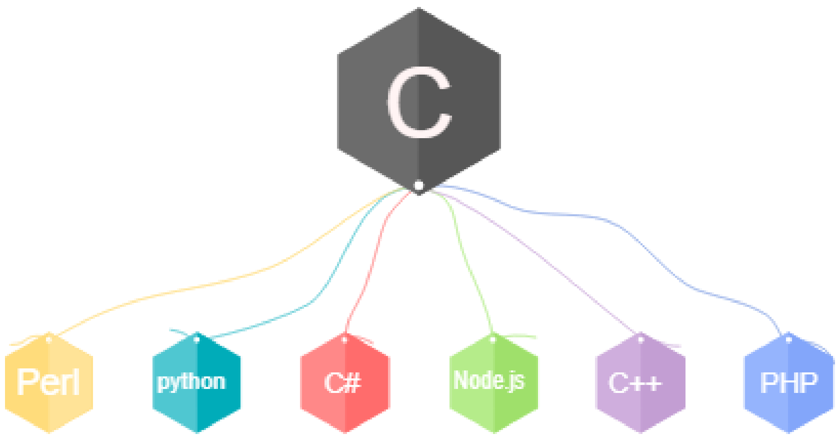
C Programming

Welcome to the world of C programming, C is a general purpose, server side, structured programming language developed by Dennis Ritchie at Bell Laboratories in the early 1970's. A story started with Common Programming Language (CPL) which is designed by Martin Richards. Later it is known Basic Combined Programming Language (BCPL). BCPL allow the user to direct access to the computer memory. Ken Thompson at BELL Laboratories wrote his own variant of BCPL and named as B. C is a middle level language which is an outgrowth of BCPL and B. C reduces the gap between high level language and low level language. Hence C is known as Middle Level Language. In year of 1983 an ANSI standard for C emerged. Thus ANSI C is Internationally accepted.

C History

There are several programming languages like C++, java, C#, Python, Perl etc. All these languages are partially inherited from C. Even though, C is considered to be the most common programming language in the world today. C is a core language, having knowledge in C will help you to learn other programming language with ease. C is a stable programming language as it is not revised since 1983 (year C recognized by ANSI standard).



Applications of C

- 100% Linux kernel is written using C.
- 90% of Unix operating System code are written in C.
- Both iOS and OSX are written using a combination of C and Objective C.
- Google.com and youtube.com using C as a server side(back end) programming language.
- C is widely used in Embedded system application.
- MATLAB is partially written in C.
- Python interpreter is completely written in C.
- Though javascript is client side programming language, node.js become server side programming language only because of C and C++.

C Extensions

In operating systems, a file name extension is an optional addition to the file name in a suffix of the form ".xyz" where "xyz" represents a limited number of alphanumeric characters depending on the operating system. The file name with extension allows a file's format to be described as part of its name so that users can quickly understand what type of file it is without having to (open).

Example

- .c stands for c file
- .java stands for java file
- .cpp stands for c++ file
- .py stands for python file

Extensions in C Programming

Source code file extension (.c)

Source code are versions before running of program by the compiler.

Header file extension (.h)

Header files contain predefined function declaration and various preprocessor statements. Header files allows source code to access externally defined function

Object code file extension (.obj or .o)

Object files are versions after running of program by the compiler. Object file consist of statements and function definition that are defined in binary form. Object files are not executable by themselves. In some operation system the file extension for Object file is .o, but in windows they often ends with .obj

Binary Executables file extension (.exe)

Binary Executables file produced as the output of the program called the linker. The linker link together a number of object files to produce a binary file that can be directly executed. In UNIX operating system Binary executable files does not have any extensions. In other operating system Binary Executables file have .exe as extension.

C Compiler

What Compiler Means?

Complier is nothing more than translating one thing to another to do some useful work.

Compiler in C

In C, **Compiler** is a kind of translator that translate source code (.c) into machine code (.obj). This machine code is used by your computer's microprocessor to do the work as per the command written in source code by the programmer.

Error In Source Code

During translation process the **Compiler** reads the source code (.c) and checks the syntax errors. If any error, the **Compiler** generates an error message, which is usually displayed on a screen. In case of any errors a object code will not be created by the **Compiler**. Until all the errors are rectified.

Where Compiler Resides

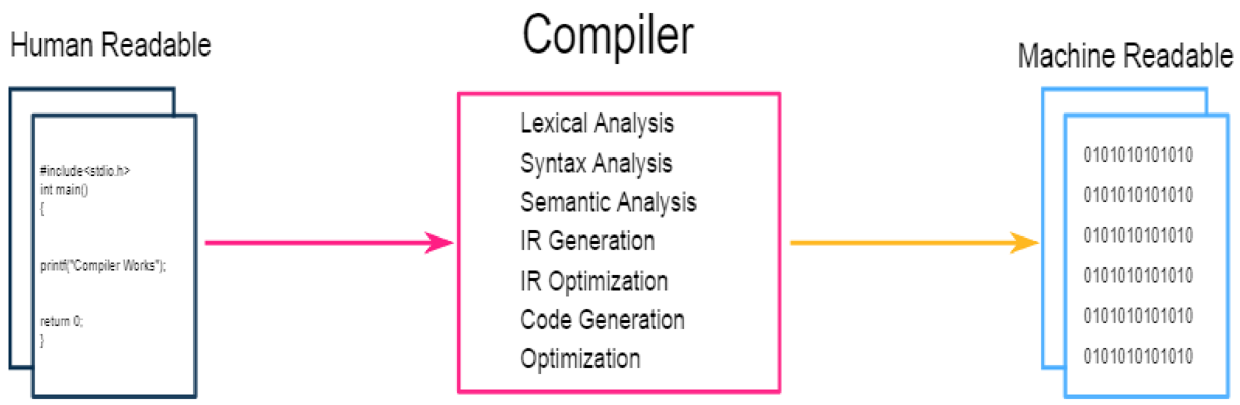
The **Compiler** usually resides on a disk and not in Random Access Memory (**RAM**).

Does Source Code Actually Needed?

Once the program has been compiled the resulting machine code is saved in an executable file, which can be run on its own at any time. Once the executable file (.obj) is generated there is no need of actual source code file. Execution file (.obj) generation will not affect a source code file (.c).

How Compiler Works

The **Compiler** replaces all statements with a series of machine language instruction. A single statement is a line terminated by semicolon. The following diagram clearly demonstrate how **compiler** translate source code to a machine code.



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

printf("Compiler Works");

