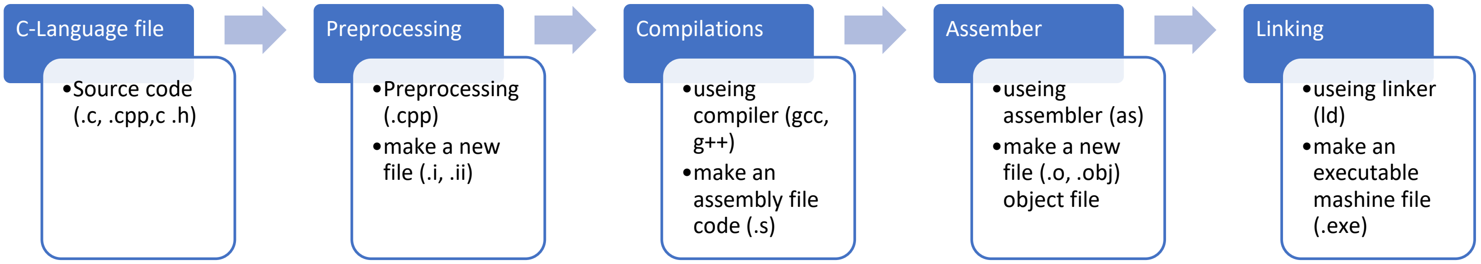
C Basics

- When we make a C file, we should make (main.C) file to be the main file the all-main functions it will putted there.
- But we should know how compilation will process?

Compilation process



C-Language file
- Source code (.c, .cpp,c .h)

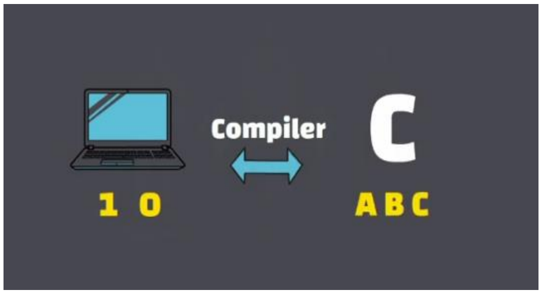
Preprocessing
- Preprocessing (.cpp)
- make a new file (.i, .ii)

Compilations
- useing compiler (gcc, g++)
- make an assembly file code (.s)

Assember
- useing assembler (as)
- make a new file (.o, .obj) object file

Linking
- useing linker (ld)
- make an executable mashine file (.exe)

- اللغات البرمجية تعتمد على text language
- دائما هناك متجرم يقوم بتحول اللغة البرمجية الى لغة يفهمها الحاسب الالي compiler

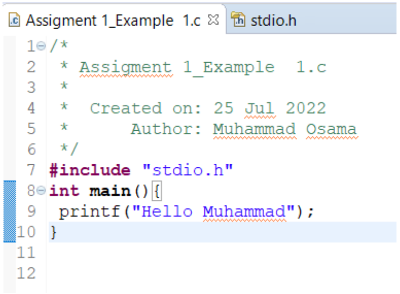


- ال compiler المستخدم في لغة Cهو clang
- هي عمليات تحويل من لغة high language to low languageيفهمها الحاسب الالي
- high languageمثل (C – python – C++ - etc.)
- low languageهي لغة الحاسوب 101010
- تم تلك العملية على 4 خطوات

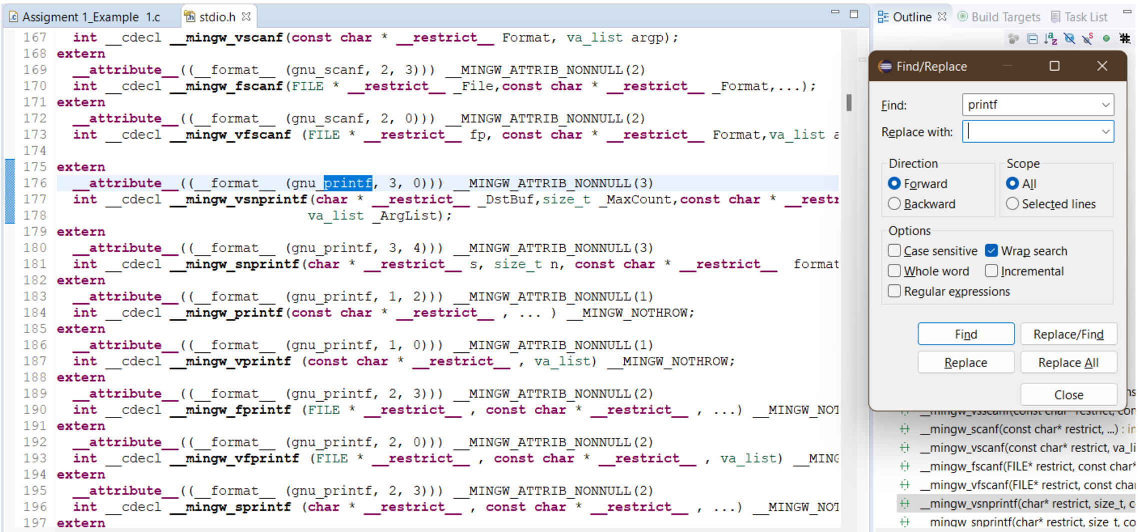
Preprocessing	- تقوم باستعمال او باستخراج الاوامر المستخدمة داخل الملف من المكتبات المعرفة داخله مثال - استعمال او استخراج printf من مكتبة stdio.h التي تم تعريفها داخل الملف #include <stdio.h> - أي أنه كل ال # library يزيله من الكود ويضع المستخدم منها فقط داخل الكود - يخرج على هيئة file .i
Compiling	- هو تحويل اللغة المكتوبة داخل ال file الى لغة assembly language وهى اقرب لغة الى لغة الالة وهى اسرع لغات - ينتج file .s
Assembling	- هى تحويل لغة assembly الى لغة 0101010 - يخرج على هيئة file .o
Linking	- يقوم بربط المكتبات المستخدمة مع ال files المحول وتجميعهم بلغة 101001 - File .o هو file ينادي فقط لذلك يجب جمعه مع file اخر ليعمل - ال file binary الناتج هو يقوم بعمل address لكل ما بالداخل ليتم حرقة بعد ذلك على ال flash memory - أي انه يخرج على هيئة file .exe

For Example

- Make a c file
- Then open **stdio.h** library, (CRTL+F) then search about **printf** function



```
/*
 * Assigment 1_Example  1.c
 *
 *  Created on: 25 Jul 2022
 *      Author: Muhammad Osama
 */
#include "stdio.h"
int main(){
 printf("Hello Muhammad");
}
```



```
  int __cdecl __mingw_vscanf(const char * __restrict__ Format, va_list argp);
extern
  __attribute__((__format__ (gnu_scanf, 2, 3))) __MINGW_ATTRIB_NONNULL(2)
  int __cdecl __mingw_fscanf(FILE * __restrict__ _File,const char * __restrict__ _Format,...);
extern
  __attribute__((__format__ (gnu_scanf, 2, 0))) __MINGW_ATTRIB_NONNULL(2)
  int __cdecl __mingw_vfscanf (FILE * __restrict__ fp, const char * __restrict__ Format,va_list a

extern
  __attribute__((__format__ (gnu_printf, 3, 0))) __MINGW_ATTRIB_NONNULL(3)
  int __cdecl __mingw_vsnprintf(char * __restrict__ _DstBuf,size_t _MaxCount,const char * __restr
                              va_list _ArgList);
extern
  __attribute__((__format__ (gnu_printf, 3, 4))) __MINGW_ATTRIB_NONNULL(3)
  int __cdecl __mingw_snprintf(char * __restrict__ s, size_t n, const char * __restrict__  format
extern
  __attribute__((__format__ (gnu_printf, 1, 2))) __MINGW_ATTRIB_NONNULL(1)
  int __cdecl __mingw_printf(const char * __restrict__ , ... ) __MINGW_NOTHROW;
extern
  __attribute__((__format__ (gnu_printf, 1, 0))) __MINGW_ATTRIB_NONNULL(1)
  int __cdecl __mingw_vprintf (const char * __restrict__ , va_list) __MINGW_NOTHROW;
extern
  __attribute__((__format__ (gnu_printf, 2, 3))) __MINGW_ATTRIB_NONNULL(2)
  int __cdecl __mingw_fprintf (FILE * __restrict__ , const char * __restrict__ , ...) __MINGW_NOT
extern
  __attribute__((__format__ (gnu_printf, 2, 0))) __MINGW_ATTRIB_NONNULL(2)
  int __cdecl __mingw_vfprintf (FILE * __restrict__ , const char * __restrict__ , va_list) __MING
extern
  __attribute__((__format__ (gnu_printf, 2, 3))) __MINGW_ATTRIB_NONNULL(2)
  int __cdecl __mingw_sprintf (char * __restrict__ , const char * __restrict__ , ...) __MINGW_NOT
extern
```

Find: printf
Replace with:

Direction: ● Forward ○ Backward
Scope: ● All ○ Selected lines
Options: □ Case sensitive ☑ Wrap search □ Whole word □ Incremental □ Regular expressions

```
_mingw_scanf(const char* restrict, ...) : ir
_mingw_vscanf(const char* restrict, va_li
_mingw_fscanf(FILE* restrict, const char*
_mingw_vfscanf(FILE* restrict, const char
_mingw_vsnprintf(char* restrict, size_t, c
mingw_snprintf(char* restrict, size_t, co
```

- All of this functions in library its call prototype (name of functions and definitions), so that's mean when we type any function in our .c file the compiler will see if this functions included in libraries or not, the compiler just need the name of functions to be there, not its definition to make a (instructions) branch for it (branch means: tell the processor to go to another address in memory (ex. 0x100)) (for this function), then make a .s file including all this instructions.
- The assembly take the .s file and convert it to binary file (010101) (convert all lines in file to binary code) and make an .o file (object file).
- The linker take the .o file and see the branches that made it (by the compiler) and look at the library then take binary code of all functions (branches) from included libraries and takes:

1- The address for all branches (ex. 0x100) from assembly file.
2- The assembly file (that branch the functions) and put it on the address for every branch (ex. 0x100)
3- Then now we have assembly file (include address ex.0x100) and binary library file (include address ex.0x100), now the linker links the same addresses together in an .exe file can be read it.

- Then make an .exe file
- So, when we use the .exe file the CPU open the .exe file and will find branches (functions) (jump) so, it will go to this address (branches) to do the instructions.

Toolchain

- In software, a toolchain is a set of programming tools that is used to perform a complex software development task or to create a software product, which is typically another computer program or a set of related programs. In general, the tools forming a toolchain are executed consecutively so the output or resulting environment state of each tool becomes the input or starting environment for the next one, but the term is also used when referring to a set of related tools that are not necessarily executed consecutively.

- A simple software development toolchain may consist of a

1- compiler and linker (which transform the source code into an executable program)

2- libraries (which provide interfaces to the operating system)

3- debugger (which is used to test and debug created programs).

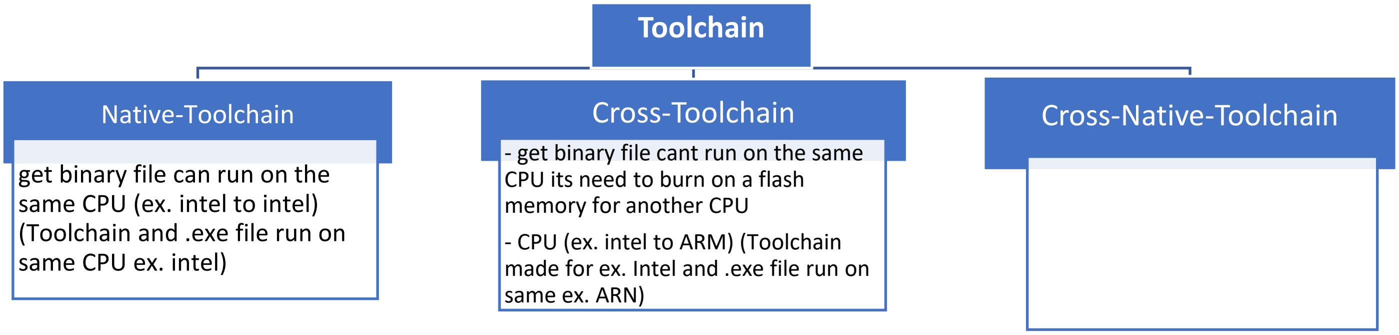
- A complex software product such as a video game needs tools for preparing sound effects, music, textures, 3-dimensional models and animations, together with additional tools for combining these resources into the finished product.
- When talking about toolchains, one must distinguish three different machines:

1- The build machine, on which the toolchain is built

2- The host machine, on which the toolchain is executed

3- The target machine, for which the toolchain generates code

- From these three different machines, we distinguish four different types of toolchains building processes:
- **A native toolchain**, as can be found in normal Linux distributions, has usually been compiled on x86, runs on x86 and generates code for x86.
- **A cross-compilation toolchain**, which is the most interesting toolchain type for embedded development, is typically compiled on x86, runs on x86 and generates code for the target architecture (be it ARM, MIPS, PowerPC or any other architecture supported by the different toolchain components)
- **A cross-native toolchain**, is a toolchain that has been built on x86, but runs on your target architecture and generates code for your target architecture. It's typically needed when you want a native GCC on your target platform, without building it on your target platform.
- A Canadian build is the process of building a toolchain on machine A, so that it runs on machine B and generates code for machine C. It's usually not really necessary.



Toolchain

Native-Toolchain

get binary file can run on the same CPU (ex. intel to intel) (Toolchain and .exe file run on same CPU ex. intel)

Cross-Toolchain

- get binary file cant run on the same CPU its need to burn on a flash memory for another CPU

- CPU (ex. intel to ARM) (Toolchain made for ex. Intel and .exe file run on same ex. ARN)

Cross-Native-Toolchain

Toolchain components

Binutils (Binary utilities)

- The GNU Binutils is the first component of a toolchain. The GNU Binutils contains two very important tools:

  1- The assembler, that turns assembly code (generated by GCC) to binary.

  2- ld, the linker, that links several object codes into a library, or an executable.

- Binutils also contains a couple of other binary file manipulation or analysis tools, such as objcopy, objdump, nm, readelf, strip, and so on. The Binutils website has some documentation on all these tools.

C, C++, Java, Ada, Fortran, Objective-C compiler

- The second major component of a toolchain is the compiler. In the embedded Linux, the only realistic solution today is GCC, the GNU Compiler Collection. Nowadays, as input, it not only supports C, but also C++, Java, Fortran, Objective-C and Ada. As output, it supports a very wide range of architectures.

C library

- The C library implements the traditional POSIX API that can be used to develop user space applications. It interfaces with the kernel through system calls and provides higher-level services.
- Realistically, there are nowadays two options for the C Library:
  1- **glibc** is the C library from the GNU project. It's the C library used by virtually all desktop and server GNU/Linux systems. It's feature-full, portable, complies to standards, but a bit bloated.
  2- **Embedded GLIBC (EGLIBC)** is a variant of the GNU C Library (GLIBC) optimized for embedded systems. Its goals include reduced footprint, support for cross-compiling and cross-testing, while maintaining source and binary compatibility with GLIBC. The project is discontinued.
  3- **uClibc** is an alternate C library, which features a much smaller footprint. This library can be an interesting alternative if flash space and/or memory footprint is an issue. However, the space advantages gained using **uClibc** are becoming less important as the price of memory and flash continues to drop. It is still useful C library for embedded systems without an MMU.
  4- **uClibc-ng** is a spin-off of **uClibc** C library. The main goal of the spin-off is to do regular releases and do a lot of automatic runtime testing.
  5- **musl** New standard C library. **musl** is lightweight, fast, simple, free, and strives to be correct in the sense of standards-conformance and safety.
- The C library has a special relation with the C compiler, so the choice of the C library must be done when the toolchain is generated. Once the toolchain has been built, it is no longer possible to switch to another library.

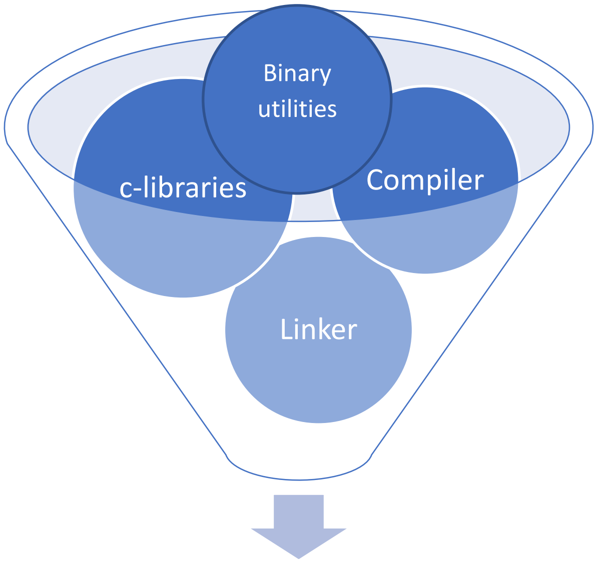
Debugger

- The debugger is also usually part of the toolchain, as a cross-debugger is needed to debug applications running on your target machine. In the embedded Linux world, the typical debugger is GDB.

Lazarus and Free Pascal

- Free Pascal is a professional but free 32 bit / 64 bit compiler for Pascal and Object Pascal.

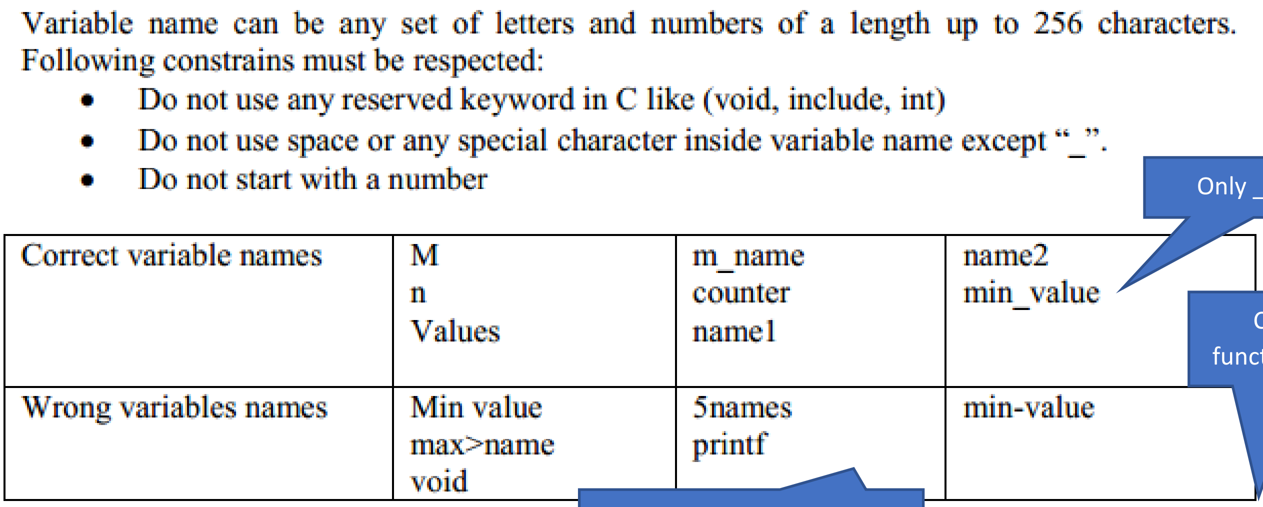
- It supports a wide variety of processors and Linux distributions including the Raspberry Pi.
- The Free Pascal toolchain is widely independent from GCC and other external tools.
- Major components are the Free Pascal compiler (FPC), a command-line tool, a text-mode IDE and, as an optional component, Lazarus, a full-featured GUI-based IDE. FPCUnit is a framework allowing for unit-testing.
- On most platforms Free Pascal makes use of the GDB debugge



Toolchain

- For c-language the native-toolchain is MinGW.
- Gcc is a Cross-toolchain (ex. made it on intel to another architecture).
- IDE Eclipse has an GUI editor help you to make an .exe file.

