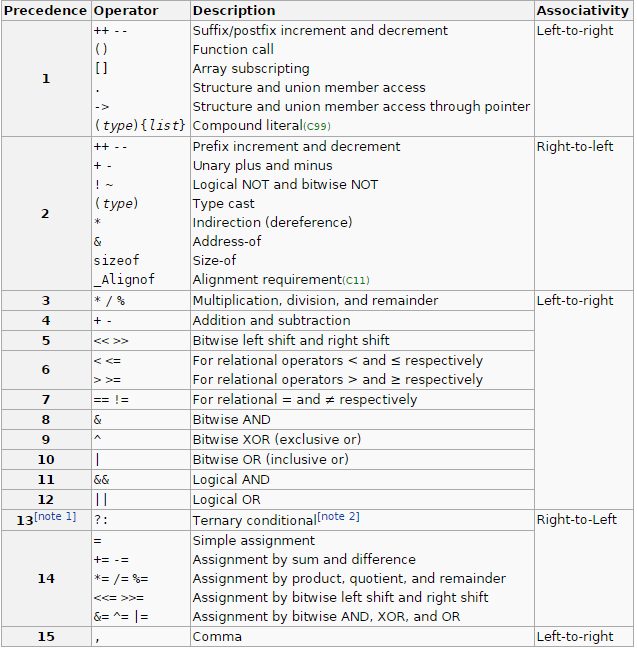
**Part 1**

* **C Operators**



**-Operator types:**

-Arithmetic Operators; +, -, \*, /, %

-Increment and Decrement Operators; ++, --

-Relational Operators; ==, !=, >, <, <=, >=

-Logical Operators; &&, ||, !

-Cast Operators; (float)sum; (int)fred;

-Bitwise Operators; ~, &, |, ^, <<, >>

-Assignment Operators; =, +=, -=, \*=, /=, %=, >>=, <<=, &=, ^=, |=

- “->” Operatörü ise main dışında pointer ile gösterilen struct üyesine ulaşmak için kullanılır.

“.” Operatörü “->” ile aynı işi yapar ama main içinde kullanılır.

- x = ++y operatörünün x = y++ dan farkı, loop için kullanılmıyorsa ++y kullanılırsa önce rakam arttırılır sonra işlem yapılır(mesela x = ++y), y++ kullanılırsa önce atama yapılır sonra arttırılır. Loop içinde kullanılıyorsalar bir fark yoktur. (tag: postfix, prefix)

- **Advanced Assignment** -> x = x - 5 eşittir x -= 5

while (x --> 0) = while( (x--) > 0 )

**-Data Type Changes:**

**-Cast operator**

İki int değeri böldüğünde elde ettiğin sonuç yine int olacaktır. Float’a atasan bile int bir değer atarsın. Dolayısıyla cast operatör kullanman lazım.

*degisken\_1 = ( tip ) degisken\_2;*

int bolunen = 12, bolen = 8;

float bolum;

bolum = (float) bolunen / bolen;

printf("Sonuc: %f\n",bolum);

**-Domine değişken**

İki farklı tür değişken veya literal ile işlem yapılrken az yer kaplayan data tipi yok yer kaplayan data tipine dönüşür.

DÜŞÜK- char <-> short <-> int <-> long <–> long long <-> float <-> double -YÜKSEK

**-C de değişkenleri manipule etme:** printf( “%c”, A ); derken sağ tarafa A yazmamalısın. Çünkü A demek değişken demek. Yani değeri demek. Ama bize değeri değil ekrana değişkenin ismi yazdırmak önemli. Bu yüzden ascii değerini karakter olarak yaz diyeceğiz. printf( “%c”, ‘A’ );

***-*** Tam bir sayıyı **10 a bölmek** son rakamı yok eder. **10 a bölümden kalanı almak** son haneyi alır. Rakam sayısı kere 10 a bölüncede geriye 0 kalıyor.

### -Kısa Devre Değerlendirme/ Short Circuit Evaluation

### Bir koşul içersinde AND ( && ) operatörü kullandığınızda, koşulun sol tarafı yanlışsa, sağ tarafı kontrol edilmez. Çünkü artık sağ tarafın doğru veya yanlış olmasının önemi yoktur; sonuç her şekilde yanlış olacaktır.

Benzer bir mantık OR ( || ) operatörü içinde geçerlidir. Eğer sol taraf doğruysa, sağ tarafın kontrol edilmesine gerek yoktur. Çünkü OR operatöründe taraflardan birinin doğru olması durumunda, diğerinin önemi kalmaz ve sonuç doğru döner.

Nasıl kullanılabilir: if( i == 1 & j++ ) , j nin değerini bir arttırır. If’e girmez

Bunun kötü yanı şöyle olabilir. (a <b) && (++c < 10) kodunda c arttırılmaz. Ama yazarken buna bilerek yazarsan için sorun olmaz.

* **C Data types**

**-** In C and C++ **sizes of data types** change from platform to platform.

**-Basic Types:** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.

**-Enumerated types(enum):**They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.

**-The type void:** The type specifier void indicates that no value is available.

**-Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

|  |  |  |
| --- | --- | --- |
| **Type** | **Storage size** | **Value range** |
| char - c | 8 bit - 1 byte | -128 to 127 or 0 to 255 |
| signed char – c or hhi for numerical output | 8 bit - 1 byte | -128 to 127 |
| unsigned char – c or %hhu for numerical output | 8 bit - 1 byte | 0 to 255 |
| short – hi | 16 bit - 2 byte | -32,768 to 32,767 |
| unsigned short - hu | 16 bit - 2 byte | 0 to 65,535 |
| int – d or i | 32 bit - 4 byte | -2,147,483,648 to 2,147,483,647 |
| unsigned int - u | 32 bit - 4 byte | 0 to 4,294,967,295 |

|  |  |  |
| --- | --- | --- |
| pointer - p | 64 bit - 8 byte | 0 to FFFFFFFFFFFFFFFF which is equal to (18,446,744,073,709,551,615)10 |

-Diğer değişkenler gibi pointerında büyüklüğü OS’in, compiler’ın vs. kaç bit olduğuna göre değişiyor.

-Signed sadece char için bir anlam ifade ediyor. Bu anlam ise char tek başına signed’mı unsigned’mı olduğunu bilmiyor diğer veri tiplerinin aksine.

-Eğer bir char tırnak içinde değilse karakter değil ascii değeri tutar. Mesela print fonksiyonuna x değerini yazdır dersek x’in ascii değerini yazdırır. ‘x’ i yazdır dersek x karakterini yazdırır. “x” i yazdır dersek x string’inin yazdırır.

- char c = ‘a’ yada char c = 97 şeklinde atama yapabilirsin.

- Printf(“%c”, c-10); şeklindede yazdırabilirsin

- ++ndigit[c – ‘10’]; kalibi belli bir ard arda gelen ascii grubunu 0 indisinden baslayarak kullanmaya yarar. Chars are just small integers. So c - ‘0’ is an integer expression.

**Printf”%c”, x);** x değişkeninin sayısal değerini char olarak yazdırır.

**Printf”%c”, ‘x’);** x karakterini ekrana yazdırır.

**Printf”%c”, “x”);** hiçbirşey yazdırmaz. Çünkü tek karakter string değildir.

**Printf”%d”, x);** x değişkeninin sayısal değerini int cinsinden yazdırır

**Printf”%d”, ‘x’);** x karakterinin sayısal karşılığını(ascii) int cinsinden yazdırır

x++ dediğimizde x’in sayısal değeri artar tekrar yazdırınca sonraki ascii gelir.

**Printf”%d”, “x”); undefined behaviour**

**-Undefined behaviour:** c de tanımlı olmayan işlemler

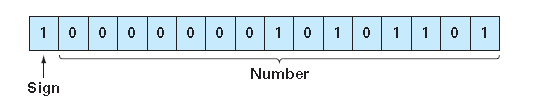
-signed variables in c use one bit as sign flag. Char is 8 bit, 2 byte. But it is 7 bit long. -128 to 127. Left most digit is sign flag. Short is 16 bit, 4 byte. But it is 15 bit long. -32,768 to -32,767. İnt is 32bit, 8byte. But its 31 bit long. 2,147,483,648. If you define an unsigned variable, it will be twice as long. For example an unsigned short will have a limit of 64k not 32k. But it cant take negative values. İf we have a 64 bit signed variable(16 byte), it can be 63 bit long. 9.22337204E+18.

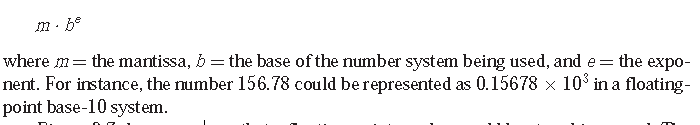
-The negative is -128 and the positive is 127 because negative can be 10000000(left most is both acting as sign bit and data bit), but positive can only be 01111111(left most cant be 1 because that means negative number. İf you want to be able to use it, you need to use unsigned variable).

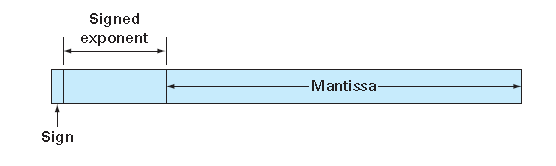
|  |  |  |
| --- | --- | --- |
| Long long –llu | 64 bit - 8 byte | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| Unsigned long long –lli | 64 bit - 8 byte | 0 to 18,446,744,073,709,551,615 |

- Değişkenlerin kaç bit olduğu compiler’dan compiler’a(ve bit sayısı) değişiyor. Yeni derleyicilerde int 16 bit olduğu için daha büyük sayıları ihtiyaç olmadıkça long ve long long gibi değişkenlere ihtiyaç yok.

