

UNIT-I Part- 1

INTRODUCTION ABOUT C PROGRAMMING

- C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie at Bell Labs, USA in 1972
- It was actually B-language that was developed by Ken Thompson and was improved and transformed into C language by Dennis Ritchie
- One of the flexible, efficient and widely used languages in the hierarchy of the programming languages
- Unix operating system was developed in C. In fact C was introduced for building the Unix operating system
- C has now become a widely used professional language for various reasons:-
 1. Major part of the web browsers are written in C e.g. Mozilla, Firefox, Google Chrome etc.
 2. Core libraries of Android are also written in C language
 3. Device drivers are mostly written in C language
 4. Some of the popular and widely used database management software's are written in C (e.g. Oracle, MySQL)
 5. It is an easy-to-learn structured language and produces efficient programs
 6. C can handle low-level activities and can be compiled on a variety of computer platforms

C program structure

A C program basically consists of the following parts: –

Preprocessor Commands, Functions, Variables, Statements & Expressions, Comments.

Lets take a simple example of C program that prints the message “welcome to the world of ‘C’ programming”

Example 1: Displaying Welcome to the World of C programming

```
#include <stdio.h>
main()
{
    /* printf function displays the content that is
    * passed between the double quotes.
    */
    printf("welcome to the World of C programming");
}
```

Output:

Welcome to the world of C programming

1. **#include <stdio.h>** – This statement tells compiler to include this stdio.h file in the program. This is a standard input output file that contains the definitions of common input output functions such as scanf() and printf(). In the above program we are using printf() function.

2. **main()** – Here main() is the function name. Every C program must have this function because the execution of program begins with the main() function.

Every C program begins with a pre-processor directive i.e. #include<stdio.h>

Let us have a look at the various parts of the above program

Main() is the function and execution of the program begins from main

C program code is enclosed within opening and closing curly braces({ , })

Printf() is also a function that is responsible to display the message on the screen

(;) semicolon shows the termination(i.e end) of the statement as well as the program

- i.)The first line of the program #include <stdio.h> is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- ii.)The next line main() is the main function where the program execution begins.
- lii.)The next line printf(...) is another function available in C which causes the message "welcome to the world of c programming!" to be displayed on the screen.That means its an output function

What IS A COMPILER

A **compiler** is a **computer program** that **translates** computer code written in one **programming language** (the source language) into another language (the target language). The name *compiler* is primarily used for programs that translate **source code** from a **high-level programming language** to a **lower level language** (e.g., assembly language, object code, or machine code) to create an **executable program**. Compilers implement different operations in phases that promote efficient design and correct transformations of **source input** to **target output**

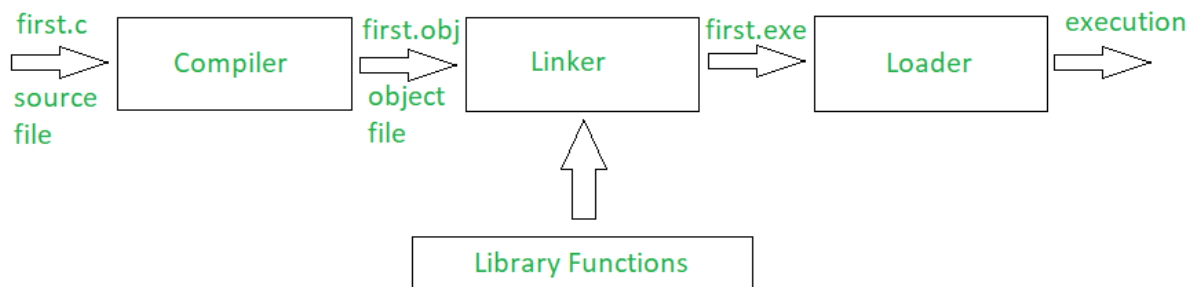
What do you mean by Execution

Execution in computer terminology is the process by which a computer executes the instructions of a computer program. Each instruction of a program is a description of a particular action which to be carried out in order for a specific problem to be solved; as instructions of a program and therefore the actions they describe are being carried out by an executing machine, specific effects are produced in accordance to the semantics of the instructions being executed.

Compilation and Execution of a 'c' program

Whenever a C program file is compiled and executed, the compiler generates some files with the same name as that of the C program file but with different extensions. So, what are these files and how are they created?

Below image shows the compilation process with the files created at each step of the compilation process:



Every file that contains a C program must be saved with '.c' extension. This is necessary for the compiler to understand that this is a C program file. Suppose a program file is named, first.c. The file first.c is called the source file which keeps the code of the program. Now, when we compile the file, the C compiler looks for errors. If the C compiler reports no error, then it stores the file as a .obj file of the same name, called the object file. So, here it will create the first.obj. This .obj file is not executable. The process is continued by the Linker which finally gives a .exe file which is executable.

Linker: First of all, let us know that library functions are not a part of any C program but of the C software. Thus, the compiler doesn't know the operation of any function, whether it be printf or scanf. The definitions of these functions are stored in their respective library which the compiler should be able to link. This is what the Linker does. So, when we write #include, it includes stdio.h library which gives access to Standard Input and Output. The linker links the object files to the library functions and the program becomes a .exe file. Here, first.exe will be created which is in an executable format.

Loader: Whenever we give the command to execute a particular program, the loader comes into work. The loader will load the .exe file in RAM and inform the CPU with the starting point of the address where this program is loaded.

Registers in CPU



CPU Registers

Instruction Register: It holds the current instructions to be executed by the CPU.

Program Counter: It contains the address of the next instructions to be executed by the CPU.

Accumulator: It stores the information related to calculations.

The loader informs Program Counter about the first instruction and initiates the execution. Then onwards, Program Counter handles the task.

Difference between Linker and Loader

LINKER	LOADER
Linker generates the executable module of a source program.	Loader loads the executable module to the main memory for execution.
Linker takes the object code generated by an assembler, as input.	Loader takes executable module generated by a linker as input.
Linker combines all the object modules of a source code to generate an executable module.	Loader allocates the addresses to an executable module in main memory for execution.
The types of Linker are Linkage Editor, Dynamic linker.	The types of Loader are Absolute loading, Relocatable loading and Dynamic Run-time loading.

