, it calls func (according to LIFO algorithm) lifo → 3 → 2 → 1

Fifo → 1 → 2 → 3

Atexit kaynaklarının geri verilmesi işleminde genellikle kullanılır (**cleanup**)

```
#include <stdlib.h>

int main() {
    // Programın normal bir şekilde sonlandırılması
    exit(0);

    // Başarısızlık durumuyla sonlandırma
    // exit(1);
}
```

#### 1. **abort()** Fonksiyonu:

- `abort` fonksiyonu aniden ve beklenmedik bir şekilde programın sonlandırılmasını sağlar.
- Programın çıkış kodu belirtilmez, genellikle işletim sisteme özel bir hata kodu döndürülür.
- `stdlib.h` başlık dosyasını içerir.

Örnek:

```
c                                     ⌂ Copy code

#include <stdlib.h>

int main() {
    // Beklenmeyen bir durum olduğunda programın aniden sonlandırılma
    if (some_unexpected_condition) {
        abort();
    }
}
```

Kullanım Yerleri:

- `exit` genellikle programın normal bir şekilde sonlandırılması durumunda kullanılır.
- `abort` ise genellikle kritik hatalar veya beklenmeyen durumlar oluştuğunda programın derhal sonlandırılması gerektiği durumlarda kullanılır.

Not: `exit` fonksiyonu kullanıldığında programın normal sonlandırılması gerçekleşirken, `abort` fonksiyonu kullanıldığında bir hata durumu veya istisna meydana gelmiş gibi işlem yapılır ve program aniden sonlandırılır.



## Dynamic memory management:

Storage classes :

- 1-)static storage class
- 2-)automatic storage class
- 3-)dynamic storage class

It is used **malloc, calloc, realloc and free** functions for dynamic memory management. (on stdlib.h)

It is different Virtual memory from that topic.(va = additionally allocate on hard disc. Not memory)

If you have a change, you must use static s.c because dynamic storage class causes less efficiency(fast, cost , etc...)

There are 2 part on memory

- 1-) stack (lifo algorithm)(fast)(easy accessibility)
- 2-) heap (it used time of work)(slow)(hard accessibility)

\*free'ye NULL pointer göndermek ub değildir.

## Multi-Dimensional Arrays dynamically:

```
6
7     int main(void)
8     {
9         printf("the array with automatic storage class(ON STACK)\n");
10
11     int a[5][5] =           // ON STACK MULTI D. ARRAY ....
12     {
13         [0] = {0, 1, 2, 3, 4},
14         [1] = {5, 6, 7, 8, 9 }
15     };
16
17     int* ptr_a = a[0];
18
19     for (int i = 0; i < 25; ++i)
20     {
21         printf("%d ", *(ptr_a + i));
22         if ((i + 1) % 5 == 0)
23             putchar('\n');
24     }
25
26     printf("\n\n\n");
```

```

printf("the dynamic array (on heap)\n");

size_t row, column;           // ON HEAP MULTI D. ARRAY.....
printf("enter row : ");
scanf("%zu", &row);
printf("enter column : ");
scanf("%zu", &column);

int** ptr = (int**)malloc(row * sizeof(int*)); // allocate for row

if (!ptr)
    exit(EXIT_FAILURE); // test: is available the memory ?

for (size_t i = 0; i < row; ++i)
{
    ptr[i] = (int*)malloc(column * sizeof(int)); //allocate for col

    if (!ptr[i]) //test: is a available the memory ?
        exit(EXIT_FAILURE);
}

srand((unsigned)time(NULL)); //for random

m_array_init(ptr, row, column);
m_array_print(ptr, row, column);

for (size_t i = 0; i < row; ++i) // set to free
    free(ptr[i]);

free(ptr); // set to free
ptr = NULL; // now, invalid.....

putchar('\n');
printf("it is done successfully....");
return 0;

```

!! stack ve heap arasında ciddi bir fark vardır stack'de tüm elemanlar ardışık iken heap üzerinde(malloc ile yapılan)'da elemanlar sadece kendi satırında ardışicktir. Bu durumu düzeltmek için tek satırlı ama zihinden satır kurulabilir.

!!malloc(0) çağrısı unspecified behaviour' a sebep olur pointer kullanılması ub fakat free edilmesi legaldir.

!! reallocation takes time.(reducing efficiency)

!! realloc için malloc gerekmeyebilir, null ile çağrılabilebilir.

## Dynamic Array Data Structure:

Dinamik dizi oluşturulurken mümkün oldukça realloctan kaçınılabilir çünkü reallocation takes time , bunun için geliştirilen işletim sistemi algoritmaları vardır ve dizinin capacity > size ilişkisi bunlar eşit halde geldikten sonra tekrardan reallocation durumunda artık yeni bir heap tutulur ve capacity/size 1.5 gibi bir sabit sayı ile korunur.kullanıcı burada pseudo dynamid array de tanımlayabilir.

Memory leak = free(ptr) yapmayı unutmak!!!