Variables Name



Variable name can be any set of letters and numbers of a length up to 256 characters. Following constrains must be respected:

- Do not use any reserved keyword in C like (void, include, int)
- Do not use space or any special character inside variable name except "_".
- Do not start with a number

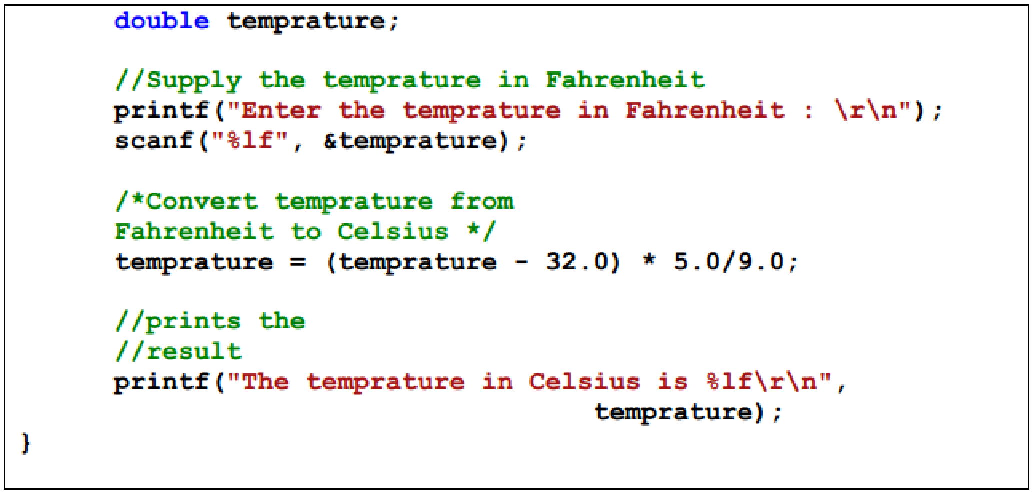
Correct variable names	M n Values	m_name counter name1	name2 min_value
Wrong variables names	Min value max>name void	5names printf	min-value

Only _ can use

Can't use an function named in c

Variable not applicable to begin in number

- To choose a name for a variable there is a conditions according to last table.
- For comment use // for only one line and /* your comment */ for multi line.

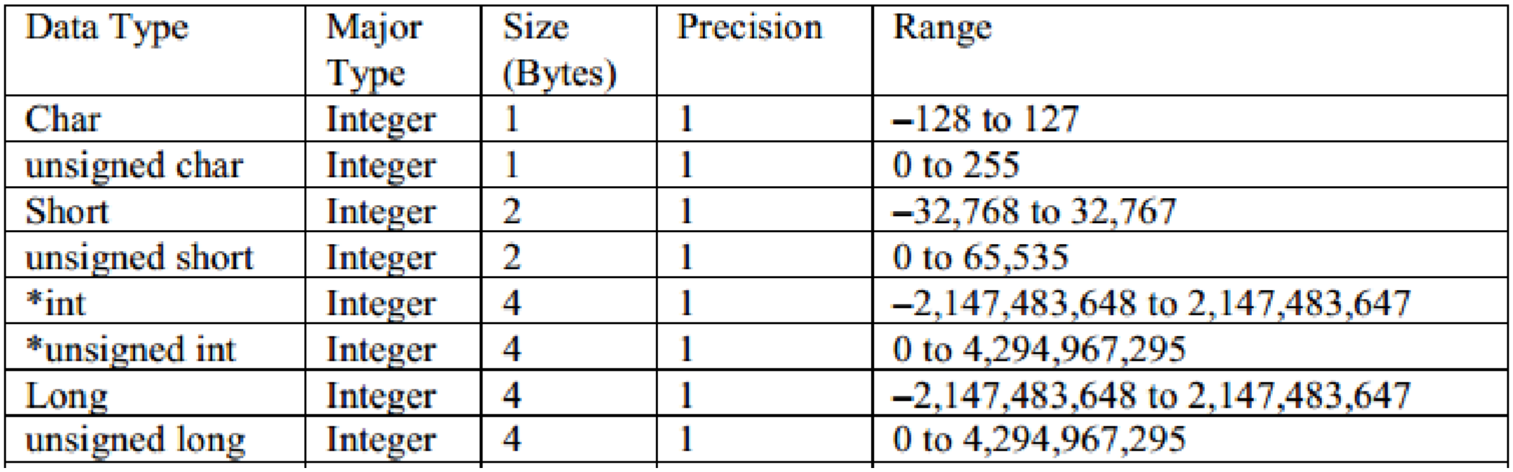


```
        double temprature;

        //Supply the temprature in Fahrenheit
        printf("Enter the temprature in Fahrenheit : \r\n");
        scanf("%lf", &temprature);

        /*Convert temprature from
        Fahrenheit to Celsius */
        temprature = (temprature - 32.0) * 5.0/9.0;

        //prints the
        //result
        printf("The temprature in Celsius is %lf\r\n",
                                    temprature);
}
```

Data Types

No				
1	User Defined	enum		
		typedef		
2	Derived	Arrays		
		Structure		
		union		
		Pointer		
3	Primitive/Basic Types	Real Value		- And + and point (fraction) الكسور
		Integer Value	Unsigned	- And +
			Signed	+

Integer Value



Data Type	Major Type	Size (Bytes)	Precision	Range
Char	Integer	1	1	–128 to 127
unsigned char	Integer	1	1	0 to 255
Short	Integer	2	1	–32,768 to 32,767
unsigned short	Integer	2	1	0 to 65,535
*int	Integer	4	1	–2,147,483,648 to 2,147,483,647
*unsigned int	Integer	4	1	0 to 4,294,967,295
Long	Integer	4	1	–2,147,483,648 to 2,147,483,647
unsigned long	Integer	4	1	0 to 4,294,967,295

long long	8	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	0 to 18,446,744,073,709,551,615

- When we declare a variable that mean we get an space on memory with an address to this variable
- The address for variable (ex. X) = (& X) (ex. 0x1000)
- For embedded it's very important to look at size column in last table.

Introduction in computer science

- 0 و 1 هما فقط اللغة الوحيدة التي يفهمها الحاسوب
- 0 يرمز الى انقطاع الكهرباء اما 1 يرمز الى مرور التيار الكهربائي
- Bit هو وحدة واحدة مرة تساوي 0 ومرة تساوي 1 (ترانزستر)
- Byte = bit bit bit bit bit bit bit bit
- Byte = 8 bit (8 ترانزستور)

Decimal system النظام العشري

الخانة 100	الخانة 10	الخانة 1	الناتج
3	2	1	**321**
4	0	0	**400**

- 0 1 2 3 4 5 6 7 8 9 نظام يتكون من عد عن طريق الخانات **مثال**

- الخانة 1 = 10^0 -------------------------- اي شئ هنا سيتم ضربه في 10^0
- الخانة 10 = 10^1 ------------------------- اي شئ هنا سيتم ضربه في 10^1
- الخانة 100 = 10^2 ----------------------- اي شئ هنا سيتم ضربه في 10^2
- ....الخ

Binary system العد الثنائي

- كل حرف في اللغة الإنجليزية له تعريف في لغة الحاسب الالي حتى الصور والفيديوهات وغيرها من البيانات ونظام العد هذا يسمى بنظام العد الثنائي binary system
- وهو ايضا مثل النظام العشري ولكن على رقمين فقط (ليس 10 ارقام كما في النظام العشري)

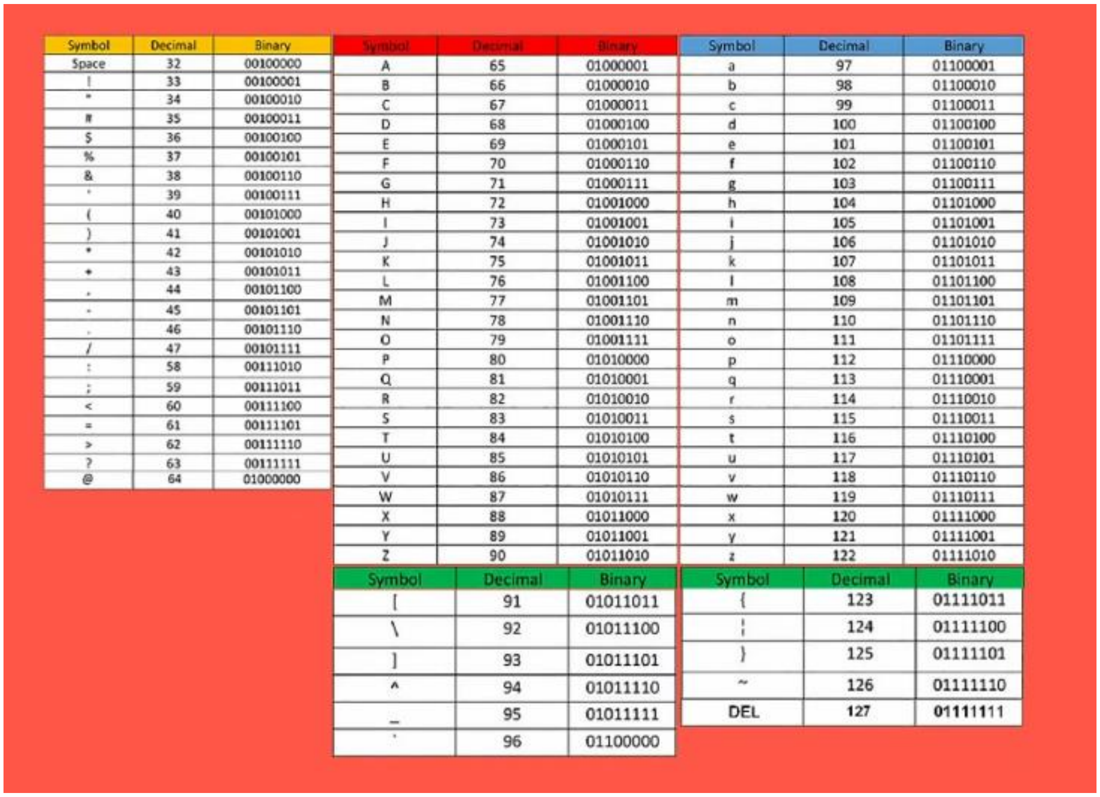
الخانة 128	الخانة 64	الخانة 8	الخانة 4	الخانة 2	الخانة 1	الناتج
1	1	1	0	1	0	**202**
0	0	1	1	0	1	**13**

- الخانة 1 = 2^0 -------------------------- اي شئ هنا سيتم ضربه في 2^0
- الخانة 2 = 2^1 -------------------------- اي شئ هنا سيتم ضربه في 2^1
- الخانة 4 = 2^2 ----------------------- اي شئ هنا سيتم ضربه في 2^2
- ....الخ

ASCII code American standard information interchange

- هو نظام يحول اي بيانات الى رموز لكي يقرؤها الحاسوب
- لكن خلل هذا النظام انه يحتوي على 7 bit فقط اي انه ان اقصى قيمة يمكن كتابتها هى 127

- لذلك ظهر نظام جديد اسمه



Symbol	Decimal	Binary
Space	32	00100000
!	33	00100001
"	34	00100010
#	35	00100011
$	36	00100100
%	37	00100101
&	38	00100110
'	39	00100111
(	40	00101000
)	41	00101001
*	42	00101010
+	43	00101011
,	44	00101100
-	45	00101101
.	46	00101110
/	47	00101111
:	58	00111010
;	59	00111011
<	60	00111100
=	61	00111101
>	62	00111110
?	63	00111111
@	64	01000000

Symbol	Decimal	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010

Symbol	Decimal	Binary
[	91	01011011
\	92	01011100
]	93	01011101
^	94	01011110
_	95	01011111
`	96	01100000

Symbol	Decimal	Binary
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010

Symbol	Decimal	Binary
{	123	01111011
¦	124	01111100
}	125	01111101
~	126	01111110
DEL	127	01111111

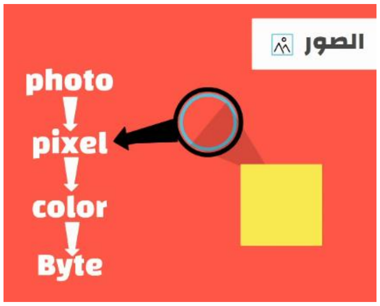
Unicode

- هو نظام يبدء من 16 bitالى 32 bit
- اي يمكن اضافة لغات اخرى واضافة تعبيرات غيرها من الأشياء

كيف يمكن للحاسب الالي فهم الالوان

- هناك نظام اسمه RGB

RGB



- Red Green Bule
- كل لون من الالوان الرئيسية له قيمة من byte
- والناتج يتكون من 3 byte
- وال byte 1يتكون من 8 bitاي ان أكبر قيمة له = 255
- انه ان أكبر قيمة لون يمكن وضعها هى 255 للون الواحد

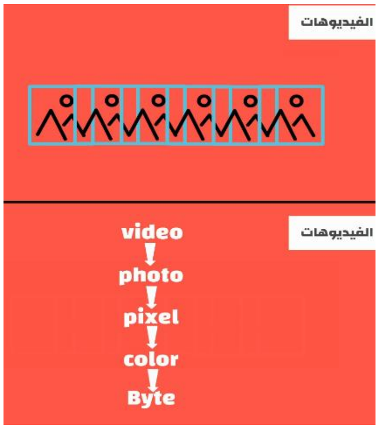
كيف يفهم الحاسب الالي الصور

- كل صورة تحتوي على عدد من ال pixels

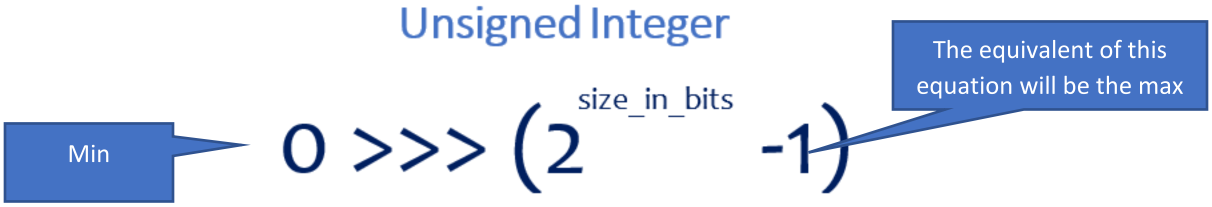
- كل pixelعبارة عن لون كل لون عبارة عن RGB

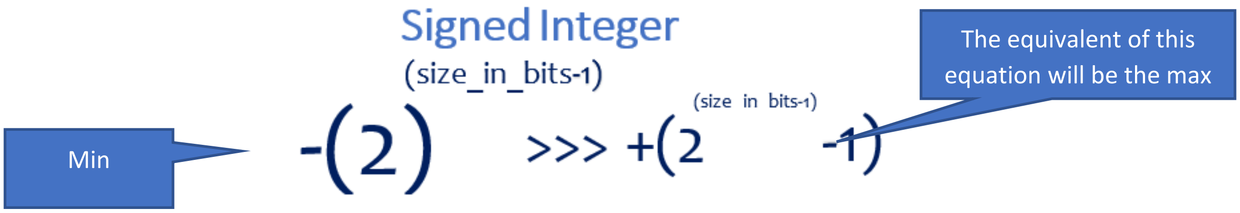
كيف يفهم الحاسب الالي الفيديو

- الفيديو عبارة عن مجموعة من الصور
- وكذلك الموسيقى



- If we declare a variable Unsigned char (size = 1 byte = 8 bits) as example min 0000 0000 and 1111 1111
- That mean the first number 0 because 0^8 = 0 and the max value is according to this equation (0 (the beginning number (min number of bits)) >>> (2^(size of bits of the character) -1) = Max number of bits in this example min = 0 and max = 2^8 = 256 -1 = 255


Unsigned Integer
Min
The equivalent of this equation will be the max
0 >>> (2^size_in_bits -1)


Signed Integer
Min
The equivalent of this equation will be the max
-(2)^(size_in_bits-1) >>> +(2^(size_in_bits-1) -1)

- The number in hexadecimal = 4 digit
- So, in last ex will be in hexadecimal = FF = 1111 1111
- The most in embedded used unsigned integer.
- Some compiler define the integer size is 4 bytes or 8 bytes but we usually defined it as 4 bytes

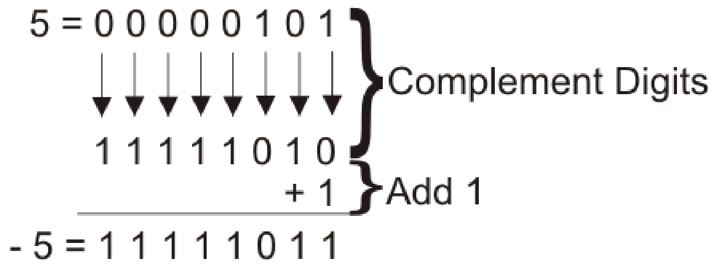
Floating points Types

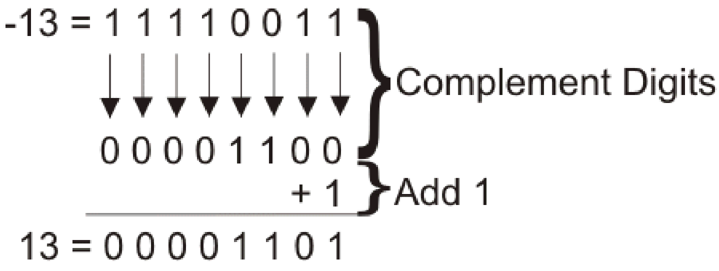
Type	**Storage size**	**Value range**	**Precision**
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places

long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Complement

- If we need to complement digit from +ve to -ve, we use this equation





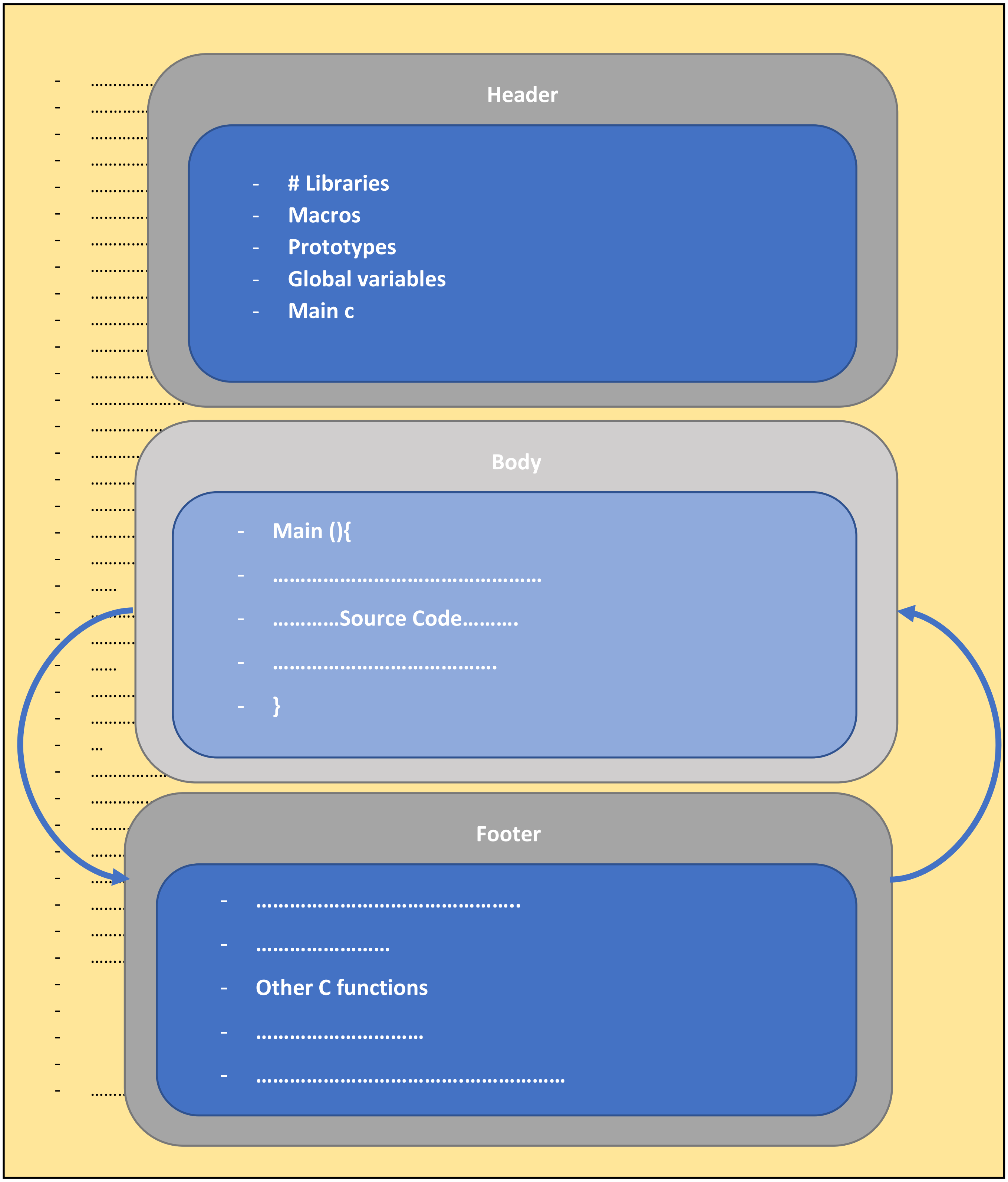
- Change 1 to 0 and 0 to 1 then add 1
- Or we can change all binary except the first one form right so, that's mean the -ve number always = 1 from the left