Data types in c

A data-type in C programming is a set of values and is determined to act on those values. C provides various types of data-types which allow the programmer to select the appropriate type for the variable to set its value

The data-type in a programming language is the collection of data with values having fixed meaning as well as characteristics.

Some of them are an integer, floating point, character, etc. Usually, programming languages specify the range values for given data-type.

C Data Types are used to:

- Identify the type of a variable when it declared.
- Identify the type of the return value of a function
- Identify the type of a parameter expected by a function

ANSI C provides three types of data types:
Primary or (Built-in) Data Types: <i>void, int, char, double and float.</i>
Derived Data Types: <i>Array, References, and Pointers.</i>
User Defined Data Types: <i>Structure, Union, and Enumeration.</i>

1.Primary data types

Every C compiler supports five primary data types:

Void	As the name suggests, it holds no value and is generally used for specifying the type of function or what it returns. If the function has a void type, it means that the function will not return any value.
Int	Used to denote an integer type.
Char	Used to denote a character type.
float, double	Used to denote a floating point type.
int *, float *, char *	Used to denote a pointer type.

Declaration of primary data types with variable names

After taking suitable variable names, they need to be assigned with a data type. This is how the data types are used along with variables:

Example:

```
int    age;  
char   letter;
```

```
float height, width;
```

Derived Data Types

C supports three derived data types:

Data Types	Description
Arrays	Arrays are sequences of data items having homogeneous values. They have adjacent memory locations to store values.
References	Function pointers allow referencing functions with a particular signature.
Pointers	These are powerful C features which are used to access the memory and deal with their addresses.

User defined data types

C allows the feature called *type definition* which allows programmers to define their identifier that would represent an existing data type. There are three such types:

Data Types	Description
Structure	It is a package of variables of different types under a single name. This is done to handle data efficiently. "struct" keyword is used to define a structure.
Union	These allow storing various data types in the same memory location. Programmers can define a union with different members, but only a single member can contain a value at a given time. It is used for
Enum	Enumeration is a special data type that consists of integral constants, and each of them is assigned with a specific name. "enum" keyword is used to define the enumerated data type.

C variables

Variables are memory locations(storage area) in the C programming language. The primary purpose of variables is to store data in memory for later use. Unlike [constants](#) which do not change during the program execution, variables value may change during execution. If you declare a variable in C, that means you are asking to the operating system to reserve a piece of memory with that variable name.

Variable definition in c

Syntax

```
type variable_name;
```

or

```
type variable_name, variable_name, variable_name;
```

Variable Definition and Initialization

Example:

```
int    width, height=5;
char   letter='A';
float  age, area;
double d;

/* actual initialization */width = 10;
age = 26.5;
```

Variable assignment is a process of assigning a value to a variable.

Example:

```
int width = 60;

int age = 31;
```

There are some rules on choosing variable names

- A variable name can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and the underscore character.
- The first character must be a letter or underscore.
- Blank spaces cannot be used in variable names.
- Special characters like #, \$ are not allowed.
- C keywords cannot be used as variable names.
- Variable names are case sensitive.
- Values of the variables can be numeric or alphabetic.
- Variable type can be char, int, float, double or void.

C Program to Print Value of a Variable

Example:

```
#include<stdio.h>

void main()
{
    /* c program to print value of a variable */    int age = 33;
    printf("I am %d years old.\n", age);
}
```

Program Output:

```
I am 33 years old.
```

Constants in c Constants are like a variable, except that their value never changes during execution once defined.. C Constants is the most fundamental and essential part of the C programming language. Constants in C are the fixed values that are used in a program, and its value remains the same during the entire execution of the program.

- Constants are also called literals.
- Constants can be any of the [data types](#).
- It is considered best practice to define constants using only *upper-case* names.

Constant Definition in C

Syntax:

```
const type constant_name;
```

const keyword defines a constant in C.

Example:

```
#include<stdio.h>

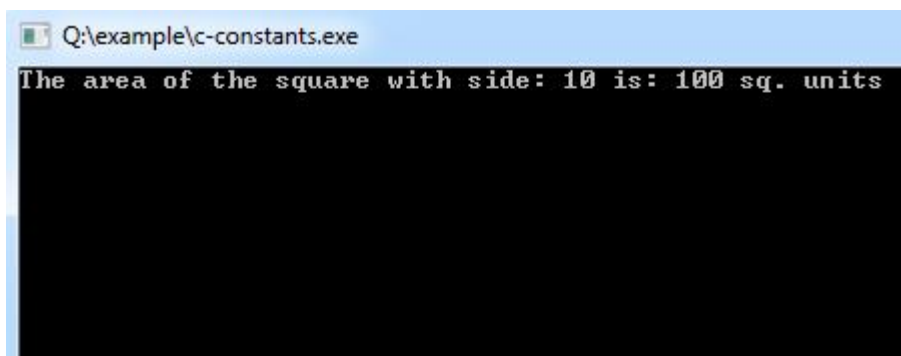
main()
{
    const int SIDE = 10;

    int area;

    area = SIDE*SIDE;

    printf("The area of the square with side: %d is: %d sq. units"
        , SIDE, area);
}
```

Program Output:



```
Q:\example\c-constants.exe
The area of the square with side: 10 is: 100 sq. units
```

Putting const either before or after the type is possible.

```
int const SIDE = 10;
```

or

```
const int SIDE = 10;
```

Constant Types in C

Constants are categorized into two basic types, and each of these types has its subtypes/categories. These are:

1.Primary Constants

Numeric Constants

- Integer Constants
- Real Constants

Character Constants

- Single Character Constants
- String Constants
- Backslash Character Constants

Note: C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning which is known to the compiler. They are also termed as "Escape Sequence".

For Example:

`\t` is used to give a tab

`\n` is used to give a new line

Constants	Meaning
<code>\a</code>	beep sound
<code>\b</code>	Backspace
<code>\f</code>	form feed
<code>\n</code>	new line
<code>\r</code>	carriage return
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab
<code>\'</code>	single quote
<code>\"</code>	double quote
<code>\\</code>	Backslash
<code>\0</code>	Null

2.Secondary Constants

- [Array](#)
- [Pointer](#)
- [Structure](#)
- [Union](#)
- [Enum](#)

C operators

C operators are symbols that are used to perform mathematical or logical manipulations. The C programming language is rich with built-in operators. Operators take part in a program for manipulating data and variables and form a part of the mathematical or logical expressions.