---

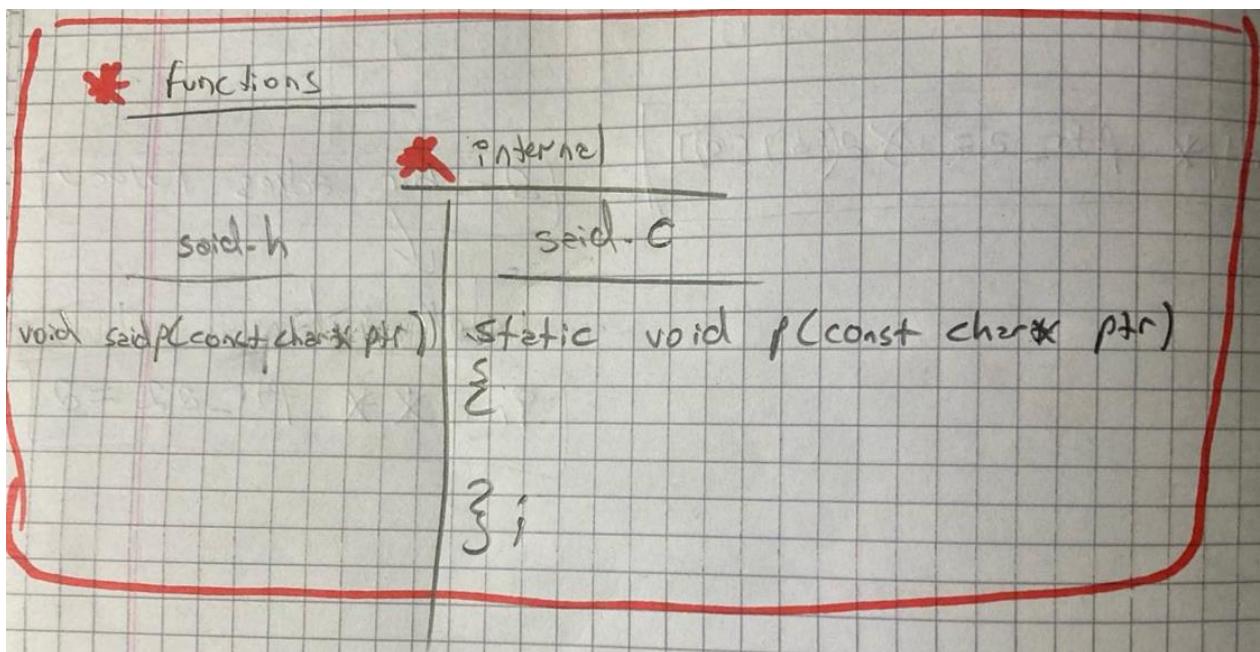
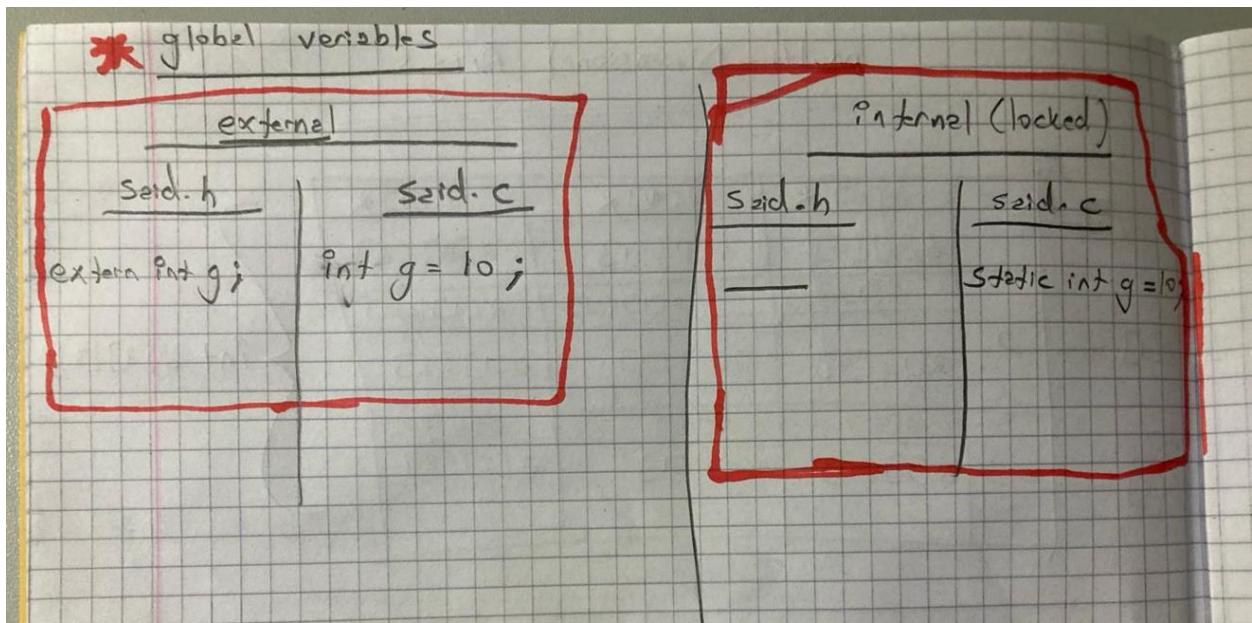
## Storage class specifiers and type qualifiers(modifiers)

Auto(not mind)		const
Register(not mind)		volatile
Extern		restrict(c99)
Static		

## (identifiers (functions and objects))The Concept of Linkage :

There are 3:

- 1-) external linkage(it can be used all modules)
- 2-) internal linkage(it can be used only one module)
- 3-) no-linkage (in only scope)



## Volatile Keyword:

Volatile anahtar kelimesi derleyicinin optimizesini keser ve örneğin volatile ile tanımlanmış bir değişken dış etkenler tarafından değiştirilebilir. Optimizesi kesilmeyen derleyici bunu göremez ve lojik hata meydana gelir bu yüzden gömülü alanda genellikle volatile anahtar kelimesi kullanılır (yani derleyici kısa yol kullanmadan direkt olarak x'in değerini gider bulur.)

Volatile int\* volatile x;

Hem x'in adresi hem de x'in kendisi volatile.

## Restrict keyword :

2. **Restrict (Kısıtlamak):** "Restrict" kelimesi İngilizce bir terimdir ve genel olarak sınırlama, kısıtlama veya sınıflandırma anlamına gelir. Ancak, bağlamına bağlı olarak farklı anımlar kazanabilir. Programlama bağlamında, bir değişkenin veya değerin belirli bir etki alanı veya kapsam içinde kullanılabilirliğini sınırlamak için "restrict" ifadesi kullanılabilir.

Örnek (C dilinde):

```
c                                     Copy code

void exampleFunction(int *restrict arr1, int *restrict arr2, int size) {
    // restrict, arr1 ve arr2'nin örtüşmediğini belirtir
    // Bu, derleyiciye daha iyi optimizasyon yapma şansı tanır
    // ...
}
```

Burada, "restrict" kelimesi özellikle C dilinde belirli bir optimizasyonu ifade etmek için kullanılır ve derleyicilere, belirli bir işlevin argümanlarının birbirine örtüşmediğini bildirir.



## USER DEFINED TYPES :

STRUCTURES: (oop with c)

```
1 #include <stdio.h>
2
3
4
5 struct Data {
6     int x, y, z;
7 };
8
9
10
11 int main()
12 {
13     struct Data mydata;
14     struct Data* pdata = &mydata;
15     mydata.x = 2;
16
17     printf("%d\n", mydata.x);
18     printf("%d\n", (*pdata).x);
19     printf("%d\n", pdata->x);
20 }
```

(->) = adres içindeki nesneye erişim

(.) = nesneye erişim

Aynı struct nesneleri birbirine atanabilir ama farklı türdeki structlardaki nesneler birbirlerine atanamaz!

