History of C programming

C is a mid-level structured oriented programming language, used in general-purpose programming, developed by Dennis Ritchie at Bell Labs, the USA, between 1972.

Facts about C

- C is the most widely used System Programming Language.
- It was developed to overcome the problems of previous languages such as B, BCPL, etc.
- C was invented to write **UNIX** operating system.
- **Linux OS**, **PHP**, and **MySQL** are written in C.
- C has been written in assembly language.

Why to use C ?

Because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Data Bases
- Language Interpreters
- Utilities

Advantages of C

- C is the building block for many other programming languages.
- Programs written in C are highly portable.
- Several standard functions are there (like in-built) that can be used to develop programs.
- C programs are collections of C library functions, and it's also easy to add functions to the C library.
- The modular structure makes code debugging, maintenance, and testing easier.

Disadvantages of C

- C does not provide Object Oriented Programming (OOP) concepts.
- C does not provide binding or wrapping up of data in a single unit.
- C does not provide Constructor and Destructor.

History of C Language/ Other Languages

The programming languages that were developed before C language are as follows:-

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson

Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

Characteristics/ Features of C Language

Features of C are as follows:

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

1) Simple

It provides a structured approach (to break the problem into parts), the rich set of library functions, data types, etc.

2) Machine Independent or Portable

c programs can be executed on different machines with some machine specific changes.

3) Mid-level programming language

C is intended to do low-level programming but it also supports the features of a high-level language. That is why it is known as mid-level language.

4) Structured programming language

we can break the program into parts using functions. So, it is easy to understand and modify.

5) Rich Library

C provides a lot of inbuilt functions that make the development fast.

6) Memory Management

It supports the feature of dynamic memory allocation.

7) Speed

The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

8) Pointer

We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array, etc.

9) Recursion

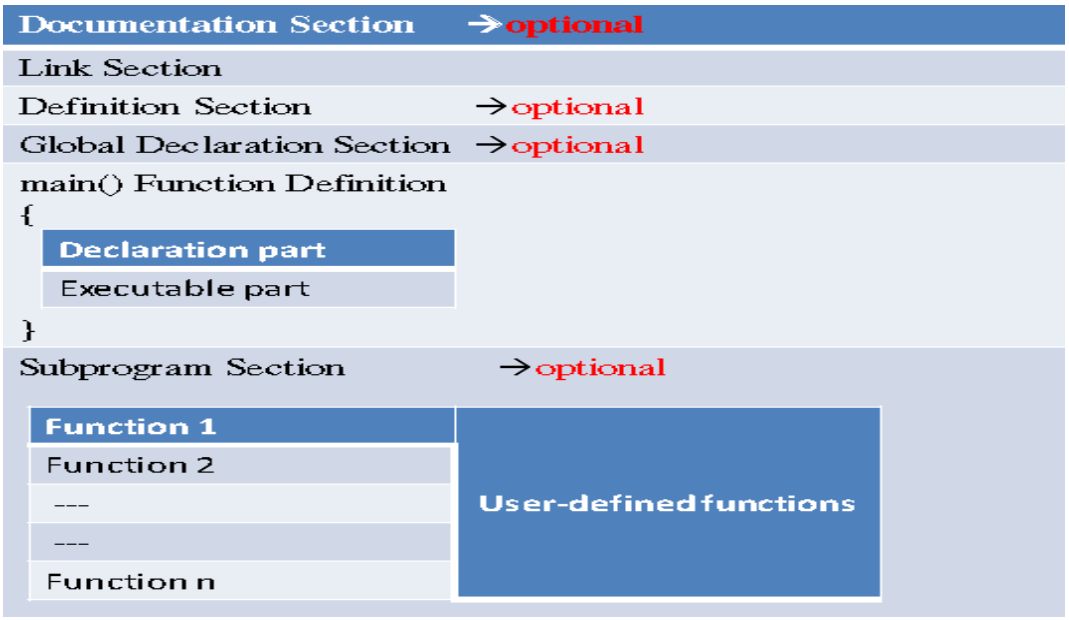
In C, we can call the function within the function.

10) Extensible

it can easily adopt new features.

Structure Of C Program

The structure of a C program means the specific structure to start the programming in the C language. Without a proper structure, it becomes difficult to analyze the problem and the solution.



C program is divided into different sections. There are six main sections to a basic c program. The six sections are-

1. Documentation section/Comment Section
2. Pre-processor section/ link section/Header Section
3. Definition section
4. Global declaration
5. Main function
6. User defined functions/ sub program section

- **Documentation section**

A set of comment lines giving the name of the program, the author and other details.

Example

```
/**
* File Name: Helloworld.c
* Author: Martin
* date: 09/08/2019
* description: a program to display hello world
*              no input needed
*/
```

- **Link section**

Provides instructions to the compiler to link functions from the system library.

Example

#include<stdio.h>

Definition section

defines all symbolic constants.

Example #define PI=3.14

- **Global declaration section**

- There are some variables that are used in more than one function.
- Such variables are called global variables.
- Global variables are declared in the global declaration section that is outside of all the functions.
- Also declares all the user-defined functions.