Types of Operators in C

C programming language offers various types of operators having different functioning capabilities.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and Decrement Operators
6. Conditional Operator
7. Bitwise Operators
8. Special Operators

Arithmetic Operators

Arithmetic Operators are used to performing mathematical calculations like addition (+), subtraction (-), multiplication (*), division (/) and modulus (%).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

C Program to Add Two Numbers

[Example:](#)

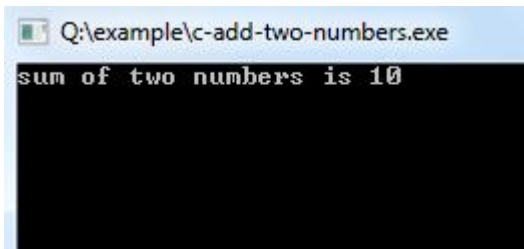
```

#include <stdio.h>

void main()
{
    int i=3,j=7,k; /* Variables Defining and Assign values */ k=i+j;
    printf("sum of two numbers is %d\n", k);
}

```

Program Output:



Increment and Decrement Operators

Increment and Decrement Operators are useful operators generally used to minimize the calculation, i.e. ++x and x++ means $x=x+1$ or $-x$ and $x--$ means $x=x-1$. But there is a slight difference between ++ or -- written before or after the operand. Applying the pre-increment first add one to the operand and then the result is assigned to the variable on the left whereas post-increment first assigns the value to the variable on the left and then increment the operand.

Operator	Description
++	Increment
--	Decrement

Example: To Demonstrate prefix and postfix modes.

```

#include <stdio.h>

//stdio.h is a header file used for input.output purpose.

void main()
{
    //set a and b both equal to 5.
    int a=5, b=5;

    //Print them and decrementing each time.
    //Use postfix mode for a and prefix mode for b.
}

```

```

printf("\n%d %d", a--, --b);

printf("\n%d %d", a--, --b);

printf("\n%d %d", a--, --b);

printf("\n%d %d", a--, --b);

printf("\n%d %d", a--, --b);

}

```

Program Output:

```

5 4

4 3

3 2

2 1

1 0

```

Relational Operators

Relational operators are used to comparing two quantities or values.

Operator	Description
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Logical Operators

C provides three logical operators when we test more than one condition to make decisions. These are: && (meaning logical AND), || (meaning logical OR) and ! (meaning logical NOT).

Operator	Description
&&	<i>And</i> operator. It performs logical conjunction of two expressions. (if both expressions evaluate to True, result is True. If either expression evaluates to False, the result is False)

	<i>Or</i> operator. It performs a logical disjunction on two expressions. (if either or both expressions evaluate to True, the result is True)
!	<i>Not</i> operator. It performs logical negation on an expression.

Bitwise Operators

C provides a special operator for bit operation between two variables.

Operator	Description
<<	Binary Left Shift Operator
>>	Binary Right Shift Operator
~	Binary Ones Complement Operator
&	Binary AND Operator
^	Binary XOR Operator
	Binary OR Operator

Assignment Operators

Assignment operators applied to assign the result of an expression to a variable. C has a collection of shorthand assignment operators.

Operator	Description
=	Assign
+=	Increments then assign
-=	Decrements then assign
*=	Multiplies then assign
/=	Divides then assign
%=	Modulus then assign

<<=	Left shift and assign
>>=	Right shift and assign
&=	Bitwise AND assign
^=	Bitwise exclusive OR and assign
=	Bitwise inclusive OR and assign

Conditional Operator

C offers a ternary operator which is the conditional operator (?: in combination) to construct conditional expressions.

Operator	Description
?:	Conditional Expression

Special Operators

C supports some special operators

Operator	Description
sizeof()	Returns the size of a memory location.
&	Returns the address of a memory location.
*	Pointer to a variable.

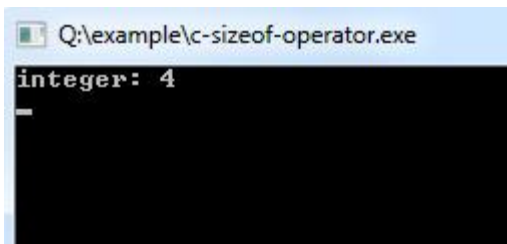
Program to demonstrate the use of sizeof operator

Example:

```
#include <stdio.h>

void main()
{
    int i=10; /* Variables Defining and Assign values */
    printf("integer: %d\n", sizeof(i));
}
```

Program Output:



Keywords

The C Keywords must be in your information because you can not use them as a variable name.

You can't use a keyword as an identifier in your C programs, its reserved words in C library and used to perform an internal operation. The meaning and working of these keywords are already known to the compiler.

C Keywords List

A list of *32 reserved keywords* in c language is given below:

auto	Double	int	Struct
break	Else	long	Switch
case	Enum	register	Typedef
char	Extern	return	Union
const	Float	short	Unsigned
continue	For	signed	Void
default	Goto	sizeof	Volatile
do	If	static	While

Example Where and How Keywords are Used in the Program

Example:

```
#include<stdio.h>

main()
{
float a, b;
printf("Showing how keywords are used.");
return 0;
}
```

In the above program, `float` and `return` are keywords. The float is used to declare variables, and return is used to return an integer type value in this program.

C Type Casting

Type Casting in C is used to convert a variable from one data type to another data type, and after type casting compiler treats the variable as of the new data type. Type Casting in C is used to convert a variable from one data type to another data type, and after type casting compiler treats the variable as of the new data type.