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

#pragma pack(1)

struct Data {
    int x, y;
};

struct Nec {
    int x, y;
};

int main()
{
    struct Data mydata;
    struct Nec mynec;

    mydata = mynec;
}
```

```
1 #include <stdio.h>
2 #include <stddef.h>
3 #include <string.h>
4
5 #define      SIZE          20
6
7
8 struct Data {
9     char name[SIZE];
10    char surname[SIZE];
11 };
12
13
14 int main()
15 {
16     struct Data mydata = { "said", "mod" };
17     struct Data* pmydata = &mydata;
18     strcpy(pmydata->name, "muhammed said");
19     strcat(pmydata->surname, "ali");
20
21     printf("name = %s\nsurname = %s", pmydata->name, pmydata->surname);
22     return 0;
23 }
24 }
```

```
1 struct {
2     int a, b, c;
3 }x;
4
5 struct {
6     int a, b, c;
7 }y;
8
9 int main()
10 {
11     /*x = y;
12 }
```

(mülakat)

```
1 #include <stdio.h>
2 #include "nutility.h"
3
4
5 typedef struct Data {
6     int x, y;
7 }Data, *Dataptr;
8
9 int main(void)
10 {
11     Data mydata = { 1,2 };
12     Dataptr ptr = &mydata;
13 }
14
15
16
17
```

```
udy
1 #include <stdio.h>
2 #include "nutility.h"
3
4
5
6 typedef struct tagData {
7     int x, y;
8 }Data;
9
10
11
12 int main(void)
13 {
14     Data mydata = { 1,2 };
15     return 0;
16 }
17
```

Time library :

## tm

Defined in header `<time.h>`

`struct tm;`

Structure holding a calendar date and time broken down into its components.

### Member objects

<code>int tm_sec</code>	seconds after the minute - [ 0 , 61 ] (until C99) [ 0 , 60 ] (since C99) (public member object)
<code>int tm_min</code>	minutes after the hour - [ 0 , 59 ] (public member object)
<code>int tm_hour</code>	hours since midnight - [ 0 , 23 ] (public member object)
<code>int tm_mday</code>	day of the month - [ 1 , 31 ] (public member object)
<code>int tm_mon</code>	months since January - [ 0 , 11 ] (public member object)
<code>int tm_year</code>	years since 1900 (public member object)
<code>int tm_wday</code>	days since Sunday - [ 0 , 6 ] (public member object)
<code>int tm_yday</code>	days since January 1 - [ 0 , 365 ] (public member object)
<code>int tm_isdst</code>	Daylight Saving Time flag. The value is positive if DST is in effect, zero if not and negative if no information is available (public member object)

### Complete type and incomplete type

Complete type : bir struct bildiriminin tanımlamasıyla birlikte yapılması.

Incomplete type : bir struct bildiriminin sadece isminin bildirilmesi.

::: incomplete type'da pointer tanımlanabilir ama nesne tanımlanamaz çünkü derleyici kaç byte ayırması gerektiğini bilemez.

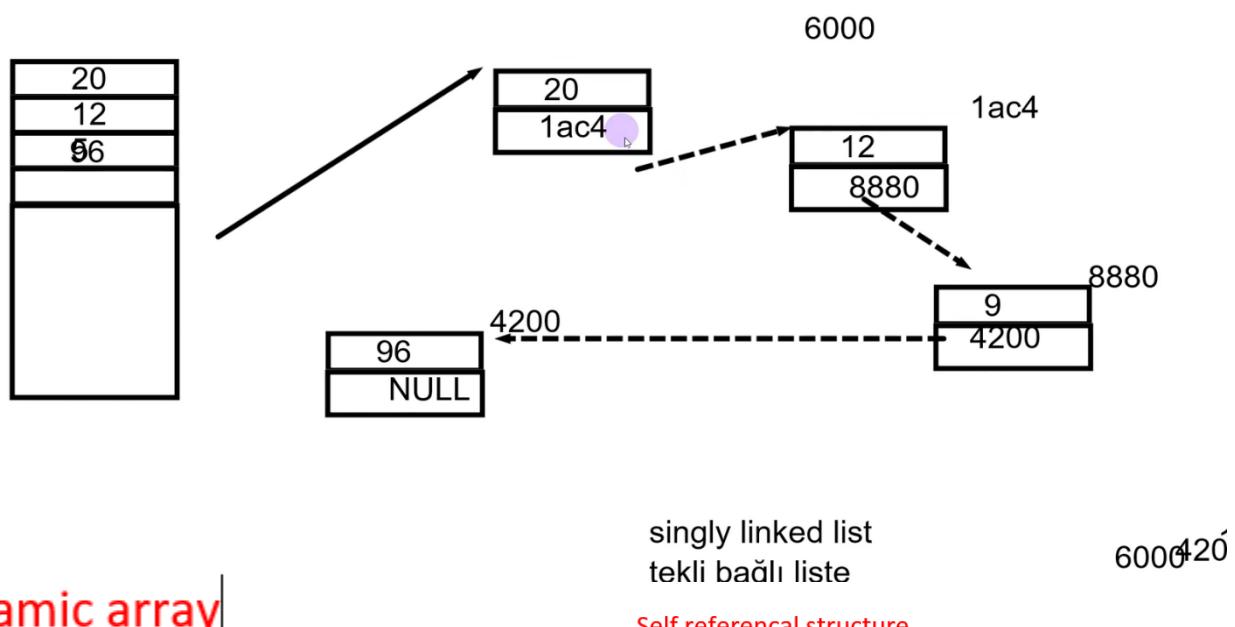
Self referencial structure :