-Bilgisayarda sayılar şekildeki gibi tutulur;

**

****

****

**-Floating-Point Types**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Storage size** | **Value range** | **Precision** |
| float - f | 32 bit - 4 byte | 1.2E-38 to 3.4E+38 | 6 decimal places |
| double – F, lF(scanf) | 64 bit - 8 byte | 2.3E-308 to 1.7E+308 | 15 decimal places |
| long double - LF | 80 bit - 10 byte | 3.4E-4932 to 1.1E+4932 | 19 decimal places |

**-Void tipleri**

-Function returns as void: void exit (int status);

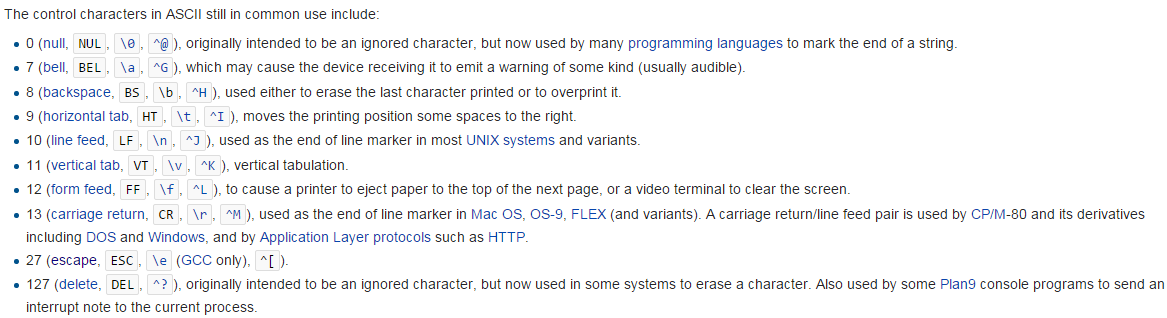
-Function arguments as void:  int rand(void);

-Pointers to void: A pointer of type void \* represents the address of an object, but not its type. For example, a memory allocation function **void \*malloc( size\_t size );** returns a pointer to void which can be casted to any data type.

**-Escape character and Escape sequences(Control sequences):** (triggers escape sequences): “\” is the escape character. Escape character is used to print anything you want to be printed without compiler trying to recognize it as someting else. For example \c will print c and \\c will print \c \n is an escape sequence.

-\ is not the backwards slash. \ is the escape sequence. If you want to say if c equals to back slash you say c == \\. And as you know if you want to print \\ you gotta say print \\\\. When you use backslash in odd numbers the unused back slash takes the doubles quotes and other characters around it and uses them to form escape sequences.



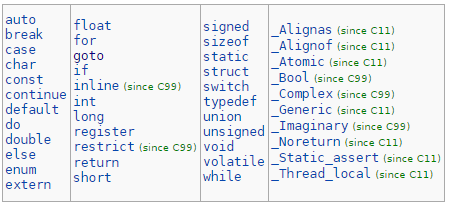


- %o for octal and %x for hexadecimal.

- Escape sequences are single characters. \n is a single character. '\n' stands for the value of the newline character, which is 10 in ASCII. You should note carefully that '\n' is a single character, and in expressions is just an integer; on the other hand, '\n' is a string constant that happens to contain only one character.

**-\r carriage return:** Satır başı yapar ve satırda yazanları siler.

* **C Keywords(Reserved Words)**



**-Auto:** These variables are local to their block. Their storage space is reclaimed on exit from the block. Declared when program enters its scope and they are removed from memory when program exits the scope.

**-Const/Literal**(const int x) değişkenin değeri sabittir değişmez. Ama bir detay var.

C:\Users\Anıl\Desktop\Capture.PNG

-Bir int verisinin tek değeri vardır tek const yeter. Ama bir pointer’ın iki değeri vardır. Pointer’ın tuttuğu adres. Ve adresteki değer. Eğer tek const kullanırsan diğerini değiştirebilirsin. Mesela yukardaki örnekte pointer’ın başında const yok ama yıldızdan sonra var. Bu neyin adresini gösterdiğini değiştirememize sebep oluyo ama adresteki değeri değiştirebiliyoruz. Değerinde sabit olmasını istiyorsak const pointer’ın gösterdiği adreste bir const değişken olabilir.

-Also we call 10 in “int a = 10;” an integer constant. We call A in “char c = `A`;” a char constant. We call A in “char string1[] = “A”;” is a string constant(character string). Its not a good idea to use integer values of characters like “ c = ‘65’ “ because character sets change from machine to machine.

-**Dynamic**(int x)ise program tekrar çalıştığında farklı bir değere sahip olabilir. Dynamic kelimesi kullanımadan direk int x; şeklinde kullanılır.

-**Extern:** Extern variable is a variable that is outside every function. An external variable. Generally it is used to access a variable in the header file you are using.

-**Static**(static int x): Eğer static kelimesini bir foksiyonda bir veri için kullanıyorsan, o veri artık kaydedilir. O fonksiyonu ikinci kere çağırışında ilk çağırıştaki değişmiş hali ile kullanılır. Sıfırdan kullanılmaz, atama yapılmaz. Verinin ömrü fonksiyonun ömrü kadar değil tüm programın ömrü kadar oluyor. Eğer global bir veriyi yada bir fonksiyonu static olarak tanımlıyorsan bu .c dosyası dışındada tanınsın demektir.

**-Static external:**

**-Volatile:** Kod bu değeri değiştirmese bile çeşitli sebeplerden dolayı bu değer her an değişebilir anlamına gelir.

**-Register:** If possible, the variable will be stored in a processor register. May give faster access to the variable. If register storage is not possible, then the variable will be auto. Not recommended to use. Compiler will decide what to save in register instead.

**Part 2**

* **History and Physiology of C**

-C dilini Dennis Ritchie, Bell Laboratuarlarında 1972 de UNIX işletim sistemini yazmak için geliştirmiştir. B ve B’nin etkilendiği dillerden esinlenilmiştir. C dili genel amaçlı, yüksek seviye bir dildir.

-Paradigms: Structured, Procedural(Imperative).

-Typing dicipline: Static, weak, manifest, nominal. <https://en.wikipedia.org/wiki/C_(programming_language)>

- C dilinde **compiler ve linker** çalışma mantığı: Bir veya birden çok fonksiyon yazılır. Fonksiyonlar compiler tarafından .c den .obj’e çevrilir. .obj’ler ise linker tarafından birleştirilip .exe’ye çevrilir. Bu .exe ram e yüklenip ordan işlemciye yüklenip çalıştırılır.

-C dilini command prompt’ta kullanmak için .c değil .exe dosyasına, yani makine koduna ihtiyacımız var. Çalıştırırken .exe’nin adresine “C:”, “D:” ve “cd” ile gelip programın ismini yazmamız yeterlidir. Eğer main fonksiyonu’nun argüman istediği bir yazılımsa isim den sonra argümanarı tek tek giriyoruz.

-Command propmt’ta bir dosyanın içindeki bir dosyaya girmek için “cmd alt dosya ismi” yazmamız yeterlidir.

* **-Warning(warning shown, does run):** Değişken yaratıp değer atamamak, kullanmamak.

**-Error:** Yazım hataları, tanımlanmış değişkenleri kullanmak.

**-Compile time errors:**

**-Syntax error(error shown, doesnt run):** Wrong syntax. They are also called compiler errors since compile show them during compliation phase. Also called compiler error, compile-time error, compliation error.

Using variable without declaration.

Forgetting a “;”

When one brace of a block is mising.

**-Semantic error/Logic error(error shown, doesnt run):** Things that are not meaningful to complier.

b+c=a;

Not using both brace of a block might cause problems like “Dangling-else Problem”.

**-Run time(execution time) errors/ Logical errors:**

**-Design error (error not shown, does run):** Produces wrong output. Cant be detected by complier.

(a+b) / 2 yerine a + b / 2 kullanmak.

**-Fatal error (error not shown, does run and crash):** Hata compile sırasında yok. Program çalışırken oluşur. Sonsuz döngü, bölen sayının 0 olması, scanf te ampersandın unutulması, yaratılmamış bi dosyayı açmaya çalışmak, yeterli belleğin olmaması. İnput istenilen formatta değil veya input data file belirtilen adreste bulunamadı.. Hardware hataları. Harwarenin belleğinin dolu olması.

i = 0, dizi[--i] = a;

Warnings, errors in c: <http://www.c4learn.com/c-programming/c-pragma-warn-macro-directive/>

* **Programming Style / Conventions**

-A C **identifier** is a name used to identify a variable, function, or any other user-defined item. C’de değişken ismi sayıyla başlayamaz(karakter yada alt çizgi ile başlayabilir), isimde tire(-), numara işareti(#) olamaz. C case sensitive’dir. Define isimlerinin tüm karakterleri büyük harftir. Nesneye yönelik struct gibi yapıların ilk harfi büyüktür. Birden fazla kelime varsa veri isminde ya büyük harfle(referred to as camel case) yada define identifier ise alt tire ile ayrılır.