Boolean

- Traditionally, there was no booleantype in C. However, C99 defines a standard booleantype under <sdbool.h> header file. A booleantype can take one of two values, either true or false. For example:

```
- #include<stdio.h>
- #include<stdbool.h>
- intmain()
- {
- boola = true;
- return0;
- }
```

How to divide you source code page



Header

- # Libraries
- Macros
- Prototypes
- Global variables
- Main c

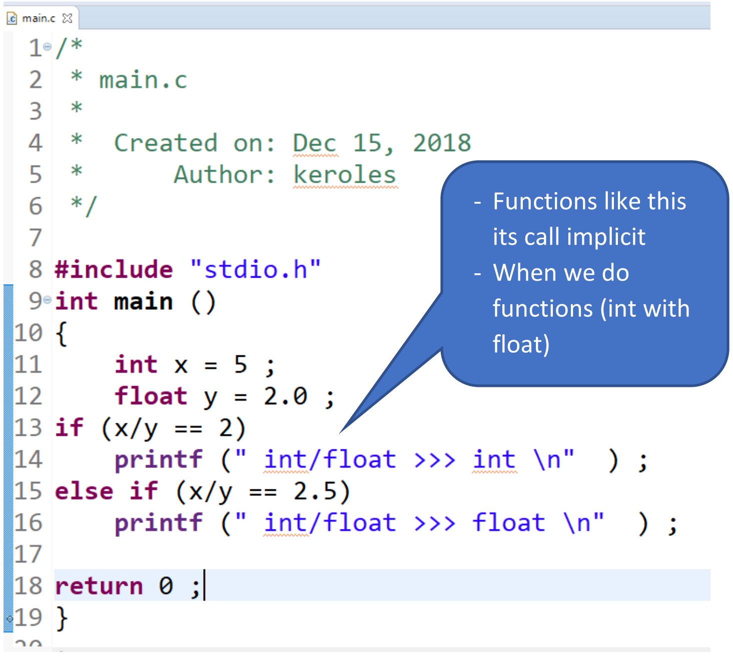
Body

- Main (){
- ……………………………………………
- …………Source Code……….
- ……………………………………
- }

Footer

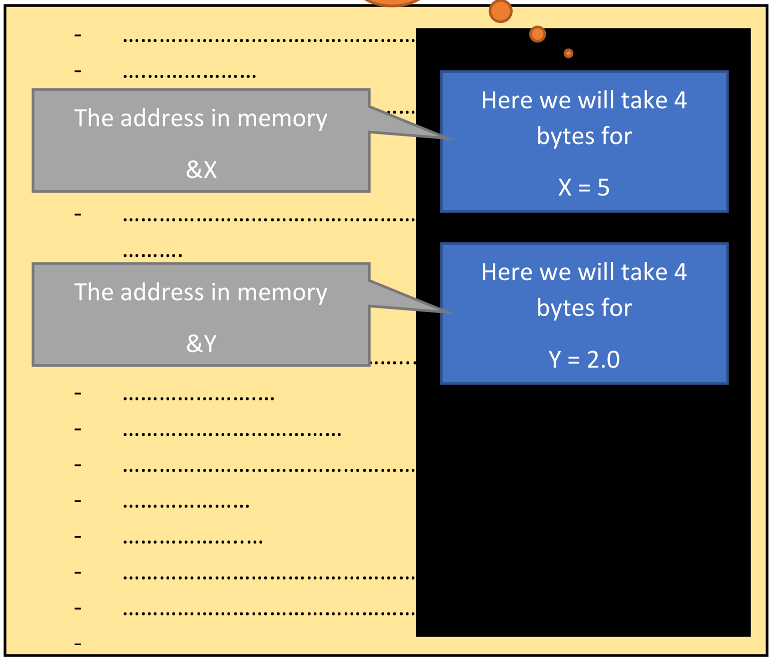
- ……………………………………………
- ………………………
- Other C functions
- ……………………………
- ……………………………………………………

How to solve problems



```
/*
 * main.c
 *
 *  Created on: Dec 15, 2018
 *      Author: keroles
 */

#include "stdio.h"
int main ()
{
	int x = 5 ;
	float y = 2.0 ;
if (x/y == 2)
	printf (" int/float >>> int \n"  ) ;
else if (x/y == 2.5)
	printf (" int/float >>> float \n"  ) ;

return 0 ;
}
```

main.c
- Functions like this its call implicit
- When we do functions (int with float)



This is our memorv
The address in memory &X
Here we will take 4 bytes for X = 5
The address in memory &Y
Here we will take 4 bytes for Y = 2.0

- First, we should think in memory data storage.
- So, here we have 2 variables (int and float) int size = 4 bytes and float size = 4 bytes.
- When we do functions (int with float) the float higher than int so the int it will be smaller than float so, the result of the equation it will be float (real).
- So, in this example (X/Y) = 2.5 so, the fist condition will not happen, we will go to else if condition.
- We will fide that (X/Y) = 2.5 == 2.5 so, this condition will happen.

Integer and Float Conversions

In order to effectively develop C programs, it will be necessary to understand the rules that are used for the implicit conversion of floating point and integer values in C. These are mentioned below. Note them carefully.

- An arithmetic operation between an integer and integer always yields an integer result.
- An operation between a real and real always yields a real result.
- An operation between an integer and real always yields a real result. In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Type Conversion in Assignments

EX 1

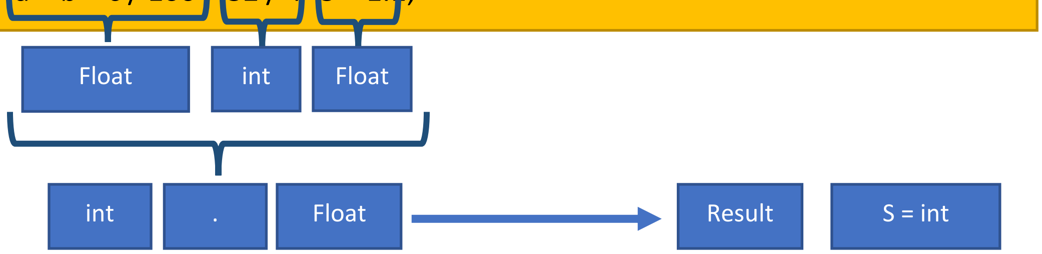
- Here in the first assignment statement though the expression's value is a float (3.5) it cann stored in I, since it is an int.
- In such a case the float is demoted to an int and then its value is stored. Hence what gets s I, is 3.
- Exactly opposite happens in the next statement. Here, 30 is promoted to 30.000000 and th stored in b, since b being a float variable cannot hold anything except a float value.

```
- inti;
- float b;
- i= 3.5;
- b = 30;
```

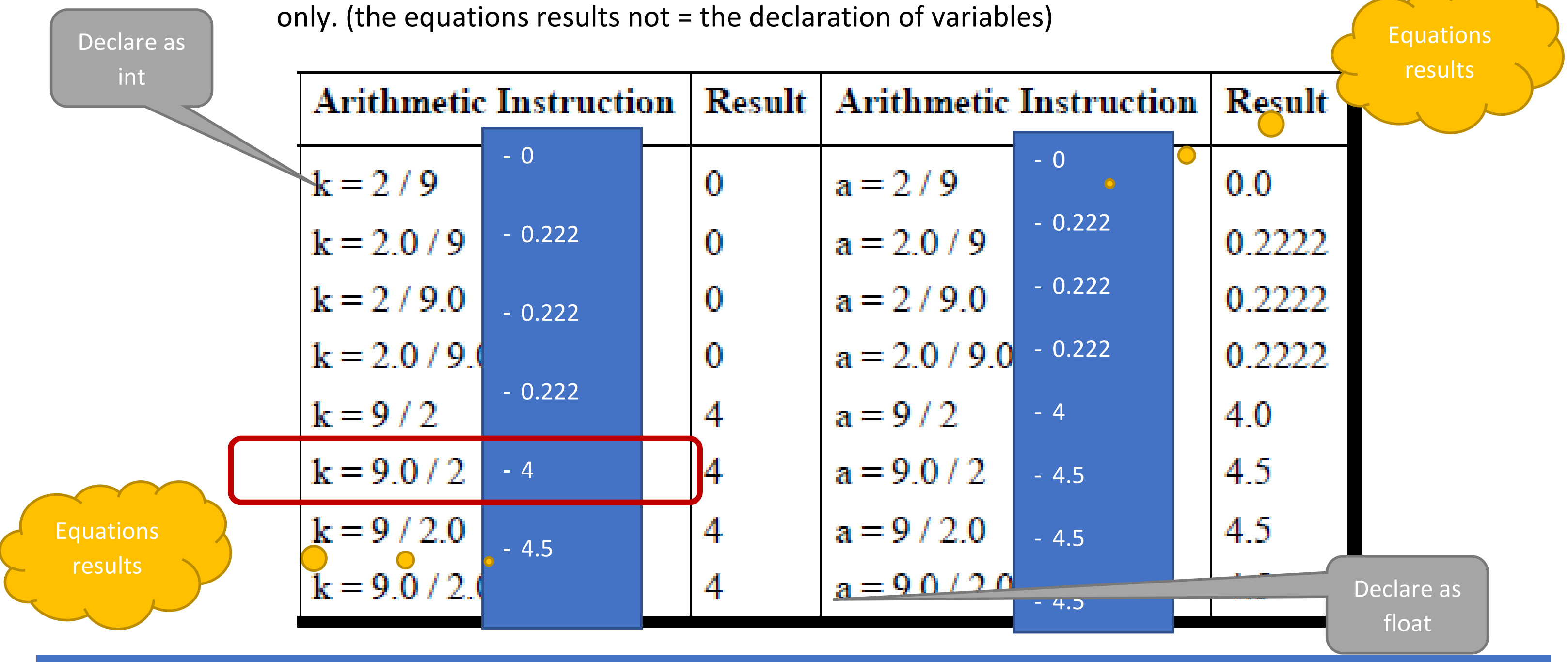
EX 2

- In the assignment statement some operands are ints where as others are floats. As we know, during evaluation of the expression
- The ints would be promoted to floats and the result of the expression would be a float. But when this float value is assigned to s it is again demoted to an int and then stored in s.

```
- float a, b, c;
- int s;
- s = a * b * c / 100 + 32 / 4 -3 * 1.1;
```



- The result should be float but s variable here declares as an int so, it will take the int part only. (the equations results not = the declaration of variables)

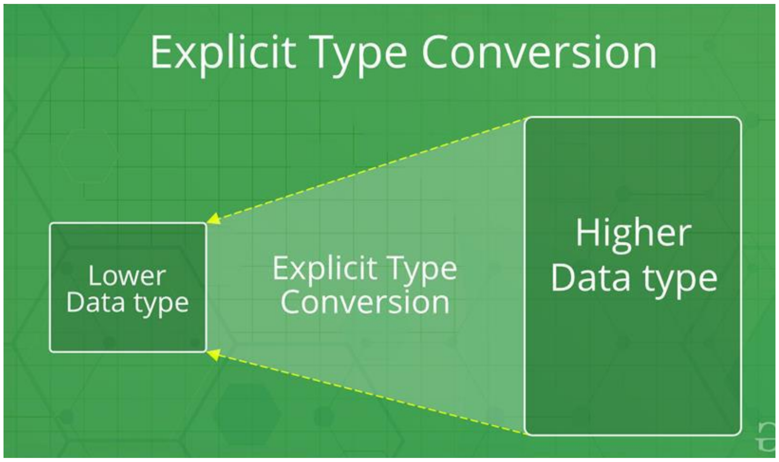


Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.2222
k = 2 / 9.0	0	a = 2 / 9.0	0.2222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.2222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

Declare as int
Equations results
- 0
- 0.222
- 0.222
- 0.222
- 4
- 4.5
- 0
- 0.222
- 0.222
- 0.222
- 4
- 4.5
- 4.5
- 4.5
Equations results
Declare as float

- In last ex we can see that variable k is declare as int but the result of equation should be float so the result of equation not equal the variable result that will store in memory.

Type Conversion in C



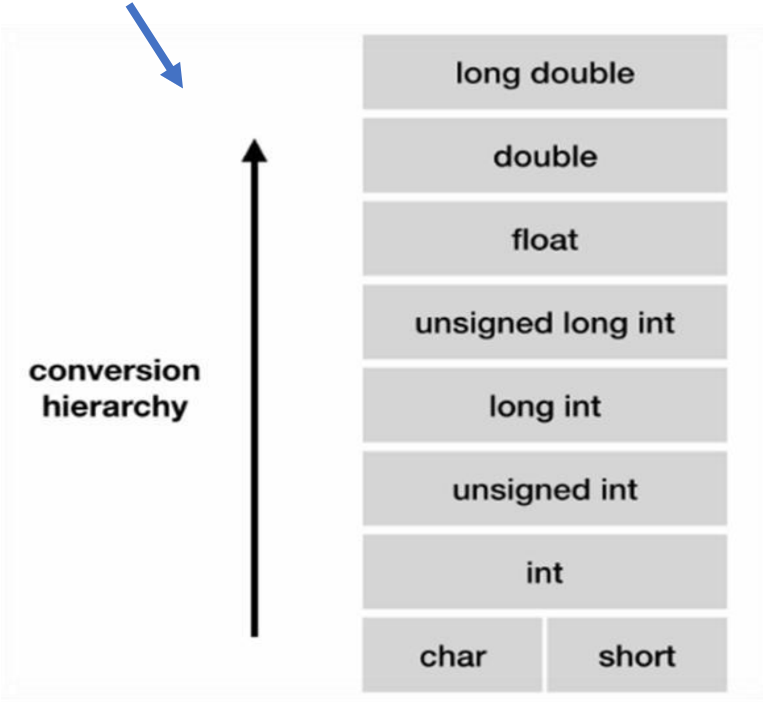
Explicit Type Conversion

Lower Data type

Explicit Type Conversion

Higher Data type

- Implicit happened automatic.
- Explicit that mean I told the program to do that.



conversion hierarchy

long double

double

float

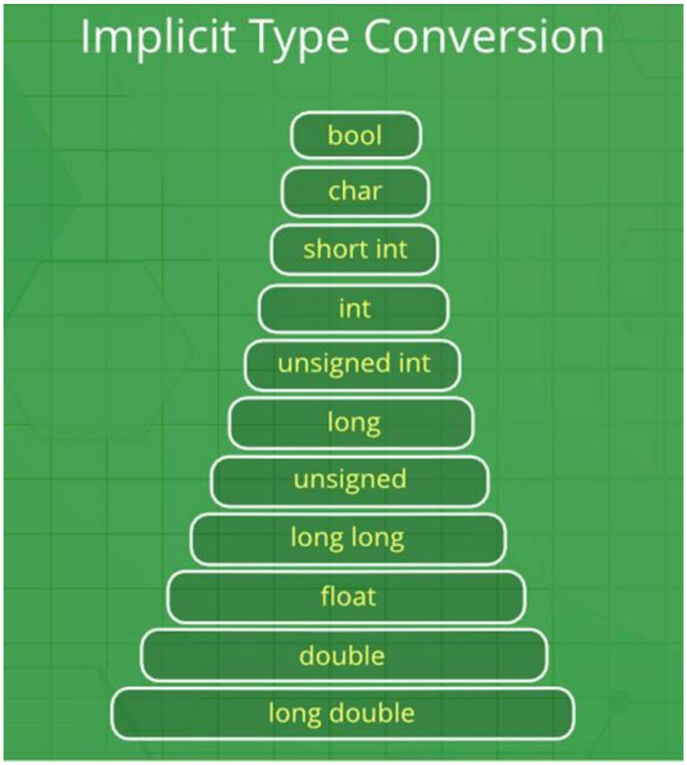
unsigned long int

long int

unsigned int

int

char | short



Implicit Type Conversion

bool

char

short int

int

unsigned int

long

unsigned

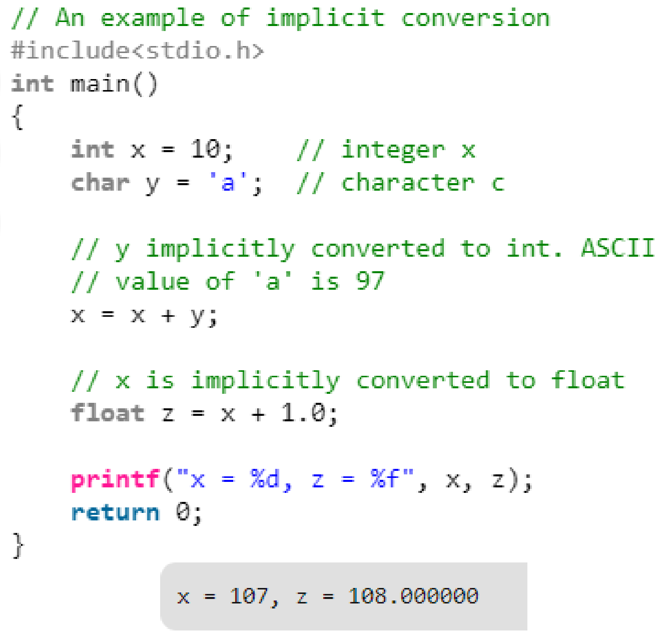
long long

float

double

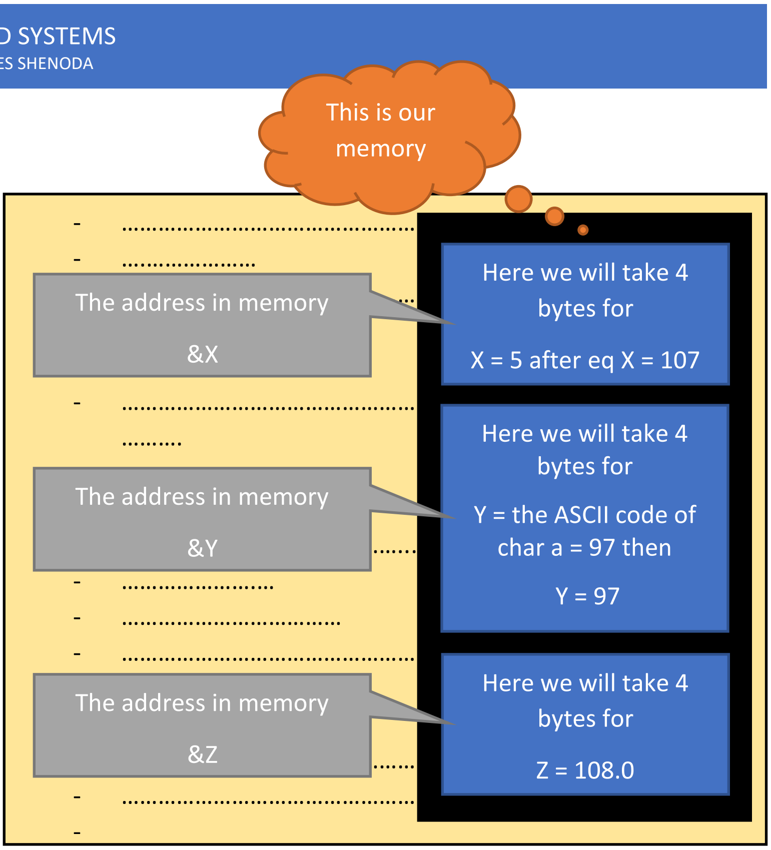
long double

EX 1 (implicit)