```
struct Data {  
    int x;  
    //  
    struct Data* p;  
};
```

→ böyle bir bildirim yapılabılır çünkü sizeof'u belli.

Composition : struct içinde struct.

\*\* dinamik bellek dizileri heap'te tutulur ömrleri belirli olmadığı için. Bir dinamik dizi tanımlandığında boyutu küçük ise genellikle cache de tutulur. Cache işlemciğe en yakın ön bellektir. Ama yapılarda bu özellik yoktur bunun yerine linkage list kullanılır her biri bir sonrakinin adresini tutar.

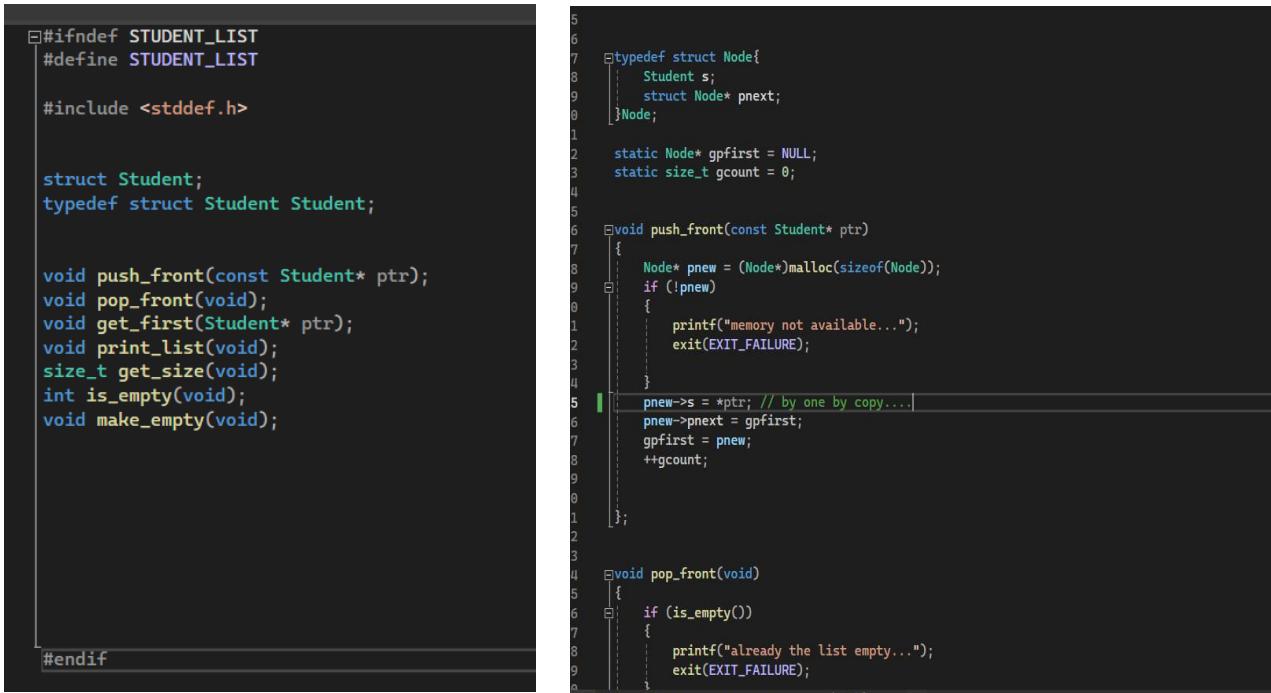


Cache hit : cache'ye yakın

Cache miss : cache'ye uzak.

---

## List handle:



```
#ifndef STUDENT_LIST
#define STUDENT_LIST

#include <stddef.h>

struct Student;
typedef struct Student Student;

void push_front(const Student* ptr);
void pop_front(void);
void get_first(Student* ptr);
void print_list(void);
size_t get_size(void);
int is_empty(void);
void make_empty(void);

#endif

5   typedef struct Node{
6     Student s;
7     struct Node* pnext;
8   }Node;
9
10 static Node* gpfirst = NULL;
11 static size_t gcount = 0;
12
13 void push_front(const Student* ptr)
14 {
15   Node* pnew = (Node*)malloc(sizeof(Node));
16   if (!pnew)
17   {
18     printf("memory not available...");
19     exit(EXIT_FAILURE);
20   }
21   pnew->s = *ptr; // by one by copy...
22   pnew->pnext = gpfirst;
23   gpfirst = pnew;
24   ++gcount;
25 }
26
27 void pop_front(void)
28 {
29   if (is_empty())
30   {
31     printf("already the list empty...");
32     exit(EXIT_FAILURE);
33   }
34 }
```

Burada yapılan sadece linked list kullanmak yani ortada tek bir data var . handle teknigi ile bu data verileri , yani ayrı ayrı içinde isimlerin olduğu liste1, liste2 gibi yapabiliriz ve bu ayrı ayrı görevleri kullanmamızı yani sınıf yapısı oluşturmamıza yardımcı olacaktır. Tek ihtiyacımız olan şey globalde tanımlanan gpfirst ve gcount nesnelerini ayrı bir struct içinde toplamak olacaktır.

```
#ifndef STUDENT_LIST
#define STUDENT_LIST

#include <stddef.h>

typedef struct Student Student;
typedef struct List List;
typedef List* ListHandle;

ListHandle create_list();
void destroy_list(ListHandle h);
void push_front(ListHandle h, const Student* ptr);
void pop_front(ListHandle h);
void get_first(ListHandle h, Student* ptr);
void print_list(ListHandle h);
size_t get_size(ListHandle h);
int is_empty(ListHandle h);
void make_empty(ListHandle h);

#endif
```

```
1  #include "student_list.h"
2  #include "student.h"
3  #include <stdlib.h>
4
5
6
7  typedef struct Node{
8      Student s;
9      struct Node* pnext;
10 }Node;
11
12 typedef struct List {
13     Node* gpfirst;
14     size_t gcount;
15 }List, *ListHandle;
16
17
18
19
20
21
22 ListHandle create_list()
23 {
24     ListHandle h = (ListHandle)malloc(sizeof(List));
25
26
27     if (!h)
28     {
29         printf("memory not available....");
30         exit(EXIT_FAILURE);
31     }
32     h->gpfirst = NULL;
33     h->gcount = 0;
34
35     return h;
36
37
38
39 }
```

Ve ek olarak iki fonksiyon tanımlanması ve diğer fonksiyonlara da bu yeni oluşturulan struct yapısını göndermemiz gerekiyor.

## ALIGNMENT:

Alignment(hizalama):

```
int main(void)
{
    printf("alignof(char) = %zu\n", _Alignof(char));
    printf("alignof(short) = %zu\n", _Alignof(short));
    printf("alignof(int) = %zu\n", _Alignof(int));
    printf("alignof(short) = %zu\n", _Alignof(short));
    printf("alignof(long) = %zu\n", _Alignof(long));
    printf("alignof(long long) = %zu\n", _Alignof(long long));
    printf("alignof(double) = %zu\n", _Alignof(double));
    printf("alignof(float) = %zu\n", _Alignof(float));
}
```