-When declaring methods in C and C++(unlike Java and C#) if there are no parameters then write (void), dont leave it empty. Leaving it empty means unspecified number of parameters.

**-**Identifier names dont start with a number because compiler needs to know if this is a number or an identifier. It is easy to understand if first character is never a number.

-Put an end of line comment (//) to the end bracers of functions that say // end class Car, // end main.

-Put end of line comments before functions and even parts of functions etc. that explain things shortly and cleanly. Update comments as you update your program.

- You dont have to have a **main function** in java or c but then the program is not execuble. It is a resource file, a header file. There can be maximum of 1 main function per .c file.

The return value for main should indicate how the program exited. Normal exit is generally represented by a 0 return value from main. Abnormal termination is usually signalled by a non-zero return but there is no standard for how non-zero codes are interpreted. Also as noted by others, void main() is explicitly prohibited by the C++ standard and shouldn't be used. The valid C++ mainsignatures are:

int main()

and

int main(int argc, char\* argv[])

which is equivalent to

int main(int argc, char\*\* argv)

It's also worth noting that in C++, int main() can be left without a return value at which point it defaults to returning 0. This is also true with a C99 program. Whether return 0 should be omitted or not is open to debate. The range of valid C program main signatures is much greater.

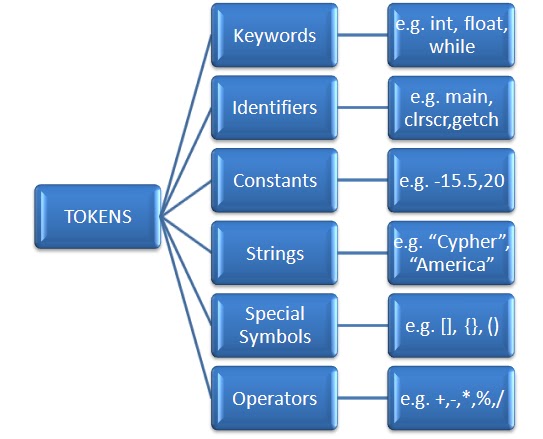
- Binary operatörlerin iki tarafına birer boşluk konulması okunurluğu arttırır.

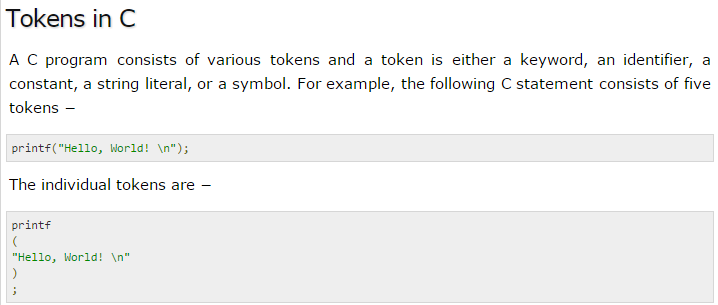
-For döngüsünde standart biçim kullan. i= 0; i > range.

-Use symbolic constants instead of magic numbers.

* **Terms in C**

-**Tokens**





**-Expression:** An expression is any legal combination of symbols that represents a value. Consists of operands and operators.

-**Semicolon** is a statement terminator. Null statement(empty statement) is a semicolon used in an empty body.

for ( i = 0; i < 10; line[i++] = 0 )

;

-**Whitespace:** A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments.

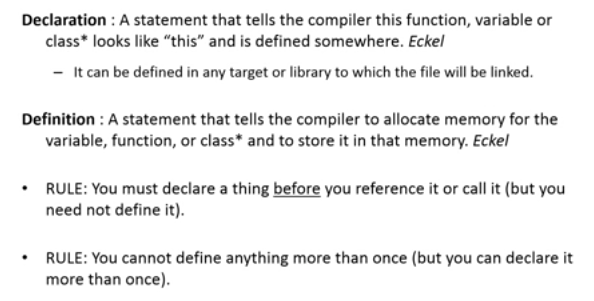
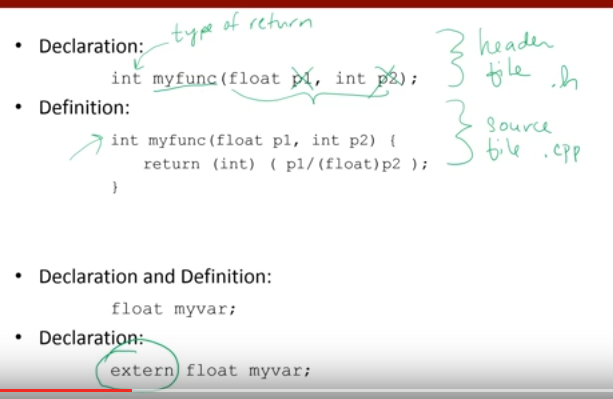
**-Declaration:** Reserve space for variable in memory.

int x;

**Initialization:** Sets value of variable. (tag: initizalize)

x = 10;

int x = 10;



- **Initial values:** external and static variables are initialised to zero if not explicitly done.

Initialization done once only. These variables cannot be initialized with a value that is not known until run-time(an unknown variable).

Pointers are initialized to NULL if not explicitly done. Initialization is done each time block is entered.

If you look at locals window in visual studio during debugging you will see variables are declared and initialized a random value when they get into scope. The line in which you declare it doesnt do anything and the line where you initizalied it gives its first non random value.

**-Boolean:** C de **'HAYIR', 0** değerindedir. **'EVET'** ise, sıfırdan değişik herhangi bir şeydir. Bir EVET/HAYIR testinin sonucu herhangi bir integer yada karakter değişkenine atanabilir. Genelde 1 atanır doğru cevabı için. İf(false) yada if(f) gibi bir ifade kullanılamaz ama if(‘f’), if(‘fa’), if(“fa”), if(“\n”) gibi ifadeler kullanılabilir.

**-Comments**

// end of line comment

/\* \*/ traditional comment

/\*\*\*\*\*\*\*\* boxed comments

\* \*

\*\*\*\*\*\*\*\*/

**Part 3**

* **Binary işlemler:**

**-Negatif bir sayının 2'lik tabandaki karşılığını bulmak** için, önce (*i*) sayıyı pozitif olarak ikilik tabanda yazarız. Daha sonra, (*ii*) ikilik tabanda yazılmış sayının 1 yazan rakamları 0, 0 yazan rakamları 1'e çevrilir. Son olarak (*iii*) çıkan sayıya, 1 eklenir. Bu size istediğiniz sayının ikilik tabanındaki eşini verecektir. Şimdi bir uygulama yapalım ve -7 sayını ikilik tabana çevirmeye çalışalım:

*i* ) -7 ==> ( 7 )10 = ( 0000 0111 )2

*ii* ) ( 0000 0111 ) ==> ( 1111 1000 )

*iii* ) ( 1111 1000 ) + 1 = ( 1111 1001 ) ==> ( -7 )10 = ( 1111 1001 )2

* **Input/Output**

**Printf** deki f formatted demektir. Kullanıcıya bir iş yaptıracak çıktılara **prompt** denir.

-print(“%c”, a); // a değişkeninde ne varsa onu yazdır. char a = ‘5’ ise 5 karakterini yazdır. char a = 5 ise 5 in ascii karşılığı neyse onu yazdır.

-print(“%c”, ‘a’); // a karakterini yazdır. burda %d kullanıp ascii değerini görebiliriz

-printf("Sonuc: %f\n", 2 / 4);

-printf("Benim adım "

"Çağatay Çebi.\n");

**-scanf**(%d, &x); //adres gösteren bir değişken kullanırsan &’ye gerek yok.

**-fprintf**(fp, "%s %s %s %d", "We", "are", "in", 2012);

- fputs("We are in 2012", fp);

rewind(fp);

**fscanf**(fp, "%s %s %s %d", str1, str2, str3, &year);

-scan de & kullanılıp printte kullanılmamasının sebebi printin sadece değere, scanin ise adrese ihtiyacı olmasıdır. Direk adres gösteren değişkenleri scanf’te kullanırken ampersand kullanmayız çünkü zaten bir değer değil adres tutarlar. Bunlara örnek diziler, structlar, pointerlar.

-while (scanf("%d", &num) == 1)

-You dont have to print a variable you can print an integer, floating point, string, char, etc. contstant.

**-Formatlı Gösterim:**

char cumle[20] = "Denemeler";

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

// 20 karakterlik alan ayirir ve 20 karakterlik alanın sağına dayalı şekilde yazar. Mesela 2 boşluğa sağa dayalı 1 yazdın 01 olur yani boşluk-1

printf( "%20s\n", cumle );

// 20 karakterlik alan ayirir ve en saga dayali sekilde, katarin ilk bes kelimesini yazar

printf( "%20.5s\n", cumle );

// 5 karakterlik alan ayirir ve en saga dayali sekilde yazar.Eger girilen kelime 5 karakterden buyukse, kelimenin hepsi yazilir.

printf( "%5s\n", cumle );

// 20 karakterlik alan ayirir ve sola dayali sekilde yazar.Sola dayali yazilmasi icin yuzde isaretinden sonra, (eksi) isareti konulur.

printf( "%-20s\n", cumle );

int **getc**(FILE \*stream) //File \*stream yerine dosya yada stdin out konulabilir.

int **putc**(int char, FILE \*stream)

getw

putw

int **getch** (void) // getchar’dan farkı girilen karakterin ekranda gösterilmemesi.

int **getchar**(void)

kr = getchar();

int **putchar**(int char)

gets c11 de silinmiş

char \***gets**(char \*str)

int **puts**(const char \*str)

[**http://stackoverflow.com/questions/1694036/why-is-the-gets-function-so-dangerous-that-it-should-not-be-used**](http://stackoverflow.com/questions/1694036/why-is-the-gets-function-so-dangerous-that-it-should-not-be-used)

**tüm fonksiyonlar için birer örnek koy**

**fputc**

**fgetc tek karakter alır**

int **fputs**(const char \*str, FILE \*stream)

char \***fgets**(char \*str, int n, FILE \*stream): tüm satırı(imleç başta değilse kalanını alır)

**-Random number generation**: rand() and srand() needs stdlib.h and time() needs time.h. Randomization is performed as a function that has a starting value, namely the seed. So, for the same seed, you will always get the same sequence of values.