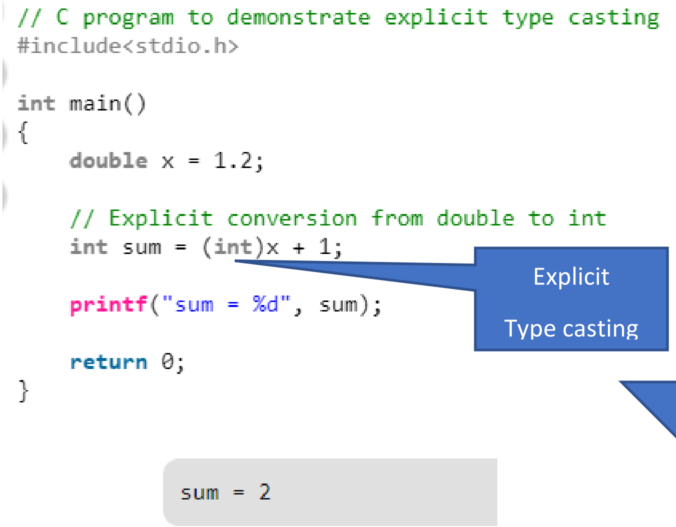


```
// An example of implicit conversion
#include<stdio.h>
int main()
{
    int x = 10;    // integer x
    char y = 'a';  // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

```
x = 107, z = 108.000000
```

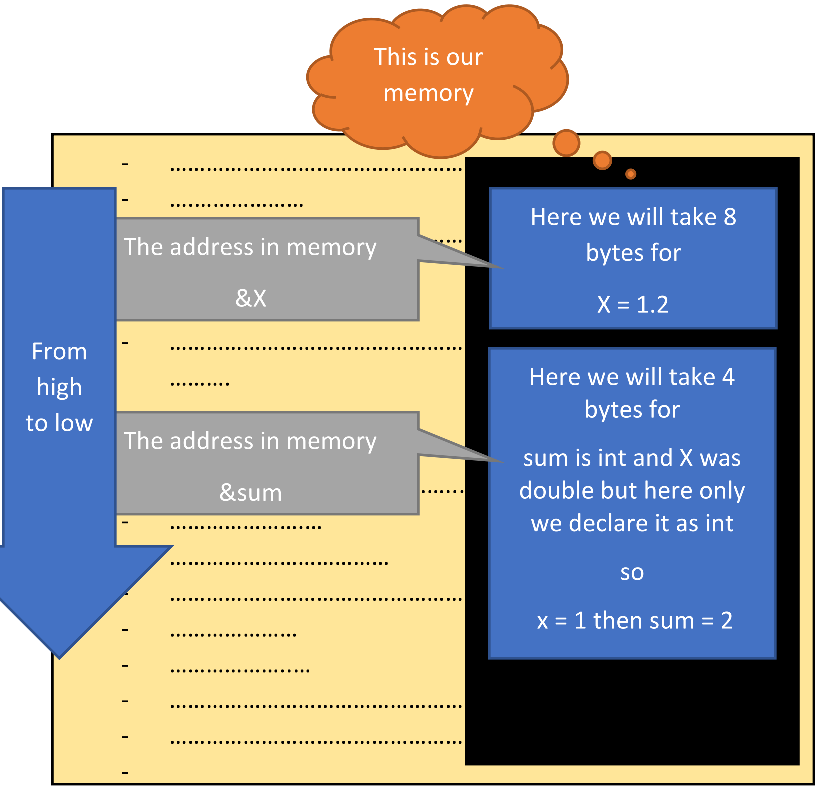

This is our memory
The address in memory
&X
Here we will take 4 bytes for
X = 5 after eq X = 107
The address in memory
&Y
Here we will take 4 bytes for
Y = the ASCII code of char a = 97 then
Y = 97
The address in memory
&Z
Here we will take 4 bytes for
Z = 108.0

EX 2 (Explicit)

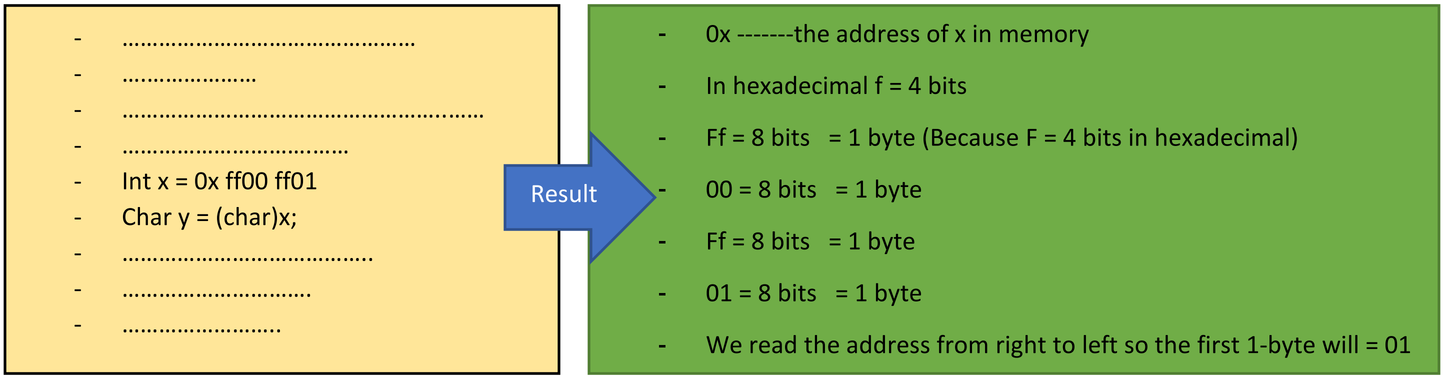


```
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

```
sum = 2
```



Ex



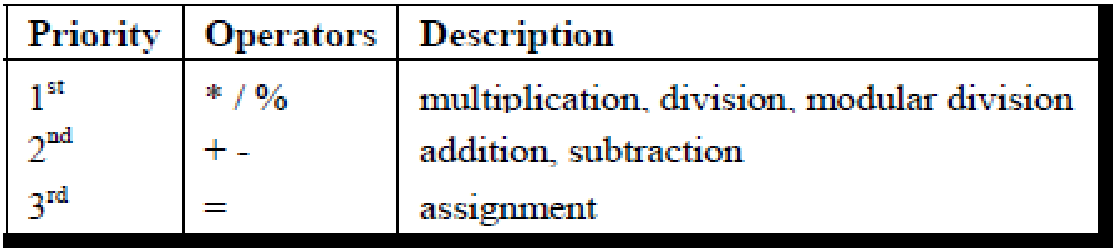
- ……………………………………………
- …………………
- ……………………………………………………
- ………………………………
- Int x = 0x ff00 ff01
- Char y = (char)x;
- ……………………………………
- …………………………
- ……………………..

- 0x -------the address of x in memory
- In hexadecimal f = 4 bits
- Ff = 8 bits = 1 byte (Because F = 4 bits in hexadecimal)
- 00 = 8 bits = 1 byte
- Ff = 8 bits = 1 byte
- 01 = 8 bits = 1 byte
- We read the address from right to left so the first 1-byte will = 01

Hierarchy of operations

- While executing an arithmetic statement, which has two or more operators, we may have some problems as to how exactly does it get executed.
- For example, does the expression 2 * x -3 * y correspond to (2x)-(3y) or to 2(x-3y)?
- Similarly, does A / B * C correspond to A / (B * C) or to (A / B) * C?
- To answer these questions satisfactorily one has to understand the ‘hierarchy’ of operations.

Example 1.1: Determine the hierarchy of operations and evaluate the following expression:



Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

i = 2 * 3 / 4 + 4 / 4 + 8 -2 + 5 / 8

- i= 6 / 4 + 4 / 4 + 8 -2 + 5 / 8 operation: *
- i= 1 + 4 / 4 + 8 -2 + 5 / 8 operation: /
- i= 1 + 1+ 8 -2 + 5 / 8 operation: /
- i= 1 + 1 + 8 -2 + 0 operation: /
- i= 2 + 8 -2 + 0 operation: +
- i= 10 -2 + 0 operation: +
- i= 8 + 0 operation : -
- i= 8 operation: +

C Programming Input Output (I/O): printf() and scanf()

- C programming has several in-built library functions to perform input and output tasks.
- Two commonly used functions for I/O (Input/Output) are **printf()** and **scanf()**.
- The **scanf()** function reads formatted input from standard input (keyboard) whereas the **printf()** function sends formatted output to the standard output (screen).
- But in microcontroller we didn’t do that so we not used **printf()** or **scanf()**.

No	Order	Syntax	What it do?
1	include	#include <….name of library….>	لستدعاء الملف الذي يحتوي على الاوامر الخاصة باللغة
2	Int main	Int main(Function data) {…C language code…. }	- هى حاوية خاصة يتم كتابة الكود البرمجي الخاص بلغة C بداخله - Void أي ان المعادلة لا ترجع بقيمة
3	Printf	Printf("…Data….,%data type of variable", Variable name);	طباعة أي شئ
4	String	String Variable name =" ……..";	في حالة تعريف نص كامل
5	Float	Float Variable name = ……..;	في حالة تعريف رقم عشري صغير
6	Integer	Int Variable name = ……..;	في حالة تعريف رقم صحيح
7	double	double Variable name = ……..;	في حالة تعريف رقم عشري كيير
8	long	long Variable name = ……..;	رقم صحيح لكن كبير
9	Bool (Boolean)	Bool Variable name = ……..;	لتعريف متغير صح أم خطأ
10	Char (character)	Char Variable name =' ……..';	- لتعريف حرف - يتم استخدام ' بدل من " في حالة Char

11	if	If (…comparison data….){
		C language code….
		}
			معادلة if condition اي في حالة تحقق الشرط قم بفعل كذا

12	Else	Else {
		C language code….
		}
			دالة توضع بعد if اي في حالة عدم تحقق شرط if يتحقق ما بداخل else

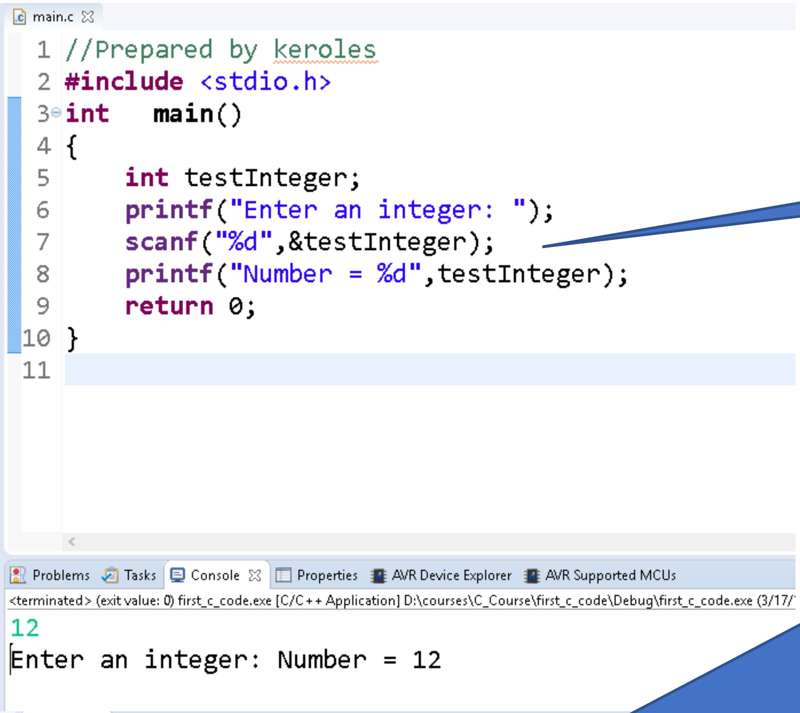
13	Else if	Else if (…Comparison data….){
		C language code….
		}
			- دالة توضع بعد if اي في حالة عدم تحقق شرط if يتحقق ما بداخل elseif في حالة تحقق شرطها
			- تستخدم في حالة وجود اكثر من شرط

14	While	While (…Boolean Expiration….){
		C language code….
		}
			- من دوال عمل التكرار وهى تقوم بعمل الكود طالما الشرط متحقق (عد لا نهائي من المرات) عندما لا يحقق الشرط يقف التكرار
			- يجب ان يكون الشرط من نوع Boolean اي شرط لا يقبل إلا قيمتين إما صح أو خطأ

15	Do While	do {
		C language code….
		} while (…Boolean Expiration….);
			- من دوال عمل التكرار وهى تقوم بعمل الكو (مرة واحدة أولا ثم تبدأ في تنفيذ الشرط) طالما الشرط متحقق (عد لا نهائي من المرات) عندما لا يحقق الشرط يقف التكرار
			- يجب ان يكون الشرط من نوع Boolean اي شرط لا يقبل إلا قيمتين إما صح أو خطأ

16	for	For (data type of variable (space key) Variable name = …Variable Value ; …Boolean Expiration…. ; … Variable condition){
		C language code….
		}
			- من دوال عمل التكرار وهى تقوم بعمل الكود طالما الشرط متحقق (عد لا نهائي من المرات (وهو محدد على حسب ال Variable condition)) عندما لا يحقق الشرط (Boolean Expiration) يقف التكرار
			- يجب ان يكون الشرط من نوع Boolean اي شرط لا يقبل إلا قيمتين إما صح أو خطأ
			- يتم بداخلها تعريف المتغير وقيمته ونوعه (بداية العداد)
			- يتم تعريف الشرط الذي ستتكرر فيه الاكواد طالما متحقق (نهاية العداد)
			- يتم تعريف فيه الحد الموضوع للمتغير مع كل دورة
			- Variable condition هو decrement (post / pre) أو Increment (post / pre)

17	Const (constant)	Const data type of variable Variable name =…..;	- لتعريف متغير ثابت دائما



```c
//Prepared by keroles
#include <stdio.h>
int   main()
{
     int testInteger;
     printf("Enter an integer: ");
     scanf("%d",&testInteger);
     printf("Number = %d",testInteger);
     return 0;
}
```

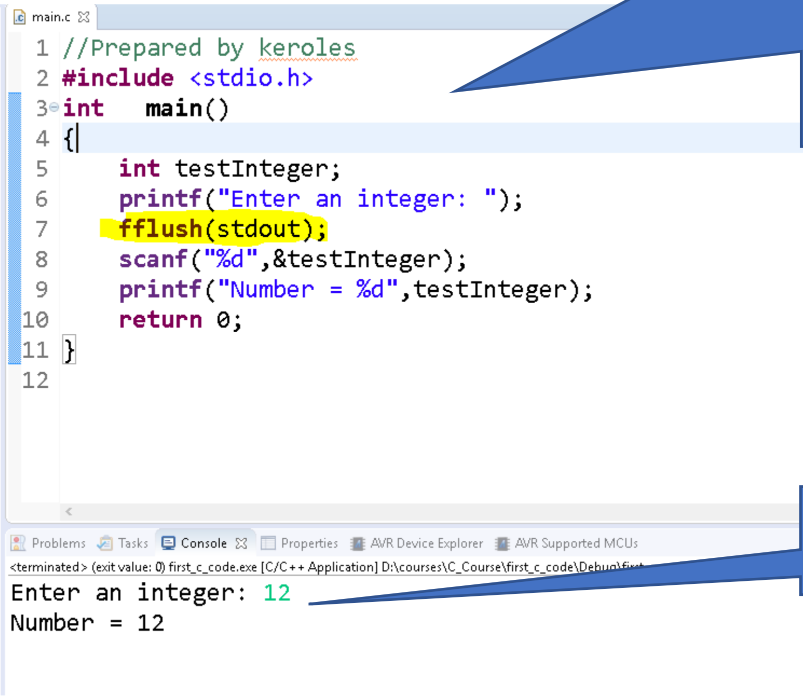
```
<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\first_c_code.exe (3/17/
12
Enter an integer: Number = 12
```

What scanf() do:

- Take an int input from user ("%d") (the value of int)
- Put it in (&) (the address of) (variable = testinteger)

Why the result before the input?

- There is a bug in Eclipse
- The library stdio.h talking the window (I/O)
- Then the buffer (the place that Eclipse read the output of (I/O) that come from library (stdio.h)
- To solve that we add (fflush(stdout);)
- Fflush is a function
- Stdout and stdin (standerd in and standard out)
- Printf() functions used(standerd in and standard out) in stdio.h library to read from screen in windows



```c
//Prepared by keroles
#include <stdio.h>
int   main()
{
     int testInteger;
     printf("Enter an integer: ");
     fflush(stdout);
     scanf("%d",&testInteger);
     printf("Number = %d",testInteger);
     return 0;
}
```

```
<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\first
Enter an integer: 12
Number = 12
```

- add (fflush(stdput);)
- then the result be correct now

'\r\n'	Makes a newline
'\t'	Inserts a tab
'\\'	Prints ‘\’
'\"'	Prints ‘"’
'%d'	Prints or scans an integer (int) value
'%x' or '%X'	Prints or scans an integer value in small or capital hexadecimal format
'%f'	Prints or scans a real (float) value
'%lf'	Prints or scans a real (double) value
'%u'	Prints or scans an (unsigned int) value
'%c'	Prints or scans a single character (char)

- Print the character that equivalent this decimal in ASCII code

Code	Output	Description
printf("A\r\nB\r\nC");	A B C	"\r\n" makes a line break where ‘\r’ is the carriage return and ‘\n’ is the newline command.

printf("A\tB\tC\r\n");
printf("D\tE\tF\r\n");
printf("N\tO\tP\r\n");
	A B C
	D E F
	N O P
		'\t' makes a tab separator.

printf("A\\B\\C ");	A\B\C	To print the ‘\’ letter you must place ‘\\’ instead.
printf("Say \"Hello\"");	Say "Hello"	To print the ‘"’ letter you must place ‘\"’ instead.

int a = 20*30;
printf("Area is %d",a);
	Area is 600	The directive '%d' is replaced with an integer value (a).

printf("If the width is %d and the height is %d then the area is %d",20, 30, 20*30);	If the width is 20 and the height is 30 then the area is 600	There are three '%d' directives and three integer values each value is printed instead of one of the '%d' directives, Number of '%d' directives must equals to the number of numeric values.

scanf("%d/%d",&W,&H);
printf("\r\nArea is %d
",W*H);
		integer value.
		The combination '%d/%d' is used to scan two integer value separated by '/'.

int x = 172;
printf("X equals %x
",x);
	X equals ac	The directive '%x' prints the integer value in small hexadecimal format.

int X = 172;
printf("X equals %X
",X);
	X equals AC	The directive '%x' prints the integer value in capital hexadecimal format.