```
ion
in st alignof(char) = 1
dy alignof(short) = 2
Refe alignof(int) = 4
Exte alignof(short) = 2
Head alignof(long) = 4
h c alignof(long long ) = 8
h r alignof(double) = 8
h s alignof(float) = 4
h s C:\Users\said_\Desktop\study\study\Debug\study.exe (process 10940) exited with code 0.
Resc Press any key to close this window . . .
Sour
C
C
r
D
r
D
```

Bir struct yapısının sizeofunu tahmin etmek kolay değildir çünkü tuttuğu nesnelerin byte'ının yanında bilinmesi gereken hizalama yöntemidir.

Alignment requirement : her bir tür için öyle bir 2'nin katıdır ki derleyici buna göre memoryde hizalama yapar örneğin char değerinin \_Alignof(char) değeri 1 dir ve derleyici 1'in katı olan adres değerlerine bu char nesnesinin değerini atar ve böylelikle işlemcinin işi kolaylaşır.

Bu işlem memory maliyetini arttıran ama hızı da arttıran bir yöntemdir.

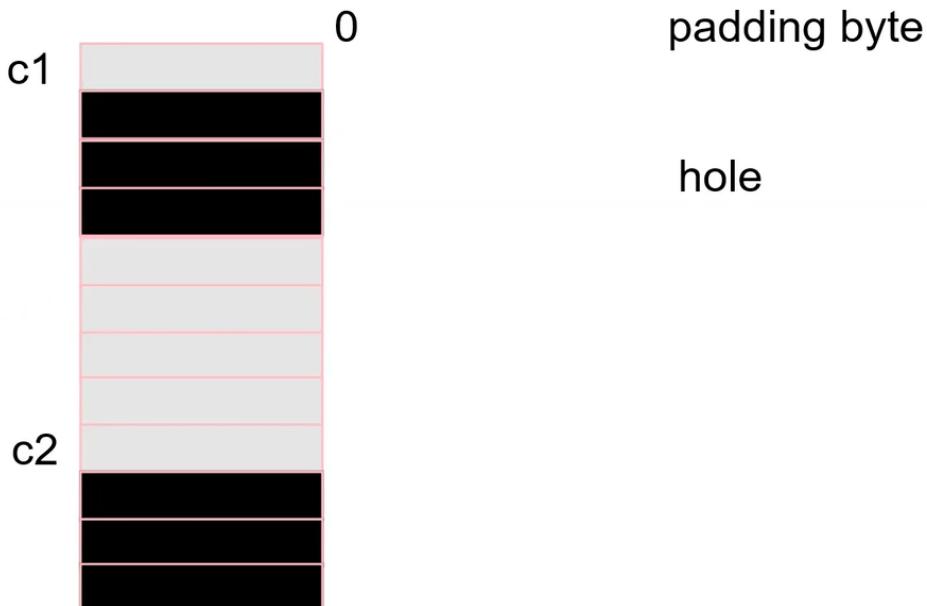
Arada oluşan boşluklara padding byte denir ve minimum padding bay için büyükten küçüğe nesne tanımlamak yerindedir:

Örneğin :

```
struct data {  
    char a;  
    int b;  
    char c;  
    int d;  
};
```

```
struct data {  
    int d;  
    int b;  
    char a;  
    char c;  
};
```

Soldaki kodu sağa dönüştürmek padding bytları azaltacaktır.



→ bu padding byte sayısını azaltmak için eğer donanım uygun ise şu komut kullanılabilir

```
#pragma pack(1)
```

bu komut bütün türlerin alignof'unu 1 yapar ve minimum sayıda padding byte kullanır.hız azalır ama memory kullanımı da azalır.

Offsetof functional macro :

```
1  #include <stdio.h>
2  #include <stddef.h>
3
4
5  typedef struct Data {
6      char a;
7      int b;
8      char c;
9  }Data;
10
11
12  int main(void)
13  {
14      printf("%zu\n", offsetof(Data, a));
15      printf("%zu\n", offsetof(Data, b));
16      printf("%zu\n", offsetof(Data, c));
17      Data obj1 = { 1, 145, 4 };
18      char* pfirst = &obj1.a;
19      int* ptwo = (int*)(pfirst + offsetof(Data, b));
20      printf("%d", *ptwo);
21
22
23
24
25
26 }
```

Offsetof macrosu iki değer alır birincisi sınıf ve ikincisi sınıfın kaçinci elemanı. Bunları kullanarak sınıf ilk üyesinden değer olarak yazılan üyeye kadar adrese kaç eklenmesi gerekiğinin değerini size\_t türünden gönderir biz de bunu kullanarak istediğimiz elemana ulaşırız.yani yukarıdaki kod çıktısının outputu : 145 dir.

```
#define moffsetof(s, e) ((size_t)&(((s*)0)->e))
```

Mülakat sorusu.

Offsetof(s, e) macrosu stddef.h'dadır.

## UNIONS :

Unionlar içерdiği nesnelerin en büyüğünün byte kadar yer ayırır ve tüm nesneler aynı bellek alanını kullanırlar.

The screenshot shows a Microsoft Visual Studio code editor and a debug output window. The code defines two structures: a regular struct with fields x and y, and a union with the same fields. Both have a size of 4 bytes. The debug output window shows the sizes of these structures.

```
1 #include <stdio.h>
2
3
4
5 #pragma pack(1)
6
7
8
9 typedef struct {
10     int x, y;
11 }Stype;
12
13
14 typedef union {
15     int x, y;
16 }Utype;
17
18
19
20 int main(void)
21 {
22     printf("struct : %zu\n", sizeof(Stype));
23     printf("union : %zu", sizeof(Utype));
24     return 0;
25 }
```

Microsoft Visual Studio Debug

```
struct : 8
union : 4
C:\Users\said_\Desktop\study\study\Debug\stu
Press any key to close this window . . .
```

Birliklerin tüm nesnelerinin adresleri aynıdır .

For embedded systems anonymous structure using:

The screenshot shows a code editor with a file named 'y'. It contains a C program using anonymous structures. The union 'Utype' contains an unnamed struct with 'low\_bytes' and 'high\_bytes' fields, and an unnamed 'obj' field of type 'uint32\_t'.

```
y
1 #include <stdio.h>
2 #include <stdint.h>
3
4
5 typedef union {
6     struct {
7         uint16_t low_bytes;
8         uint16_t high_bytes;
9     };
10    uint32_t obj;
11 }Utype;
12
13
14
15 int main(void)
16 {
17 }
```

Diğer kullanımı : as a variant (any type you can use)

Tagged union : ne tuttugunu bilen.

```
#define INT 0
#define DOUBLE 1
#define NAME 2
#define DATE 3

typedef struct {
    int w_type;
    union {
        int int_t;
        double double_t;
        char name_t[20];
        Date date_t;
    };
}Mytype;

void get_type(Mytype* mytype);
void write_type(Mytype* mytype);
```

## ENUMERATIONS:

```
3
4
5 typedef enum {
6     White, Black, Purple, Pink, Yellow, Blue
7 }
8 Color;
9 //thats int type.
10
11
12
13
14 int main(void)
15 {
16     Color x = White;
17     printf("%d\n", x);
18     return 0;
19 }
20
21 }
```

The screenshot shows a Microsoft Visual Studio interface. On the left is a code editor window titled "Study" containing the following C code:

```
1 #include <stdio.h>
2
3
4
5 typedef enum {
6     White, Black, Purple, Pink, Yellow, Blue
7 }Color;
8
9 //thats int type.
10
11
12
13
14 int main(void)
15 {
16     printf("%d\n", White );
17     printf("%d\n", Black );
18     printf("%d\n", Purple);
19     printf("%d\n", Pink);
20     printf("%d\n", Yellow );
21     printf("%d\n", Blue);
22     return 0;
23
24
25
26 }
```

To the right is a "Microsoft Visual Studio Debug" window showing the output of the program:

```
0
1
2
3
4
5
C:\Users\said_\Desktop\study\study\Debug
Press any key to close this window . . .
```

This screenshot shows the same Microsoft Visual Studio interface, but the code in the editor has been modified:

```
2
3
4
5 typedef enum {
6     White = 12, Black, Purple, Pink, Yellow, Blue
7 }Color;
8
9 //thats int type.
10
11
12
13
14 int main(void)
15 {
16     printf("%d\n", White );
17     printf("%d\n", Black );
18     printf("%d\n", Purple);
19     printf("%d\n", Pink);
20     printf("%d\n", Yellow );
21     printf("%d\n", Blue);
22     return 0;
23
24
25
26 }
```

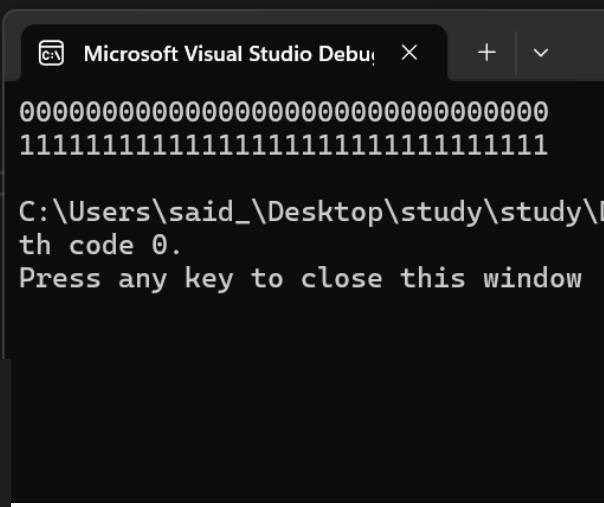
The "Microsoft Visual Studio Debug" window now displays the output:

```
12
13
14
15
16
17
C:\Users\said_\Desktop\study\study\Debug
Press any key to close this window . . .
```

Like a macro.

## BITWISE OPERATIONS :

```
2 ~ (bitwise not)
5 >> << (bitwise left/right shift)
8 & bitwise and
9 ^ bitwise exor
10 | bitwise or
14 &= ^= |= >>= <<=
```



The screenshot shows a code editor and a debug window. The code editor contains C++ code that prints the binary representation of an integer. The debug window shows the output of the program.

```
2 #include "nutility.h"
3
4
5
6
7
8 int main()
9 {
10     int x = 0;
11     base2_print(x);
12     base2_print(~x);
13     return 0;
14 }
15
16
17
```

```
void base2_print(int val)
{
    static char ptr[40];
    _itoa(val, ptr, 2);
    printf("%032s\n", ptr);
```

Microsoft Visual Studio Debug

```
00000000000000000000000000000000
11111111111111111111111111111111
```

C:\Users\said\_\Desktop\study\study\I  
th code 0.  
Press any key to close this window

\*\*bitsel sola veya sağa kaydırırmada

Örneğin : int a =1; int 4 byte ise

a<< (buraya maksimum 31 gelir ve negatif değer gelemez ub!)

bitwise operationslarda atama hariç side effect yoktur yani değerin kendisi asla değişmez.

x<<1 = bitsel sola kaydırma, sola 1 kaydırır sağdan 1 ekler.

x>>1 = bitsel sağa kaydırma, sağa 1 kaydırır sola bir ekler(eğer negatif ise 0 veya 1 eklemesi sağa kaydırma için sadece derleyiciye bağlıdır.)

& = bitwise and operators

| = bitwise or operations

^ = bitwise xor operators

Mülakat sorusu : üçüncü değişken olmadan swap.

The screenshot shows a Microsoft Visual Studio interface. On the left, the code editor displays a C program named 'Study'. The code defines a main function that prompts the user to enter two numbers, swaps them using XOR assignment, and then prints the swapped values. On the right, the 'Microsoft Visual Studio Debug' window shows the program's output. It asks for two numbers, receives '10' and '20', and then prints the swapped values as 'x = 20, y = 10' and 'x = 10, y = 20'.

```
1 #include <stdio.h>
2 #include "nutility.h"
3
4
5
6 int main(void)
7 {
8     int x, y;
9     printf("enter two number : ");
10    scanf("%d%d", &x, &y);
11    printf("x = %d, y = %d\n", x, y);
12    x ^= y, y ^= x, x ^= y;
13    printf("x = %d , y= %d", x, y);
14    return 0;
15
16 }
```

Microsoft Visual Studio Debug

enter two number : 10  
20  
x = 10, y = 20  
x = 20 , y= 10  
C:\Users\said\_\Desktop\study\|  
Press any key to close this window

Sayının n.bitini set etmek :

$$x |= (1 << n)$$

The screenshot shows a Microsoft Visual Studio interface. On the left is a code editor with a dark theme containing C++ code. The code includes `#include <stdio.h>`, `#include "nutility.h"`, and a `main` function that initializes `x` to 0, prints its base-2 representation, shifts it left by 5 bits, prints it again, and then returns 0. On the right is a debugger window titled "Microsoft Visual Studio Debug". It displays the memory dump of variable `x` in hex format: `00` followed by `0001000000`. Below the dump, the path `C:\Users\said\_\Desktop\study\study\Debug` is shown, along with the message "Press any key to close this window . . .".

Sayının n.bitini reset etmek :

$$x \&= \sim(1 << n)$$

Basixc using :

The screenshot shows a code editor with a dark theme containing a C program. The program defines a `main` function that initializes `x` to -1, measures the start time using `clock()`, and then enters a loop. In each iteration of the loop, it performs a bitwise AND operation with the complement of a bit shifted left by 32 positions (effectively clearing the nth bit). It then prints the current value of `x` using `base2\_print(x)` and sleeps for 50 milliseconds using `Sleep(50)`. Finally, it prints a message indicating the process is done and provides the execution time in seconds.

Sayının n. Bitini flip etmek :

```
x ^= (1 << n);
```

Sayının n. Bitini get etmek :

```
if (x & (1 << n))
```

## COMMAND LINE ARGUMENTS

```
int main(int argc, char* argv[])
{
    if (argc != 4) {
        printf("kullanim: <hes> <sol operand> <islem +-*> <sag operand>\n");
        return 1;
    }

    int op_left = atoi(argv[1]);
    int op_right = atoi(argv[3]);

    switch (argv[2][0]) {
        case '+': printf("%d\n", op_left + op_right); break;
        case '-': printf("%d\n", op_left - op_right); break;
        case '*': printf("%d\n", op_left * op_right); break;
        case '/':
            if (op_right == 0)
                printf("sifira bolme hatasi\n");
            else
                printf("%f\n", (double)op_left / op_right);
            break;
        default: printf("gecersiz operator\n"); break;
    }
}
```

Null pointer :