return 0;
}
```

Interpreter

What Interpreter Means?

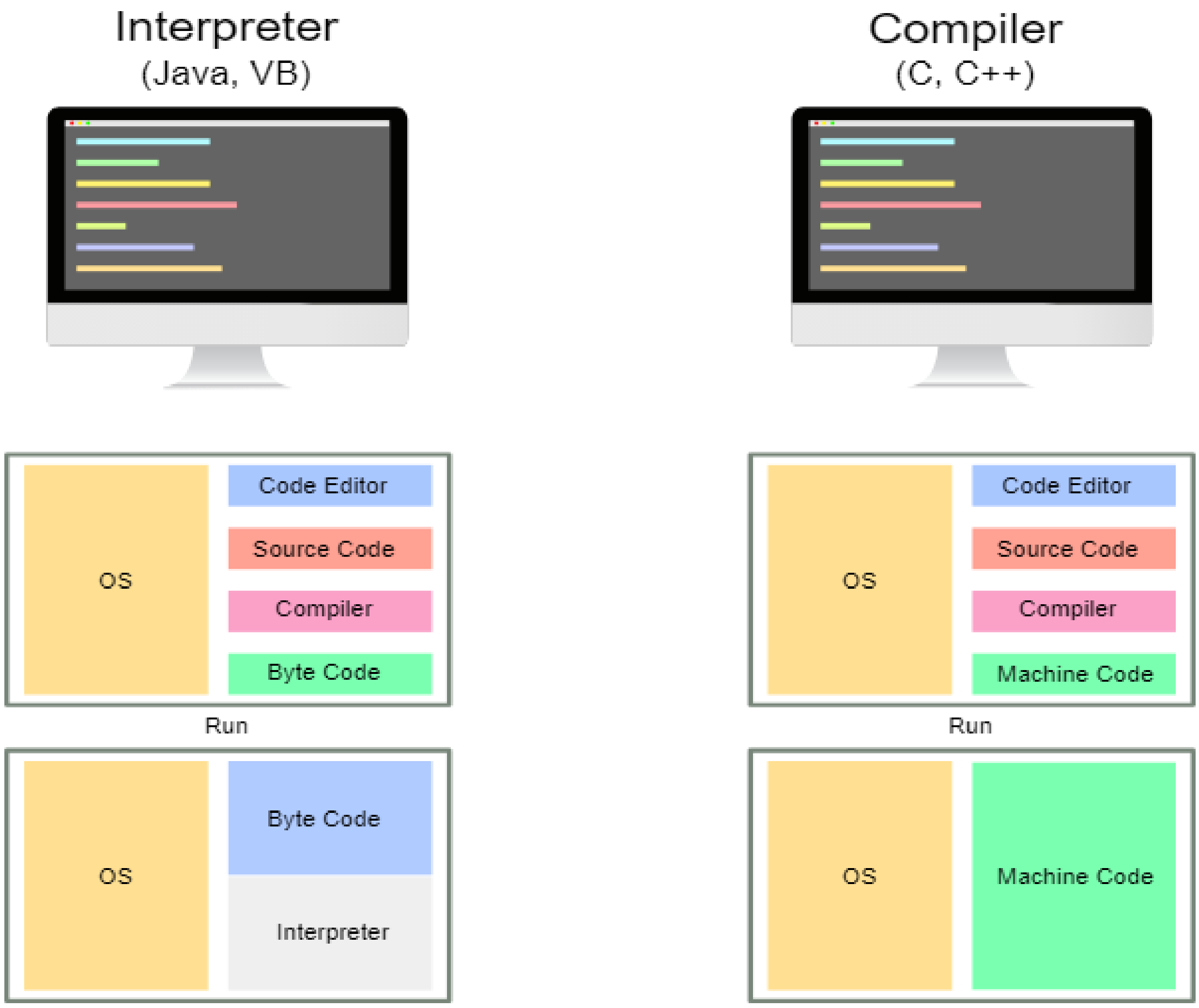
Interpreter is also a language translator that translates a source code(.c) into a machine code(.obj). But it converts only one source code instruction to a machine code instruction at a time. Therefore a compiler processes the entire program in one go while interpreter processes one instruction at a time.

Error In Source Code

The interpreter checks for errors in each source code instruction and indicates it to the programmer instantly. Thus clearing errors for a programmer is not that much headache. As interpreter execute the instruction, only after the error in a instruction is corrected. This process continues until the last line of a program is reached.

How Interpreter Works

Interpreter replaces a single statements to a single machine language instruction at a time. A single statement is a line terminated by semicolon. The following diagram clearly demonstrate how interpreter translate source code to a machine code.



Compiler vs Interpreter

Compiler	Interpreter
Scans entire program before translating it into object code(machine readable).	Translates one instruction at a time.
Slow in debugging.	Fast in Debugging.
Execution time is less.	Execution time is more.
C, C++ are the best examples which uses compiler.	Python, Ruby, Matlab and Perl are the best examples which uses interpreter.

C Basic Program

Basic Program

Here we starts with very basic C program

basicprogram.c

```
#include <stdio.h> //header file section
int main() //main section
{
printf("I will display "); /*Output statement*/
return 0; /*Program ends here*/
}
```

Header File Section

C contains many inbuilt functions which are all predefined by its appropriated header files. Here we used printf() inbuilt function which is defined in **stdio.h** header files or otherwise said to be library. The file should be included by using **#include** directive e.g.) #include<stdio.h>.

Main Section

Every C program must contain **main()** function which tells the compiler that the program starts from here. Empty paranthesis must after every main in C programming. main function cannot have any parameters.

Executable Section

Executable section contains single (or) set of statement for example printf is an executable statement.

8 Programming Rules In C

- **Rule 1:** All statement should be written in lowercase letters.
- **Rule 2:** Uppercase letter can be used only for symbolic constants.
- **Rule 3:** Blank spaces may be inserted between the words. This improves the readability of statements.
- **Rule 4:** Blank spaces should not be allowed while declaring variables, constants, keyword, function.
- **Rule 5:** Programmer can write the statement anywhere between the "2 braces".
- **Rule 6:** Programmer can write one or more statement in one line, separating that by semicolon (;).
- **Rule 7:** Opening and closing "braces" must be balanced.

- **Rule 8:** Program must contain a main function.

C Data Types

What Is Data Type?

Data Type is nothing more than what type the data belongs to. In other words Data types are the storage format to store a data.

Why Data Types?

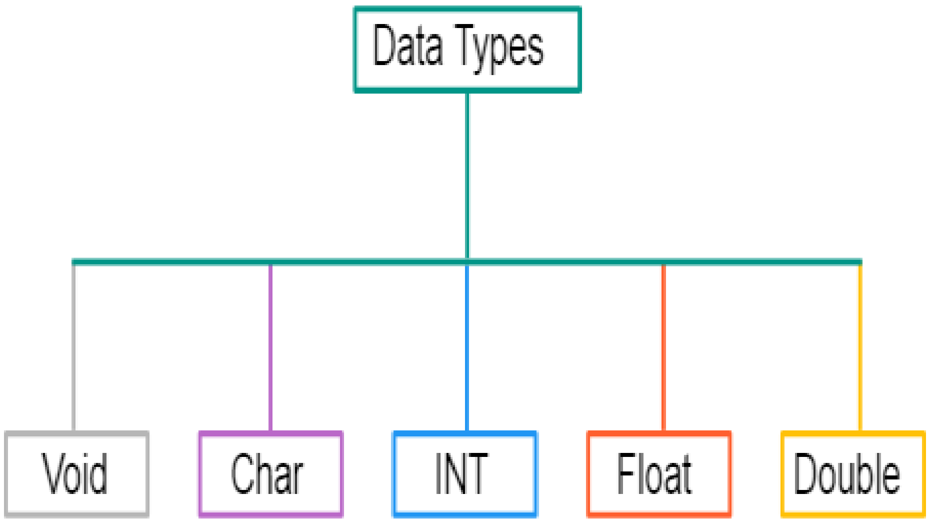
Data types are widely used in 90's programming languages like C, java and much more. Data types tells the compiler how to treat the data while performing any mathematical operation in it. Modern Compliers and Interpreters are capable of finding the types of data automatically so that users are provided with no-headache of mentioning data types. Languages like python, javascript, php etc are datatype free programming language.

C programming have minimal set of basic data types. Complex data types can be built using these basic data types. Data types are also known as primitive types. In C programming, the memory size of data types may change according to 32 (4 bytes) or 64 (8 bytes) bit operating system.

Types of Data Types

In C, **5** basic data types to define data.

- **int** represent integer.
- **char** represent character.
- **float** represent floating point.
- **double** represent 2 * integer.
- **void** represent valueless.



Data Type's Size and Range

The following table will demonstrate Data type's size and Data type's range for clarity view.

Data Type	Size (bits)	Range
void	1 byte or 8 bits	nothing
char	1 byte or 8 bits	-128 to 127
int	2 byte or 16 bits **||** 4 byte or 32 bits	-32768 to 32767 or -2,147,483,648 to 2,147,483,647
float	4 bytes or 32bits **||** 8 byte or 64 bits	3.4e^{-38} to 3.4e^{+38}
double	8 byte or 64 bits	1.7e^{-308} to 1.7e^{+308}

Conceptual Data Types

There are 2 types of conceptual data types in C programming

Derived data type	Enumeration Data Type
array, pointer, structure, union	enum

Format Specifiers

Format Spectifiers in Programming language tells us which type of data to store and which type of data to print or to output.

Data Type	Keyword	Format Specifier
character	char	%c
signed integer	int	%d or %i of %l
unsigned integer	unsigned int	%u or %lu
long double	long double	%ld
string	char	%s
floating point	float	%f
double floating point	double	%lf

Char Data Type in C

Char is a Data Type to store a single character. Generaly, **char** Data Type occupies 1 bytes i.e) 8 bits.

char Data Type Program

chardatatype.c

```
#include <stdio.h>
int main()
{
char a = 'b';
printf("The character of a = %c ",a);
return 0;
}
```