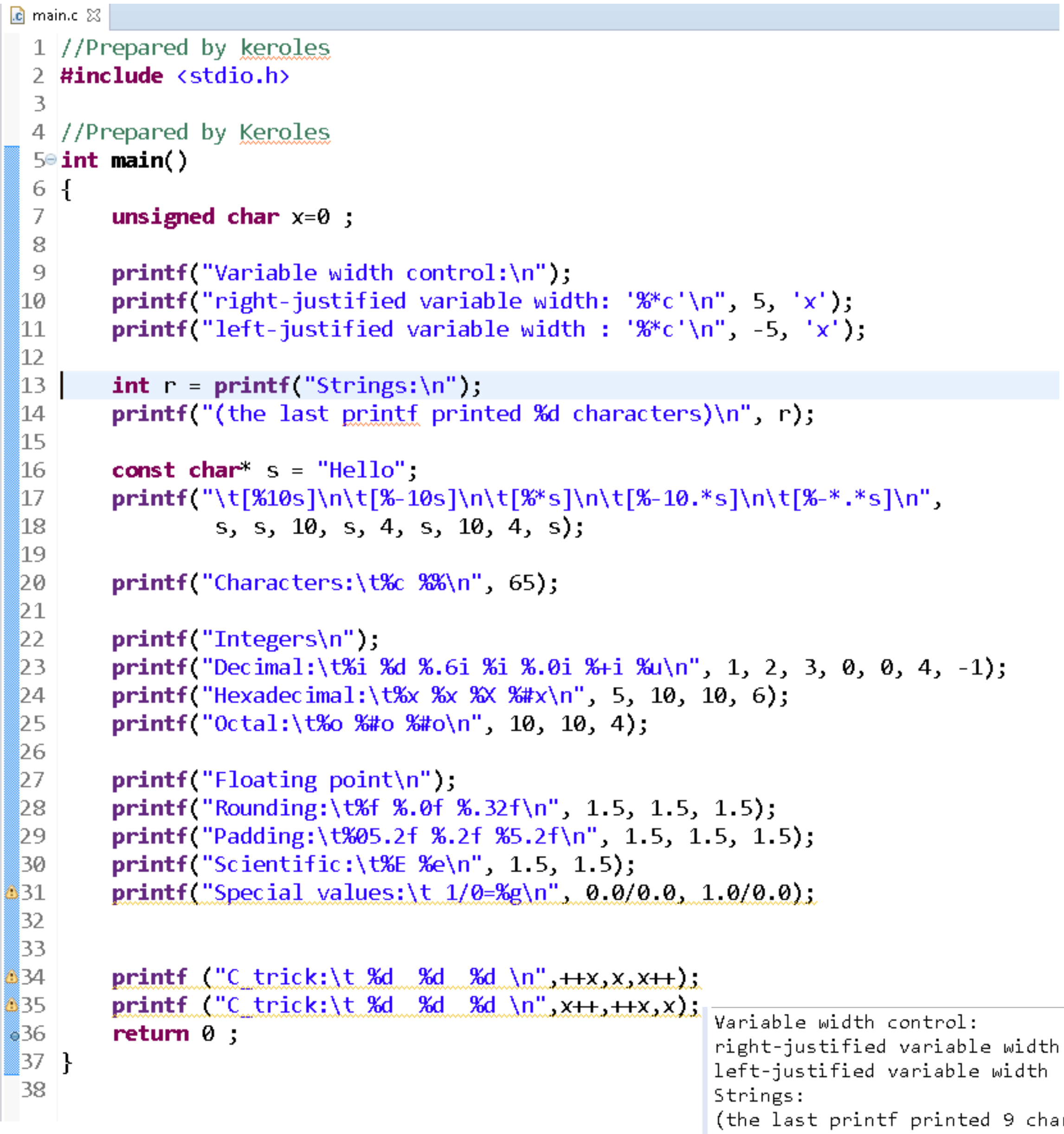
int X;
printf("Enter X in
hexadecimal format:");
scanf("%X",&X);
printf("\r\nX equals %d
",X);
	Enter X in
	hexadecimal format:
	AC
	X equals 172
		The directive '%X' also used to scan values in hexadecimal format.

float R = 2.5;
printf("R equals %f
",R);
	R equals 2.5	The directive '%f' prints a real (float) value.

int X = 6235;
printf("X equals
%10d",X);
	X equals 6235
	------------------- 	
		Prints the number in 10 digits including the '.' and 2 digits in the fraction part.

float R = 8372.5675365;
printf("R equals %10.2f
",R);
	R equals 8372.56
	------------------- 	
		Prints the number in 10 digits including the '.' and 2 digits in the fraction part.

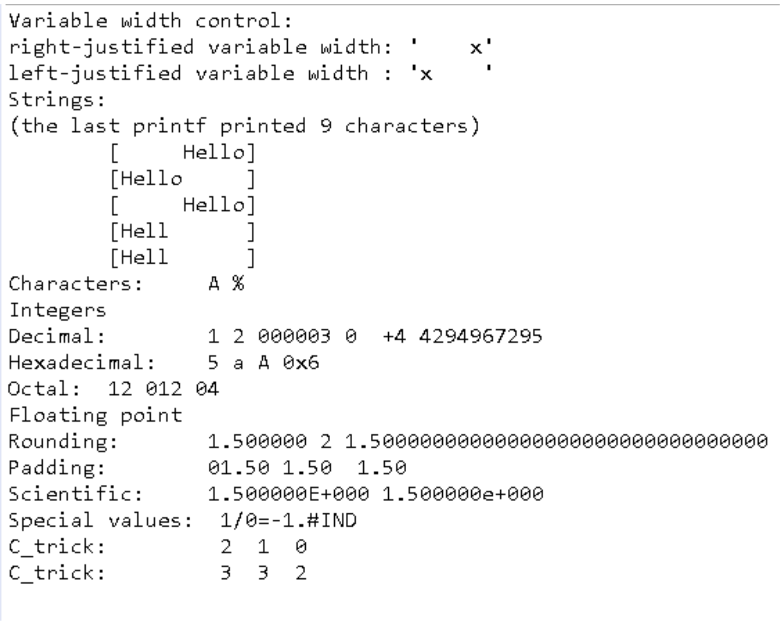
int X = 15;
printf("X equals
%05d",X);
	X equals 00015
	-------------- 	
		Prints the number in 5 digits and pad it with zeros.



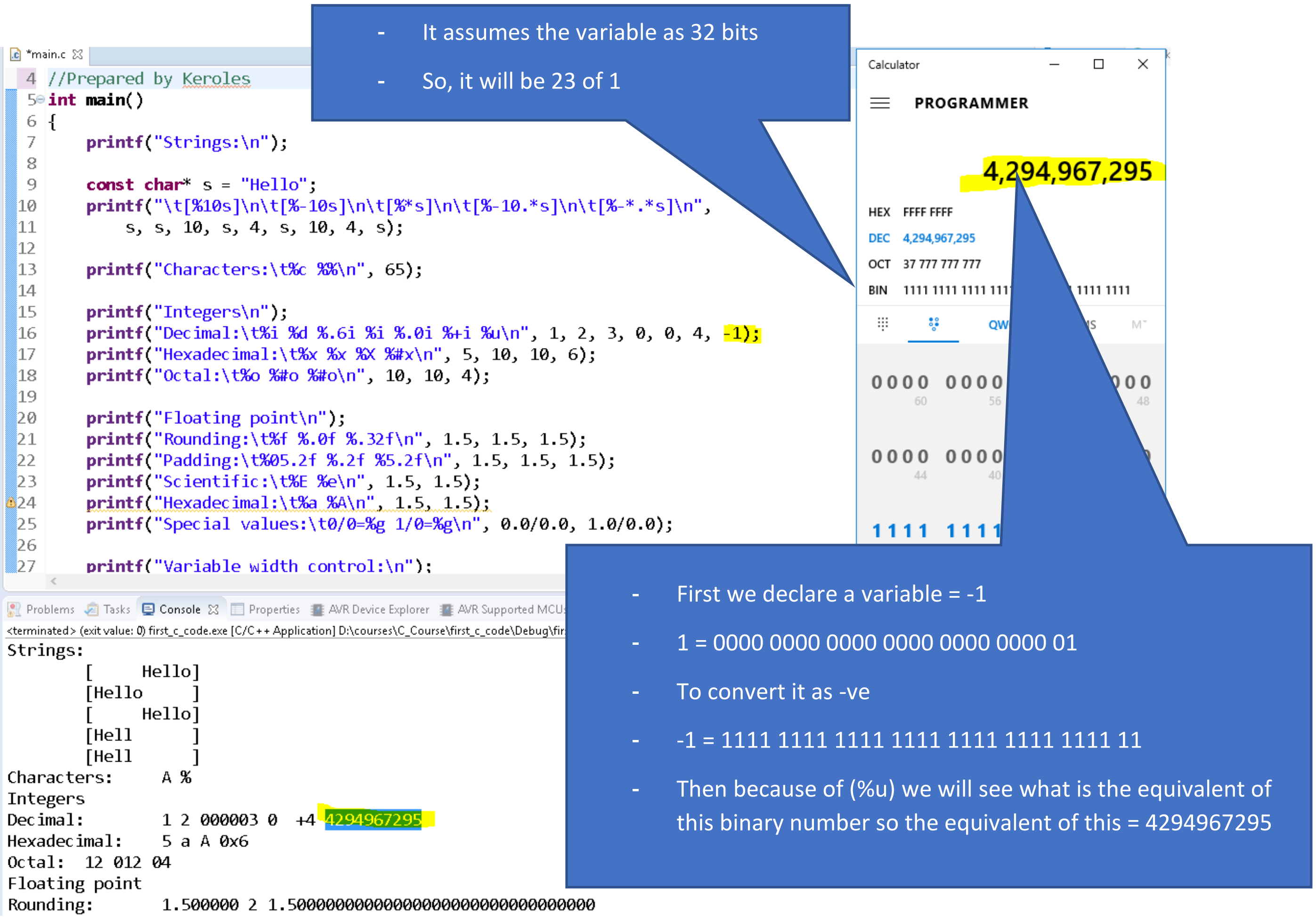
main.c

```c
//Prepared by keroles
#include <stdio.h>

//Prepared by Keroles
int main()
{
    unsigned char x=0 ;

    printf("Variable width control:\n");
    printf("right-justified variable width: '%*c'\n", 5, 'x');
    printf("left-justified variable width : '%*c'\n", -5, 'x');

    int r = printf("Strings:\n");
    printf("(the last printf printed %d characters)\n", r);

    const char* s = "Hello";
    printf("\t[%10s]\n\t[%-10s]\n\t[%*s]\n\t[%-10.*s]\n\t[%-*.*s]\n",
            s, s, 10, s, 4, s, 10, 4, s);

    printf("Characters:\t%c %%\n", 65);

    printf("Integers\n");
    printf("Decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 3, 0, 0, 4, -1);
    printf("Hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
    printf("Octal:\t%o %#o %#o\n", 10, 10, 4);

    printf("Floating point\n");
    printf("Rounding:\t%f %.0f %.32f\n", 1.5, 1.5, 1.5);
    printf("Padding:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
    printf("Scientific:\t%E %e\n", 1.5, 1.5);
    printf("Special values:\t 1/0=%g\n", 0.0/0.0, 1.0/0.0);


    printf ("C_trick:\t %d  %d  %d \n",++x,x,x++);
    printf ("C_trick:\t %d  %d  %d \n",x++,++x,x);
    return 0 ;
}
```

Result



```
Variable width control:
right-justified variable width: '    x'
left-justified variable width : 'x    '
Strings:
(the last printf printed 9 characters)
        [     Hello]
        [Hello     ]
        [     Hello]
        [Hell      ]
        [Hell      ]
Characters:     A %
Integers
Decimal:        1 2 000003 0  +4 4294967295
Hexadecimal:    5 a A 0x6
Octal:  12 012 04
Floating point
Rounding:       1.500000 2 1.50000000000000000000000000000000
Padding:        01.50 1.50  1.50
Scientific:     1.500000E+000 1.500000e+000
Special values:  1/0=-1.#IND
C_trick:         2  1  0
C_trick:         3  3  2
```



```
//Prepared by Keroles
int main()
{
    printf("Strings:\n");

    const char* s = "Hello";
    printf("\t[%10s]\n\t[%-10s]\n\t[%*s]\n\t[%-10.*s]\n\t[%-*.*s]\n",
        s, s, 10, s, 4, s, 10, 4, s);

    printf("Characters:\t%c %%\n", 65);

    printf("Integers\n");
    printf("Decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 3, 0, 0, 4, -1);
    printf("Hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
    printf("Octal:\t%o %#o %#o\n", 10, 10, 4);

    printf("Floating point\n");
    printf("Rounding:\t%f %.0f %.32f\n", 1.5, 1.5, 1.5);
    printf("Padding:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
    printf("Scientific:\t%E %e\n", 1.5, 1.5);
    printf("Hexadecimal:\t%a %A\n", 1.5, 1.5);
    printf("Special values:\t0/0=%g 1/0=%g\n", 0.0/0.0, 1.0/0.0);

    printf("Variable width control:\n");
```

```
<terminated> (exit value: 0) first_c_code.exe [C/C++ Application] D:\courses\C_Course\first_c_code\Debug\fir
Strings:
        [     Hello]
        [Hello     ]
        [     Hello]
        [Hell      ]
        [Hell      ]
Characters:     A %
Integers
Decimal:        1 2 000003 0  +4 4294967295
Hexadecimal:    5 a A 0x6
Octal:  12 012 04
Floating point
Rounding:       1.500000 2 1.50000000000000000000000000000000
```

- It assumes the variable as 32 bits
- So, it will be 23 of 1

- First we declare a variable = -1
- 1 = 0000 0000 0000 0000 0000 0000 01
- To convert it as -ve
- -1 = 1111 1111 1111 1111 1111 1111 1111 11
- Then because of (%u) we will see what is the equivalent of this binary number so the equivalent of this = 4294967295

Mathematical and Logical Expressions

C language supports following expression operators:

=	Equal operator. X = Y; means copy the value of Y into X
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Division Reminder (Mod) (EX: 15%4 → 3)
( )	Mathematical Parenthesis
++	Increment by one operator
--	Decrement by one operator
|	Logical OR operator
&	Logical AND operator
^	Logical XOR operator
~	Logical NOT operator
>>	Shift Right Operator
<<	Shift Left Operator
+=	Self addition operator. Ex: X+=2 means X = X + 2
Other self operators -=, *=, /=,%=, |=, &=, ^=, >>=, <<=	

Marks

No	Order	Syntax	What it do?
1	\	\…	اي قم بعمل escape لاي شئ بعده
2	=	Variable name = data	ليس معناه المساواة لكن معناه ان ما بعد العلامة يتم حفظه فى ما قبل العلامة
3	%	%…Data type	لتعريف المتغير داخل معادلة printf
4	-	… data … - … data ….	لعملية الطرح
5	+	… data … + … data ….	لعملية الجمع
6	/	… data … / … data ….	لعملية القسمة
7	*	… data.. * … data …..	لعملية الضرب
8	**	… data ……**… data ……	الاس الرياضي Exponentiation
9	%	…… data … %… data ……	باقي القسمة Modulus (Division remainder)
10	++	… data ……++… data ……	- Increment (post / pre) يوضع المتغير قبل أو بعد العلامة - يقوم بزيادة المتغير ب 1
11	--	… data ……--…… data …	decrement (post / pre) يوضع المتغير قبل أو بعد العلامة - يقوم بانقاص المتغير ب 1
12	+=	Variable name+=… data ……	- يقوم بزيادة المتغير بقيمة
13	-=	Variable name-=…… data …	- يقوم بانقاص المتغير بقيمة
14	*=	Variable name*=… data ……	- يقوم بضرب المتغير بقيمة
15	/=	Variable name/=…… data …	- يقوم بقسمة المتغير بقيمة
16	<	… data … <… data ….	- مقارنات بين الارقام (اصغر من)
17	>	… data … >… data ….	- مقارنات بين الارقام (اكبر من)
18	==	… data … ==… data ….	- مقارنات بين الارقام (يساوي)
19	>=	… data … >=… data ….	- مقارنات بين الارقام (اكبر من أو يساوي)
20	<=	… data … <=… data ….	- مقارنات بين الارقام (اصغر من أو يساوي)
21	||	… Comparison data ….||… Comparison data …	- في حالة وجود أكثر من حالة مقارنة (بمعنى أو)
22	&&	Comparison data ….&&… Comparison data …	- في حالة وجود أكثر من حالة مقارنة (بمعنى و)

- post decrement Or Post increment   مثال   n++   n-- يعني انه يسطبع n ثم سيضيف عليها أو ينقص 1
- pre decrement Or Pre increment   مثال   ++n   --n يعني انه سيضيف أو ينقص 1 ثم سيطبع الرقم

- Mean that go to the memory and add this to n and finally print it after that

- Mean that go to the memory and print n then finally add to n

- They are not defiance between Prefix or postfix when we write
- ++X: or   X++;   because it’s not in equation here

Following examples provides some specific C expressions:

C Expression	Meaning
`X = X + 9;`	Calculate X+9 then stores the result in X
`X++;`	Add one to X
`X--;`	Subtract one from X

`X = 10;`
`Y = 5;`
`X = X + Y++;`
	Add X+Y → 15
	then increment Y → 6
	Store 15 in X

`X = 10;`
`Y = 5;`
`X = X + ++Y;`
	Increment Y → 6
	Add X+Y → 16
	Store 16 in X

`unsigned char X = 0xA4;`
`unsigned char Y = 156;`
`unsigned char Z = X|Y;`
	Calculate the X OR Y → 0xBC (188)

`unsigned char X = 0xA4;`
`unsigned char Y = 156;`
`unsigned char Z = X&Y;`
	Calculate the X AND Y → 0x84 (132)

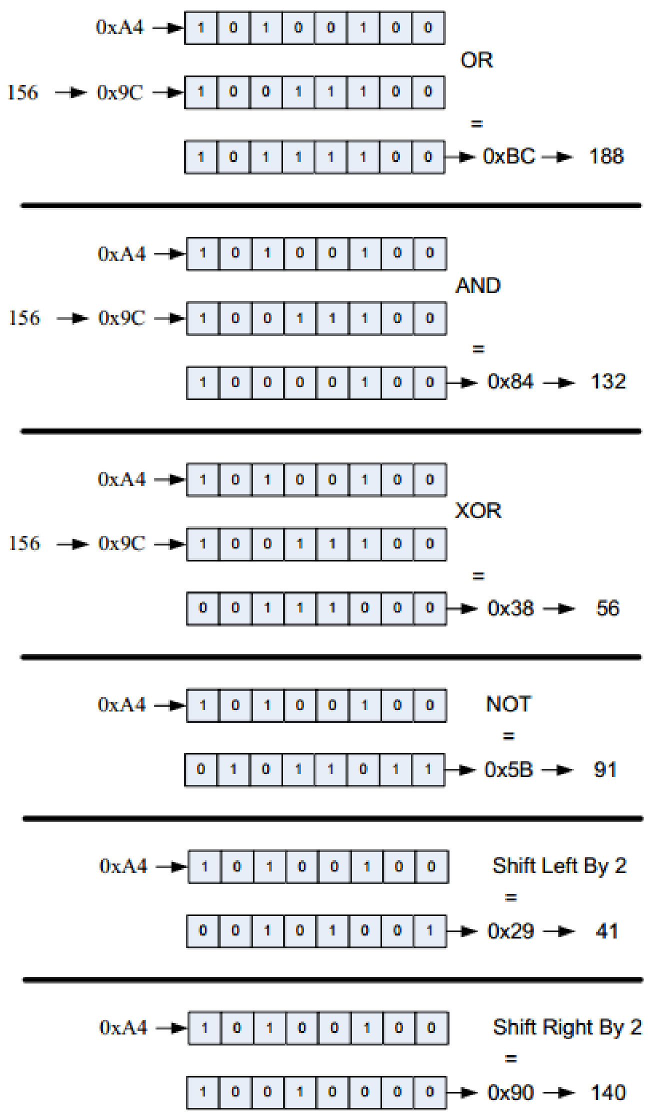
`unsigned char X = 0xA4;`
`unsigned char Y = 156;`
`unsigned char Z = X^Y;`
	Calculate the X XOR Y → 0x38 (56)

`unsigned char X = 0xA4;`
`unsigned char Z = ~X;`
	Calculate the (NOT X) → 0x5B (91)

`unsigned char X = 0xA4;`
`unsigned char Z = X>>2;`
	Calculate the X shifted by two bits to right → 0x29 (41)

`unsigned char X = 0xA4;`
`unsigned char Z = X<<2;`
	Calculate the X shifted by two bits to right → 0x90 (140)

C Expression	Meaning
`X = X + Y * Z;`	Multiply Y by Z then Add X
`X = (X + Y) * Z;`	Add X to Y then multiply by Z



- XOR (^) when the 2 numbers are defiance = 1
- When the 2 numbers are the same = 0