Example

```
float area(float r);
int a=7;
Main() function
{
Statement ------
----------
}
```

This section contains two parts

- **Declaration part**
- **Executable part**

These two parts must appear between the opening and closing braces ({ }).

Declaration part-

Declares all the variables used in the executable part

.**Executable part**

There is atleast one statement in the executable part.

The closing brace of main function sections is the logical end of the program.

All statements in the declaration and executable parts end with the semicolon (;).

- **Subprogram section**

Contains all user-defined functions that are called in the main function.

Example

```
#include <stdio.h>
int main()
{
        /* my first program in C */
        printf("Hello, World! \n");
        return 0;
}
```

NOTE :-

- 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.
- The next line *int main()* is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.

- The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line **return 0;** terminates the main() function and returns the value 0.

EXECUTING A 'C' PROGRAM

Steps :-

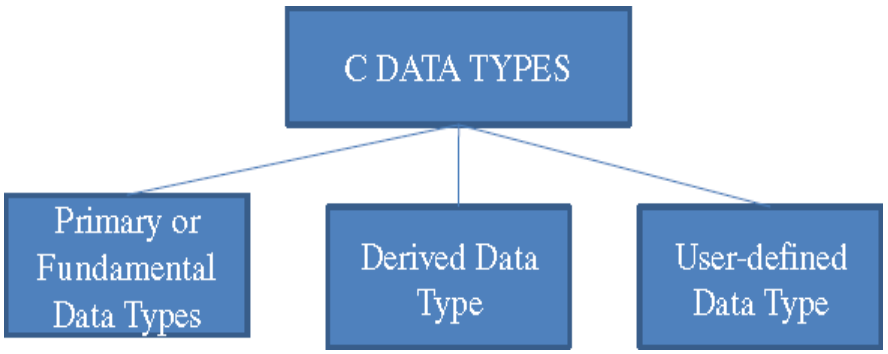
1. Creating the program.
2. Compiling the program.
3. Linking the program with functions that are needed from the C library
4. Executing the program.

Data Types

Data types refer to an extensive system used for declaring variables or functions of different types before its use.
The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

C Data Types are used to:

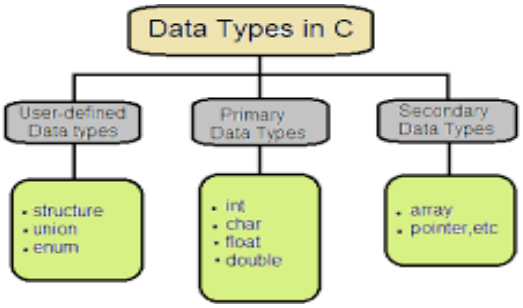
- Identify the type of a variable when it declared.
- Identify the type of the return value of a function.
- Identify the type of a parameter expected by a function.



C provides three types of data types:

1. **Primary(Built-in)**
   void, int, char, double and floa
2. **Derived**
   - Arrays,
     References
     Pointers

3. **User- Define**
   - Structure,
   - Union,
   - Enumeration



INTEGER TYPES

- Integer occupy one word of storage, and word sizes of machines vary (16 or 32 bits).
- The signed integer uses one bit for sign and 15 bits for the magnitude of the number.
- Smallest to largest.

FLOATING POINT TYPE

Floating point numbers are stored in 32-bits, with 6 digits of precision.

CHARACTER TYPE

- A single character can be defined as a character (char) type data.
- Characters are stored in 8 bits of internal storage.
- The qualifier signed or unsigned may be explicitly applied to char.

VOID TYPE

- There has no values.
- The type of function is said to be void when it does not return any value to the callingfunction.

EX- it can represent any of the other standard type.

TYPE	KEYWORD	SIZE (BITS)	RANGE
Character	char or signed char	8	-128 to 127
	unsigned char	8	0 to 255
Integer	int or signed int	16	-32,768 to 32767
	unsigned int	16	0 to 65535
	short int or signed short int	8	-128 to 127
	unsigned short int	8	0 to 255
	long int or signed long int	32	-2,147,483,648 to 2,147,483,647
	unsigned long int	32	0 to 4,294,967,295
Real	float	32	3.4E-38 to 3.4E+38
	double	64	1.7E-308 to 1.7E+308
	long double	80	3.4E-4932 to 1.1E+4932

2. 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	used to access the memory and deal with their addresses.

3.User Define

C Allows the feature called type definition which allows programmers to define their identifier that would represent an axisting 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.
Enum	It consists of integral constants, and each of them is assigned with a specific name. "enum" keyword is used to define the enumerated data type.

Variables in C

- Variable is a data name which is used to store data value .
- The value of the variable can be change during the execution.
- The rule for naming the variables is same as the naming identifier.

Rules :

1) They begin with a letter. Some systems permit underscore as the first character.
2) Upto 31 characters (ANSI)
3) Case Sensitive
4) It should not be a keyword
5) White space is not allowed.