```
The character of a = b
```

Note:

Here char data type store a character **b** in variable a.

Integer Data Type

Integer is a Data Type to store an integer value. Generaly, Integer Data Type occupies 2 bytes or 4 bytes its highly depend on your operating system bit rates.

Integer Data Type Program

intdatatype.c

```
#include <stdio.h>
int main()
{
int a = 8;
printf("The value of a = %d ",a);
return 0;
}
```

```
The value of a = 8
```

Note:

Here int data type store an integer value **8** in variable a.

Float Data Type

Float is a Data Type to store decimal point values. Generaly, Float Data Type occupies 4 bytes.

Float Data Type Program

floatdatatype.c

```
#include <stdio.h>
int main()
{
float a = 8.987654321;
printf("The value of a = %0.9f",a);
return 0;
}
```

```
The value of a = 8.987654686
```

Note:

Here float data type store an decimal point value **8.987654686** in variable a. Note that float data types precision is limited up to **6** digit after a decimal point. After six digit some garbage value will be displayed by the compiler. **%0.9f** represent, Display 9 numbers after decimal point.

Double Data Type

Double is a Data Type to store decimal point values with more accuracy than float. Generaly, Double Data Type occupies 8 bytes.

Double Data Type Program

doubledatatype.c

```
#include <stdio.h>
int main()
{
double a = 8.987654321;
printf("The value of a = %lf",a);
return 0;
}
```

```
The value of a = 8.987654321
```

Note:

Here double data type store an decimal point value **8.987654321** in variable a with more precision.

C Keywords

Basic Program

Keywords are those words which tells the compiler what to do . In C Programming, keywords are otherwise said to be reserved words.

Keywords Rules

- Keywords are always in lowercase letters.
- Keywords cannot be changed by a programmer.
- Keywords cannot be used as variable name, function name, array name by a programmer.

Keywords Bad Practices

Keywords in Uppercase letters can be used as identifiers but considers as a poor programming practice.

Program For Verification

keywords.c

```
#include <stdio.h>
int main()
{
int BREAK = 1;
printf("The value of BREAK = %d ",BREAK);
return 0;
}
```

```
The value of BREAK = 1
```

Note:

Here we used **BREAK** as an identifier, but the compiler haven't through any error instead it replied with an output.

List Of All Keywords

There are **32** keywords defined in C Programming.

auto	extern	sizeof
break	float	static
case	for	struct

char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while
enum	signed	-

C Identifiers

What is Identifier?

An identifier is a **name** that identifies unique things. here things may be a person, object, idea or anything. The following diagram clearly represents the concept of identifier.

Identifiers In C

- Identifier as the name suggest are used identify elements in C Programming.
- Identifiers is a sequence of characters and digits created by a programmer to identify various program elements.
- Identifiers are the names of variables, arrays, functions, structure.
- C Programming language is case-sensitive, so uppercase, lowercase, camelcase letters will be treated differently.
- Here is some example are: firstname, FIRSTNAME, FirstName will be treated differently.
- But in general practice the lowercase and CamelCase letters can be used as a variable name, function name, array name and structure name, while UPPERCASE letters are used for symbolic constants.
- Programmer can use underscores `_` in between identifiers. e.g.) first_name.

Identifier C Program

identifier.c

```
#include <stdio.h>
int main()
{
int number = 25;
int Number = 47;
printf("The number is %d ",number);
printf("\nThe Number is %d ",Number);
return 0;
}
```

Did You Know?

Once identifier is declared under any datatype it cannot be redeclared even my its current datatype itself.

Program For Verification

identifier.c

```
#include <stdio.h>
int main()
{
int number = 25;
int Number;
int Number = 47.00;
printf("The number is %d ",number);
printf("\nThe Number is %d ",Number);
return 0;
}
```

Note:

The above program will cause error as identifier **Number** is redeclared.

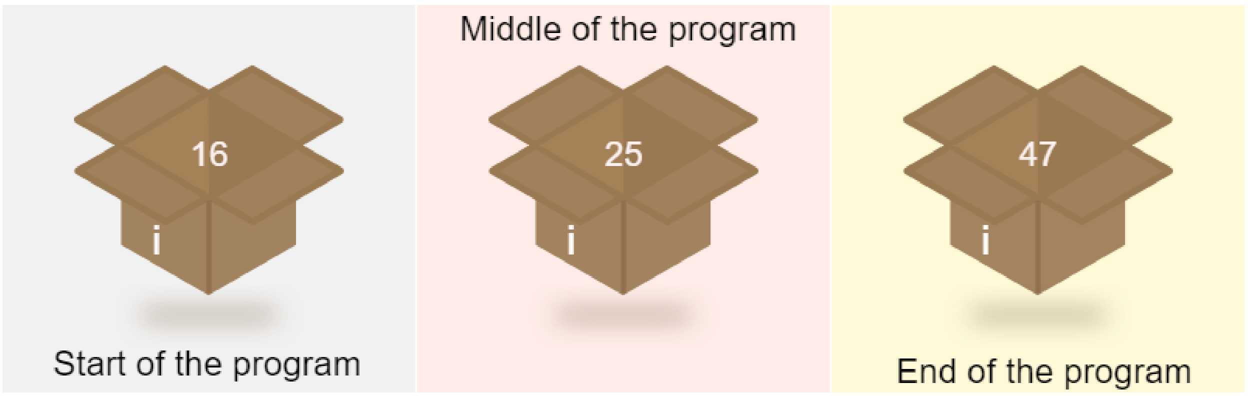
C Variables

What Is Variables?

Generally, Variables are things which are all changable. We encounter several variables in our day-to-day life.

Example

- **Time** is variable which changes for every single seconds.
- **Climate** is variable which changes as per season.
- **Rheostat** is variable which changes to increase or to decrease circuit resistance.
- **Speed** is variable, either one can accelerate to increase the speed or decelerate to decrease the speed.



Variables in C

In C programming, variable is an identifier for a memory location for storing data. Once data is stored with a variable name, it can be retrieved by its variable name for various computations in a C program. A data that can be changed by assigning a new data to the variable. Variable can be numerical quantity or character constant.

Variable Purpose

Different operations can be performed using data stored in variables during program execution Such opereations are Addition, Subtraction, Mutiplication etc

A data can be accessed later in the program by simply referring the name of the variable.

Rules for declaring Variables

- Variable name should not be a C keyword.
- Variable must be declared before it can be used.
- Variable name should not be starts with a digit.
- Variable name may be a combination of UPPERCASE and lowercase. e.g.)

firstName

- Variable name cannot be repeated within the same scope.
- A data type (int, float, char) is established when the variable is defined. Once defined the type of variable cannot be changed

Variables C Program

variables.c