```
argv[argc]
```

## FILE OPERATIONS

```

5965 =====
5966 =====
5967 açış modu           varsa/yoksa          ne yapabilirim
5968 okuma (read)       varsa açılacak yoksa açılmayacak okuyabilirim / yazamam
5969 yazma (write)      varsa truncate yoksa oluşturulacak yazabilirim / okuyamam
5970 sona ekleme (append) varsa açılacak yoksa oluşturulacak sona yazabilirim / okuyamam
5971
5972 okuma +(read)      varsa açılacak yoksa açılmayacak okuyabilirim / yazabiliyorum
5973 yazma +(write)     varsa truncate yoksa oluşturulacak yazabilirim / okuyabiliyorum
5974 sona ekleme + (append) varsa açılacak yoksa oluşturulacak sona yazabilirim / okuyabiliyim
5975
5976
5977
5978
5979
5980
5981
5982
5983
5984

```

Default mode : txt.

"r"	"rb"	
"w"	"wb"	
"a"	"ab"	
<hr/>		
"r+"	"r+b"	"rb+"
"w+"	"w+b"	"wb+"
"a+"	"a+b"	"ab+"

```

int main(int argc, char** argv)
{
    // the usage : <write.exe> <filename>

    if (argc != 2)
    {
        fprintf(stderr, "the usage : <write.exe> <filename>\n");
        return 1;
    }

    FILE* f = fopen(argv[1], "r");
    if (!f)
    {
        fprintf(stderr, "the file doesnt exist...\n");
        return 2;
    }

    int ch;

    while ((ch = fgetc(f)) != EOF)
        putchar(ch);

    fclose(f);
    return 0;
}

```

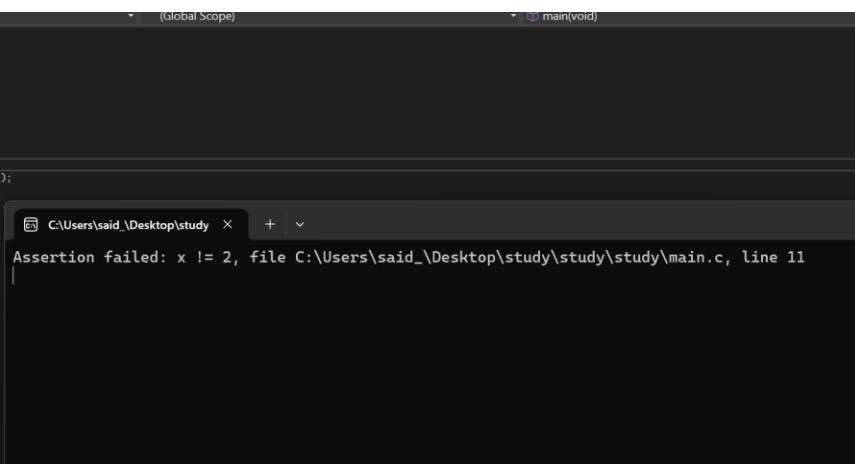
\*\* dosyanın yazı karakterleriyle ilgisi yok ise binary modda açılması gereklidir.

Önemli : rb+ modunu kullanırken seek pointer göstericisi ayarlanmalıdır.

```
while (fread(&s, sizeof(s), 1, f)) {
    if (!strcmp(poldname, s.name)) {
        strcpy(s.name, pnewname);    I
        fseek(f, -(long)sizeof(s), SEEK_CUR);
        fwrite(&s, sizeof(s), 1, f);
    }
}
```

```
13 fseek(f, -30L, SEEK_CUR);
14 fseek(f, 40L, SEEK_CUR);
15 fseek(f, 0L, SEEK_CUR);    I
16
17
18 EN SON dosyadan yapılan işlem okuma işlemi ise ancak bir sonraki yazma işlemi olacak ise
.0 EN SON dosyadan yapılan işlem yazma işlemi ise ancak bir sonraki okuma işlemi olacak ise
1.
2. file pointer mutlaka konumlandırılmalı
3.
```

#include <assert.h> => it includes assert(write an expression must be.)



The screenshot shows a terminal window with the following output:

```
C:\Users\said_\Desktop\study x + v
Assertion failed: x != 2, file C:\Users\said_\Desktop\study\study\study\main.c, line 11
```

## Qsort and bubble sort algorithm :

```
11
12
13 int main(void)
14 {
15     int a[SIZE];
16     srand((unsigned)time(NULL));
17
18     set_array_random(a, SIZE);
19     print_array(a, SIZE);
20     printf("\n-----\n");
21
22
23     for (int i = 0; i < SIZE; ++i)
24         for (int j = 0; j < SIZE - 1; ++j)
25         {
26             if (a[j] > a[j + 1])
27             {
28                 int temp = a[j];
29                 a[j] = a[j + 1];
30                 a[j + 1] = temp;
31             }
32         }
33
34     print_array(a, SIZE);
35     return 0;
36
37
38
39
40 }
```

```
study (Global)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "nutility.h"
4
5
6
7 #define      SIZE      1000
8
9
10 int my_cmp(const void* vp1, const void* vp2)
11 {
12     return *(int*)vp1 - *(int*)vp2;
13 }
14
15
16 int main(void)
17 {
18     int a[SIZE];
19
20     set_array_random(a, SIZE);
21     print_array(a, SIZE);
22     qsort(a, SIZE, sizeof(int), my_cmp);
23     print_array(a, SIZE);
24     return 0;
25
26
27 }
```

The Libraries....