Seed is usually taken from the current time, which are the seconds, as in time(NULL), so if you always set the seed before taking the random number, you will get the same number as long as you call the srand/rand combo multiple times **in the same second**.

To avoid this problem, srand is set only once per application, because it is doubtful that two of the application instances will be run in the same second, so each instance will then have a different sequence of random numbers.

- x = rand();

İstediğimiz şekilde bi sayı almak için rand() % 6 tarzında kullanılır veya rand() % 6 + 1 . rand() ı sabit bi sayı gibi kullanırız. Ama bu şekilde test amaçlı kullanılır sabit bi rastgele sayı gelir. Her defa farklı rastgele sayı istiyosa ya time yada besleme kullancaz

-Unsigned feed;

Scanf feed

Srand(feed);

x = rand();

-srand(time(NULL));

coinToss = rand() % 2;

* **if:** koşulu sağlayan bütün if işlemleri yapılır.

- if( !mark[j] && x )

- if( x && y )

Bir değişkene bir değer atanmışsa veya bir değer if in içindeyse bu değer yada değeri olan değişken 1 se doğru 0 sa yanlış bu ikisi dışında herhangi birşeyse doğru kabul edilir. Ama değeri olmayan bir değişkeni veya 0 olmayan bir değeri !x veya !253 şeklinde yazarsak yanlış kabul edilir. !0 ise bir kabul edilir.

* **if else/if,else,if/if, else if... :** koşulu sağlayan ilk if işlemi yapılır diğerleri yapılmaz

*if(koşul){*

*komut(lar)*

}

else {

*komut(lar)*

}

* **switch**( degisken ) {

case sabit1:

komut(lar)

[break]

case sabit2:

komut(lar)

[break]

default:

komut(lar);

}

-Eğer break kullanmazsan, alttaki case’lerin içindeki kodlarda çalışır. Dolayısı ile sadece bir case çalışsın istiyosan her case’e break koy.

-Case içinde sabit değer olmak zorunda. Dolayısı ile ya sayı yada ‘a’, “anıl“ gibi sabit değerler kullanabilirsin.

Not:switch - case, if - else if yapısına oldukça benzer bir ifadedir. Ancak aralarında iki fark vardır. Birincisi, switch - case yapısında, aralık değeri girmezsiniz. Direkt olarak ifadelerin bir şeylere eşit olup olmadığına bakarsınız. İkinci farksa, switch - case yapılarında, illa ki uygun koşulun sağlanmasıyla yapının kesilmek zorunda olmayışıdır. 'break' komutu kullanmadığınız takdirde, diğer şartların içindeki işlemleri de yapma imkanınız olabilir.

Örnek: printf("Lütfen notu giriniz> ");

scanf("%d",&not);

switch( not / 10 ) {

case 10:

case 9: printf("NOT: A\n"); break;

case 8: printf("NOT: B\n"); break;

case 7: printf("NOT: C\n"); break;

case 6: printf("NOT: D\n"); break;

case 5:

case 4:

case 3:

case 2:

case 1:

case 0: printf("NOT: F\n"); break;

default:

* **while** (i<5)

while( *koşul* ) {

*komut(lar)*

}

-while’ı for gibi kullanmak:

int i = 0;

while(i < 5)

i--;

-we can use multiple variables in for header. İnitialize them all, check them all, increment them all.

for(i = 0, j = 1; i < 100 && j <= 100; i+=2, y+=1)

* **do while(**while’in aynısı ama şart sağlanmasa bile bir kere çalışıyor)

do {

*komut(lar)*

} while( *koşul* );

Not: do while sadece başlanğıçta kesin bir kere çalışıp sonra while gibi davranmaz her adım kesin işlemi yapar sonra şarta bakar.

* **for:** Belli bi sayıda / belli bir yere kadar işlem yapmak için kullanırız.

-For içinde tanımlama yapılabilir.

-Ayrıca bir bir arttırıp azaltmak zorunda değiliz.

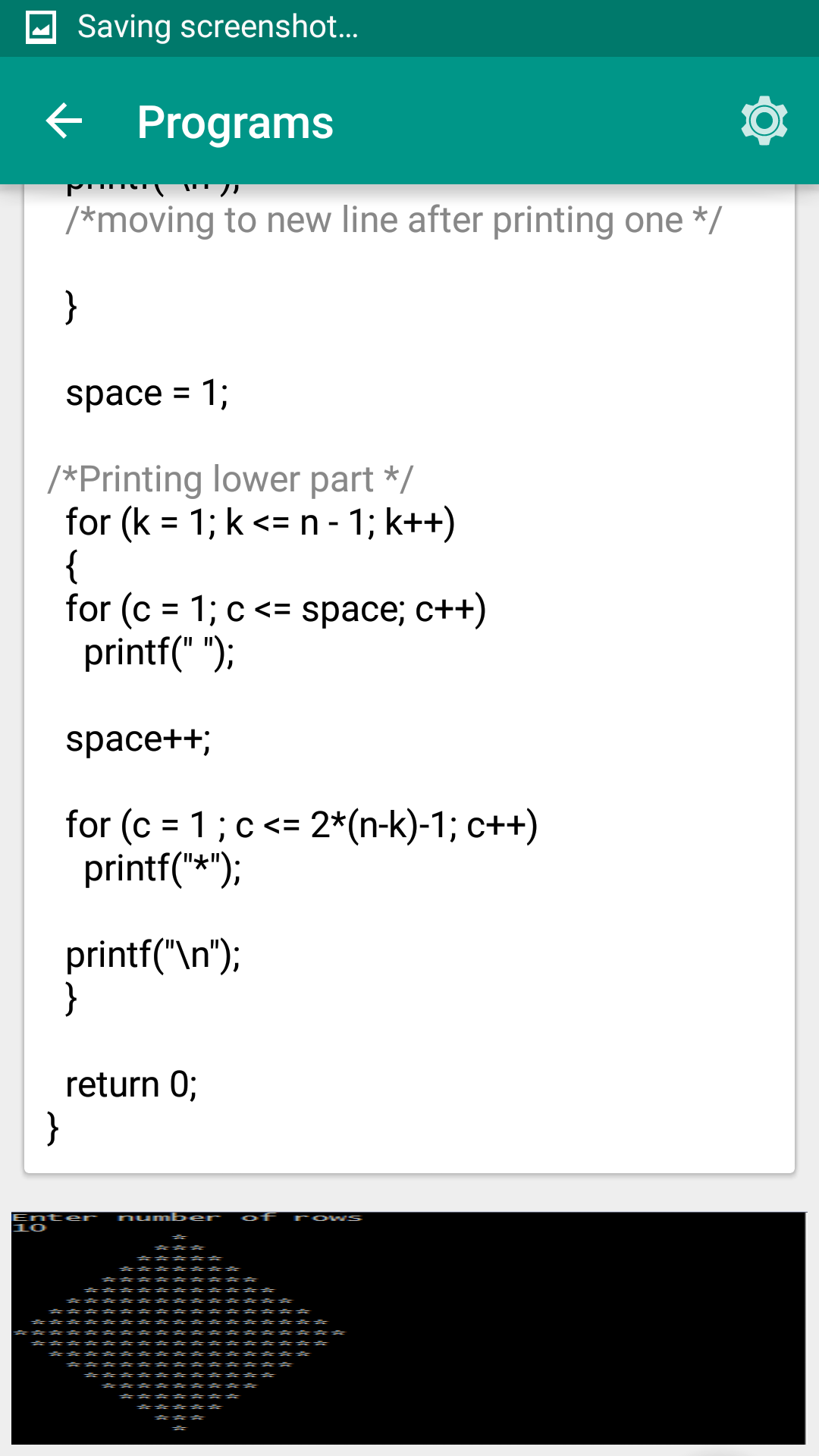
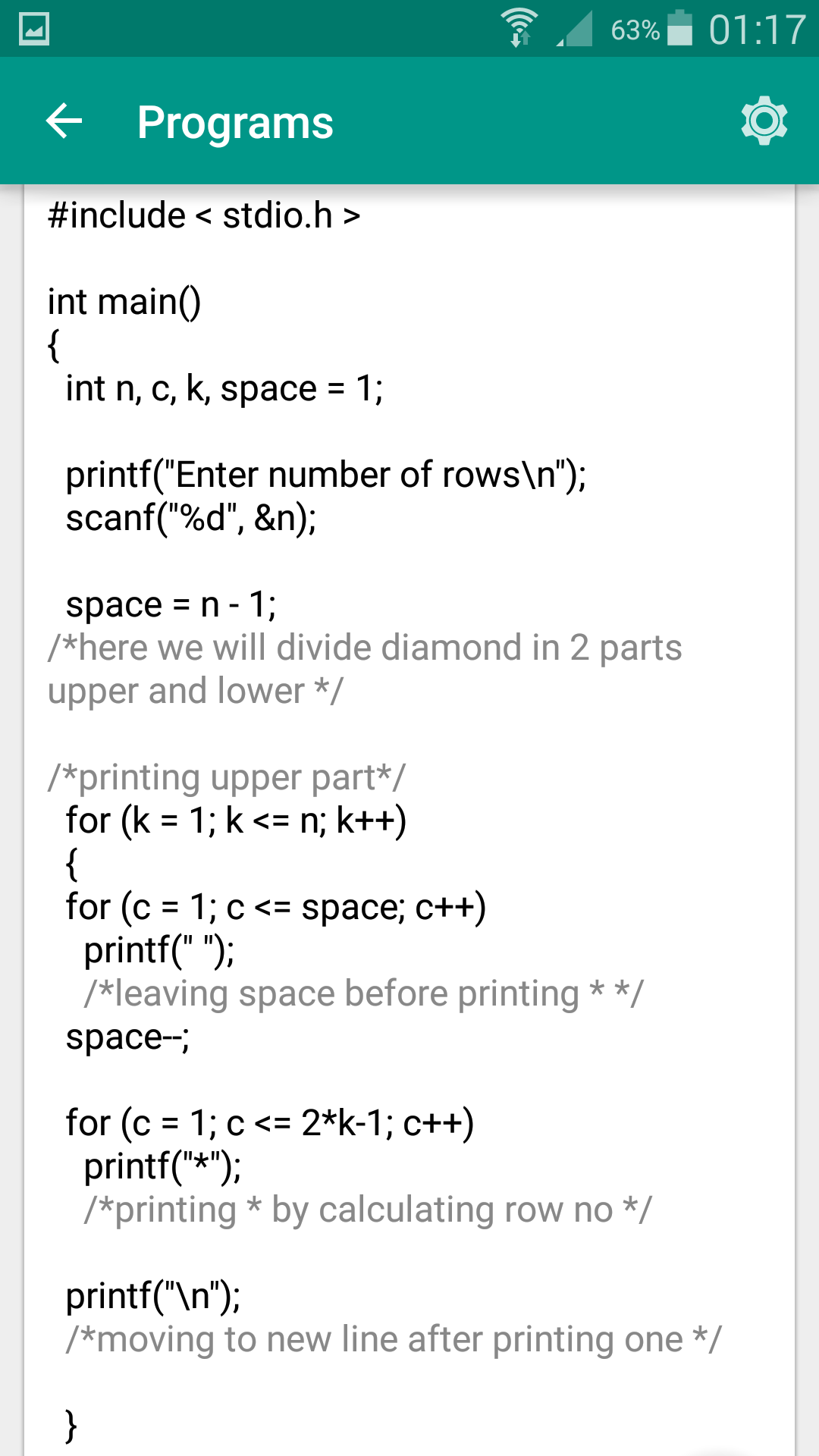
For( i=0; i<5; i++ )