```
#include <stdio.h>
int main()
{
int number = 25;
int Number = 47;
printf("The number is %d ",number);
printf("\nThe Number is %d ",Number);
return 0;
}
```

- The number is 25
- The Number is 47

Note:

The values 25 and 47 are stored in a memory location with its unique variable name **number** and **Number** respectively.

Helpful Tips

- Variable can be declared at the start of block of code which is more helpful for later reference
- Variable name may be declared based on the meaning of the operation. e.g.) result, average, sum etc.

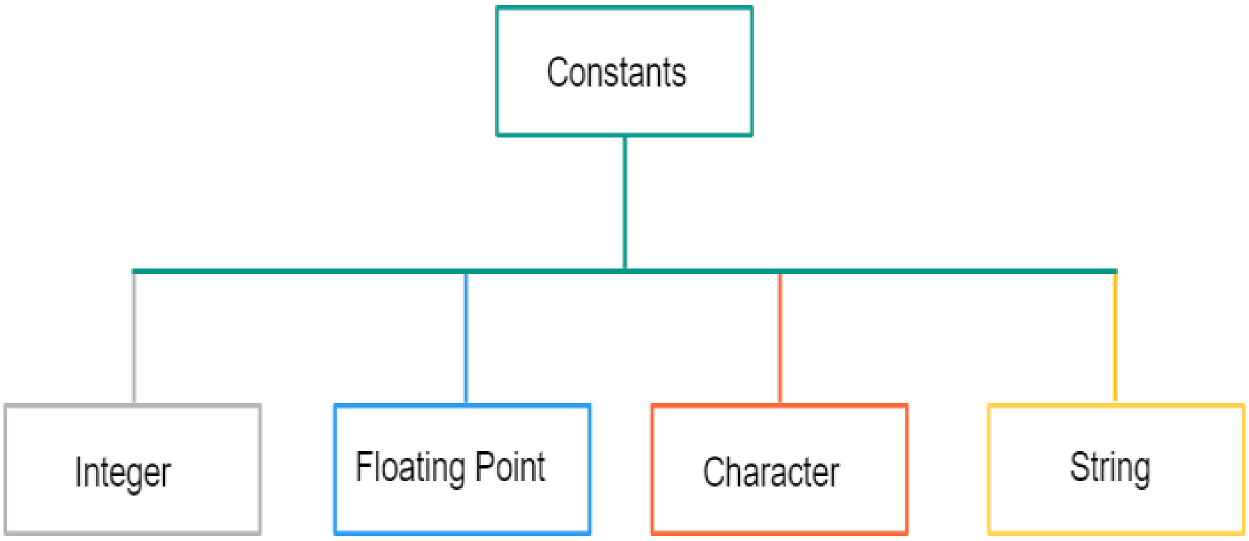
C Constants

What is Constant Variable?

Constant variable, does not change its value during the execution of a program. A constant is a data value written by a programmer. Constants may be belonging to any of the data type(int, float, char). There are **4** basic types of constants. They are

- Integer Constant.
- Floating Point Constant.
- Character Constant.
- String Constant.

Constants Types



Integer Constants

- An integer constants consists of sequence of Integer numbers from **0 to 9**.
- Decimal points, black spaces, commas cannot be included within an integer constants.
- An Integer constants can either be positive or negative or zero.
- The number without sign can be assumed as positive
- Integer constants can be classified into **3** ways . They are
    - Decimal Integer Constants.
    - Octal Integer Constants.
    - Hexadecimal Integer Constants.

Decimal Integer Constants

A Decimal integer constants consist of any combination of digits from 0 to 9. A Decimal integer constants can contain two or more digits, but first digit should not be 0. Base value of decimal integer is 10.

Valid Decimal integer Constants in C Programming

0	12	856	456844

Invalid Decimal Integer Constants

45,565	invalid character (,)
25.0	invalid character(.)
1 2 3	invalid character(blank space)
12-23-34-456	invalid character(-)
051	first digit cannot be a zero

Octal Integer Constants

An Octal integer should starts with 0 in order to identify the constant as an octal integer number. An Octal integer constants consist of any combination of numbers between 0 to 7. Base value of Octal integer constants is 8.

Valid Octal Integer Constants in C Programming

0	012	0856	0456844

Invalid Octal Integer Constants

4725	does not starts with 0
01491	invalid digit(9)
012.23	invalid character(.)
051-43	invalid character(-)

Hexadecimal Integer Constants

A Hexadecimal integer constants can be a combination of any numbers in the range of 0 to 9 and a to f or A to F. Here A to F denotes 0 to 15. A Hexadecimal integer constants should starts with either 0x or 0X. Base value of Hexadecimal integer constants is 16.

Valid Hexa Integer in C Programming

0x	0X12	0x8A6	0x4a84b4

Invalid Hexa Integer Constants in C Programming

4725	does not starts with 0x
0x1H9	invalid charcter(H)
0X12.23	invalid character (.)
0X51-43	invalid character(-)

Unsigned or Long Integer Constants

Unsigned or Long Integer Constants may exceed the range of Integer. Unsigned or Long Integer Constants are identified by the suffix u or l. Here

- u represents unsigned int.
- L represents long int.

Example of unsigned and long integer constants

4541U	unsigned int
4558L	long int

Floating point constant

A floating point constant consist of integer part, decimal point and fractional part. A floating point constant contains exponential field e or E followed by an integer. A normal decimal point value will be treated as double until a suffix f is added to the decimal point value. Decimal point value using double will be more accurate than float. Adding suffix l or L to the Decimal point value will be treated as long double. A floating point constant with an exponent is essentially the same as scientific notation, except that base 10 is replaced by e or E. Exponential notation is useful in representing the numbers whose magnitudes are very large or very small.

Valid and Invalid Floating Point Constants

Valid Floating Point Constants	Invalid Floating Point Constants
47.00	47
0.4	0 5

What is Mantissa and Exponent?

Consider the number 257.47. This can also be return as 0.25747E^3. The sequence of digits after decimal point (25747) is a mantissa. The power of E (3) is known as exponent.

Valid and Invalid Exponential notation

Valid Exponential notation	Invalid Exponential notation
$542.254E^6$	$7.5.3E^4$
$542.254e^6$	$7.5.3e\quad^4$

Character Constants in C Programming

A character constant is a single character enclosed within single quotes `''`. A character constant also represented with a single digit or a single special character or white space enclosed within a single quotes. All PC make use of ASCII(American Standard Code for Information Interchange) character set, where each integer character is numerically encoded. A character constant occupies a single byte.

Valid and Invalid Character Constants

Valid Character Constants	Invalid Character Constants
'a'	'ab'
'1'	'22'

String Constants

A string constant is a sequence of characters enclosed within double quotes `""`. The Characters may consists of letters, digits, escape sequences and spaces. A string constant always ends with null (/0). A string constant occupies a two bytes.

Valid and Invalid String Constants

Valid String Constants	Invalid String Constants
'Hello Programmer'	'Hello Programmer"
"Hello Programmer"	"Hello Programmer'

C Escape Sequences

What Escape Sequences Means?

In C Programming, the word escape sequences means to**escape** the character from the **sequences** of words and give special meaning to the escaped character.

Example

Let us consider the following word "hello**\n**world". Clearly, **n** is the character which escapes from the sequences of word, helloworld and a special meaning is given to n i.e) new line.

Escape Sequences Facts

- An Escape sequences are non-printing characters.
- An Escape sequences are certain characters associated with \ (backslash).
- An Escape sequences always begin with \ (backslash) and is followed by one character.
- An Escape sequences are mostly used in **printf()** statement.

Escape Sequences Table

There are 12 escape sequences in C Programming. They are

Escape Sequence	Use	ASCII Value
\n	New Line	010
\t	Horizontal tab	009
\b	Backspace	008
\r	Carriage Return	013
\a	Alert Bell	007
\'	Single quote	039
\"	Double quote	034
\?	Question	063
\\\\	Backslash	092
\f	Form feed	012
\v	Vertical tab	011
\0	Null	000

\n Escape Sequence

newline.c

```
#include <stdio.h>
int main()
{
printf("Hello \nProgrammer ");
return 0;
}
```

- Hello
- Programmer

Note:

Statement followed by **\n** will be printed in New Line.

\t Escape Sequence

horizontalspaces.c

```
#include <stdio.h>
int main()
{
printf("Hello\tProgrammer ");
return 0;
}
```

```
Hello    Programmer
```

Note:

Statement followed by **\t** will be printed after 8 space.

\b Escape Sequence

backspace.c

```
#include <stdio.h>
int main()
{
printf("Hello\b Programmer ");
return 0;
}
```

```
Hell Programmer
```

Note:

\b will backspace a character.

\r Escape Sequence

returncarriage.c

```
#include <stdio.h>
int main()
{
printf("\nab");
printf("\bsi");
printf("\rha");
return 0;
}
```

```
hai
```

Note:

- **\n** will print its defined characters in New line. ie) ab.
- **\b** will backspace a character. ie) asi.
- **\r** will replace the existing characters by its defined characters(ha) ie) hai.

\a Escape Sequence

alarm.c

```
#include <stdio.h>
int main()
{
printf("This will make a bell sound\a ");
return 0;
}
```

```
This will make a bell sound
```

Note:

\a will audio you a bell sound 🔔.

\' Escape Sequence

singlequote.c

```
#include <stdio.h>
int main()
{
printf("\'Hello Programmer\' ");
return 0;
}
```

```
'Hello Programmer'
```

Note:

\' will print single quote '.

\" Escape Sequence

doublequote.c

```
#include <stdio.h>
int main()
{
printf("\"Hello Programmer\" ");
return 0;
}
```

```
"Hello Programmer"
```

Note:

\\"\\" will print double quote ".

\\? Escape Sequence

questionmark.c

```
#include <stdio.h>
int main()
{
printf("How are you\? ");
return 0;
}
```

```
How are you?
```

Note:

\\? will print? (Question mark).

\\\\ Escape Sequence

backslash.c