Syntax:

```
(type_name) expression
```

Without Type Casting

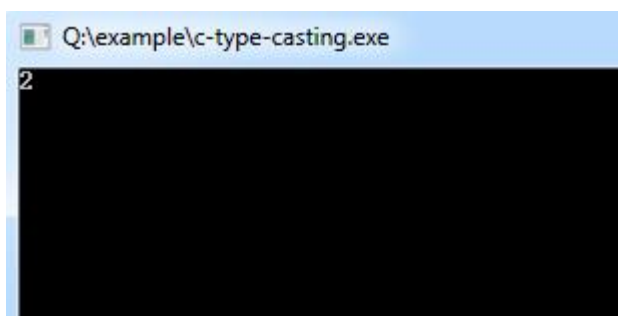
Example:

```
#include <stdio.h>

main ()
{
    int a;
    a = 15/6;
    printf("%d",a);
}
```

Program Output:

In the above C program, 15/6 alone will produce integer value as 2.



After Type Casting

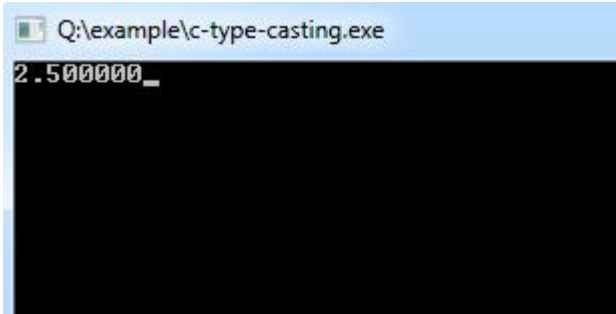
```
#include <stdio.h>

main ()
{
    float a;
    a = (float) 15/6;
```

```
printf("%f", a);  
}
```

Program Output:

After type cast is done before division to retain float value 2.500000.



Functions

C function is a self-contained block of statements that can be executed repeatedly whenever we need it.

Benefits of using functions

- The function provides modularity.
- The function provides reusable code.
- In large programs, debugging and editing tasks is easy with the use of functions.
- The program can be modularized into smaller parts.
- Separate function independently can be developed according to the needs..

There are two types of functions in C

- **Built-in(Library) Functions**
 - The system provided these functions and stored in the library. Therefore it is also called *Library Functions*. e.g. `scanf()`, `printf()`, `strcpy`, `strlwr`, `strcmp`, `strlen`, `strcat` etc.
 - To use these functions, you just need to include the appropriate C header files.
- **User Defined Functions** These functions are defined by the user at the time of writing the program.

Parts of Function

1. Function Prototype (function declaration)
2. Function Definition
3. Function Call

1. Function Prototype

Syntax:

```
dataType functionName (Parameter List)
```

Example:


```
int addition();
```

2. Function Definition

Syntax:

```
returnType functionName (Function arguments) {  
    //body of the function  
}
```

Example:

```
int addition()  
{  
  
}
```

3. Calling a function in C

Program to illustrate the Addition of Two Numbers using User Defined Function

Example:

```
#include<stdio.h>  
  
/* function declaration */int addition();  
  
int main()  
{  
    /* local variable definition */    int answer;  
  
    /* calling a function to get addition value */    answer = addition();  
  
    printf("The addition of the two numbers is: %d\n",answer);  
    return 0;  
}  
  
/* function returning the addition of two numbers */int addition()  
{  
    /* local variable definition */    int num1 = 10, num2 = 5;
```

```
    return num1+num2;
}
```

Program Output:

The addition of the two numbers is: 15

C Function Arguments

While calling a function, the arguments can be passed to a function in two ways, Call by value and call by reference.

Type	Description
Call by Value	The actual parameter is passed to a function. New memory area created for the passed parameters, can be used only within the function. The actual parameters cannot be modified here.
Call by Reference	Instead of copying variable; an address is passed to function as parameters. Address operator(&) is used in the parameter of the called function. Changes in function reflect the change of the original variables.

Call by Value

Example:

```
#include<stdio.h>

/* function declaration */int addition(int num1, int num2);

int main()
{
    /* local variable definition */    int answer;

    int num1 = 10;
    int num2 = 5;

    /* calling a function to get addition value */    answer =
    addition(num1,num2);

    printf("The addition of two numbers is: %d\n",answer);

    return 0;
}

/* function returning the addition of two numbers */int addition(int a,int
b)
{
    return a + b;
}
```

Program Output:

```
The addition of two numbers is: 15
```

Call by Reference

Example

```
#include<stdio.h>

/* function declaration */int addition(int *num1, int *num2);

int main()
{
    /* local variable definition */    int answer;

    int num1 = 10;

    int num2 = 5;

    /* calling a function to get addition value */    answer =
    addition(&num1,&num2);

    printf("The addition of two numbers is: %d\n",answer);

    return 0;
}

/* function returning the addition of two numbers */int addition(int *a,int
*b)
{
    return *a + *b;
}
```

Program Output:

```
The addition of two numbers is: 15
```

Variable scope

A scope is a region of the program, and the scope of variables refers to the area of the program where the variables can be accessed after its declaration.

In C every variable defined in scope. You can define scope as the section or region of a program where a variable has its existence; moreover, that variable cannot be used or accessed beyond that region.

In C programming, variable declared within a function is different from a variable declared outside of a function. The variable can be declared in three places. These are:

Position	Type
----------	------

Inside a function or a block.	local variables
Out of all functions.	Global variables
In the function parameters.	Formal parameters

So, now let's have a look at each of them individually.

Local Variables

Variables that are declared within the function block and can be used only within the function is called local variables.

Local Scope or Block Scope

A local scope or block is collective program statements put in and declared within a function or block (a specific region enclosed with curly braces) and variables lying inside such blocks are termed as local variables. All these locally scoped statements are written and enclosed within left ({} and right braces (}) curly braces. There's a provision for nested blocks also in C which means there can be a block or a function within another block or function. So it can be said that variable(s) that are declared within a block can be accessed within that specific block and all other inner blocks of that block, but those variables cannot be accessed outside the block.

[Example:](#)

```
#include <stdio.h>

int main ()
{
    /* local variable definition and initialization */    int x,y,z;

    /* actual initialization */    x = 20;
    y = 30;
    z = x + y;

    printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

    return 0;
}
```

Global Variables

Variables that are declared outside of a function block and can be accessed inside the function is called global variables.

Global Scope

Global variables are defined outside a function or any specific block, in most of the case, on the top of the C program. These variables hold their values all through the end of the program and are accessible within any of the functions defined in your program.

Any function can access variables defined within the global scope, i.e., its availability stays for the entire program after being declared.