DECLARATION OF VARIABLES

Declaration does two things-

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

Primary type declaration

A variable can be used to store a value of any data type.

data_type v1,v2,v3,….,vn;

v1,v2,v3,…vn are the name of the variables.

- Variables are separated by commas.
- Declaration statement end with a semicolon.

Example:

```
int count;
float total;
double ratio;
```

ASSIGNING VALUE TO VARIABLES
Variablename = constant;
Example:-
n=5;
count=total;
datatype variable_name=constant;
Example:
int n=5;
float x=5.5;
User-defined type declaration

1. **"type definition"** – allows users to define an identifier that would represent an existing datatype.
The user-defined data type identifier can be used to declare variables later.
typedef type identifier;
Where **type** refers to an existing data type, **identifier** refers to the new name given to the data type.
The existing data type may belong to any class of type, including the userdefined type.
Example:
typedef int units;
typedef float marks**;**

Advantage of typedef

- We can create meaningful data type names for increasing the readability of the program.

2. "enumerated"
Enum identifier (value1,value2,…..,valuen);
The 'identifier' is a user-defined enumerated data type which can be used to declare variables
That can have one of the values enclosed within the braces – enumerated constants.
enum identifier v1,v2,…..,vn;
The enumerated variables v1,v2,…..,vn can have one of the values value1,value2,…..,valuen.
V1=value3;
V3=value5;
Example:
Enum day(Moday,Tuesday,…..,Sunday);
Enum day week_st,week_end;
Week_st=Monday;
Week_end=Sunday;

Defining Global Variables

- Global variables are known throughout the program.
- The variables hold their values throughout the programs execution. Global variables are created by declaring them outside of any function.It need not be declared in other function.
- A global variable is also known as ***external variable***.
- Global variables are defined above main () .

Syntax-

int n, sum;
int m,l;
char letter;
main()

{
}
it is also possible to pre-initialize global variables using the = operator for assignment.
Example:
float sum = 0.0;
int n= 0;
char c=`a';
main()
{
}

Printing Out and Inputting Variables

printf() and scanf() in C
The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

1. **printf() function -**

The printf() function is used for output. It prints the given statement to the console.
The syntax of printf() function is given below:

printf("format string",argument_list);

The format string can be %d (integer), %c (character), %s (string), %f (float) etc.

---

2. **scanf() function**

The scanf() function is used for input. It reads the input data from the console.

scanf("format string",argument_list);

Program to print cube of given number

.

```
1. #include<stdio.h>
2. int main()
3. {
4. int number;
5. printf("enter a number:");
6. scanf("%d",&number);
7. printf("cube of number is:%d ",number*number*number);
8. return 0;
9. }
```

Output
enter a number:5
cube of number is:125

Program to print sum of 2 numbers

```
1. #include<stdio.h>
2. int main(){
3. int num1,num2,sum;
```

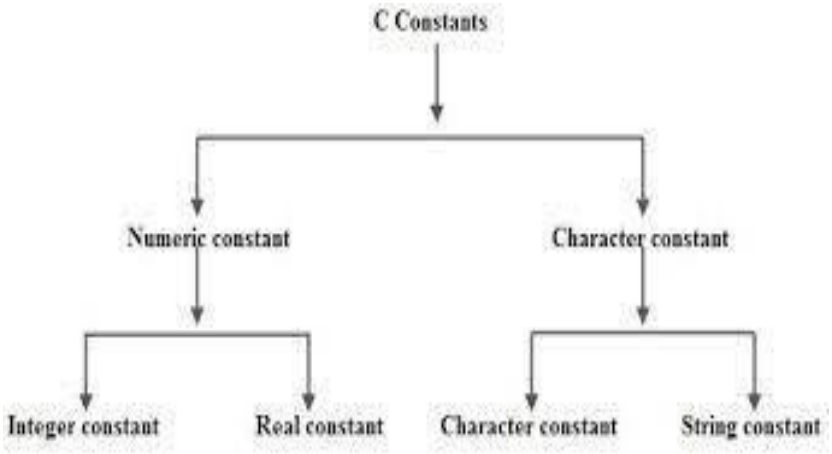
```
5.  printf("enter two numbers:");
6.  scanf("%d%d",&num1,&num2);
7.  sum=num1+num2;
8. printf("The sum is: %d\n", sum);

9.   return 0;
10. }
```

Output

enter two numbers-10
20
The sum is – 30

Constants/ literals

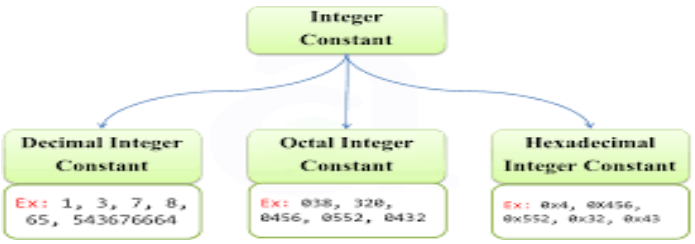
Fixed values that do not change during the execution of a program.



1. Integer Constants

An integer constant refers to a sequence of digits.