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

printf("%2d: Merhaba Dünya\n",(i+1));

i = i + 1;

**-**Karmaşık senaryolarda for kullanma mantığı şudur: Her birden çok olacak işlem için for gerekir. Bu for’ların nasıl bir iç içe yapıya sahip olacağına yapıya göre karar vereceksin. Mesela aşağıdaki şekili yapmak için önce şekli ikiye bölmemiz lazım. Üst+orta ve alt. (veya üst ve orta+alt). Bundan sonra öncelikle bir satır döngümüz olmalı. Bu satırlarda ne yazdırılıyor? Boşluk ve yıldız. Üst kısımda boşluklar azalıyor, yıldızlar artıyor. Bu artma azalma kısmını ayarlarken ise önce satır sayacı, sütun sayacı ve bir sabit kullanıyorsun. İlk değerlerle istenecek sonucu verecek bir sabit verdikten sonra satır ve sütun değişkenlerini ve işaretlerinide ayarlıyorsun.



Note: When you have a calculation in a for header expression or in while header expression it is calculated every time unless the ide/compiler has a optimizer. So do your own calculation one time and use the caculated value.

* **break:**

for( ; ; ) {

printf("Lütfen bir sayı giriniz> ");

scanf("%d",&girilen\_sayi);

if( girilen\_sayi < 0 )

break;

sayi\_toplami += girilen\_sayi;

girilen\_sayi\_adedi++;

}

-Break içinde bulunduğu döngüyü bitirir (burada for)

-Break sadece loop ve switch içinde bulunabilir. Bunların dışında hiçbiryerde kullanılamaz. if(break == 1) break; şeklindede kullanılamaz.

* **continue:** döngünün başına döner(değişiklikten sonra),

Örnek:

//Sadece tek sayıları yazdıran birprogram

int i;

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

if( i%2 == 0 ) continue;

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

Örnek:

int i;

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

if( i%2 == 0 ) continue;

printf("%2d\n",i);

* **go to**

int i = 0;

baslangic\_noktasi:

printf( "Merhaba Dünya\n" );

i++;

if( i<10 ) goto baslangic\_noktasi;

### Conditional Operator

* Conditional Operator, if-else ile tamamen aynı yapıdadır. Hiçbir farkı yoktur. Tek farkı koda bakıldığında anlaşılmasının biraz daha zor oluşudur. Bir de if - else gibi yazıyla ifade edilmez. Onun yerine soru işareti (?) ve iki nokta üst üste ( : ) kullanarak yazarız. **İfade1 ? İfade2:İfade3; if(ifade1) ifade2 else ifade3 anlamına gelir.**
* **İf else’i tek satırda yazmaya yarar.**
* s = ( x < 0 ) ? -1 : x \* x
* sonuc = not >= 50 ? "Basarili" : "Basarisiz";
* printf( "%s\n", count % 2 ? "\*\*\*\*" : "++++++++" );
* printf( “%s\n ”, not>=60? “Geçtiniz” : Kaldınız” );

Eğer x, 0'dan küçük bir değerse, s değişkenine, -1 değeri atanır. Eğer x, 0'dan büyükse,

x'in karesi, s değişkenine atanır. (Gördüğünüz gibi ilk bakışta kodu yorumlamak biraz

sıkıcı olabiliyor. Sanırım yukarda ne demek istediğimi anlamışsınızdır.)

* **Pre Processors (Symbolic constants)**

<http://www.csharpnedir.com/articles/read/?id=1073>

Ön işlemcilerin normal işlemcilerden farkı compile’dan önce çalışmalarıdır. Ve koşulun yanlış olduğu kısmı etmezler(yorum satırı gibi görürler). Yani koşulun doğru olup olmadığıı test etmeye yararlar. Negatifi kontrol etmek için infdef vardır. Kullanımları çeşitli zorluklara yol açtığı için javada bulunmazlar. Yeni diillerde gittikçe kullanımları azalmıştır.

**-#include(header file):** Çalışma ortamımızla aynı klasörde olan bir başlık dosyasını(header file), programımıza eklemek için #include "file.h" şeklinde yazarız.

For #include "filename" the preprocessor searches in the same directory as the file containing the directive. This method is normally used to include programmer-defined header files.

For #include <filename> the preprocessor searches in an implementation dependent manner, normally in search directories pre-designated by the compiler/IDE. This method is normally used to include standard library header files.

**-#define**

**#define PI 3.14**

**#define ekrana\_yazdir printf**

-Kodda 3.14 yerine PI yazmana yarar ama bu kodun yaptigi iş PI’leri 3.14 ıle degistırmektır.

**-Using define for functions:**

#define merhaba\_yazdir( x ) int i; for ( i = 0; i < (x); i++ ) printf( "Merhaba\n" );

int main( void )

{

int yazdirma\_adedi;

printf( "Kaç defa yazdırılsın> " );

scanf( "%d", &yazdirma\_adedi );

merhaba\_yazdir( yazdirma\_adedi ); // I truncated rest of the program

-you can also use

#define merhaba\_yazdir( x ) \

int i; \

for ( i = 0; i < (x); i++ ) \

printf( "Merhaba\n" );

**-#undef**

Örneğin #undef PI yazdığınız da, o noktadan itibaren PI tanımsız olacaktır.

**-#ifdef - #endif**

PI değeri tanımlanıyor ve sonrasında tanım kaldırılıyor. Biz de sürprizlerle karşılaşmak istemediğimizden, PI değerinin tanım durumunu kontrol ediyoruz.

#ifdef PI

//PI tanımlıysa, daire alanı hesaplanıyor.

alan = PI \* yaricap \* yaricap;

printf( "Çember alanı: %.2f\n", alan );

#else

//PI değeri tanımsızsa, HATA mesajı veriliyor.

printf("HATA: Alan değeri tanımlı değildir.\n");

#endif

**-#ifndef - #endif**

**#ifndef PI**

#define PI 3.14

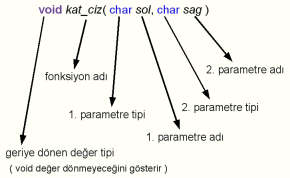
**#endif**

alan = PI \* yaricap \* yaricap;

printf( "Çember alanı: %.2f\n", alan );

-#if #else #endif #if #elif(else if) #endif

* **Fonksiyonlar**



**-** Fonksiyonu çağırırken **argüman** alırız. Fonksiyonda kullandığımız değişkene ise **parametre** deriz aldığımız değerlere. Fonksiyonda yeni bir değişken yaratılır ve argümanın değeri parametrenin değerine atanır.