Example:

```
#include <stdio.h>

/* global variable definition */int z;

int main ()
{
    /* local variable definition and initialization */    int x,y;

    /* actual initialization */    x = 20;
    y = 30;
    z = x + y;

    printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

    return 0;
}
```

Global Variable Initialization

After defining a local variable, the system or the compiler won't be initializing any value to it. You have to initialize it by yourself.

It is considered good programming practice to initialize variables before using. Whereas in contrast, global variables get initialized automatically by the compiler as and when defined. Here's how based on datatype; global variables are defined.

datatype	Initial Default Value
int	0
char	'\0'
float	0

double	0
pointer	NULL

C Header Files

C language is famous for its different libraries and the predefined functions pre-written within it. These make programmer's effort a lot easier. In this tutorial, you will be learning about C header files and how these header files can be included in your C program and how it works within your C language.

What are the Header Files

Header files are helping file of your C program which holds the definitions of various functions and their associated variables that needs to be imported into your C program with the help of pre-processor `#include` statement. All the header file have a '.h' an extension that contains C function declaration and macro definitions. In other words, the header files can be requested using the preprocessor directive `#include`. The default header file that comes with the C compiler is the `stdio.h`.

Including a header file means that using the content of header file in your source program. A straightforward practice while programming in C or C++ programs is that you can keep every macro, global variables, constants, and other function prototypes in the header files. The basic syntax of using these header files is:

[Syntax:](#)

```
#include <file>
```

or

```
#include "file"
```

This kind of file inclusion is implemented for including system oriented header files. This technique (with angular braces) searches for your file-name in the standard list of system directories or within the compiler's directory of header files. Whereas, the second kind of header file is used for user-defined header files or other external files for your program. This technique is used to search for the file(s) within the directory that contains the current file.

The C's `#include` preprocessor directive exertions by going through the C preprocessors for scanning any specific file like that of input before abiding by the rest of your existing source file. Let us take an example where you may think of having a header file `karl.h` having the following statement:

[Example:](#)

```
char *example (void);
```

then, you have a main C source program which seems something like this:

[Example:](#)

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

```

int x;

#include "karl.h"

int main ()
{
    printf("Program done");

    return 0;
}

```

So, the compiler will see the entire C program and token stream as:

Example:

```

#include<stdio.h>

int x;

char * example (void);

int main ()
{
    printf("Program done");

    return 0;
}

```

#include<stdio.h>

stdio is standard for input/output, this allows us to use some commands which includes a file called stdio.h. Or This is a preprocessor command. That notifies the compiler to include the header file stdio.h in the program before compiling the source-code.

Conio.h

The conio.h header file used in C programming language contains functions for console input/output. Some of its most commonly used functions are clrscr, getch, getche, kbhit etc. They can be used to clear screen, change color of text and background, move text, check whether a key is pressed or not and to perform other tasks.

conio.h is a **C** header file used mostly by **MS-DOS** compilers to provide console **input/output**. This header declares several useful library functions for performing "console input and output" from a program

All C inbuilt functions which are declared in conio.h header file are given below.

LIST OF INBUILT C FUNCTIONS IN CONIO.H FILE:

Functions	Description
clrscr()	This function is used to clear the output screen.

getch()	It reads character from keyboard
getche()	It reads character from keyboard and echoes to o/p screen
textcolor()	This function is used to change the text color
textbackground()	This function is used to change text background

Comments in c

/* Comments */	<p>Comments are a way of explaining what makes a program. The compiler ignores comments and used by others to understand the code. OR</p> <p>This is a comment block, which is ignored by the compiler. Comment can be used anywhere in the program to add info about the program or code block, which will be helpful for developers to understand the existing code in the future easily.</p>
----------------------	---

As we all know the three essential functions of a computer are reading, processing and writing data. Majority of the programs take data as input, and then after processing the processed data is being displayed which is called information. In C programming you can use `scanf()` and `printf()` predefined function to read and print data.

Example

```
#include<stdio.h>

void main()
{
int a,b,c;
printf("Please enter any two numbers: \n");
scanf("%d %d", &a, &b);
c = a + b;
printf("The addition of two number is: %d", c);
}
```

Output:

```
Please enter any two numbers:
12
3
The addition of two number is:15
```


The above program `scanf()` is used to take input from the user, and respectively `printf()` is used to display output result on the screen.

Managing Input/Output

I/O operations are useful for a program to interact with users. `stdlib` is the standard C library for input-output operations. While dealing with input-output operations in C, two important streams play their role. These are:

1. Standard Input (stdin)
2. Standard Output (stdout)

`Standard input` or `stdin` is used for taking input from devices such as the keyboard as a data stream. `Standard output` or `stdout` is used for giving output to a device such as a monitor. For using I/O functionality, programmers must include `stdio header-file` within the program.

Reading Character in C

The easiest and simplest of all I/O operations are taking a character as input by reading that character from standard input (keyboard). `getchar()` function can be used to read a single character. This function is alternate to `scanf()` function.
[Syntax](#)

```
var_name = getchar();
```

[Example:](#)

```
#include<stdio.h>
void main()
{
char title;
title = getchar();
}
```

There is another function to do that task for files: `getc` which is used to accept a character from standard input.
[Syntax:](#)

```
int getc(FILE *stream);
```

Writing Character In C

Similar to `getchar()` there is another function which is used to write characters, but one at a time.
[Syntax:](#)

```
putchar (var_name) ;
```

[Example:](#)

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

```
void main()
{
char result = 'P';
putchar(result);
putchar('\n');
}
```

Similarly, there is another function `putc` which is used for sending a single character to the standard output.

Syntax:

```
int putc(int c, FILE *stream);
```

Formatted Input

It refers to an input data which has been arranged in a specific format. This is possible in C using `scanf()`. We have already encountered this and familiar with this function.

Syntax:

```
scanf("control string", arg1, arg2, ..., argn);
```

The field specification for reading integer inputted number is:

```
%w sd
```

Here the % sign denotes the conversion specification; `w` signifies the integer number that defines the field width of the number to be read. `d` defines the number to be read in integer format.