```
#include <stdio.h>
int main()
{
printf("C\\learn\\from\\2braces.com ");
return 0;
}
```

```
C \ learn\ from\ 2braces.com
```

Note:

\\\\ will print a single backslash (\\).

\\f Escape Sequence

formfeed.c

```
#include <stdio.h>
int main()
{
printf("\f - female sign. ");
return 0;
}
```

```
♀ - female sign.
```

Note:

\f will print female sign (♀).

\v Escape Sequence

formfeed.c

```
#include <stdio.h>
int main()
{
printf("\v - male sign. ");
return 0;
}
```

```
♂ - male sign.
```

Note:

\v will print male sign (♂).

\0 Escape Sequence

formfeed.c

```
#include <stdio.h>
int main()
{
printf("Hello \0 Programmer ");
return 0;
}
```

```
Hello
```

Note:

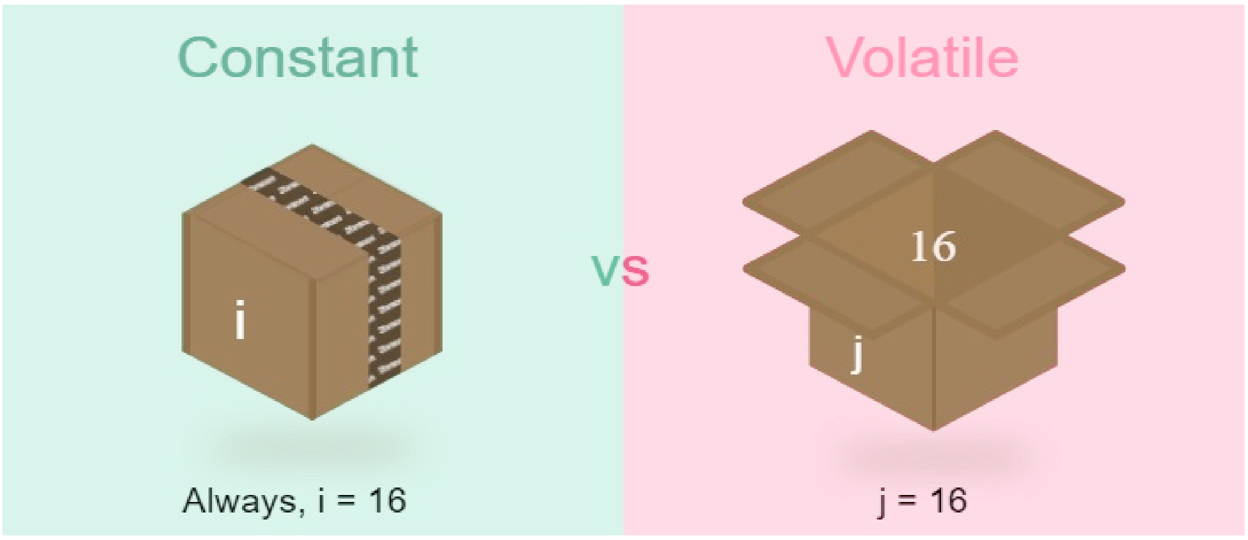
The statement followed by **\0** will not display.

Constants and Volatile

Constants

Things which are all unchangable are said to be **constant** whereas things which are all changable are said to be **volatile**. The following diagram clearly represents the relationship between constant and volatile.

In the following diagram sealed box contains a variable **i** which is initialized by 16 at the time of declaration which indicates that one cannot change the value of variable i after initialization whereas opened box contains a varibale **j** which is intialized by 30 at the time of declaration, open box indicates the one can change the value of variable **j** at anytime.



Constant vs Volatile

The following table represents the constant against volatile

Constant	Volatile
Constant variables are **unchangable**.	Volatile variables are **changable**.
Constant variable can be created by using the keyword **const**.	Volatile varibale can be created by using the keyword **Volatile**.
For example **const** int i = 16 is the way of declaring and initializing constant variable.	For example **volatile** int j = 30 is the way of declaring and initializing volatile variable.
Constant variable can only be initialized at the time of declaration.	Volatile varibale can be initialized at anytime.
By default, all variables are not constant.	By default, all variables are volatile.
The const variable is otherwise known as Read Only Variable.	The volatile variable is otherwise known as Read/Write Variable.
Another way to achieve constant variable by the use of #define preprocessor directive e.g.) **#define a 5**.	Another way to achieve volatile variable by the use of nothing e.g.) **int a = 5** is volatile.

Constant Program Using Keyword

Let's write a C program to demonstrate the purpose of constant variable

constant.c

```
#include <stdio.h>
int main()
{
const int i = 16;
i = i + 1;
printf("The value of i = %d ",i);
return 0;
}
```

Note:

Variable **i** is constant here. Thus, incrementing its value by any number is **impossible** throughout the entire program. So the above program will cause an error.

Constant Program Without Using Keyword

Let's write a C program to demonstrate the purpose of constant variable without using keyword.

noconstant.c

```
#include <stdio.h>
#define i 16
int main()
{
printf("The value of i = %d ",i);
return 0;
}
```

```
The value of i = 16
```

Note:

Variable **i** is constant here because we defined using preprocessor directive. Thus, incrementing its value by any number is **impossible** throughout the entire program.

Volatile Program Using Keyword

Let's write a C program to demonstrate the purpose of volatile variable

volatile.c

```
#include <stdio.h>
int main()
{
volatile int j = 1;
j = j + 1;
printf("The value of j = %d ",j);
return 0;
}
```

```
The value of j = 2
```

Note:

Variable 'j' is declared as volatile variable. Thus, incrementing its value by any number is **possible** throughout the entire program.

Volatile Program Without Using Keyword

Let's write a C program to demonstrate the purpose of volatile variable

novolatile.c

```
#include <stdio.h>
int main()
{
int j = 1;
j = j + 1;
printf("The value of j = %d ",j);
return 0;
}
```

```
The value of j = 2
```

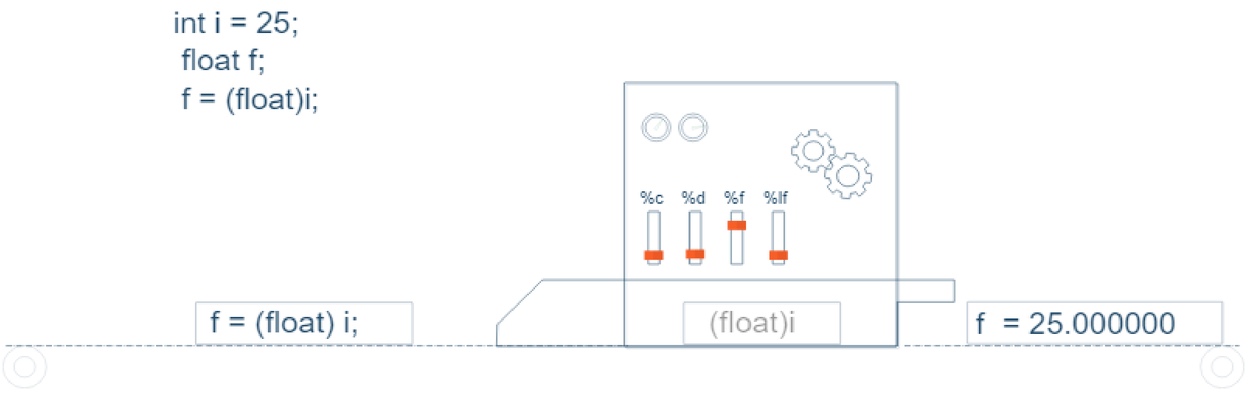
Note:

Variable 'j' is volatile because by default all variables are volatile. Thus, incrementing its value by any number is **possible** throughout the entire program.

C Type Casting

What Type Casting Is?

Generally, Type Casting is the procedure or a way to convert one thing to another. For example consider your friend bob has a bushy hairstyle, everyone of you teasing bob everyday, thus bob decided to change his hairstyle, in the very next day bob went to barber shop and got a spike hairstyle and attracted everyone. Clearly, bob is type casted from bushy hairstyle to spike hairstyle. Here typecasting in done in barber shop.



Type Casting in C

C compilers allowed programmers to convert from one data type to another data type by using type cast method. Type casting is also known as type conversion.

Type Casting Rules

- Type casting can be used to convert lower datatypes to higher datatypes and **viceversa** e.g.) int (2 bytes) to float (4 bytes).
- The type name to which conversion is to be done is placed in a closed parenthesis just before the variable. e.g.) if a float variable 'f' has a value 7.5, it can be converted into an integer type using (int)f.

Type Casting Lower Datatype Conversion

Let us write a C program to typecasting **int** datatype to **float** datatype

typecastinglower.c