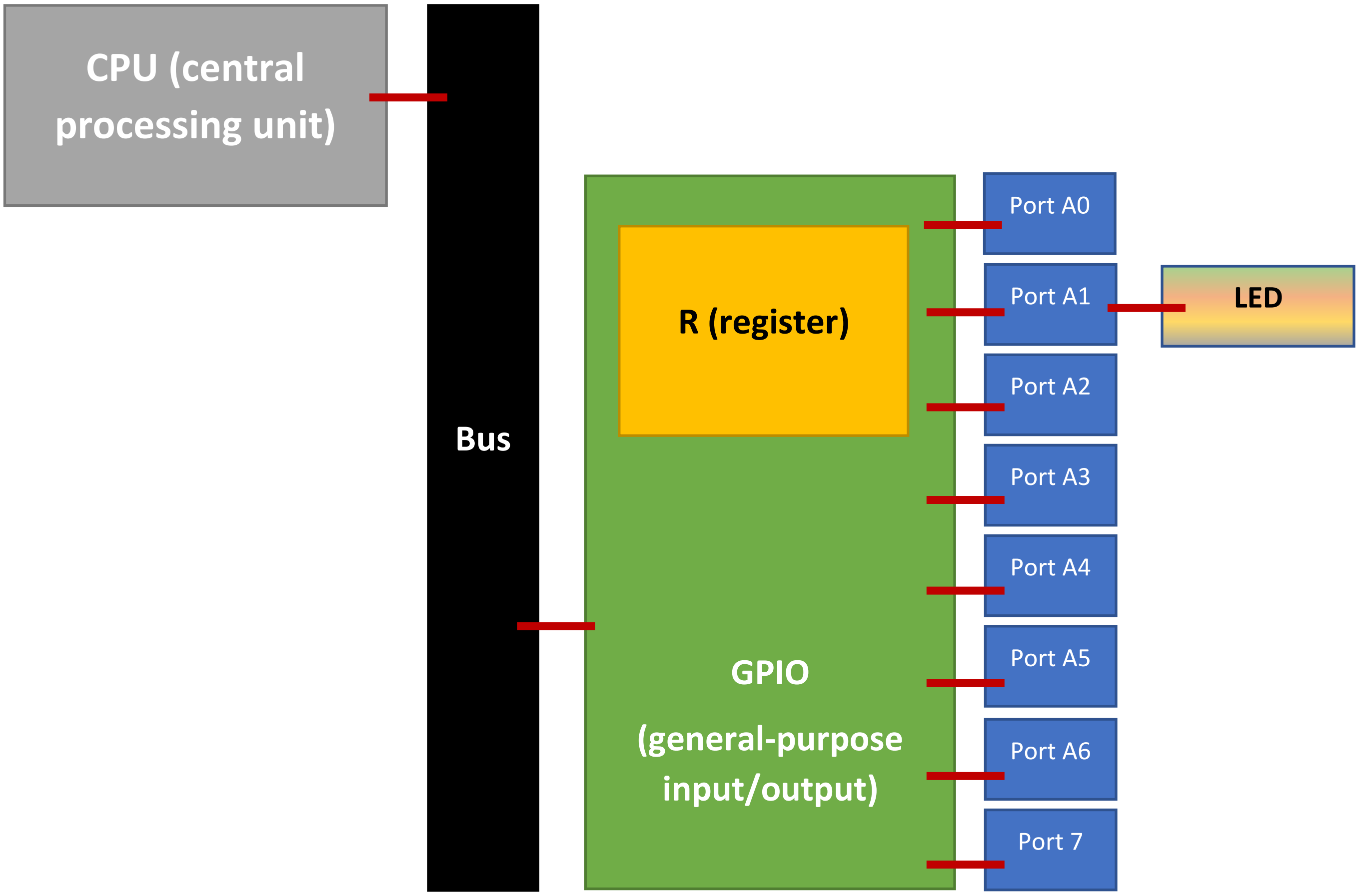
- NOT (~) change 0 to 1 and 1 to 0

- AND (&) When its deference it will = 0
- When it's the same it will if 1 wit will = 1 if 0 it will = 0

- OR (|) make an add if 1 with 0 it wills = 1
- If 1 with 1 it wills = 1
- If 0 with 0 it wills = 0

- Shift left (<<) = *2 that's mean if 0010 it will be 1000

- Shift Right (>>) = /2 that's mean if 0100 it will be 0001

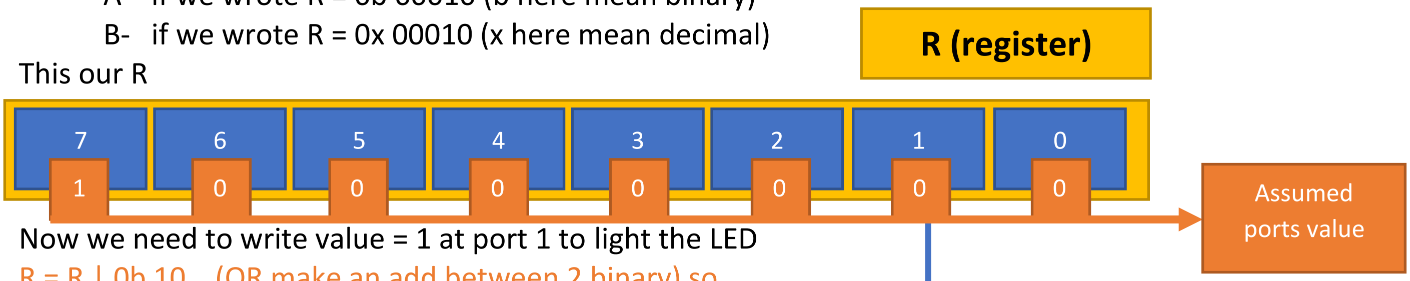


- Now we need to write a programme talking to the R (register).
- And in the specifications of R (register) told that:
  1- The R has 7 bits
  2- If you want to talk at any port on it give it value = 1
- Assume R is variable unsigned char

  Note:

  A- if we wrote R = 0b 00010 (b here mean binary)

  B- if we wrote R = 0x 00010 (x here mean decimal)
- This our R



- Now we need to write value = 1 at port 1 to light the LED
- R = R | 0b 10 (OR make an add between 2 binary) so
- Now R = 1000 0000 when we call address 0b 10 we call Port A1
- And | mean OR so R = 1000 0000 + 0000 0010 = 1000 0010

  Another solution:
- R |= 1 << n (that's mean R = R | 1 << n) (when n = the number of bit you Want to change it)

Note:

A- Shift left << that mean *2

B- Shift right that's mean /2

- We will put n = 1 because we need to shift it by 1 bit
- 1 = 0000 0001 when we shift it left it will be 0010
- So, we will assume in equation R |= 1000 0000 (OR) 0000 0010
- Then R 1000 0010

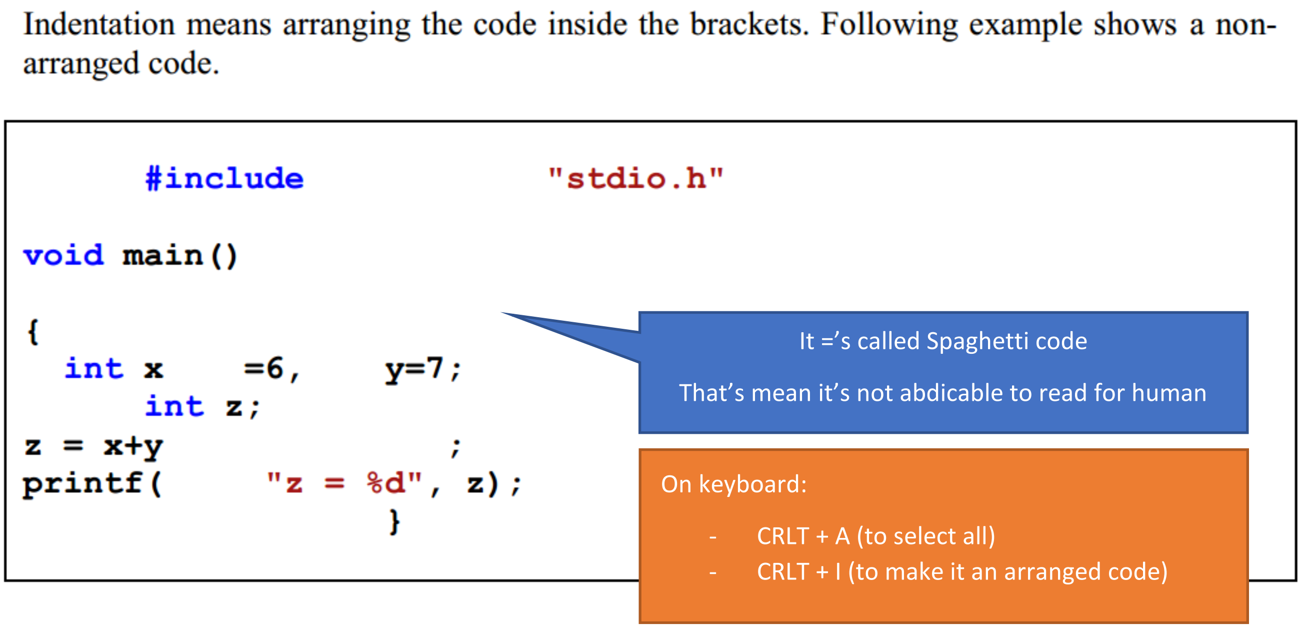
~ is a bit wise operator mean not and its convert the binary bit

Ex (0000 0010 it will be 1111 1101)

Finally

1- **If we want to make any bit value in R = 1 we can use this equation (R |= R << n) (set)**
2- **If we want to make any bit value in R = 0 we can use this equation (R &= ~ (1 << n)) (Clear)**
3- **If we want to toggle (make 0 =1 and 1 =0) we can use this equation (R ^= 1 << n) (Toggle)**

Coding Convention



Indentation means arranging the code inside the brackets. Following example shows a non-arranged code.

```
      #include            "stdio.h"

void main()

{
  int x     =6,     y=7;
      int z;
z = x+y                     ;
printf(      "z = %d", z);
                      }
```

It ='s called Spaghetti code

That's mean it's not abdicable to read for human

On keyboard:

- CRLT + A (to select all)
- CRLT + I (to make it an arranged code)

Identifiers

- Identifiers are the names that are given to various program elements such as variables, symbolic constants and functions.
- Identifier can be freely named, the following restrictions.
  - Alphanumeric characters (a ~ z, A～Z, 0～9) and half underscore (_) can only be used.
  - The first character of the first contain letters (a ~ z, A～Z) or half underscore (_) can only be used.

Here are the rules you need to know:

- Identifier name must be a sequence of letter and digits, and must begin with a letter.
- The underscore character ('_') is considered as letter.
- Names shouldn't be a keyword(such as int, float, if ,break, for etc)

- Both upper-case letter and lower-case letter characters are allowed. However, they're not interchangeable.
- No identifier may be keyword.
- No special characters, such as semicolon, period, blank-space, slash or comma are permitted

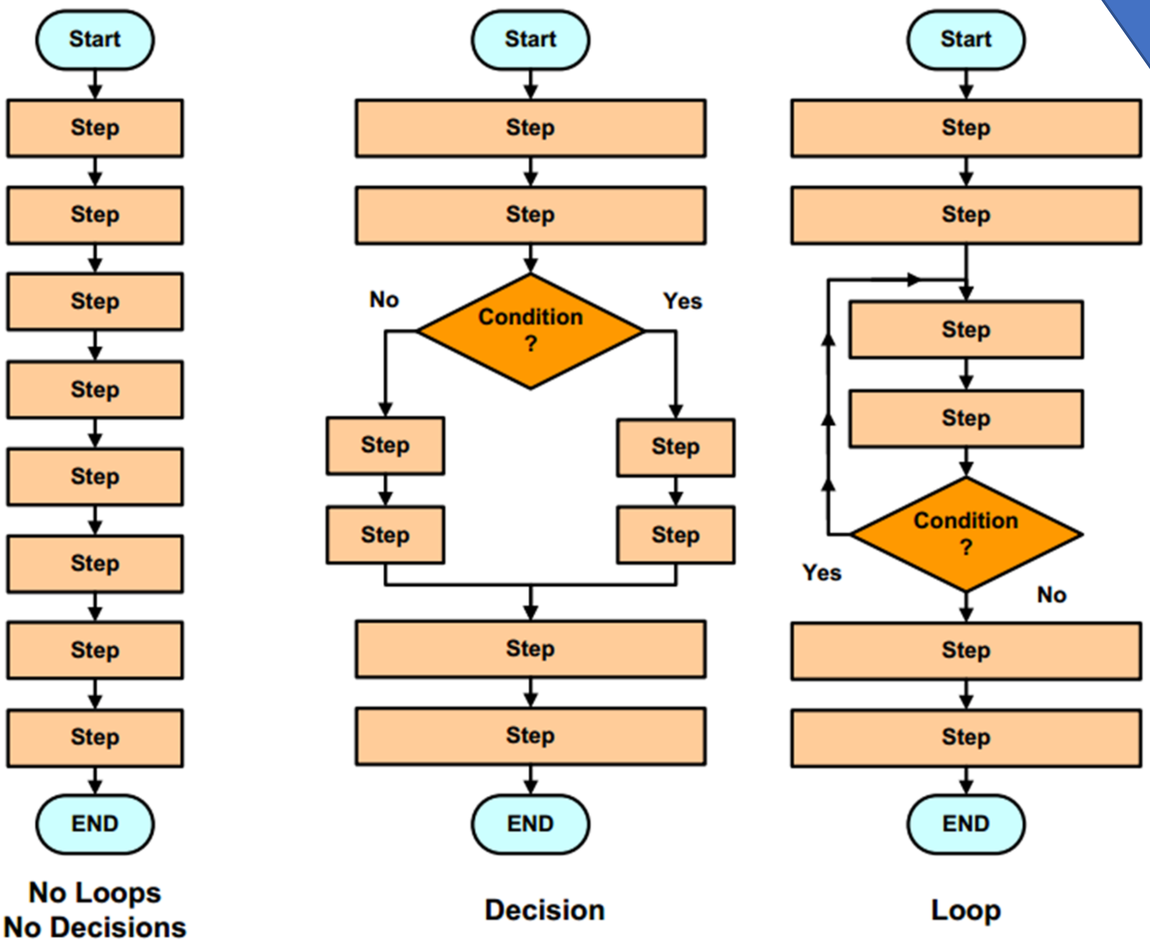
Examples of legal and illegal identifiers follow, first some legal identifiers:

- float _number;
- float a;

Keywords

- Keywords are standard identifiers that have standard predefined meaning in C.
- Keywords are all lowercase, since uppercase and lowercase characters are not equivalent it's possible to utilize an uppercase keyword as an identifier but it's not a good programming practice.
- Keywords can be used only for their intended purpose.
- Keywords can't be used as programmer defined identifier.
- The keywords can't be used as names for variables.

Controlling Program Flow



No Loops
No Decisions

Decision

Loop

That we use in Embedded C but without ending (infinite loop)

Because we program something we need it to work always on as long as there is an electricity on board

Conditions

- Conditions always = false or true
- True = 1 or -1
- False = 0

Operator	Meaning
>	Greater
>=	Greater or equal
<	Less
<=	Less than or equal
==	Equal
!=	Not equal
!	Not If the input is true the output is false if the input is false the output is true
&&	And Example: A>B && C>D If both sides are true the output is true, otherwise it gives false
||	Or Example: A>B || C>D If wither sides is true the output is true, otherwise it gives false

Example: Using Conditions

-

```
-   #include "stdio.h"
-   #include "math.h"
-   void main()
-   {
-   inta = 9;intb = 8;intc = 12;
-   printf("%d\r\n", a>b); //prints 1
-   printf("%d\r\n", b>c); //prints 0
-   printf("%d\r\n", a<=9); //prints 1
-   printf("%d\r\n", a!=9); //prints 0
-   printf("%d\r\n", (a-b)>(c-b)); //prints 0
-   printf("%d\r\n", a>b && c>b); //prints 1
-   printf("%d\r\n", a>b && c<b); //prints 0
-   printf("%d\r\n", a>b | | c<b); //prints 1
-   printf("%d\r\n", !(a<b)); //prints 1
-   printf("%d\r\n", 3 && 0); //prints 0
-   printf("%d\r\n", -15 | | 0); //prints 1
-   printf("%d\r\n", !(-15)); //prints 0
-   }
```

أنواع ال Errors

Syntax Error	**مثل نسيان ; او) او ...... أي خطأ في كتاب الكود (يمكن معرفته عن طريق help50)**
Logical Error	**الخطأ الذي يظهر في ناتج التشغيل (عمل كود يجمع 2+2 يجب ان يكون الناتج 4 لكن يعطي ناتج 5)**
Compiling Error	

- Syntax bug يعمل على Syntax code اي قبل عمل compiling للملف لذلك يتم استدعاؤه مباشرة

IF Statement Syntax

```
-   #include "stdio.h"
-   #include "math.h"
-   void main ()
-   {
-   if (/*if condition*/)
-   {
-   //if body
-   }
-   else if (/*else if condition*/)
-   {
-   //else if body
-   }
-   else if (/*else if condition*/)
-   {
-   //else if body
-   } else
-   {
-   //else body
-   }
-   }
```

Line Conditions

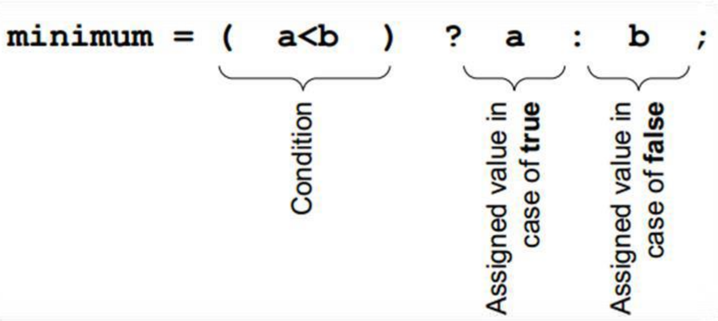
- Because it's wrote in one line

```
min = (x < y) ? x : y;
Identifier = (test expression)? Expression1: Expression2 ;
```

Name of function	(	Boolean Condition	)	?	Result if condition true	:	Result if condition False

Equation parameters

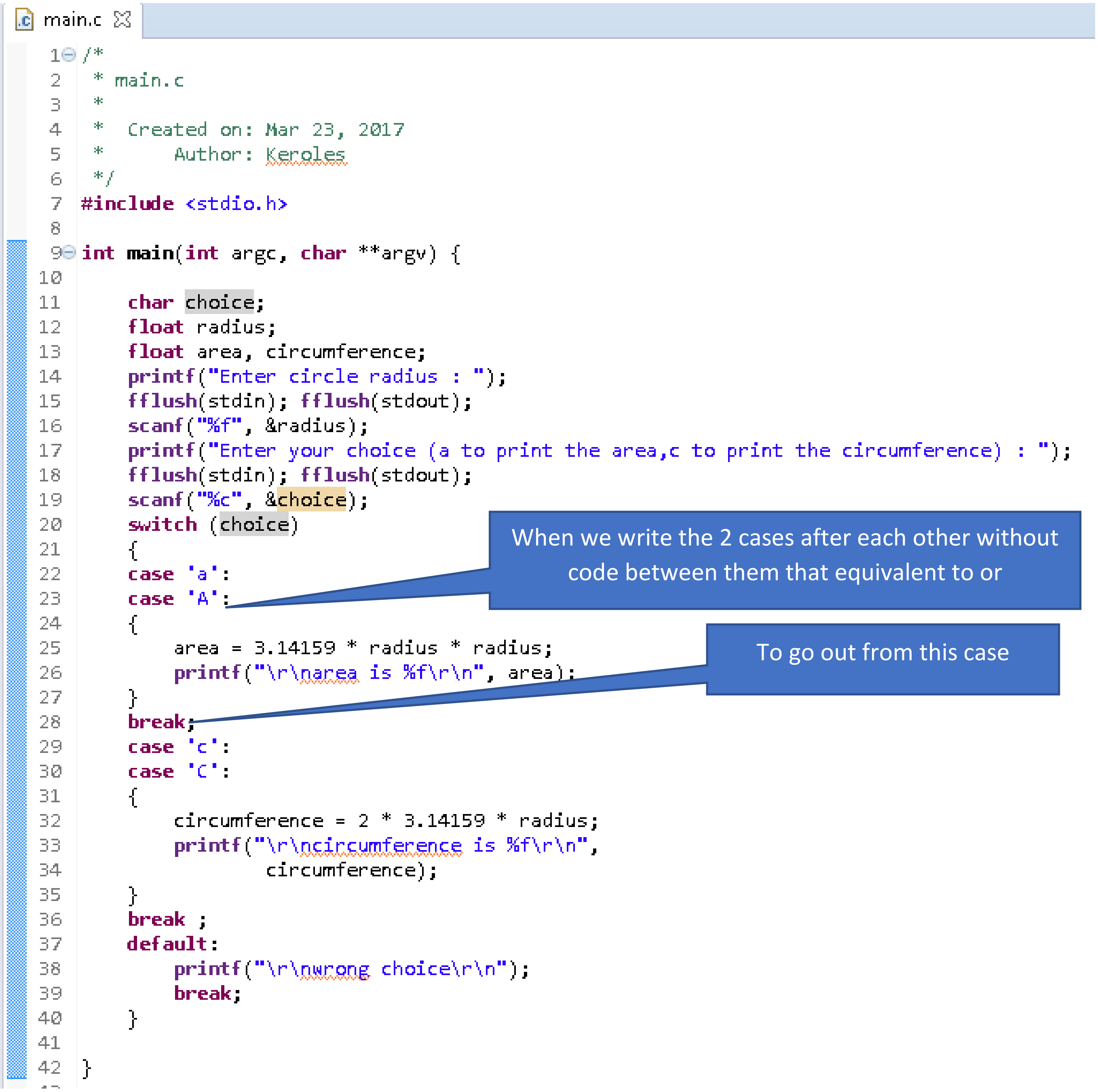
```
#include "stdio.h"

void main()
{
        int a, b, minimum;
        printf("Enter tow numbers : ");
        scanf("%d %d", &a, &b);
        minimum = (a<b)?a:b;
        printf("The minimum is %d\r\n", minimum);
}
```


minimum = (a<b) ? a : b ;
Condition
Assigned value in case of true
Assigned value in case of false