-Tüm fonksiyonlar eşit seviyededir(main dahil).

-Fonksiyonlar aritmetik sonuç(int, float etc.), pointer, struct, union, void döndürebilir(return value) ve bu argümanları alabilir. Fonksiyon çağırıldığında verilen agrümanlar parametrelere atanır.

**-Yerel değişken:** Bir fonksiyon içersinde tanımladığınız değişkenler, sadece o fonksiyon içersinde tanımlıdır. **Global değişken** include’ların hemen altında(main dahil tüm fonksiyonlarınüstünde yazılmalıdırki undeclared variable hatası vermesin) yaratılan değişkendir. Globali her fonksiyonda direk kullanabiliriz. Yereli ise sadece tanımlandığı fonksiyonda direk kullanabiliriz başka fonksiyonlarda ya parametre olarak yada pointer yoluyla göndermeliyiz. Ayrıca global a ve yerel a değişkenleri bellekte farklı yerleri işaret eder.

**-Variable shadowing:** Eğer bir fonksiyonda aynı isme sahip yerel ve global değişkenler varsa yerel olan kullanılır.

**-Global,Yerel değişken farkı:** Aynı isme sahip birden çok değişken kullanılabilir. (Birisi mainde tanımlanır diğer mainin içindeki fonksiyonlardan birinde tanımlanır vs.). Ayrıca main içerisinde bir x ve global bir x varken ve func1 fonksiyonu bir parametre olarak x i alıyorsa, ilk tercihi yerel(main içindeki) x tir. Yoksa globali alır.

-Fonksiyon kullanma: Ya üstte prototip yapıp noktalı virgül koyup mainin altında tanımlayacaksın. Yada direk yukarda tanımlayacaksın. Normal tanımlıyorsan fonksiyonu parametre türlerinin yanına bide fonksiyonda kullanılması için isim koyman lazım. Prototipte buna gerek yok. Sadece tipleri göstermen yeterli. Fonksiyon içinden dışarı bir değer döndürmek için illa global değişken kullanmak zorunda değiliz . return de kullanabiliriz.

- **return-type** is the variable type that the function returns. This can not be an array type or a function type. **If not given, then int is assumed.**

**-Normal fonksiyon örneği**

void tek\_mi\_cift\_mi( int sayi ) // direk tanımlama tipi

{

if( sayi%2 == 0 )

printf( "%d, çift bir sayıdır.\n", sayi );

else

printf( "%d, tek bir sayıdır.\n", sayi );

}

int main( void )

{

int girilen\_sayi;

printf( "Lütfen bir sayı giriniz> " );

scanf( "%d",&girilen\_sayi );

tek\_mi\_cift\_mi( girilen\_sayi );

**-Localin locali**

if (j > 0){

int i=100; /\* 'i' sadece bu if blogunda gecerli olmak uzere tanimlaniyor. \*/

**-Prototip örneği:** Bu yöntem değişkenlerde önce tanımlama sonra değer vermeye benziyor.

#include<stdio.h>

int bolme\_islemi( int, int, int \* ); // prototip tipi

int main( void )

{

int bolunen, bolen;

int bolum, kalan;

bolunen = 13;

bolen = 4;

bolum = bolme\_islemi( bolunen, bolen, &kalan );

printf( "Bölüm: %d Kalan: %d\n", bolum, kalan );

return 0;

}

int bolme\_islemi( int bolunen, int bolen, int \*kalan )

{

\*kalan = bolunen % bolen;

return bolunen / bolen;

}

**-Recursion:** Bir sistemi çalıştırdığında aynı fonksiyonu belli bir şeyi elde edene kadar kendi kendine çalıştırıp duruyorsa orada recursion vardır.   
   
unsigned long factorial(unsigned long i)  
{  
 if (i == 0) return 1;

return (i \* factorial( i- 1));  
}

* **Pointer**

-Pointer adres tutmak zorunda değildir. Değerde tutabilir.

-Null pointer değeri 0 dır. Null pointer points to nothing.

-Two pointers can be sbuctracted to find the adress amount between two adresses. Addition, multiplication, division are not allowed. You can use “++”, “—“, “+”, “-“ with adresses.

-Pointer size on 16 bit is 16 bit(2 byte), on 32 bit its 32 bit(4byte), on 64 bit its 64 bit(8 byte).

-Pointerlar adres tutan değişkenlerdir. Her tip değişkenin kendi tip pointer ı vardır.

-Pointer tanımlamak ve atamak “int \*xp = &x;” şeklinde yapılır. Pointer bir değişkenin adresini tutar böylece.

-pointer üzerinde pointer+c pointer++ gibi işlemler yapabilirsin sonraki veya önceki adreslere ulaşmak için.

-Bir pointer ın tuttuğu adresteki değeri kullanmak için başına \* koyarız. k = \*p;

Bu ifade p’nin adresini tuttuğu değişkenin değerini k’ye ata anlamına gelir.

-Kısaca bir değişkenin(pointer dahil) adresini & ile alabiliyoruz ve bir pointer ın tuttuğu adresteki değişkenin değeri(tuttuğu değer int’se bir integer, pointer’sa bir adres, pointer pointer’ıysa bir pointer adresi) \* ile alabiliyoruz.

-Pointer adresi tutucak bi pointer tanımlarken \* yerine \*\* kullanılır and so on.

- Dizinin adresini pointer a atarken ampersanda gerek yoktur çünkü dizinin kendisi zaten komple dizinin adresidir (baştan başlayıp). Ptr[i] dizi elemanlarının adresleri için kullanılır. Dizilerde \*(ptr+i) ile Dizi[i] elemanının değerine ulaşırız.

-Ptr = month+3 dizinin 3. Elemandan itibaren sona kadar olan dizinin adresini atar. Ptr = &month[i] dersek değişkenlerde olduğu gibi sadece month[i] nin adresini alır . x=(ptr+2)[2] ise month 4 ün adresini atamak için kullanılır. x=\*(month+3) ise month[3] ün değerini x e atamak için kullanılır. değişkenlerle yaptığımız şeyleri dizilerle yapıyoruz yani. X=\*(ptr + 2) ise sadece month 2 nin değerini atar. \*(ptr + 2) ise month 2 nin değerini gösterir.

-Month[5], month[0], month dizinin komple adresini verir. Üçüde pointerdır. Dolayısıyla pointer olarak kullanmak lazım. Array[n] gibi gösterimler pointerdır. Ve ordan başlayarak geri kalan diziyi ifade ederler. Tek değeri almak istiyorsak başına & koyarız &array[0].

**-Pointers to void**: A pointer of type void \* represents the address of an object, but not its type. For example, a memory allocation function **void \*malloc( size\_t size );**returns a pointer to void which can be casted to any data type.

**-Call by reference(Referansla argüman aktarımı):** Hem gönderdiğimiz değişkenin etkilenmesini istiyor hemde-yada birden çok değer döndürmek istiyorsak bunu kullanıyoruz. Değişkenleri argüman olarak kullanmak yerine fonksiyona değişkenlerin adreslerini veririz &a &b şeklinde. Fonksiyonda alırkende int \*x int \*y şeklinde alırız. Çünkü verilen şey adres, alınan şey pointer. Dolayısıyla a ve b yi değil a nın adresinin değeri ve b nin adresinin değeriyle oynarız. Fonksiyonda \*x, \*y şeklinde kullanırız. Bu bize birden fazla(return sadece bir tane değer döndürür) değer döndürme şansı veriyor. Hem return ile hemde return ile dönmeyenleri fonksiyon içerisinde değiştirerek birden fazla değişkenle oynayıp geri alabiliyoruz.

**-Call by value:** Fonksiyonu çağırırken parametre girerek bir fonksiyonda kullanmak üzere bir değer alma.

Dizilerde pointer kullanımı

int i;

// dizi'yi tanimliyoruz.

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

// ptr adinda bir pointer tanimliyoruz.

int \*ptr;

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

ptr = dizi;

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

// elemanlarini yazdiriyoruz.

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

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

* int r = 50;

int \*p;

int \*\*k;

int \*\*\*m;

printf( "r: %d\n", r );

p = &r;

k = &p;

m = &k;

\*\*\*m = 100;

printf( "r: %d\n", r );

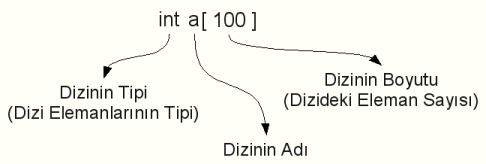
Burda m nin değerini 100 yap diyoruz. M nin değeri ise k nin adresi. Yani k nin adresindeki değişkene 100 atıyoruz. K nin adresinde ise p nin adresi var. P nin adreside ise r nin adresi var. Dolayısı ile kısaca \*\*\*m = 100 demek r = 100 demek.

pointer ın pointer ı için kullanılan \* sayısı artıyo m in tuttuğu adresteki değişkenin değeri 100 oluyo k nın tuttuğu adresteki değişkenin değeri 100 oluyo p nin tuttuğu adresteki değişkenin değeri 100 oluyo yani r nin adresinin değeri 100 oluyo.