1. The digits of an *integer constant* can range from *0 to 9.*
2. A *decimal point* should not be present in an integer constant.
3. *Positive* or *negative integer constants* are also possible.



2. Real Constants

A *fractional component* or *exponentiation* of a number is represented by *a real or floating-point constant*. It can be expressed with a decimal point, the letter *"E"*, or the symbol *"e"* in exponential or decimal notation.

Example

```
1. #include <stdio.h>
2. int main() {
3. float real = 3.14;
4. printf("The real constant is: %f\n", real);
5. return 0;
6. }
```

Output

The real constant is: 3.140000

3. Character Constants

A *character constant* represents a *single character* that is enclosed in *single quotes*.

- **String Constants**

A *series of characters* wrapped in *double quotes* is represented by a *string constant*. It is a character array that ends with the null character *\0*.

Escape Sequence in C

- It is used in output functions
- Each one of them represents one character.

These characters combinations are known as escape sequences-

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

Escape Sequence Example-

```
1. #include<stdio.h>
2. Int main()
3. {
4. Int number=50;
5. printf("You\nare\nvery\\good");
6. return 0;
7. }
```

Output

You
are
very
good

SYMBOLIC CONSTANTS

Symbolic constant is defined as follows
#define symbolic_name value-of-constant
Example:
#define PI 3.14159
#define MAXMARKS 100

Rules
1) Symbolic names have the same form as variable names.
2) No blank space between the # sign define is permitted.
3) # must be the first character in the line.
4) #define statements must not end with a semicolon.
5) Symbolic names are not declared for data types. Its data type depends on the type of constant.

Operators

• A symbol use to perform some operation on variables, operands or with the constant is known as operator.
•Some operator required 2 operand or Some required single operand to perform operation.
• C operators can be classified into a number of categories.

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Assignment operators
5) Increment and decrement operators
6) Conditional operators
7) Bitwise operators
8) Other operators

1) Arithmetic Operators

Assume variable **A** holds 10 and variable **B** holds 20 then −

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Syntax-
```
#include------
int main(){
.....;
 Sum/ sub/mul/div;
 .....}
```

2) Relational Operators

Assume variable **A** holds 10 and variable **B** holds 20 then −

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.

!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Syntax

A==B
A<=B
A>B

3) Logical Operators

Assume variable **A** holds 1 and variable **B** holds 0, then −

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B)
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B)
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B)

4) Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation.

Operator	Meaning of operator
&	Bitwise AND operator
|	Bitwise OR operator
^	Bitwise exclusive OR operator
~	One's complement operator (unary operator)
<<	Left shift operator
>>	Right shift operator

Truth tables

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	1
1	0	0	1	1

5) Assignment Operators

The following table lists the assignment operators supported by the C language –

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /=
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

6) Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

7) INCREMENT & DECREMENT OPERATORS

Operators

1. Pre-increment & decrement-prefix ++ and --
2. Post-increment & decrement-postfix ++ and--

 Prefix operator adds 1 to the operand and then the result is assigned to the variable on left.

 Postfix operator first assigns the value to the variable on left and then increments the operator.

Rules

1. Increment and decrement operators are unary operators.
2. When postfix ++ (or --) is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then the variable is Increment (or decrement) by one.
3. When prefix ++ (or decrement) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using the new value of the variable.

Examples

m=5;

operator	Expression	Result	M value
Prefix ++	y=++m	y=6	m=6
Prefix --	y=--m	y=4	m=4
Postfix ++	y=m++	y=5	m=6
Postfix --	y=m--	y=5	m=4

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others;
for example, the multiplication operator has a higher precedence than the addition operator.
For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.
Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Conditionals (The if statement, The switch statement)

Conditional statements help you to make a decision based on certain conditions. These conditions are specified by a set of conditional statements having Boolean expressions which are evaluated to a Boolean value true or false. There are following types of conditional statements in C.

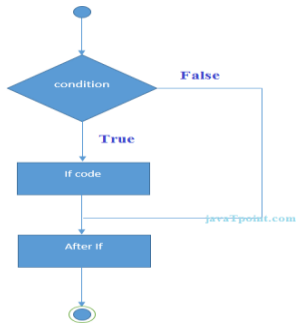
1. If statement
2. If-Else statement
3. Nested If-else statement
4. If-Else If ladder
5. Switch statement

If statement

The single if statement in C language is used to execute the code if a condition is true. It is also called one-way selection statement.