```
*****  
#include <stdio.h>  
  
**int printf(const char *format,...); ==> how many written...  
**int sprintf(char* buff, const char* format, ...) ==> it writes on buff unlike printf standard output  
int fprintf( FILE *restrict stream, const char *restrict format, ... ); ==> it writes on file...  
  
int scanf(const char * format , ...) ==> success information....  
int sscanf(char* buff, const char* ptr, ...) ==> it gets from the str  
int fscanf( FILE *restrict stream, const char *restrict format, ... ) ==> if there is no data returns EOF (be carefull what contains.)  
  
int putchar(int ch) ==> information of written character ascii code ...  
int getchar(void) ==> information of taken character ascii code ... (yes enter and eco)  
puts(str) ==> it prints str and adds \n  
  
FILE* fopen(const char* pfname, const char* popenmode);  
int fclose(FILE* pfname); the return value open file pointer.  
int fcloseall(void); the return value : succes colosed file numbers.  
int fgetc(FILE*) ==> for example, you write one character(1 byte) on txt file and you call the fgetc function and fgetc function,  
that returns ascii character code if it is empty it returns EOF(-1) macro. dont forget that the return value holds by a integer variable.  
int fputc(int c_value, FILE* f);  
int remove( const char* pathname ); ==> it deletes the file and if it success , return 0;(firstly, the file must be closed.)  
int rename(const char* oldfile_name, const char* newfile_name); ==> returns 0(success);  
char* tmpnam(char* empty_str) ==> it produces unique filename.  
char* fgets(char* buff, int size_buff, FILE* file) ==> if no any characters it returns NULL .  
//no formatted and writes byte byte functions  
size_t fread(void* vp, size_t block_size, size_t ntimes, FILE* file). returns last what reads times.(void* = &x); ==> if no any data returns 0.  
size_t fwrite(void* vp, size_t block_size, size_t ntimes, FILE* file). returns last what writes times. ,,, fread and fwrite must be binary mode files  
fseek(FILE* f, +-byte, SEEK_CUR);  
fseek(FILE* f, +- (usually 0), SEEK_SET) => it turns first byte.  
  
*****
```

#include <math.h>

```
double pow (double base, double exponent ) ==> returns the pow...  
int abs( int n ) ==> returns the absolute value of n ...  
double sqrt(double arg) ==> returns the square root of the arg ....  
exp()  
ceil()
```

```
*****  
#include <conio.h> // private for microsoft ...  
  
_getch() : writes no enter and no echo  
_getche() : writes with no enter and yes echo
```

```

79 ****
80 #include <ctype.h>
81
82
83 *****control*****
84 int isalnum (int) ==: checks the is...
85 int isalpha (int) ==: checks the is...
86 int islower (int) ==: checks the is...
87 int isupper (int) ==: checks the is...
88 int isdigit (int) ==: checks the is...
89 int isxdigi (int) ==: checks the is...t
90 int iscntrl (int) ==: checks the is...
91 int isgraph (int) ==: checks the is...
92 int isspace (int) ==: checks the is...
93 int isblank (int) ==: checks the is...
94 int isprint (int) ==: checks the is...
95 int ispunct (int) ==: checks the is...
96 *****toany*****
97
98
99 int tolower(int) : converts if is upper , if is not itself. ==> return ascii...
100 int toupper(int) : converts if is upper , if is not itself. ==> return ascii...
101
102
103
104 ****
105
106 #include <stdlib.h>(add qsort)
107
108 RAND_MAX : the macro of srand() function round 0 <= x <= RAND_MAX
109 int rand(void) ==> the random numbers in numbers chain default seed value(1)
110 void srand(unsigned int ) ==> changes to number chain.according to epoch of linux starting 1.1.1970 from the second that is over\
111
112
113 int atoi(const char* ptr) ==> it gets number by changing from str to int
114 long int atol(const char* ptr) ==> str to long int...
115 double atof(const char* ptr) ==> str to double ...
116 char* itoa(int val, char* buf, int base) ==> converts number to number system as char you wanna (not standard)
117
118 //end process
119 //normal termination:
120 void exit( int exit_code ); => it ends at that point and it guarantees buffer flush (if there is open file)
121 macros :
122 EXIT_FAILURE == non-zero      using exit(EXIT_FAILURE)
123 EXIT_SUCCESS == 0
124 int atexit( void (*func)(void) ) ==> before every exit (included return 0) call, it calls func (according to LIFO algorithm)
125
126 //abnormal termination
127 void abort(abort) ==> it stops suddenly...
128
129 // dynamic memory management
130 void* malloc(size_t n) ==> it allocates an area on memory with garbage values.
131 void* calloc(size_t n, size_t sz) ==> it allocates an area on memory with non garbage values (all 0)
132 void* realloc(void* ptr, size_t new_size) ==> it allocates new size , adds or removes on new area or old ares.(with malloc algorithmm.) *it takes time
133 void free(void* prt) ==> it set the that area free on memory with zero values automatically.
134
135
136
137
138
139
140
141
142

```

```
147 #include <time.h>
148 typedef declarations :
149 time_t : it is generally used long long etc. to portable.
150
151
152 struct tm : it is broken down time(you can get second, min , hour, day, year according to some rules..)
153 time_t time(time_t *ptr) ==> always changes second from 1970...(two options : return value with null or pointer value changes...)
154 struct tm* localtime(const time_t* timer) ==> it gives hour, min, sec....
155 struct tm* gmtime(const time_t* timer) ==> it gives hour, min, sec... but as a gm.
156 !!char* ctime(const time_t* const timer) ==> it gives a str included day,mon,year, hour.
157 char* asctime(const struct tm* timer) ==> actually it is the same ctime but it needs to return value of localtime function.
158 size_t strftime(char* ptr, size_t size, "format type", const struct tm* timer)
159 reel using random : srand((unsigned)time(NULL))
160
161 ****
162 #include <stddef.h>
163
164 some typedefs ==>
165 size_t ==> generally len (the type : sizeof's return value) so use %zu format type.
166 ptrdiff_t ==> difference of two address(return value)
167 offsetof(s, e) ==> it is a functional macro returns the size_t Z number.
168
169
170
171
172 ****
173 #include <string.h> (ADD memset....)
174
175 size_t strlen(const char* p); ==> returns lenght of str
176 char* strchr(const char* p, int c); ==> if it is founded , returns the address of character . if not returns NULL pointer.
177 char* strrchr(const char* p, int c) ==> it starts the end of str
178 char* strstr(const char* p, const char searching) ==> if it finds returns that first address if not returns null pointer.
179 char* strpbrk(const char* p, anystr) ==> is there any in anystr if there is no even one returns null pointer
180 char* strcpy(char* pdest, const char* psource) ==> (const sonda), it copies from source to pdest. it returns pdest first address.
181 be carefull(overlap) it is not the same array.
182 char* strcat(char* pdest, const char* source) ==> adss source on pdest
183 int strcmp(const char* p1, const char* p2) ==> if p1 > p2 = returns > 0, if p1 = p2 = returns = 0, if p1 < p2 returns < 0
184 //that strcmp works according to lexicographical compare or container comparison (dictionary)
185 //so , 1234 > 123(> 0), z > abcdefghijklmn....(according to ascii table)
186 void* memset(void* ptr, int set_value, size_t sizeofptr); ==> it sets with set_value all bytes....
187 void* memmove(void* dest, const void* src, size_t n); ==> it copies , "Even if the addresses overlap, it is safe." == so use memmove instead of memcpy.
188
189
190
191
192 #include <assert.h>
193 assert(write the expression must be.);
```