Example:

```
#include<stdio.h>
void main()
{
int var1= 60;
int var2= 1234;
scanf("%2d %5d", &var1, &var2);
}
```

Input data items should have to be separated by spaces, tabs or new-line and the punctuation marks are not counted as separators.

Reading and Writing Strings in C

There are two popular library functions `gets()` and `puts()` provides to deal with strings in C.

gets: The char `*gets(char *str)` reads a line from `stdin` and keeps the string pointed to by the `str` and is terminated when the new line is read or `EOF` is reached. The declaration of `gets()` function is:

Syntax:

```
char *gets(char *str);
```

Where `str` is a pointer to an array of characters where C strings are stored.

puts: The function - `int puts(const char *str)` is used to write a string to `stdout`, but it does not include null characters. A new line character needs to be appended to the output. The declaration is:

Syntax:

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

where `str` is the string to be written in C.

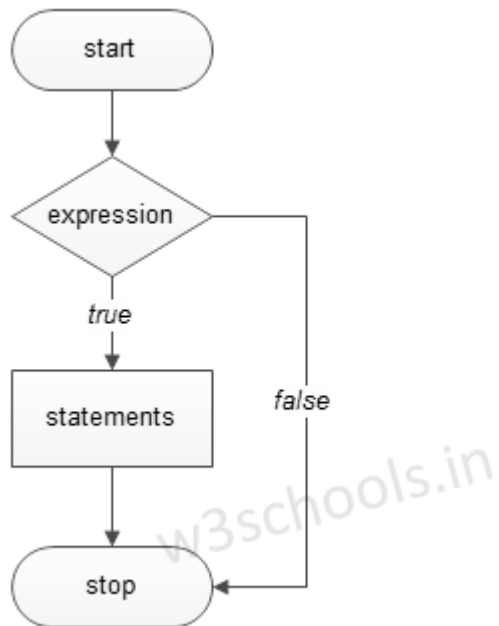
C Decision Making:

C conditional statements allow you to make a decision, based upon the result of a condition. These statements are called **Decision Making Statements** or **Conditional Statements**.

So far, we have seen that all set of statements in a C program gets executed sequentially in the order in which they are written and appear. This occurs when there is no jump based statements or repetitions of certain calculations. But some situations may arise where we may have to change the order of execution of statements depending on some specific conditions. This involves a kind of decision making from a set of calculations. It is to be noted that C language assumes any non-zero or non-null value as true and if zero or null, treated as false.

This type of structure requires that the programmers indicate several conditions for evaluation within a program. The statement(s) will get executed only if the condition becomes true and optionally, alternative statement or set of statements will get executed if the condition becomes false.

The flowchart of the Decision-making technique in C can be expressed as:



C languages have such decision-making capabilities within its program by the use of following the decision making statements:

Conditional Statements in C:

If statement

[if statement](#)

[if-else statement](#)

[switch statement](#)

If statements in C is used to control the program flow based on some condition, it's used to execute some statement code block if the expression is evaluated to true. Otherwise, it will get skipped. This is the simplest way to modify the control flow of the program.

The if statement in C can be used in various forms depending on the situation and complexity.

There are four different types of if statement in C. These are:

- Simple if Statement
- if-else Statement
- Nested if-else Statement

we have to work with the first two types. Let's see the syntax:

The basic format of if statement is:

Syntax:

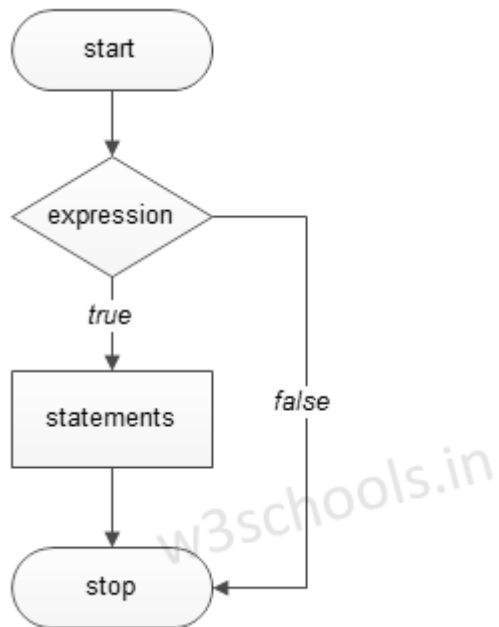
```

if (test_expression)
{
    statement 1;
    statement 2;
    ...
}
  
```

```
}
```

'Statement n' can be a statement or a set of statements, and if the test expression is evaluated to *true*, the statement block will get executed, or it will get skipped.

Figure - Flowchart of if Statement:



Example of a C Program to Demonstrate if Statement

Example:

```
#include<stdio.h>

main()
{
    int a = 15, b = 20;

    if (b > a) {
        printf("b is greater");
    }
}
```

Program Output:

```
Q:\example\c-decision-making-if.exe
b is greater_
```

If else statements

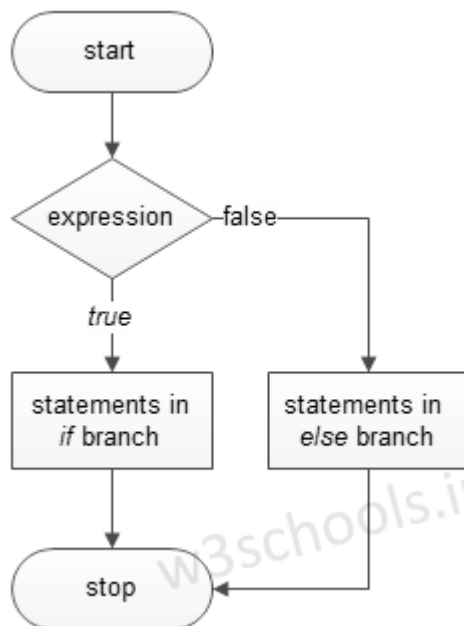
If else statements in C is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise executes else statement code block.