-New information about call by reference and call by value. Unlike Java, in C the call by value does what Java does. Passes values of primitive or reference variables hold. But in C, call by reference actually passes the pointer itself. Not pointer’s value(adress).

* **Array (Diziler)**

int a[100];



**Memset:** burda memset dışında gösterilen atama sadece ilk atamada kullanılır. ilk atama dışında tüm diziye bir değer atamak için memset kullanılır.

memset(mark, 0, 4\*sizeof(int));

-Örneğin a dizisinin, 25.elemanı gerekiyorsa, a[24] ile çağrılır. C programlama dilinde, dizi elemanları 0. pozisyondan başlar dan başlar.

-Eğer diziyi tanımlarken ilk değer atamasıda yapacaksak dizi boyutu belirtmemiz gerekmez.(Çok boyutlu dizilerde en sağdaki boyut)

-String’lerdeki gibi fonksiyonlarda argüman olarak alınınca &’a gerek yok çünkü array ve string’lerin ismi değer değil adres verir.

int dizi1[ ] = { 4, 8, 15, 16, 23, 42 };

float dizi2[ ] = { 11.5, -1.6, 46.3, 5, 21.56 };

-Dizi tanımlarken array[b] derken b bir int falan olamaz ama const olabilir. Çünkü boyutun ya sabit olması lazım yada hiç boyut tanımlanmaması lazım.

ptr = month; // month[0] adresini atar

elm = ptr[ 3 ]; // sadece month[ 3 ] adresini atar

-Aşağıda iki farklı ilk değer atama yöntemi bulunuyor. Yazım farkı olmasına rağmen, ikisi de aynı işi yapar.

int dizi[ 7 ] = { 0, 0, 0, 0, 0, 0, 0 };

int dizi[ 7 ] = { 0 };

-Fonksiyonda dizi örneği

void elemanlari\_goster( int gosterilecek\_dizi[ 5 ] )

{

int i;

for( i = 0; i < 5; i++)

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

}

int main( void )

{

int dizi[ 5 ] = { 55, 414, 7, 210, 15 };

elemanlari\_goster( dizi );

return 0;

}

-Çok boyutlu dizi

int tablo[3][4] = { 8, 16, 9, 52, 3, 15, 27, 6, 14, 25, 2, 10 }; int tablo[3][4] = { {8, 16, 9, 52}, {3, 15, 27, 6}, {14, 25, 2, 10} };

Diziyi tanımlarken, yukardaki gibi bir ilk değer atama yaparsanız, elemanların değeri aşağıdaki gibi olur:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Satır 0 | : | 8 | 16 | 9 | 52 |
| Satır 1 | : | 3 | 15 | 27 | 6 |
| Satır 2 | : | 14 | 25 | 2 | 10 |
|  |  |  |  |  |  |

-buyuk[buyuk[2][2]][buyuk[2][2]] = 177; // bu, buyuk[5][5] = 177; demek

-Tek boyutlu dizilerde ilk değer ataması yaparken, eleman sayısından az değer girerseniz, kalan değerler 0 olarak kabul edilir. Aynı şey çok boyutlu diziler için de geçerlidir; olması gerektiği sayıda eleman ya da grup girilmezse, bu değerlerin hepsi 0 olarak kabul edilir. Yani üstte yazdığımız kodun yaratacağı sonuç, şöyle olacaktır:

int tablo[3][4] = { {8, 16}, {3, 15, 27} };

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Satır 0 | : | 8 | 16 | 0 | 0 |
| Satır 1 | : | 3 | 15 | 27 | 0 |
| Satır 2 | : | 0 | 0 | 0 | 0 |

* **String**(get put lar hariç string.h gerekiyor)

-30 karakterlik bi stringin en fazla 29 karakteri olabilir. 30. karakter null(\0) olur.

-C de karakterler tek tırnak içerisinde, stringler ise çift tırnak içerisinde gösterilir. Tırnak yoksa değişkenin değeri yada karakterin ascii değeri alınır.

-Bir katar tanımı yaptığınız anda, katarın bütün elemanları otomatik olarak '\0' ile doldurulur.

-String tanımı yaparken array’ler deki gibi tanım yaparken atama yapıyorsak boyut belirtmek zorunda değiliz.

-Dizilerdeki gibi fonksiyonlarda da argüman olarak alınınca &’a gerek yok çünkü array ve string’lerin ismi değer değil adres verir.

-Karakterler, kelimeler cümleler direk atanabilir. **Char key=’a’; ( c = getchar(), putchar(c) )** ve **char key[5]=”anıl”; ( gets(s) , puts(s) )**şeklinde. Aynı şekilde karakterler get s put s ilede alınıp verilebilir.

-string++; string+c; gibi işlemler ile başka pointerlara geçebiliriz. Array, string, pointer az çok aynı şeyler.

-Atama için scanf te kullanabiliriz.

char isim[30];

printf( "İsim giriniz> ");

scanf( "%s", isim );

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

- Boşluk içeren cümleler için puts(  ) ve gets(  ) fonksiyonları kullanılmaktadır. scan yerine. Puts in printften farkı sona /n koyması.

-Karakter karakter atamada yapılabilir.

char isim[] = { 'C', 'A', 'G', 'A',

'T', 'A', 'Y', '\0' };

char soyad[5] = { 'C', 'E', 'B', 'I', '\0' };

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

-**sprintf:** Bir string’e formatted data girmeye yarar

char str[80];

sprintf(str, "Value of Pi = %f", 3.14); //output: Value of Pi = 3.14

sprintf(hashString, "%d.txt", hashResult); // dosyayı txt dosyası yapar

**-strlen:** Öncelikle bütün string fonksiyonları için #include<string.h> kullanmamız gerekiyor. Strlen string uzunluğu ölçer.

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

**-strcpy – strncpy:** Kopyalama. Strncpy n kadar karakteri kopyalar.

char kaynak[40]="Merhaba Dünya";

char kopya[30] = "";

strcpy(kaynak, “.txt” );

strcpy(kaynak , kopya);

strncpy( str3, str2, 5); // ilk beş karakteri kopyalar.

-**strcmp -** **strncmp:** Karşılaştırma:

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

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

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

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

sonuc = strncmp( ilk\_katar, ikinci\_katar, 3 ); //3 harf karşılaştırma

|  |  |
| --- | --- |
| **Dönen Değer** | **Açıklama** |
| < 0 | Katar1, Katar2'den küçüktür. |
| 0 | Katar1 ve Katar2 birbirine eşittir. |
| > 0 | Katar1, Katar2'den büyüktür. |

*-***strcat -****strncat***:* Strcat bir stringi başka bir stringe ekler. Strncat ise belli bir sayıda karakteri ekler.

char ad[30], soyad[20];

char isim\_soyad[51];

printf( "Ad ve soyadınızı giriniz> " );

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

strcat( isim\_soyad, ad );

strcat( isim\_soyad, " " );

strncat( isim\_soyad, soyad, 10 );

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

-**strstr**: Katarda kelime arama

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

char \*ptr;

//Fonksiyon buyuk-kucuk harf

// duyarlidir. Eger birden fazla eslesme varsa,

// ilk adres degeri doner. Hic eslesme olmazsa,

// pointer degeri NULL olur.

ptr = strstr( adres, "koy" );

if( ptr != NULL )

printf( "Başlangıç notkası: %d\n", ptr - adres );

else

printf( "Eşleşme bulunamadı.\n" );

#### - strchr – strrchr: Tek bir karakter arama. Strchr ilk konumu verir strrchr son konumu

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

char \*ilk\_nokta, \*son\_nokta;

ilk\_nokta = strchr( adres, 'e' );

son\_nokta = strrchr( adres, 'e' );

if( ilk\_nokta != NULL ) {

printf( "Ilk gorundugu konum: %d\n", ilk\_nokta - adres );

printf( "Son gorundugu konum: %d\n", son\_nokta - adres );

}

else

printf( "Eşleşme bulunamadı.\n" );

-**atoi – atof**: Elimizdeki bir string’i sayıya çevirip işlem yapmak istiyorsak kullanırız. Atoi tam sayılar için atof ise kayan noktalı sayaılar içindir. Rakam harici bir karakter görene kadar çalışır bu fonksiyonlar. Bu fonksiyonları kullanarak rakam yazmak yerine string ismini yazarız.

char kok\_iki[] = "1.414213";

char pi[] = "3.14";

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

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

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

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

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

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

* **Dinamik bellek yönetimi(**stdlib.h kullanır)

<https://stackoverflow.com/questions/10200628/heap-memory-in-c-programming>

<https://www2.hawaii.edu/~walbritt/ics212/examples/HeapStack.htm>

-run time sırasında atama yaparlar.

Free(x) void free(void \*ptr)

*calloc(  )* fonksiyonu aşağıdaki gibi kullanılır:

isaretci\_adi = calloc( eleman\_sayisi, her\_elemanin\_boyutu );

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

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

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

-Malloc calloc’la pointerları aynı anda kullanırsak cast operator kullanmamız gerekebilir.