Syntax

```
1. if(expression){
2. //code to be executed
3. }
```



Flow Chart

Example

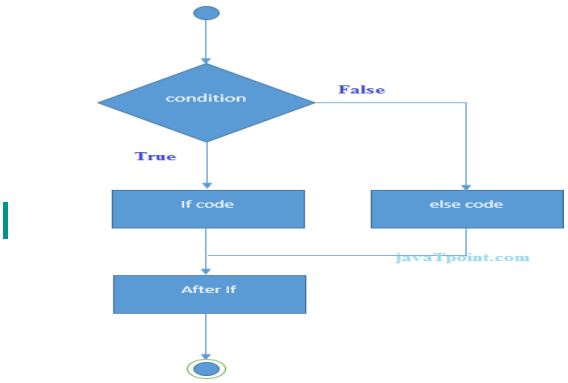
```
1) #include<stdio.h>
2) int main(){
3) int number=0;
4) printf("Enter a number:");
5) scanf("%d",&number);
6) if(number%2==0){
7) printf("%d is even number",number);
8) }
9) return 0;
10)  }
```

If-else statement

The if-else statement in C language is used to execute the code if condition is true or false. Itis also called two-way selection statement.

Syntax-

```
if(expression)
{
 //Statements
}
else
{
 //Statements
}
```



Flow Chart

How "if..else" statement works..

- If the expression is evaluated to nonzero (true) then if block statement(s) areexecuted.
- If the expression is evaluated to zero (false) then else block statement(s) areexecuted.

Example

```
1)  #include<stdio.h>
2)  int main(){
3)  int number=0;
4)  printf("enter a number:");
5)  scanf("%d",&number);
6)  if(number%2==0){
7)  printf("%d is even number",number);
8)  }
9)  else{
10) printf("%d is odd number",number);
11) }
12) return 0;
13) }
```

Nested If-else statement

The nested if...else statement is used when a program requires more than one test expression. It is also called a multi-way selection statement. When a series of the decision are involved ina statement, we use if else statement in nested form.

Syntax

```
if (condition1) {
  /* code to be executed if condition1 is true */
  if (condition2) {
    /* code to be executed if condition2 is true */
  } else {
    /* code to be executed if condition2 is false */
  }
} else {
  /* code to be executed if condition1 is false */
}
```

Example

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

2)  int main() {
3)  int num;

4)  printf("Enter a number: ");
5)  scanf("%d", &num);

6)  if (num > 0) {
7)  printf("%d is positive.\n", num);
8)  } else {
9)  if (num < 0) {
10) printf("%d is negative.\n", num);
11) } else {
12) printf("%d is zero.\n", num);
13) }
14) }

15) return 0;
16) }
```

If..else If ladder

The if-else-if statement is used to execute one code from multiple conditions. It is also called multipath decision statement. It is a chain of if..else statements in which each if statement is associated with else if statement and last would be an else statement.

Syntax

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

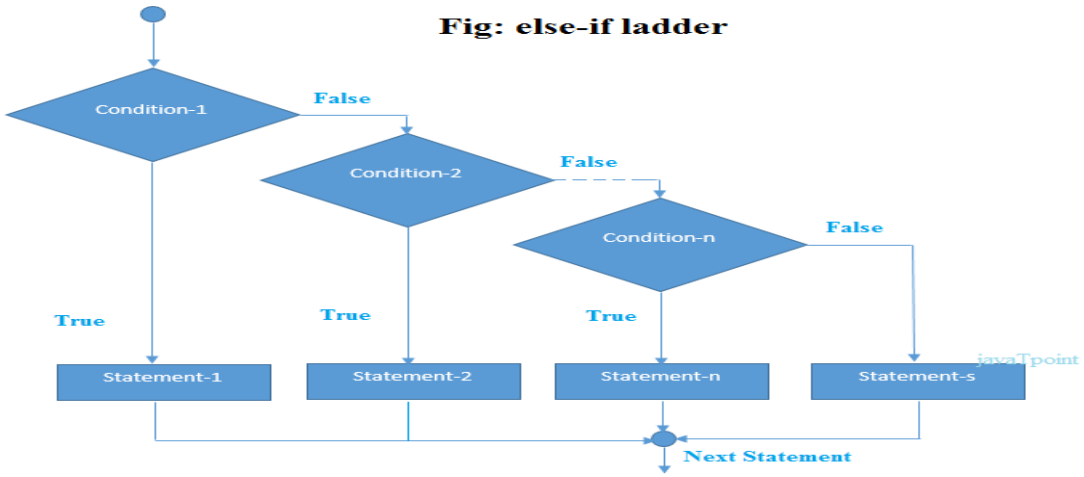


Fig: else-if ladder

Flow Chart

Example

```
1)  #include<stdio.h>
2)  int main(){
3)  int number=0;
4)  printf("enter a number:");
5)  scanf("%d",&number);
6)  if(number==10){
7)  printf("number is equals to 10");
8)  }
9)  else if(number==50){
10) printf("number is equal to 50");
11) }
12) else if(number==100){
13) printf("number is equal to 100");
14) }
15) else{
16) printf("number is not equal to 10, 50 or 100");
17) }
18) return 0;
19) }
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

Switch Statement

switch statement acts as a substitute for a long if-else-if ladder that is used to test a list of cases.A switch statement contains one or more case labels which are tested against the switch expression. When the expression match to a case then the associated statements with that casewould be executed.