The basic format of if -else statement is:

Syntax:

```
if(test_expression)
{
    //execute your code
}
else
{
    //execute your code
}
```

Figure - Flowchart of if-else Statement:



Example of a C Program to Demonstrate if-else Statement

Example:

```
#include<stdio.h>

main()
{
    int a, b;
```

```

printf("Please enter the value for a:");
scanf("%d", & a);

printf("\nPlease the value for b:");
scanf("%d", & b);

if (a & gt; b) {
    printf("\n a is greater");
} else {
    printf("\n b is greater");
}
}

```

Program Output:

```

Q:\example\c-if-else-1.exe
Please enter the value for a:10
Please the value for b:5
a is greater_

```

Example:

```

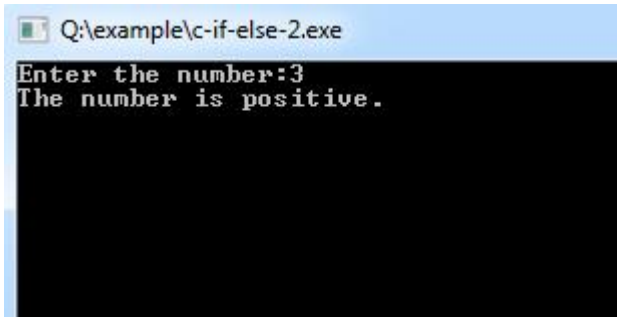
#include<stdio.h>

main() {
    int num;
    printf("Enter the number:");
    scanf("%d", num);

    /* check whether the number is negative number */ if (num < 0)
        printf("The number is negative.");
    else
        printf("The number is positive.");
}

```

Program Output:



Nested if else

Nested if else statements play an important role in C programming, it means you can use conditional statements inside another conditional statement.

The basic format of Nested if else statement is:

Syntax:

```
if(test_expression one)
{
    if(test_expression two) {
        //Statement block Executes when the boolean test expression two is
true.
    }
}
else
{
    //else statement block
}
```

Example of a C Program to Demonstrate Nested if-else Statement

Example:

```
#include<stdio.h>

main()
{
int x=20,y=30;

    if(x==20)
    {
        if(y==30)
```



```
    {  
        printf("value of x is 20, and value of y is 30.");  
    }  
}  
}
```

Execution of the above code produces the following result.

Output:

```
value of x is 20, and value of y is 30.
```

Switch Statement:

C switch statement is used when you have multiple possibilities for the if statement. Switch case will allow you to choose from multiple options. When we compare it to a general electric switchboard, you will have many switches in the switchboard but you will only select the required switch, similarly, the switch case allows you to set the necessary statements for the user.

The basic format of the switch statement is:

Syntax:

```
switch(variable)  
{  
    case 1:  
        //execute your code  
    break;  
  
    case n:  
        //execute your code  
    break;  
  
    default:  
        //execute your code  
    break;  
}
```

After the end of each block it is necessary to insert a *break statement* because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from every case onwards after matching the case block.

Contents

Example of a C Program to Demonstrate Switch Statement

Example:

```
#include<stdio.h>

main()
{
    int a;
    printf("Please enter a no between 1 and 5: ");
    scanf("%d",&a);

    switch(a)
    {
        case 1:
            printf("You chose One");
            break;

        case 2:
            printf("You chose Two");
            break;

        case 3:
            printf("You chose Three");
            break;

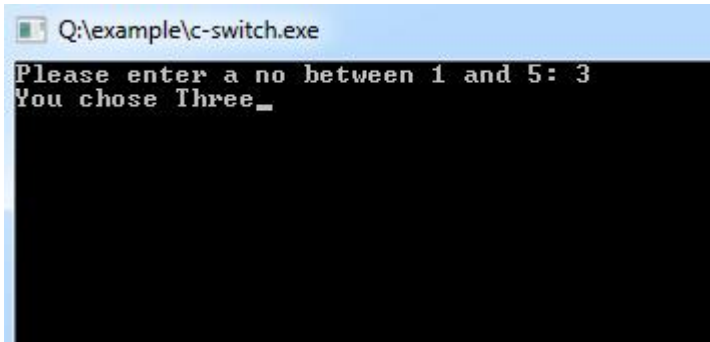
        case 4:
            printf("You chose Four");
            break;

        case 5:
            printf("You chose Five.");
            break;

        default :
            printf("Invalid Choice. Enter a no between 1 and 5");
    }
}
```

```
    break;
}
}
```

Program Output:



When none of the cases is evaluated to `true`, the default case will be executed, and `break statement` is not required for default statement

C Loops

Sometimes it is necessary for the program to execute the statement several times, and C loops execute a block of commands a specified number of times until a condition is met. In this chapter, you will learn about all the looping statements of C programming along with their use.

A computer is the most suitable machine to perform repetitive tasks and can tirelessly do a task tens of thousands of times.

Every programming language has the feature to instruct to do such repetitive tasks with the help of certain form of statements.

The process of repeatedly executing a collection of statement is called `looping`. The statements get executed many numbers of times based on the condition. But if the condition is given in such logic that the repetition continues any number of times with no

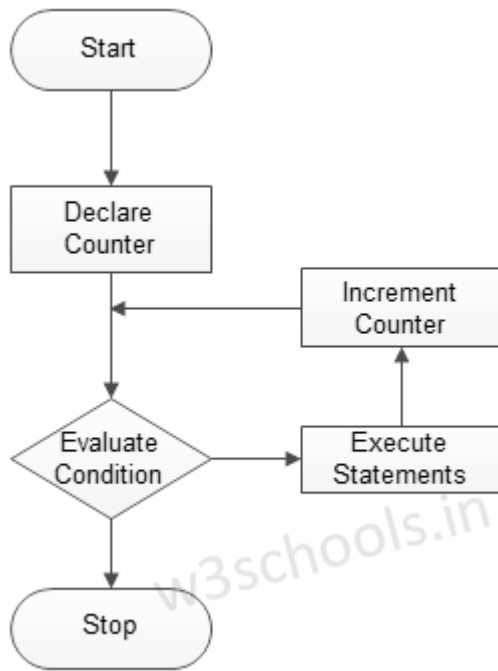
fixed condition to stop looping those statements, then this type of looping is called `infinite looping`.

C supports the following types of loops:

- [while loops](#)
- [do while loops](#)
- [for loops](#)

All are slightly different and provides loops for different situations.

Figure - Flowchart of Looping:



C Loop Control Statements

Loop control statements are used to change the normal sequence of execution of the loop.