```
#include <stdio.h>
int main()
{
int a = 3;
int b = 2;
int c;
printf("The value of c = %f ",(float)a/b);
return 0;
}
```

```
The value of c = 1.500000
```

Note:

Variables a, b, c belongs to integer data type, where actual result will be in float. Thus, we are in need to typecast the data type of variable 'c' from an **int** ⇨ **float**.

Type Casting Higher Datatype Conversion

Let us write a c program to typecasting **float** datatype to **int** datatype

typecastinghigher.c

```
#include <stdio.h>
int main()
{
float a = 3.56;
printf("The value of a = %d ",(int)a);
return 0;
}
```

```
The value of a = 3
```

Note:

Variables **a** belongs to float datatype, which is typecasted to int datatype in the printf statement. Thus integer output is displayed.

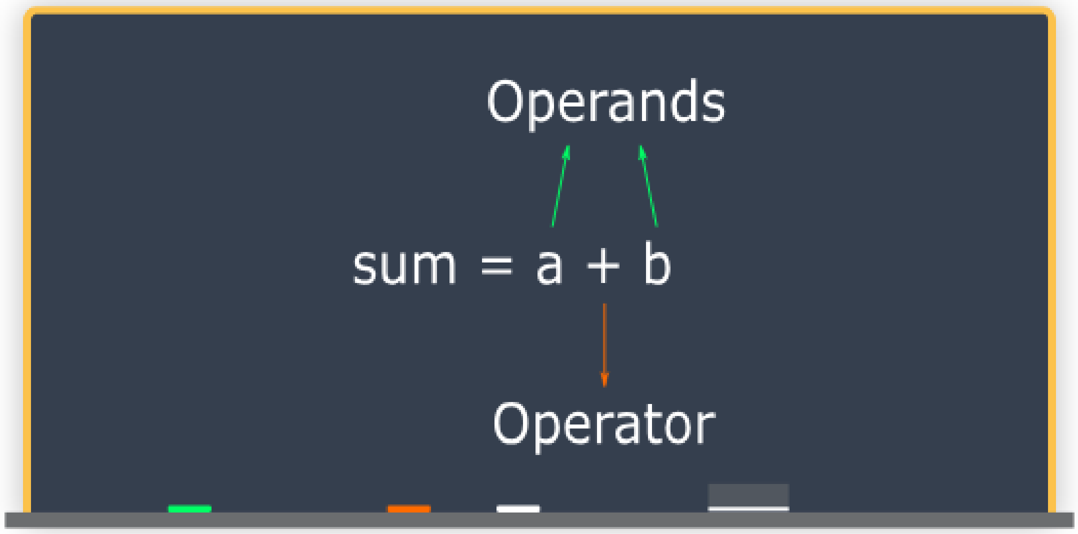
Operators

What Is Operator?

Generally, you too an operator because you operate most of the things in your day-to-day life. For example you may play music, you may drive a car and many more.

Operator In C

- In C, an operator is a symbol(+, -, *, /) which operates on a certain data type to perform mathematical, logical, relational operations.
- An operator indicates an operation to be performed on data results into a value.
- An operand is a data item (variable or constant) on which operators can perform the operations.
- An expression is a combination of variables, constants and operators.
- In C, every expression results(evaluates) into a value of certain data type.

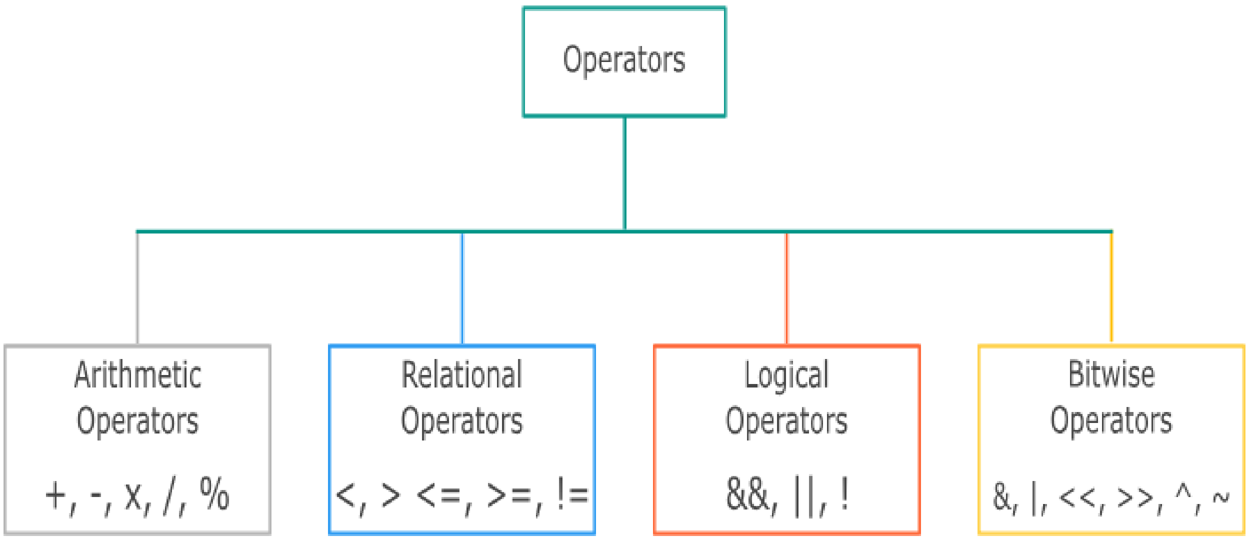


Example

Assume variable a = 2 and b = 4, these values substituted to the expression, a + (5 * b) = 22. The expression can be defined as follow, a and b are variables, 5 is a constant and 22 is a result value.

Operators Classifications

C provides **4** different kind of operators to perform different kinds of operations on operands (data items).


Operators
Arithmetic Operators
+, -, x, /, %
Relational Operators
<, > <=, >=, !=
Logical Operators
&&, ||, !
Bitwise Operators
&, |, <<, >>, ^, ~

Comma Operator

What Is Comma Operator?

The comma operator is used to separate two or more expression. Where first expression1 is evaluated, then expression2 is evaluated, and the value of expression2 is returned for the whole expression.

comma.c

```
#include <stdio.h> //header file section
int main() //main section
{
int a=5, b=10;
if(a>b,a<b)
printf("if condition executes");
else
printf("else condition executes");
return 0;
}
```

```
if condition executes
```

Note:

In the above program if condition checks only the right most codition as **comma operator** sperates two conditions.

Comma Operator Priority

Comma operator has the lowest priority among all the operators.

Comma Operator In printf

Here is an example program where **comma operator** get used in printf statement.

commaprintf.c

```
#include <stdio.h> //header file section
int main() //main section
{
printf("Addition of 4 + 4 is %d ", 4+4);
return 0;
}
```

```
Addition of 4 + 4 is 8
```

Comma Operator In scanf

Here is an example program where **comma operator** get used in scanf statement.

commascanf.c