Switch Statement

- #include "stdio.h"
- #include "math.h"
- void main ()
- {
    - switch (/*switch expression*/)
    - {
    - case /*case value*/:
    - {
    - //case body
    - }
    - break;
    - .....
    - ....
    - …….
    - case /* case value*/:
    - {
    - //case body
    - }
    - break;
    - default:
    - {
    - }
    - break;
    - }
    - }

- Switch statement faster more if statement because it working base on (lookup table) but if statement working by conditions if it not happened go to another condition…. etc.
- Switch cases should be integer constants.
- Switch expression simple but in if statement can do more complex expressions.
- Switch statement tack only integer value because the compiler looking at value at switch statement in table of cases so the values should be constants.



```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

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

    char choice;
    float radius;
    float area, circumference;
    printf("Enter circle radius : ");
    fflush(stdin); fflush(stdout);
    scanf("%f", &radius);
    printf("Enter your choice (a to print the area,c to print the circumference) : ");
    fflush(stdin); fflush(stdout);
    scanf("%c", &choice);
    switch (choice)
    {
    case 'a':
    case 'A':
    {
        area = 3.14159 * radius * radius;
        printf("\r\narea is %f\r\n", area);
    }
    break;
    case 'c':
    case 'C':
    {
        circumference = 2 * 3.14159 * radius;
        printf("\r\ncircumference is %f\r\n",
                circumference);
    }
    break ;
    default:
        printf("\r\nwrong choice\r\n");
        break;
    }

}
```

When we write the 2 cases after each other without code between them that equivalent to or

To go out from this case

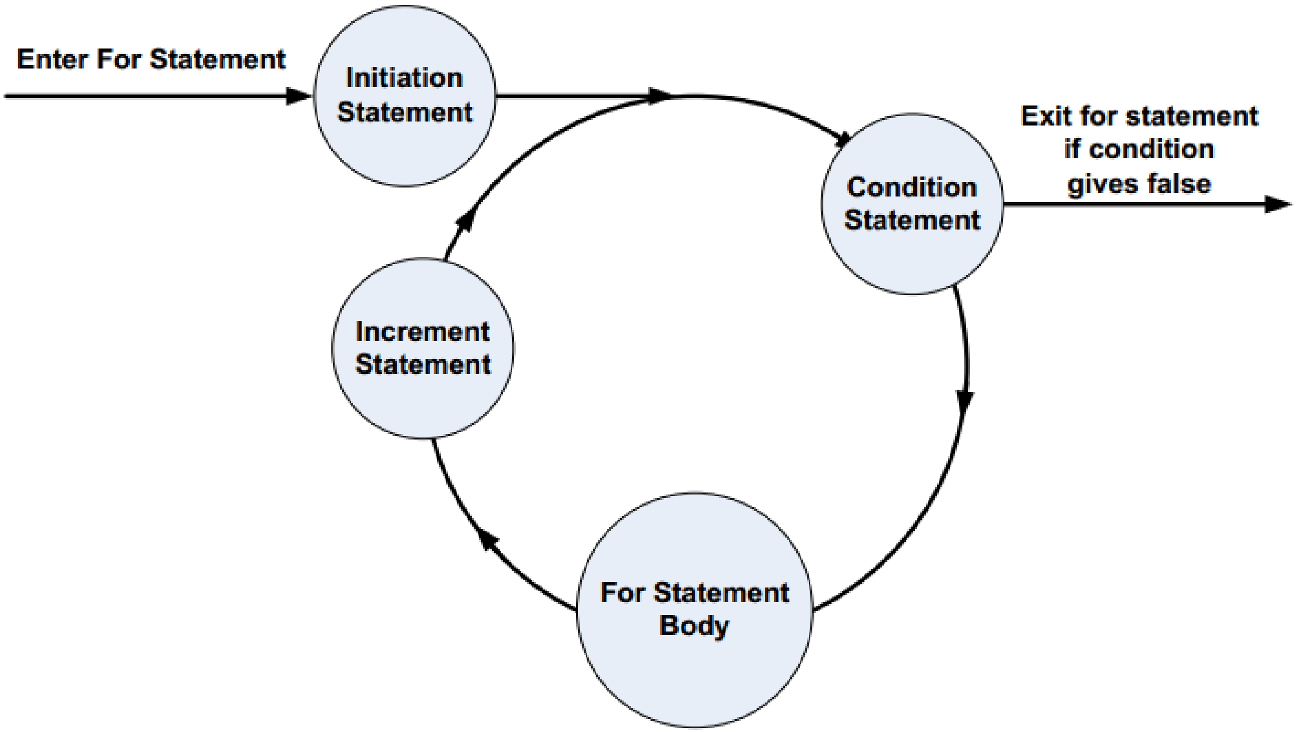
For Statement

Syntax:

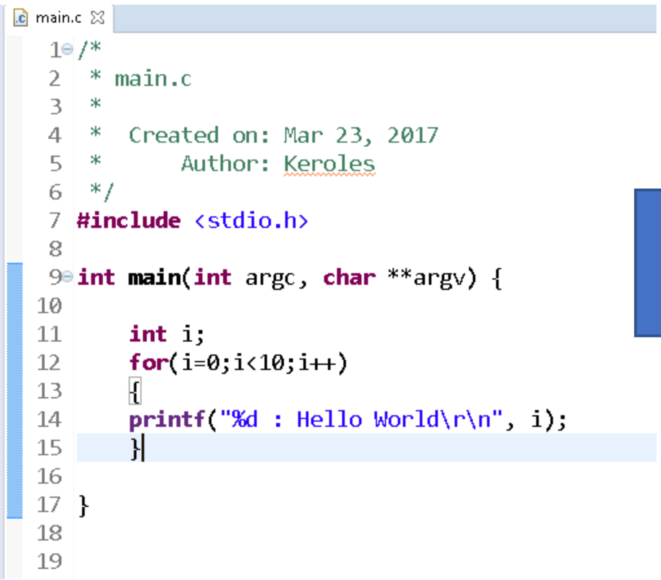
```
for(/*intiation*/;/*condition*/;/*increment*/)
{
        //for body
}
```

for statement repeats the execution of the (**for body**) until the (**condition**) statement is not succeeded any more. Computer processes **for** statement as follows:

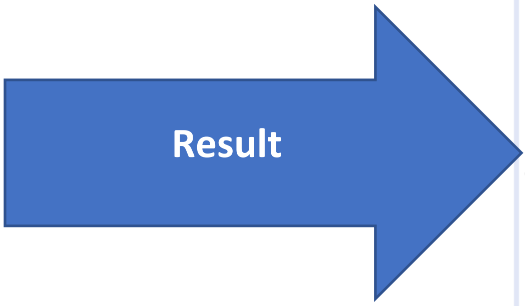
1. Execute the (**initiation**) statement to assign an initiate value to some variable if it is required.
2. Execute the (**condition**) statement, if false, go out of the for statement, otherwise, proceed to the next step
3. Execute the (**for body**)
4. Execute the (**increment**) statement to update some variables
5. Go back to (Step 2)

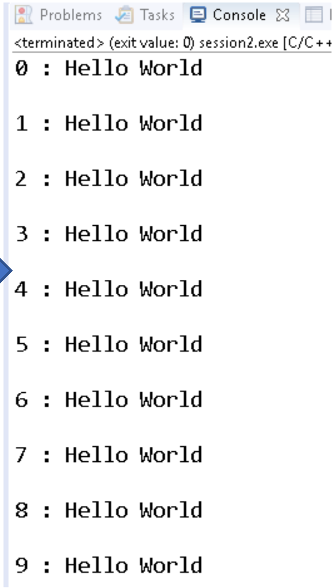


EX



```
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

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

    int i;
    for(i=0;i<10;i++)
    {
    printf("%d : Hello World\r\n", i);
    }

}
```



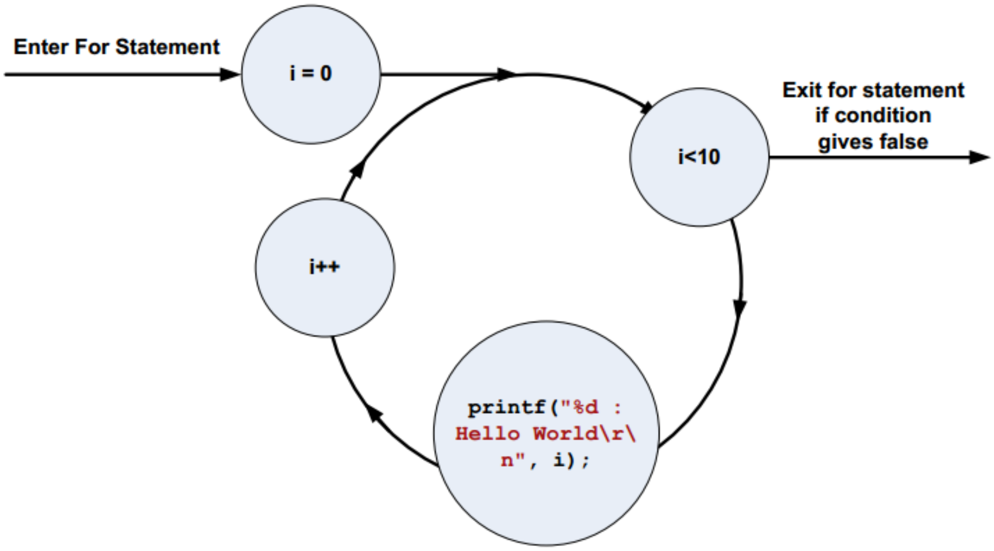


```
<terminated> (exit value: 0) session2.exe [C/C++
0 : Hello World

1 : Hello World

2 : Hello World

3 : Hello World

4 : Hello World

5 : Hello World

6 : Hello World

7 : Hello World

8 : Hello World

9 : Hello World
```

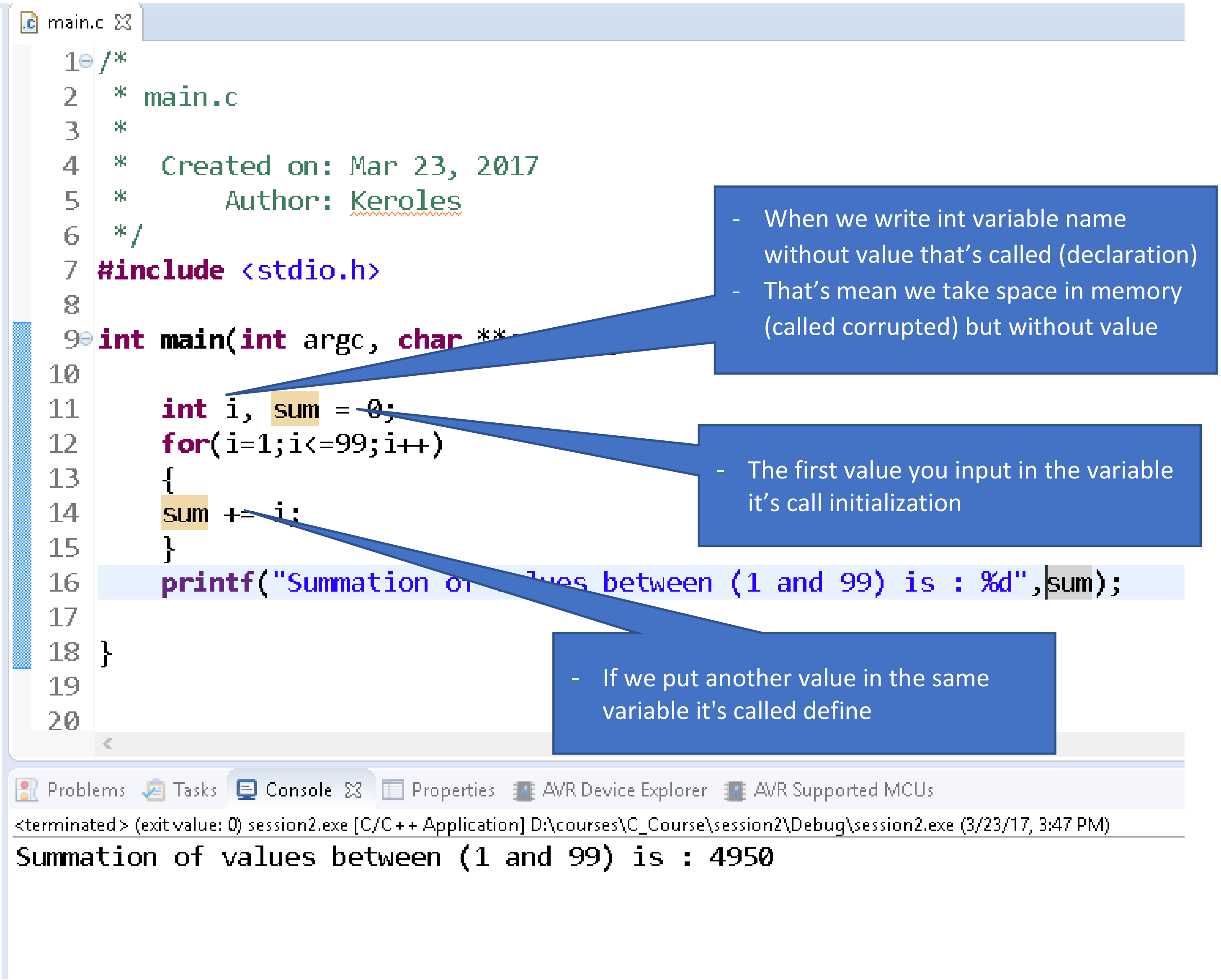
Above example prints “Hello World” 10 times. The program works as following:

1. Execute the initiation statement (i=0)
2. Execute the condition statement (i<10), if false, exit from the for statement
3. Execute the body of the for statement
4. Execute the increment statement (i++)
5. Go back to (Step 2)

Initially (i) is loaded with (0), after each loop it is incremented by (1). If (i) value reaches (10) the condition statement fails and the computer exits from the loop. It is clear that (i) variable takes the values {0,1,2,3,4,5,6,7,8,9} during the execution.



- In C there is an non-statement write (;) when we writ this non-statement that mean do nothing so when we writ the for loop like this (for (…..;…..;….) **;** {……}) that's mean the for loop do nothing because of (;).



```
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

int main(int argc, char **

    int i, sum = 0;
    for(i=1;i<=99;i++)
    {
    sum += i;
    }
    printf("Summation of values between (1 and 99) is : %d",sum);

}
```

- When we write int variable name without value that's called (declaration)
- That's mean we take space in memory (called corrupted) but without value

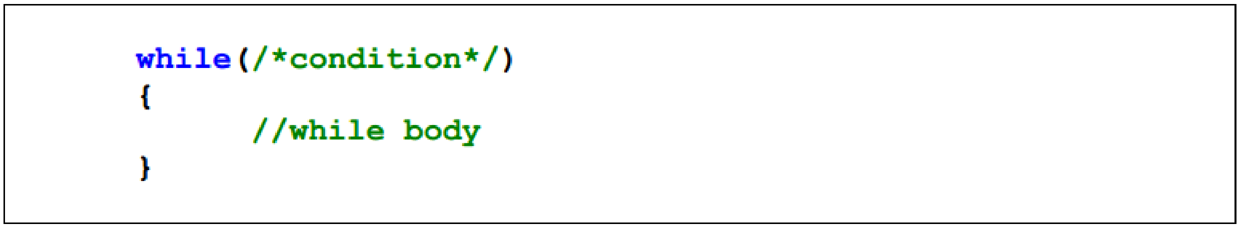
- The first value you input in the variable it's call initialization

- If we put another value in the same variable it's called define

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Debug\session2.exe (3/23/17, 3:47 PM)
Summation of values between (1 and 99) is : 4950
```

while Statement

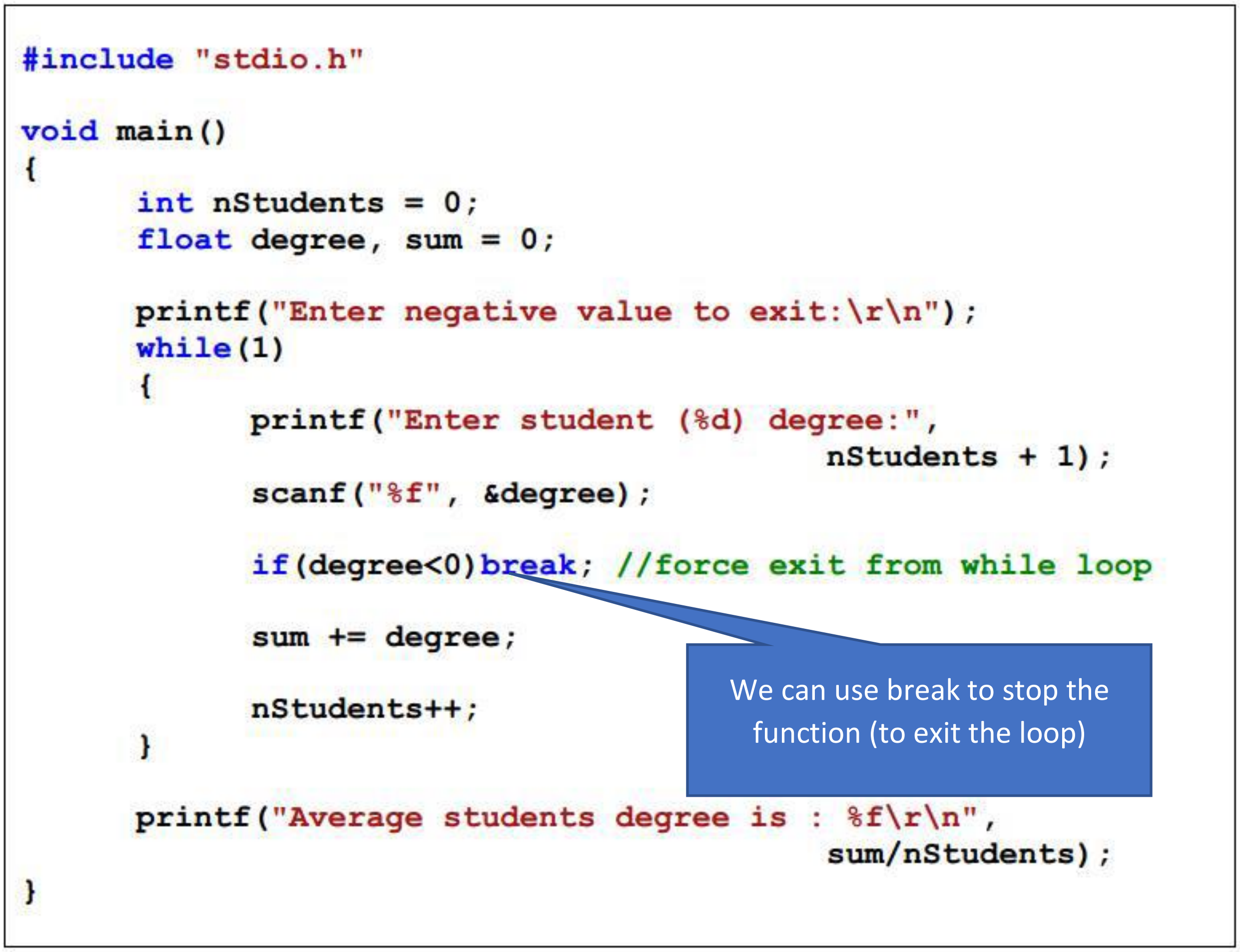
Syntax:



```
while(/*condition*/)
{
        //while body
}
```

while statement is similar to the **for** statement, however it is more simple, there is no initiation or increment statements, you have to choose where to initiate and where to increment your variables if you need this. The computer executes the while statement as follows:

1. Execute the (**condition**) statement, if false, go out of the **while** statement, otherwise, proceed to the next step
2. Execute the (**while body**)
3. Repeat (Step 1)



```
#include "stdio.h"

void main()
{
        int nStudents = 0;
        float degree, sum = 0;

        printf("Enter negative value to exit:\r\n");
        while(1)
        {
                printf("Enter student (%d) degree:",
                                                nStudents + 1);
                scanf("%f", &degree);

                if(degree<0)break; //force exit from while loop

                sum += degree;

                nStudents++;
        }

        printf("Average students degree is : %f\r\n",
                                                sum/nStudents);
}
```

We can use break to stop the function (to exit the loop)

do…while Statement

Syntax:

```
do
{
        //do...while body
}
while(/*condition*/);
```

Make the code one time then loop

do ... while statement is similar to while statement, except that the condition is checked after executing each loop, which means that, the first loop is performed without a check. The computer executes the while statement as follows:

1. Execute the (**do…while body**)
2. Execute the (**condition**) statement, if false, go out of the **do…while** statement, otherwise go to (Step 1)

```
#include "stdio.h"
#include "conio.h"

void main()
{
        float x, y;

        do
        {
                printf("\r\nEnter x value:");
                scanf("%f", &x);
                y = 5*x*x + 3*x + 2;
                printf("\r\ny(%f) = %f", x, y);

                printf("\r\ndo you want to evaluate
                                              again (y/n):");
        }
        while(getche()=='y');
}
```

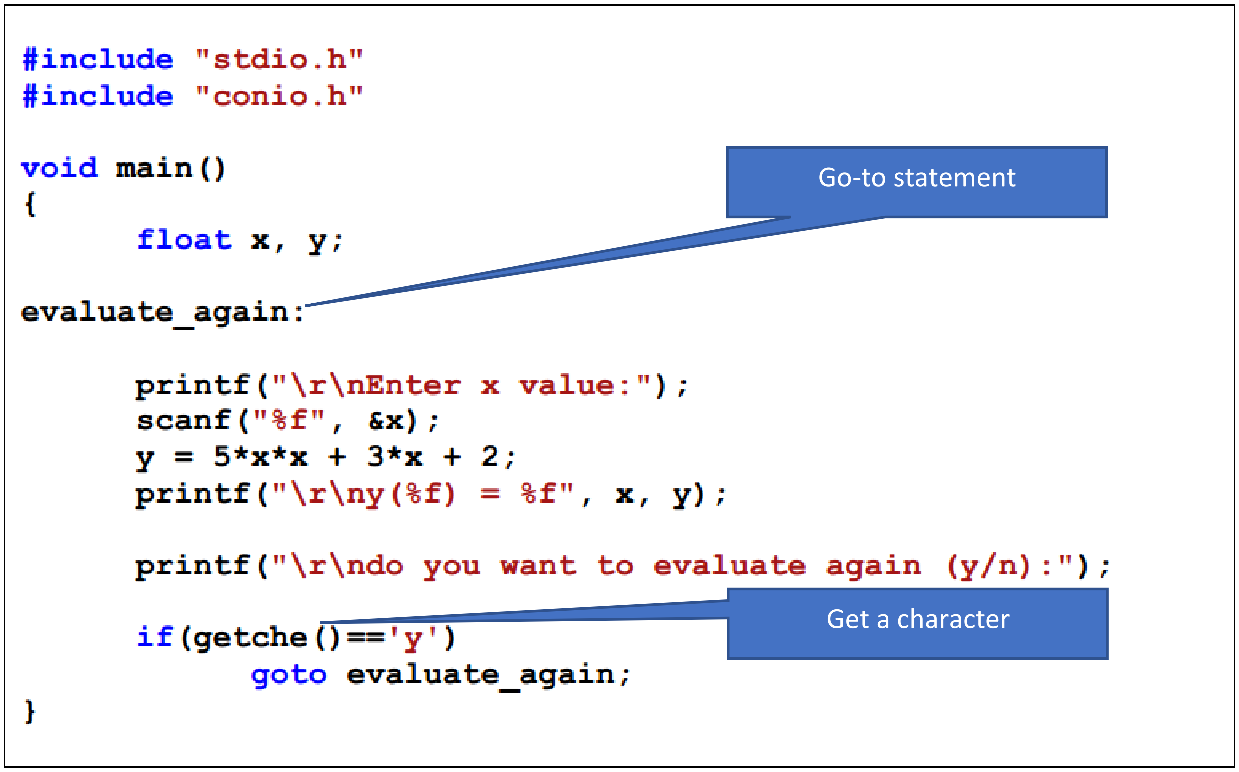
Go-to Statement

Syntax:

```
      // C Statment
labelname:
      // C Statment
      // C Statment
      goto labelname;
      // C Statement
```

```
      // C Statment
      goto labelname;
      // C Statment
      // C Statment
labelname:
      // C Statment
```

Simply **goto** statement tells the program where to jumps, it can jump forward or backward. Following example illustrates the idea.

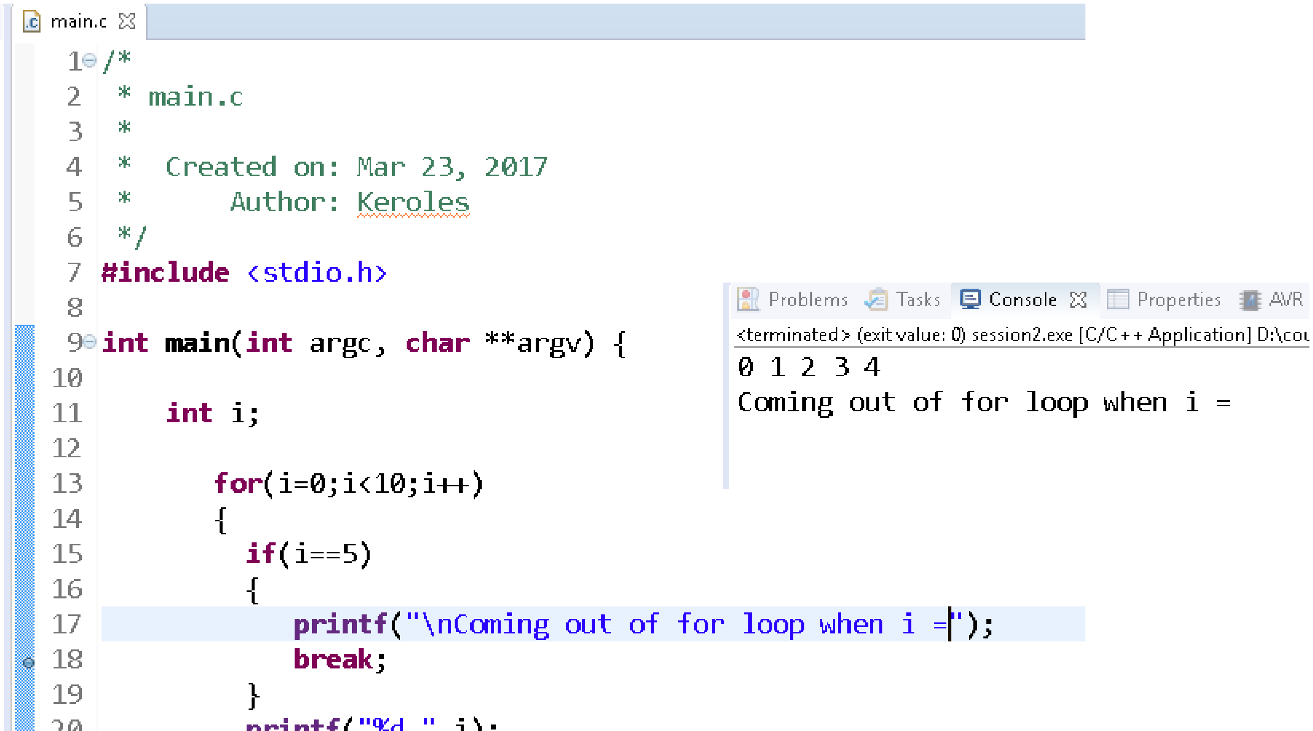


```
#include "stdio.h"
#include "conio.h"

void main()
{
      float x, y;

evaluate_again:

      printf("\r\nEnter x value:");
      scanf("%f", &x);
      y = 5*x*x + 3*x + 2;
      printf("\r\ny(%f) = %f", x, y);

      printf("\r\ndo you want to evaluate again (y/n):");

      if(getche()=='y')
            goto evaluate_again;
}
```

Important: It is **not recommended** to use **goto** statement extensively, because it allows programmers to jump anywhere in their program and this lead to unorganized and unreadable codes.

Break Statement

- The break statement is a jump instruction and can be used inside a switch construct, for loop, while loop and do-while loop.
- The execution of break statement causes immediate exit from the concern construct and the control is transferred to the statement following the loop.

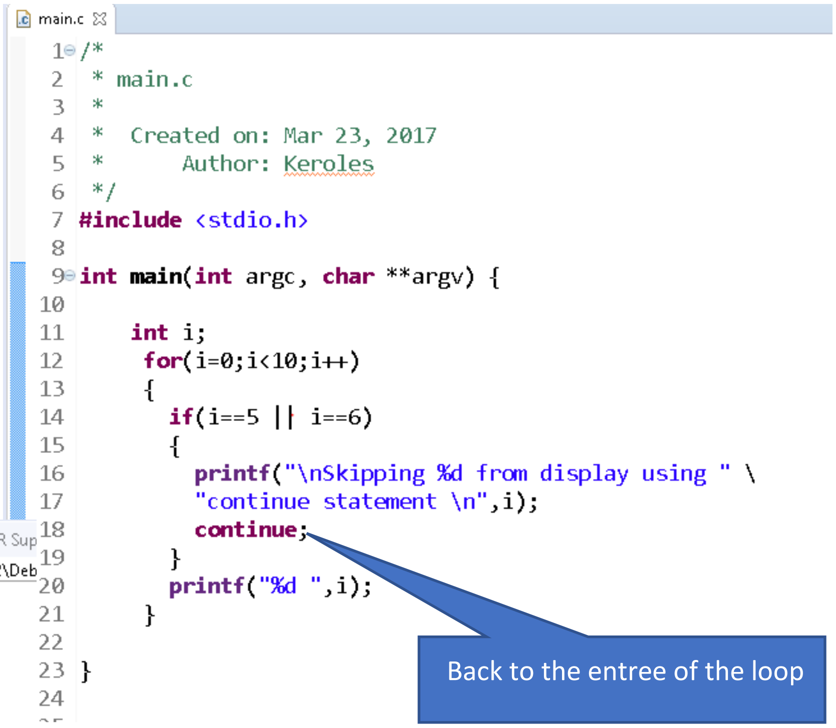


```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

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

    int i;

        for(i=0;i<10;i++)
        {
          if(i==5)
          {
              printf("\nComing out of for loop when i =");
              break;
          }
          printf("%d ",i);
        }

}
```

```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\cou
0 1 2 3 4
Coming out of for loop when i =
```

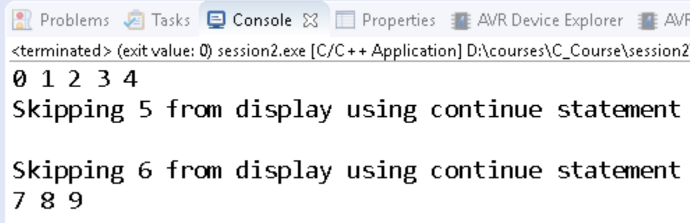
Continue statement

- Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.
- Syntax: continue;



```c
/*
 * main.c
 *
 *  Created on: Mar 23, 2017
 *      Author: Keroles
 */
#include <stdio.h>

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

    int i;
     for(i=0;i<10;i++)
     {
       if(i==5 || i==6)
       {
         printf("\nSkipping %d from display using " \
         "continue statement \n",i);
         continue;
       }
       printf("%d ",i);
     }

}
```

Back to the entree of the loop



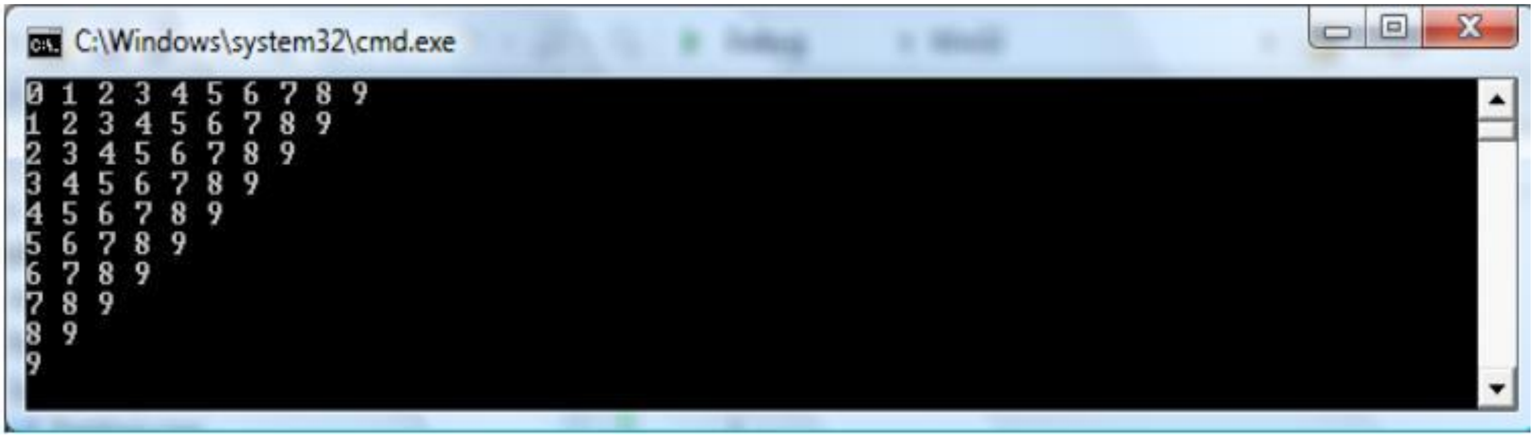
```
<terminated> (exit value: 0) session2.exe [C/C++ Application] D:\courses\C_Course\session2\Deb
0 1 2 3 4
Skipping 5 from display using continue statement

Skipping 6 from display using continue statement
7 8 9
```

Nested Loop

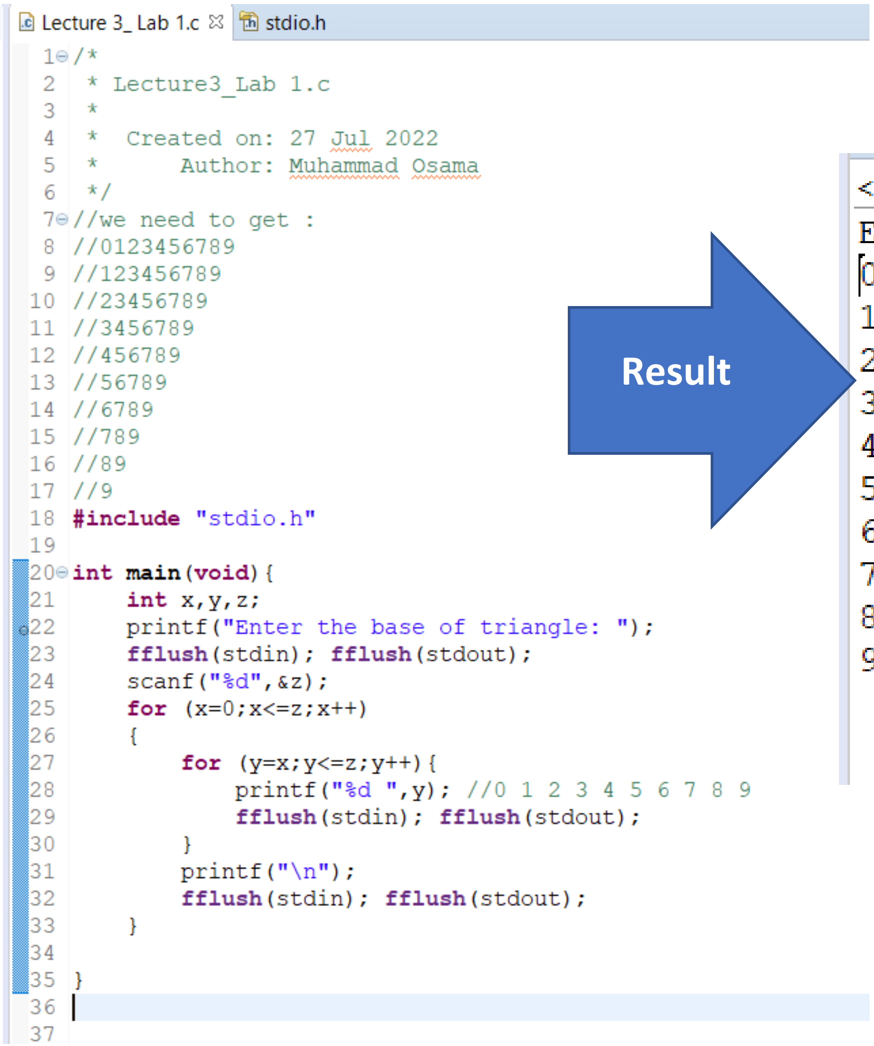
- Loop inside loop.

EX

- We need to get this result.

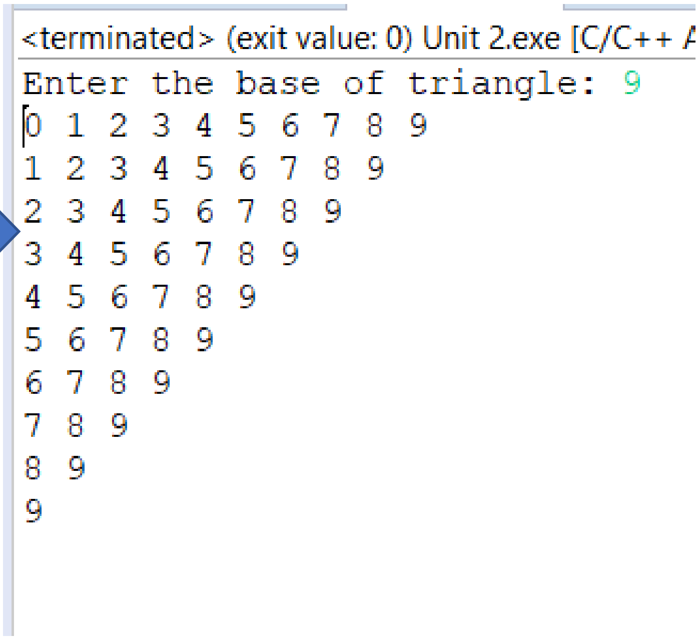


```
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
4 5 6 7 8 9
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
```



```c
/*
 * Lecture3_Lab 1.c
 *
 *  Created on: 27 Jul 2022
 *      Author: Muhammad Osama
 */
//we need to get :
//0123456789
//123456789
//23456789
//3456789
//456789
//56789
//6789
//789
//89
//9
#include "stdio.h"

int main(void){
    int x,y,z;
    printf("Enter the base of triangle: ");
    fflush(stdin); fflush(stdout);
    scanf("%d",&z);
    for (x=0;x<=z;x++)
    {
        for (y=x;y<=z;y++){
            printf("%d ",y); //0 1 2 3 4 5 6 7 8 9
            fflush(stdin); fflush(stdout);
        }
        printf("\n");
        fflush(stdin); fflush(stdout);
    }

}
```

Result



```
<terminated> (exit value: 0) Unit 2.exe [C/C++ A
Enter the base of triangle: 9
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
4 5 6 7 8 9
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
```