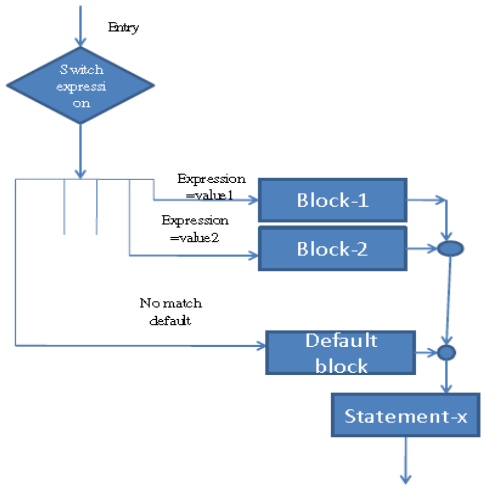
RULES FOR SWITCH STATEMENT

1. The switch expression must be an integral type.
2. Case labels must be constants or constants expressions.
3. Case labels must be unique. no two labels can have same value.
4. Case labels must end width colon.
5. The break statement transfers the control out of the switch.
6. The break statement is optional. Ie, two or more case labels may belong to the same statements.
7. The default label is optional. If present then it will executed when the expression does not find the matching case label.
8. There can be at most one default label.
9. The default may be placed anywhere but usually placed at the end.
10. It is permitted to nest switch statement.

Syntax

```
Switch (expression)
{
 case value1:
 //Statements
 break;
 case value 2:
 //Statements
 break;
 case value 3:
 //Statements
 case value n:
 //Statements
 break;
```

Flow Chart



Looping and Iteration

Iteration is the process where a set of instructions or statements is executed repeatedly for a specified number of time or until a condition is met. These statements also alter the control flow of the program and thus can also be classified as control statements in C Programming Language.
Iteration statements are most commonly know as loops.

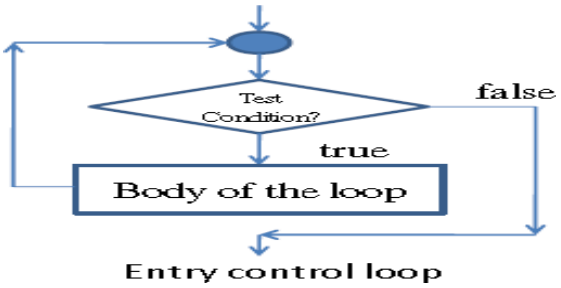
- A sequence of statement executed until some conditions for termination of the loop are satisfied.
- A program loop consists of two segments , body of the loop and control statement.
- The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.
- Depending on the position of the control statement in the loop, control structures may classified as the entry-control loop and exit-control loop.

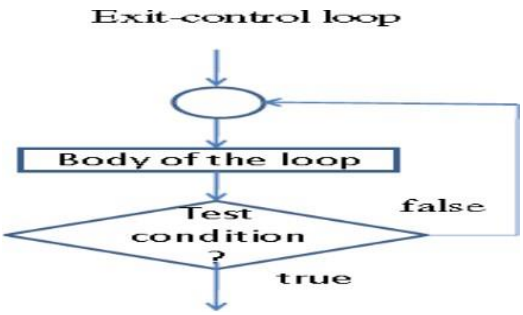
Entry-control loop

- In the entry-control loop, the control condition is tested before the start of the loop execution. If the condition is not satisfied then the body of the loop will not be executed.
- It is known as pre-test loops

Exit-control loop

- The control condition is tested at the end of the body of the loop and therefore the body is executed unconditionally for the first time.
- It is known a s post-test loops.





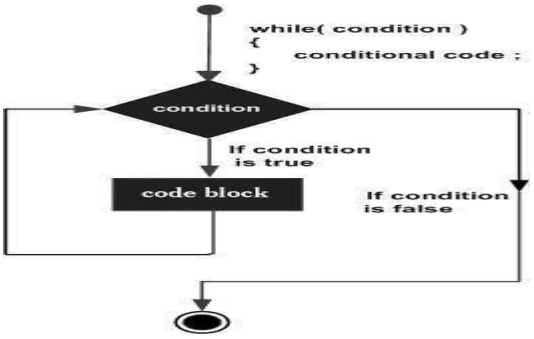
A looping process would include the following tour steps.

1. Setting and initialization of a condition variable.
2. Execution of the statements in the loop
3. Test for a specified value of the condition variable for execution of the loop.
4. Incrementing or updating the condition variable.

Looping statements

1. The while statement
2. The do statement
3. The for statement

- **WHILE STATEMENT**



While is entry-control loop.

Syntax

while (test condition)
{
Body of the loop
}

 The test condition is evaluated and if the condition is true then the body of the loop is executed.
 After execution of the body, the testcondition is once again evaluated and if it is true then the body of the loop is executed once again.
 The process of repeated execution of the body continues until the testcondition becomes false and the control is transferred out of the loop.
 On exit, the program continues with the statement immediately after the body of loop.
 The body of the loop may have one or more statements.