```
#include <stdio.h> //header file section
int main() //main section
{
int num;
printf("Enter a number : ");
scanf("%d",&num);
printf("you entered %d",num);
return 0;
}
```

- Enter a number: 2
- you entered 2

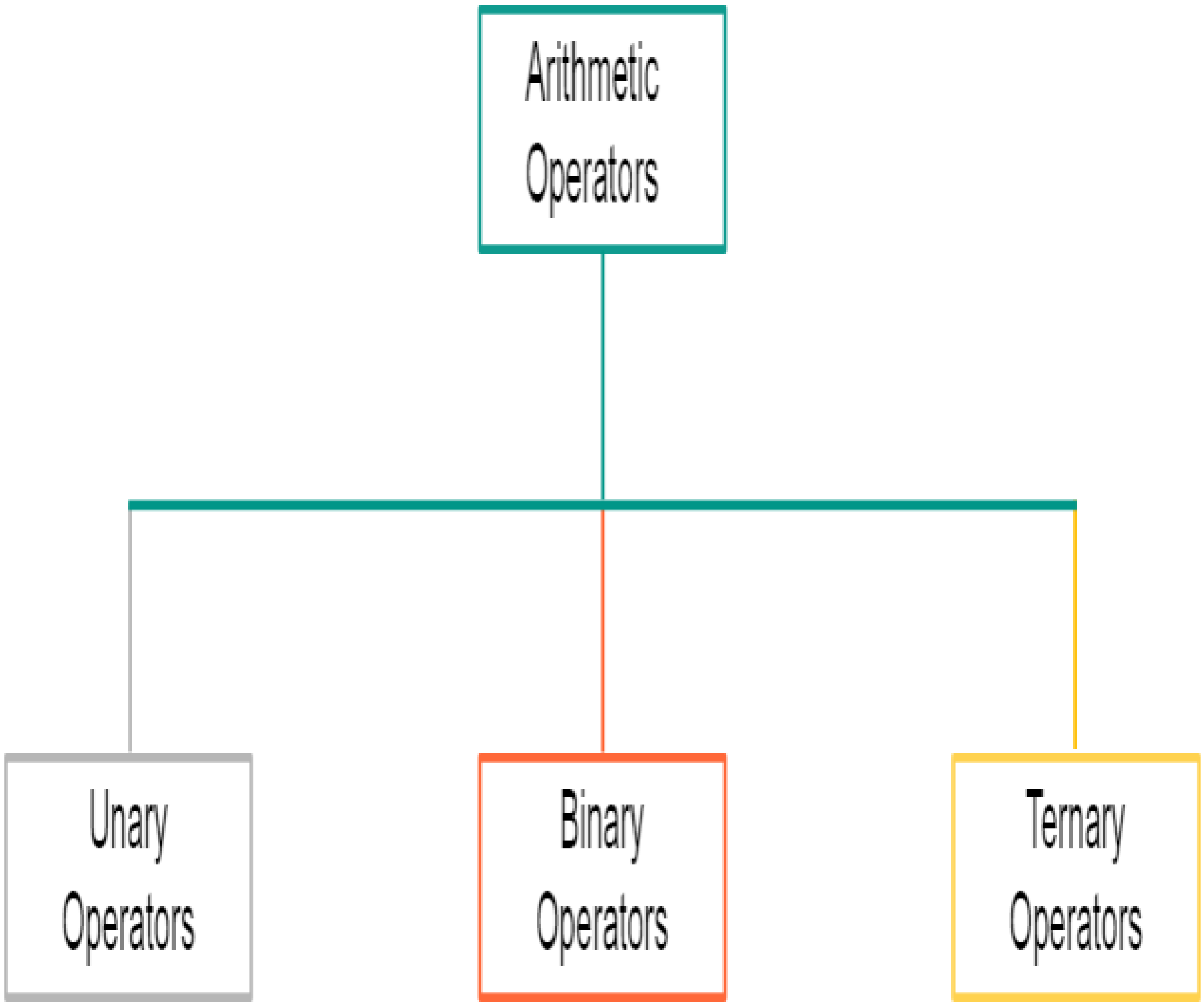
Arithmetic Operators

What is Arithmetic Operators

Arithmetic operators can be used to perform arithmetic operations such as Addition, Subtraction, Multiplication and Division. Arithmetic operators are most commonly used operators in computer programming languages. The operands involved in arithmetic operations must represent numerical value. Thus, the operands can be an integer type, floating point types, double and char(In C, characters represent numerical value according to ASCII table). Arithmetic operators can be divided into **3** types, they are

- Unary Operator
- Binary Operator
- Ternary Operator

Classifications of Arithmetic Operators



Binary Operator in C

Binary operators are act upon a two operands to produce a new value.

Operator	Description	Example
+	Addition	1 + 2 = 3
-	Subtraction	2 - 1 = 1
*	multiplication	2 * 3 = 6
/	Division	6 / 3 = 2
%	Modular Division	5 % 2 = 1 (remainder)

Binary Operators - C Program

In this program, we make use of several Binary Operators.

- + is a binary operator, which is used to sum two operands to yield a resultant value.
- - is a binary operator, which is used to subtract two operands to yield a resultant value.
- * is a binary operator, which is used to multiply two operands to yield a resultant value.
- / is a binary operator, which is used to divide two operands to yield a resultant value.
- % is a binary operator, which is used to find the remainder of two operands when divided.

binaryoperators.c

```c
#include <stdio.h> //header file section
int main() //main section
{
int a = 5;
int b = 2;
int l = a + b;
int m = a - b;
int n = a * b;
int o = a / b;
int p = a % b;
printf("Addition of a and b = %d ", l);
printf("\nSubtraction of a and b = %d ", m);
printf("\nMultiplication of a and b = %d ", n);
printf("\nDivision of a and b = %d ", o);
printf("\nRemainder of a and b = %d ", p);
return 0;
}
```

- Addition of a and b = 7
- Subtraction of a and b = 3
- Multiplication of a and b = 10
- Division of a and b = 2
- Remainder of a and b = 1

Unary Operators in C

Unary operators are act upon a single operand to produce a new value.

Operator	Description	Example

-	Minus Operator	int x = -5;
		int y = -x;

++	Increment Operator	int a = 2;
		a = ++a;

--	Decrement Operator	int a = 2 a = --a;

&	Address Operator	int n = 10;
		a = &n;

sizeof	Gives the size of operator	int x = 5;
		sizeof(x);

Unary operator - C Program

Let we make use of **minus** Unary operator and have fun

unaryoperators.c

```
#include <stdio.h> //header file section
int main() //main section
{
int x = -5;
int y = -x;
printf("The value of x = %d \nThe value of y = %d ",x, y);
return 0;
}
```

- The value of x = - 5
- The value of y = 5

Note:

'**-**' operator multiplies minus with a value in variable 'x'.

Increment Operator - C Program

Let we make use of **Increment** Unary operator and have fun

incrementoperator.c

```
#include <stdio.h> //header file section
int main() //main section
{
int a = 2;
printf("The value of a = %d ", a);
a = ++a;
printf("\nThe value of a = %d ", a);
return 0;
}
```

- The value of a = 2
- The value of a = 3

Note:

An increment operator **++** followed by a variable 'a' will be incremented immediately to the value **3**.

Purpose of increment operator

The vital purpose of increment Unary Operator is to add one to their operand.

Decrement Operator - C Program

Let we make use of **Decrement** Unary operator and have fun

decrementoperator.c

```
#include <stdio.h> //header file section
int main() //main section
{
int a = 2;
printf("The value of a = %d ",a);
a = --a;
printf("\nThe value of a = %d ",a);
return 0;
}
```

- The value of a = 2
- The value of a = 1

Note:

An **decrement** operator **--** followed by a variable 'a' will be decremented immediately to the value **1**.

Purpose of Decrement operator

The vital purpose of Decrement Unary operator is to reduce or subtract one to their operand.

Address Operator(&) - C Program

Let we make use of **Address(&)** Unary operator and have fun

addressoperator.c

```
#include <stdio.h> //header file section
int main()
{
int n = 10;
printf("The value of n = %d ",n);
printf("\nAddress of n = %u ",&n);
return 0;
}
```

- The value of n = 10
- Address of n = 2293436

Note:

Memory location for a data can be find out with & operator followed by its variable name. Addresses may differs with respect to memory management architecture of an operating system.

Where address operator get used

While the programmer plays with pointer, the address operator is the one which is most frequenty get used.

sizeof() Operator - C Program

Let we make use of **sizeof()** Unary operator and have fun

sizeofoperator.c