Statement	Syntax	Description
break statement	break;	Is used to terminate loop or switch statements.
continue statement	continue;	Is used to suspend the execution of current loop iteration and transfer control to the loop for the next iteration.
goto statement	goto labelName; labelName: statement;	It transfers the current program execution sequence to some other part of the program.

While loop

C while loops statement allows to repeatedly run the same block of code until a condition is met.

while loop is a most basic loop in C programming. while loop has one control condition, and executes as long the condition is true. The condition of the loop is tested before the body of the loop is executed, hence it is called an *entry-controlled* loop.

The basic format of while loop statement is:

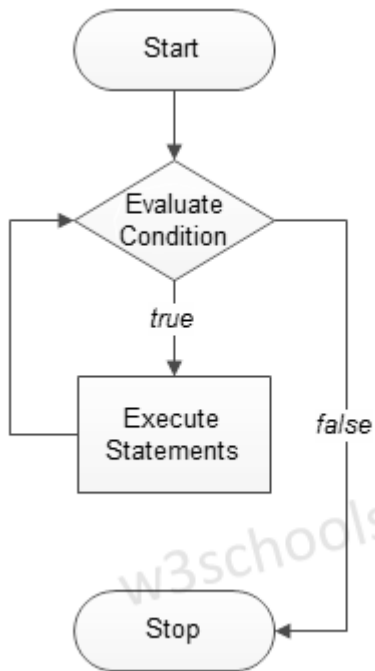
Syntax:

```

While (condition)
{
    statement(s);
    Incrementation;
}
  
```

```
}
```

Figure - Flowchart of while loop:



2_C while loops - Video Tutorial

Example of a C Program to Demonstrate while loop

Example:

```
#include<stdio.h>

int main ()
{
    /* local variable Initialization */    int n = 1,times=5;

    /* while loops execution */    while( n <= times )
    {
        printf("C while loops: %d\n", n);
        n++;
    }

    return 0;
}
```

Program Output:

```
Q:\example\c-while-loop.exe
C while loops: 1
C while loops: 2
C while loops: 3
C while loops: 4
C while loops: 5
```

Do while loop

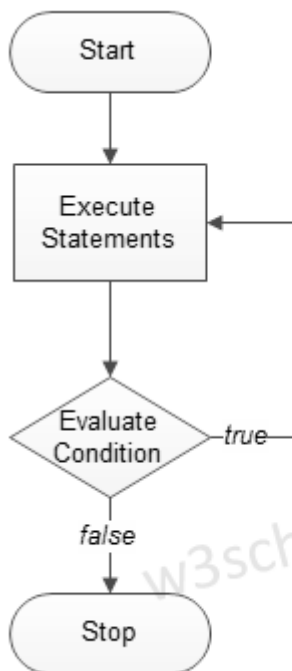
C do while loops are very similar to the while loops, but it always executes the code block at least once and furthermore as long as the condition remains true. This is an **exit-controlled loop**.

The basic format of do while loop statement is:

Syntax:

```
do
{
    statement(s);
}while( condition );
```

Figure - Flowchart of do while loop:



Example of a C Program to Demonstrate do while loop

Example:

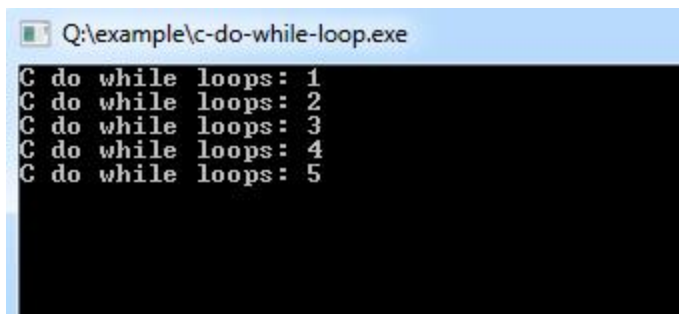
```
#include<stdio.h>

int main ()
{
    /* local variable Initialization */    int n = 1,times=5;

    /* do loops execution */    do
    {
        printf("C do while loops: %d\n", n);
        n = n + 1;
    }while( n <= times );

    return 0;
}
```

Program Output:



```
Q:\example\c-do-while-loop.exe
C do while loops: 1
C do while loops: 2
C do while loops: 3
C do while loops: 4
C do while loops: 5
```

C for loop

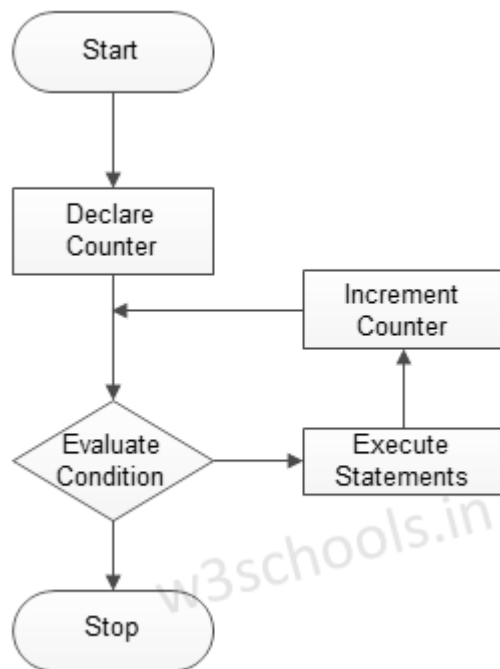
C for loops is very similar to a while loops in that it continues to process a block of code until a statement becomes false, and everything is defined in a single line. The for loop is also **entry-controlled** loop

The basic format of for loop statement is:

Syntax:

```
for ( init; condition; increment )
{
    statement(s);
}
```

Figure - Flowchart of for loop:



. 2_C for loops - Video Tutorial

Example of a C Program to Demonstrate for loop

Example:

```
#include<stdio.h>

int main ()
{
    /* local variable Initialization */  int n,times=5;;

    /* for loops execution */  for( n = 1; n <= times; n = n + 1 )
    {
        printf("C for loops: %d\n", n);
    }

    return 0;
}
```

Program Output:


```
Q:\example\c-for-loop.exe
C for loops: 1
C for loops: 2
C for loops: 3
C for loops: 4
C for loops: 5
```