Example

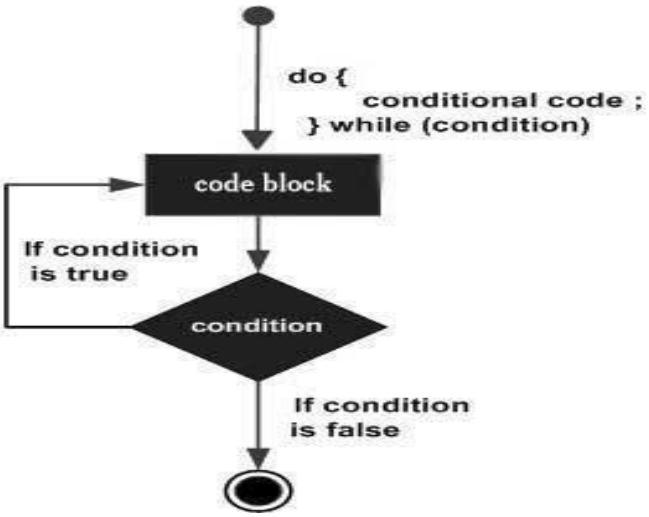
```
#include <stdio.h>

int main() {
  int i = 0;

  while (i < 5) {
    printf("%d\n", i);
    i++;
  }

  return 0;
}
```

Output

```
0
1
2
3
4
```

DO STATEMENT

Do statement is exit-controlled loop. The **do…while in C** is a loop statement used to repeat some part of the code till the given condition is fulfilled. It is a form of an **exit-controlled or post-tested loop** where the test condition is checked after executing the body of the loop



Syntax

```
do
{
body of the loop
} while (test condition);
```

Example

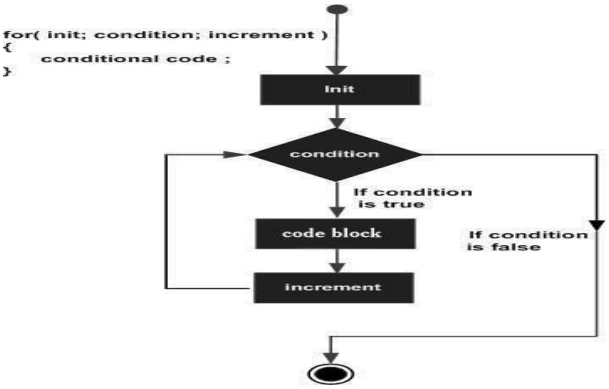
```
#include <stdio.h>

int main() {
  int i = 0;

  do {
    printf("%d\n", i);
    i++;
  }
  while (i < 5);

  return 0;
}
```

Output

```
0
1
2
3
4
```

- **FOR STATEMENT**

For statement is entry-controlled loop.



Syntax

for(initialization;test-condition,increment)
{
body of the loop;
}

The execution of the for statement is as follows -

1. Initialization of the control variable is done first, using assignment statement.
2. The **For loop** in C Language provides a functionality/feature to repeat a set of statements a defined number of times. The for loop is in itself a form of an **entry-controlled loop**.
3. Unlike the while loop and do…while loop, the for loop contains the initialization, condition, and updating statements as part of its syntax. It is mainly used to traverse arrays, vectors, and other data structures.

Example

```
1.  main()
2.  {
3.   int i, n, sum;
4.   printf("Enter the value of n\n");
5.  scanf("%d", &n);
6.  sum=0;
7.  for (i=1;i<=n;i++)
8.  {
9.  sum = sum + i;
10. }
11. printf("sum of natural numbers up
12. to %d = %d", n, sum);
13. }
```

 The for statement allows the negative increments.

Example

```
for(x=9;x>=0;x--)
printf("%d",x);
printf("\n");
```

 The above loop is executed 10 times, but the output would be from 9 to 0 instead of 0 to 9.
 Since the conditional test is always performed at the beginning of the loop, the body of the loop may not be executed at all, if the condition fails at the start.

Example

```
for(x=9;x<9;x--)
printf("%d",x);
```

 The above loop never executed because the test condition fails at the beginning itself.

COMPARISON OF THE THREE LOOPS

for	while	do

for(n=1 ;n<=10;n++)
{
 	
---------- 	
---------- 	
---------- 	
 	
}
	n=1;
	while(n<=10)
	{
	---------- 	
	---------- 	
	---------- 	
	n=n+1;
	}
		n=1;
		do
		{
		---------- 	
		---------- 	
		---------- 	
		n=n+1;
		}while(n<=10);

Example

```
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Example

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Example

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

More than one variable can be initialized at a time in the for statement. *Example*

```
p=1;
for(n=0;n<10;n++)
```

Can be written as

```
for(p=1,n=0;n<10;n++)
```

1. **Test condition may have any compound relation and the testing need not be limited only to the loop control variable.** ***Example***

```
sum=0
for (i=0;i<10&& sum <100;i++)
{
sum=sum+i;
}
```

2. **One or more section can be omitted in for loop.**----

```
m=5;
for(;m<10;)
{
printf("%d\n",m);
m=m+2;
}
```

3. **We can set up the time delay loops using for statement. Example**

```
for(j=1;j<1000;j++);
```

- **Nesting of for loops**

One for statement within another for statement is called nesting of for.

Syntax

```
for (i=0;i<n;i++)
 {
for(j=1;j<m;j++)
 {
----------
----------
 }
----------}
```

Example

Program to read the marks by n students in various subject and print the total.