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

// Eger hafiza dolmussa dizi pointer'i NULL'a

// esit olacak ve asagidaki hata mesaji cikacaktir.

if( dizi == NULL )

printf( "Yetersiz bellek!\n" );

* **Enum**
* Nesne programlamaya giriştir. Yeni bir değişken tanımlarız ve onun alabileceği değerleri gireriz. Ama değişken tek sefada bu seçeneklerden sadece birisini kullanabilir. Kullandığı değişkenin datası yer kaplar sadece.
* Enum’da datalar aynı adreste saklanır.
* Bilgisayar otomatik olarak en üsttekine 0 bi alttakine 1 şeklinde sayılar atıyor. Kendi değerlerinide girebilirsin.
* Array ve stringlerin aksine struct ve enumda fonksiyonlara argüman olar verirken & kullanmak gerekir. Çünkü bu ikisi direk adres göstermez.

Yöntem 1

// Enum kullanarak boolean isminde yeni bir değişken tipi tanımlıyoruz.

enum boolean {

false = 0,

true = 1

};

// boolean tipinde doru\_mu isminde bir değişken tanımlıyoruz.

enum boolean dogru\_mu;

// Tanimladigimiz 'dogru\_mu' degiskenine deger atayip, bir alt satirda da

// kontrol yapiyoruz.

dogru\_mu = true;

if( dogru\_mu == true )

printf( "Doğru\n" );

-Farklı 2 enum değişkeninin alabileceği değere aynı sayıyı atamak bi hata değil. Enumda benzer seçeneklere aynı sayı verilebilir.

Yöntem 2

// Yeni veri tipini olusturuyoruz ayrica yeni veri tipinden bir degisken

//tanimliyoruz.

enum boolean {

false = 0,

true = 1

} dogru\_mu;

dogru\_mu = true;

if( dogru\_mu == true )

printf( "Doğru\n" );

-2 adet örnek yazılımlarda. Typedef’de gösteriliyor.

* **Struct**

-structs can be made local or global.

-push(&stack, value); //mainde push fonksiyonu çağırılıyor

void push (struct stack \*stack1, value1); //adres gösteren argümanı alırken ‘\*’ kullanmamızın sebebi bir değer değil, bir adres göstericisi(pointer) olması argümanın

-typdef struct node \*NODEPTR; derken \* kullanmamızın sebebi değer değil adres tutmasını istememiz NODEPTR’ın.

-Enum içindeki değşikenlerden bir defada sadece birini kullanabiliyorduk ama struct’ta hepsi kullanıma açıktır. Nesneye yönelik programlamaya doğru ikinci adımdır.

-Struct mainin üstünde tanımlanmalıdır.

-Array ve stringlerin aksine struct ve enumda fonksiyonlara argüman olarak verirken & kullanmak gerekir. Çünkü bu ikisi direk adres göstermez.

-Sizin bilgilerinizi, erkek kardeşinize kopyalamak için tek yapmanız gereken, "*erkek\_kardes = siz*" yazmaktır.

### Struct Etiketleri (isimleri)

struct sahis\_bilgileri {

char isim[40];

int boy;

};

struct sahis\_bilgileri kisi\_1;

strcpy( kisi\_1.isim, "AHMET" );

kisi\_1.boy = 170;

-Etiket koyduğunuz zamansa, programın herhangi bir yerinde istediğiniz kadar yapı değişkeni tanımlayabilirsiniz.

### -Structlarda İlk Değer Atama

struct {

char isim[40];

int boy;

} kisi = { "Ali", 167 };

* İsim , boy ve doğum tarihi alan bi programda { "Mehmet", 160, 23, 3, 1980 } yerine { "Mehmet", 160, {23, 3, 1980} } yazmakta mümkündür. Ama ikincisi daha okunurdur.

### -Struct Dizileri

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

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

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

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

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

printf( "Ad: %s\n", kisi[i].isim );

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

printf( "Doğum Tarihi: %d/%d/%d\n\n", kisi[i].tarih.gun,

kisi[i].tarih.ay,

kisi[i].tarih.yil );

}

### -Struct Dizilerine Pointer ile Erişim

### Struct bilgilerini pointerdan almakta mümkün. Yapı dizisinin adresini pointer a aşağıdaki şekilde atadık.

struct sahis\_bilgileri \*ptr;

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

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

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

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

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

ptr->tarih.ay,

ptr->tarih.yil );

}

### -Structlar ve Fonksiyonlar

### Bu örnekte mainde ve fonksiyonda iki farklı struct kullanıp fonksiyondakini retunle maine getirip maindekine atamışız. Ama genelde biz fonksiyona adres verip direk maindeki struct’ı değiştiricez.

void struct sahis\_bilgileri bilgileri\_al( void )

{

struct sahis\_bilgileri sahis;

printf( "İsim> " );

gets( sahis.isim );

printf( "Boy> " );

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

return sahis;

}

void bilgileri\_goster( struct sahis\_bilgileri sahis )

{

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

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

}

İnt main()

{

kisi = bilgileri\_al( );

bilgileri\_goster( kisi );

return 0;

}

### -Dinamik Structlar

struct sahis\_bilgileri \*ptr;

ptr = calloc( 1, sizeof( struct sahis\_bilgileri ) );

free( ptr );

### -Structlarda typedef Kullanımı

### Porgram çalışmadan önce çalışır preprocessorler. Define preprocessor’ı sağdakini gördüğü yere soldakini yazar. Biz uzun, karmaşık şeyleri yazmaktan kurtulmuş oluruz bunu bizim yerimize proprocessor yapar.

typedef struct *sahis\_bilgileri* *kisi\_bilgileri*;

### -Typedef programda sağdaki cümleyi gördüğünde soldaki ile değiştirir. Bu şekilde biz soldaki ni yazmak zorunda olmayız, sağdakini yazarız ama program bizim için sağdakini soldaki ile değiştirir program çalışmadan önce. Çünkü preprocessor’dır.

### -Başka bir örnek:

typedef int bool;

#define TRUE 1

#define FALSE 0

bool f = FALSE;

if (f) { ... }

### Yada,

typedef enum { FALSE, TRUE } boolean;

boolean b = FALSE;

if (b) { ... }

### Main function argument

### -int main(int argc, char \*argv[]); // ilk önce eleman sayısını sonra elemanları giriyosun.

### -You pass all the command line arguments separated by a space, but if argument itself has a space then you can pass such arguments by putting them inside double quotes "" or single quotes ''.

### - These values are called command line arguments and many times they are important for your program specially when you want to control your program from outside instead of hard coding those values inside the code.

### -argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program.

### -It should be noted that argv[0] holds the name of the program(c file) itself and argv[1] is a pointer to the first command line argument supplied, and \*argv[n] is the last argument. If no arguments are supplied, argc will be one, otherwise and if you pass one argument then argc is set at 2.

### Dosya, File: A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices.

### - You can use the fopen( ) function to create a new file or to open an existing file. This call will initialize an object of the type FILE, which contains all the information necessary to control the stream.

FILE \*word;

word = fopen(“10harf.txt”, “w”);

fclose(word);

Açma modları:

**r:** Var olan dosyayı okuma amaçlı açar.

**w:** Dosyayı yoksa yaratır varsa sıfırlar ve baştan yazmaya başlar. Yazma amaçlı.

**a:** Dosyayı yoksa yaratır varsa sonran başlayıp ekleme yapar. Yazma amaçlı.

**r+:** Var olan dosyayı okuma yazma amaçlı açar.

**w+:** Dosyayı yoksa yaratır varsa sıfırlar ve baştan yazmaya başlar. Yazma okuma amaçlı.

**a+:** Dosyayı yoksa yaratır varsa sonran başlayıp ekleme yapar. Yazma okuma amaçlı.

-If you are going to handle binary files, then you will use following access modes instead of the above mentioned ones

"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

### -The fclose(word) function returns zero on success, or EOF if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file stdio.h. EOF is equal to CTRL+D in UNIX/Linux/MacOS and CTRL+Z in windows.

-fputc( c, fp);

fputs( s, fp);

fputs ile fputs(string1... yada fputs(“string”... yapabiliriz

fprintf( fp, “x = %d”, x);

fscanf

fscanf bir dosyadan bir string veya başka bir değişken okuduğunda, imleç o değişkenin sap tarafına gider. Fscanf boşluk görene kadar okur. İmlecin yerini rewind yada başka yöntemlerle değiştirebilirsin.

### feof() fonksiyonu: argüman olarak dosya alır. Eğer imleç dosya sonuna gelmişse 1 gelmemişse 0 döndürür.

### rewind(fp);

-if(!fp)

Print dosya açılmadı

dosya açılırsa fp 1 döndürür

-if(!feof(stdin)) komutu ile end of file sinyali vermediğimiz sürece data girişi yapabiliriz

- if(!feof(fp)) komutu ile eof gelmediği sürece data okuyabiliriz

-line atlamak için line’ı fgets okuyup ignore etmeyi kullanabilirsin.

fopen

fclose

fprintf

fscanf: imleçten başlayıp ne okumasını söylediysen onu alır

fwrite

fread

### Veri yapıları, Data structures

### (veri yapıları notlarından veri yapıları, arama algoritmaları(hepsi değil) , ve gerekli gördüğün diğer şeyleri ekle. Hem buraya hem c yazılımlara veri yapıları diye bi dosyaya.

### File organization

* Binary işlemler , yukardada var biraz binary. Onlara ekleme yap. Derste gördüğün binary şeyler. 2-8-16 çevirme falan. X den 10 tabanına 10 dan x tabanına falan.