```
1.  #include<stdio.h>
2.  #include<conio.h>
3.  void main()
4.  {
5.    int n, m, i, j, rno, marks, total;
6.    printf("Enter the no. of students &
      subjects\n");
7.  scanf("%d%d",&n,&m);
8.  for(i=1;i<=n;i++)
9.  {
10.  printf("Enter the roll no. :");
11.  scanf("%d",&rno);
12.  total=0;
13.  printf("Enter marks of %d subjects for
      roll
14.  no %d\n", m, rno);
15. for(j=1;j<=m;j++)
16. {
17. scanf("%d",&marks);
18. total= total+marks;
19. }
20. printf("total marks =%d" ,total);
21. }
22. getch();
23. }
```

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements.

Sr.No.	Control Statement & Description
1	**break statement** Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	**continue statement** Causes the loop to skip the remainder of its body and immediately retest its condition priorto reiterating.
3	**goto statement** Transfers control to the labeled statement.

1. break statement in C

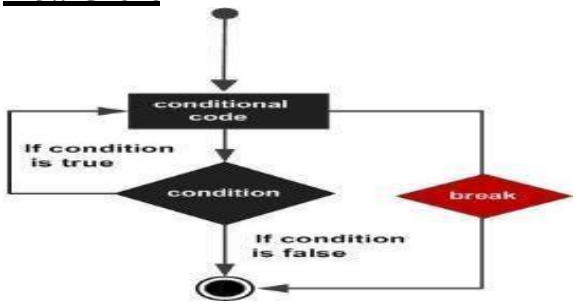
The **break** statement in C programming has the following -

The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops.

Syntax

```
//loop of switch case
break;
```

Flow Chart



Example

```
#include <stdio.h>
int main() {
  int i;
  for (i = 0; i < 10; i++) {
    if (i == 4) {
      break;
    }
    printf("%d\n", i);
  }
   return 0;
```

Output

```
0
1
2
3
```

2. Continue statement in C

The continue statement in C programming works somewhat like the break statement. Insteadof forcing termination, it forces the next iteration of the loop to take place, skipping any codein between.

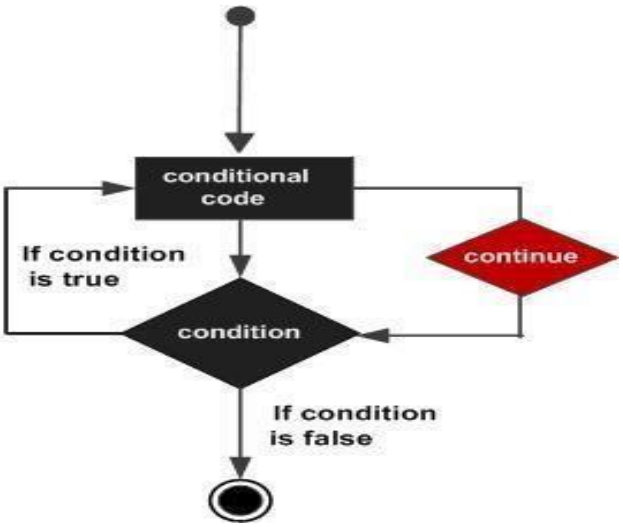
For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control to pass to the conditional tests.

Syntax

The syntax for a **continue** statement in C is as follows −

```
continue;
```

Flow Chart



Example

```
#include<stdio.h>
int main()
{
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
return o;
}
```

Output

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

3. Goto statement in C

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

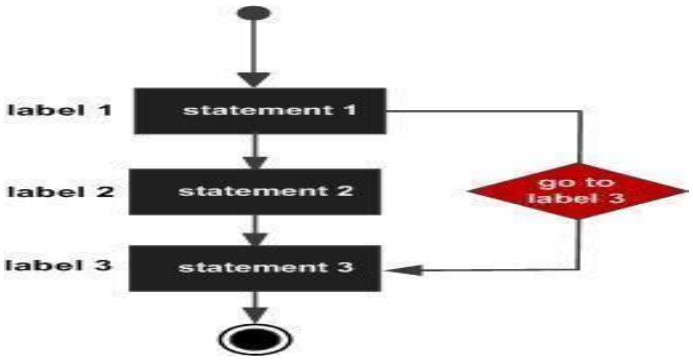
NOTE − Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.Syntax

Syntax

```
goto label;
..
.
label: statement;
```

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

Flow Chart -



Example

```
1.  #include <stdio.h>
2.  int main () {
3.  /* local variable definition */
4.  int a = 10;
5.  /* do loop execution */
6.  LOOP:do {
7.  if( a == 15) {
8. /* skip the iteration */
9. a = a + 1;
10. goto LOOP;
11. }
12. printf("value of a: %d\n", a);
13. a++;
14. }while( a < 20 );
15. return 0;
16. }
```

Output −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```