```
#include <stdio.h>
int main()
{
int x = 5;
float y = 10;
printf("sizeof (x) = %d \nsizeof (y) = %d ",sizeof(x), sizeof(y));
return 0;
}
```

- sizeof (x) = 2
- sizeof (y) = 4

Note:

Here, the variables declared and initialized as x and y. The variable x is an integer and y is a float data type. Using **sizeof()** operator, we can identified the size of the variable.

Where sizeof() operator get used

While the programmer plays with dynamic memory allocation, the sizeof() operator is the one which is most frequenty get used.

C Relational Operators

What Is Relational Operator?

- Relational operators are used to compare arithmetic, logical and character expressions.
- If the condition is "true" it returns **1**, otherwise, it returns **0**.

Relational operator Types

C Programming provides **6** relational operators for comparing numeric quantities.

Operator	Description	Example	Return Value
>	Greater than	5 > 3	1
<	Less than	5 < 3	0
<=	Less than equal to	5 <= 5	1
>=	Greater than equal to	6 >= 5	1
==	Equal to	5 == 3	0
!=	Not equal to	3 != 3	0

Relational Operator Program

Let us write a C program to demonstrate

relationaloperators.c

```
#include <stdio.h>
int main()
{
printf("\nCond.\t Return"); //Cond. stands for Condition
printf("\n5 > 3  :\t %d",5 > 3);
printf("\n5 < 3  :\t %d",5 < 3);
printf("\n5 <= 5 :\t %d",5 >=5);
printf("\n6 >= 5 :\t %d",6 >=5);
printf("\n5 == 3 :\t %d",5 == 3);
printf("\n3 != 3 :\t %d",3 != 3);
return 0;
}
```

- Cond.   Return
- 5 > 3 :   1
- 5 < 3 :   0
- 5 <= 5 :   1
- 6 >= 5 :   1

- 5 == 3 :   0
- 3 != 3 :   0

Note:

Here true condition returns **1** and false condition returns **0**.

Relational Operator In Conditional Statement

relationaloperators.c

```
#include <stdio.h>
int main()
{
int a = 5;
int b = 10;
if(a < b)
printf("a is the smallest number");
else
printf("b is the smallest number");
return 0;
}
```

```
a is the smallest number
```

Note:

Here **less than** conditional operator is used to check whether **a** is smaller than **b**

C Logical Operators

Why Logical Operator?

By learning relational operator, you should have a general idea how to use conditions in if statements by now, but imagine if you what to check two condition to execute certain sets of statement. what you will be doing there

Option A

Using nested if statement(2 if statements one after another) to evaluate two condition.

Option B

Make use of **Logical Operators**

Answer

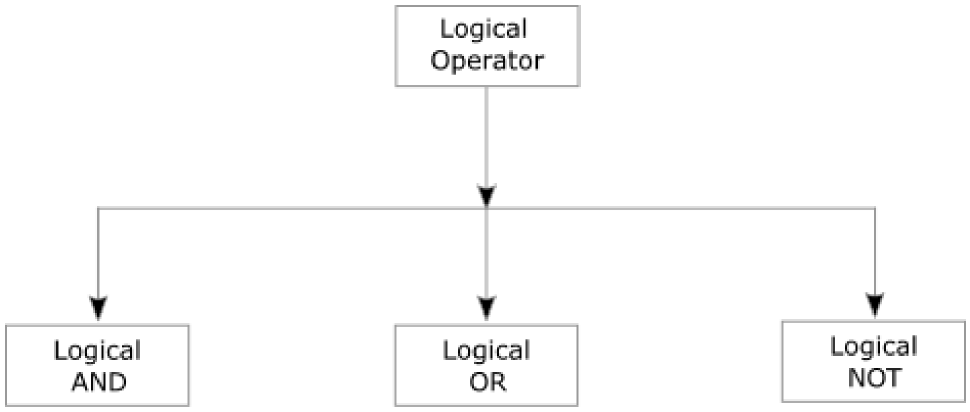
Though option A looks classic it fails in **either or condition**. In such case you are insisted to make use of **Logical Operators** with no choices.

Logical Operator in C

- Logical operators are used to check (or) compare the logical relations between the expressions.
- Logical operator, returns **1** if given condition is true, **0** if given condition is false.

Logical Operator flow

C provides **3** logical operator for comparing numeric quantities.



Logical Operator Table:

Operator	Description	Example	Return Value
&&	Logical AND	7 > 3 && 8 > 5	1
||	Logical OR	7 > 3 || 8<5	1
!	Logical NOT	5 != 5	0

Logical AND Operator

Let us write a C program to demonstrate **logical AND operator**

logicaland.c

```
#include <stdio.h>
int main()
{
int a = 20;
int b = 10;
int c = 15;
if(a<b && b<c)
printf(" C is the greatest number of all");
else
printf(" C is not greatest number of all");
return 0;
}
```

```
C is not greatest number of all
```

Note:

Here printf statement next to if conditional statement will execute only both conditions inside if statement is true otherwise else part will be executed.

Logical OR Operator

Let us write a C program to demonstrate **logical OR operator**

logicalor.c

```
#include <stdio.h>
int main()
{
int a = 20;
int b = 10;
int c = 15;
if(c>a || c>b)
printf(" C is not smallest and may not biggest of all ");
else
printf(" C is smallest of all");
return 0;
}
```

```
C is not smallest and may not biggest of all
```

Note:

Here printf statement next to if conditional statement will execute even either condition is true.

Logical NOT Operator

Let us write a C program to demonstrate **logical NOT operator**

logicalnot.c

```
#include <stdio.h>
int main()
{
int a = 20;
int b = 10;
if(a != b )
printf("a is not equal to b");
else
printf(" a is equal to b");
return 0;
}
```

```
a is not equal to b
```

Note:

Here printf statement next to if conditional statement executes as a and b are different in numbers.

C Bitwise Operators

We knew that, all integer variables represented internally as binary numbers. A value of type **int** consists of 32 binary digits, known to us as bits. We can operate on the bits that make up integer values using the bitwise operators.

Bitwise Operator's Facts

- A bitwise operator which operates on each bit of data.
- Bitwise operators only operates on integer operands such as int, char, short int, long int.

All Bitwise Operators

C provides **6** bitwise operators to operate on individual bits in an integer quantity.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
>>	Right shift
<<	Left shift
~	One's Complement

Bitwise AND Operator

Bitwise AND operator, &, combines corresponding bits in its tow operands such that if both bits are 1, the result is 1 otherwise the result is 0.

Bitwise AND Operator Table

Input		Output
X	**Y**	**Z**
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise AND Operator Program

Let us write a C program to demonstrate **Bitwise AND operator**

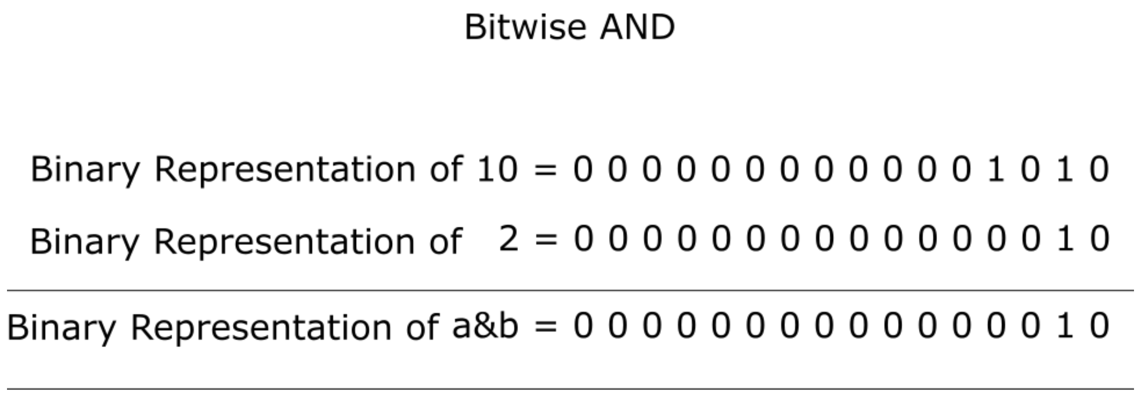
bitwiseand.c

```
#include <stdio.h>
int main()
{
int a = 10, b = 2;
printf("a & b = %d ", a & b);
return 0;
}
```

```
a & b = 2
```

Note:

Binary representation is given below.



Bitwise AND

```
Binary Representation of 10 = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
Binary Representation of  2 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
Binary Representation of a&b = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Bitwise OR Operator

The bitwise OR operator, |, combines corresponding bits such that if either or both bits are 1, then the result is 1. Only if both bits are 0 is the result 0.

Bitwise OR Operator Table

Input		**Output**
X	**Y**	**Z**
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise OR Operator Program

Let us write a C program to demonstrate **Bitwise OR operator**

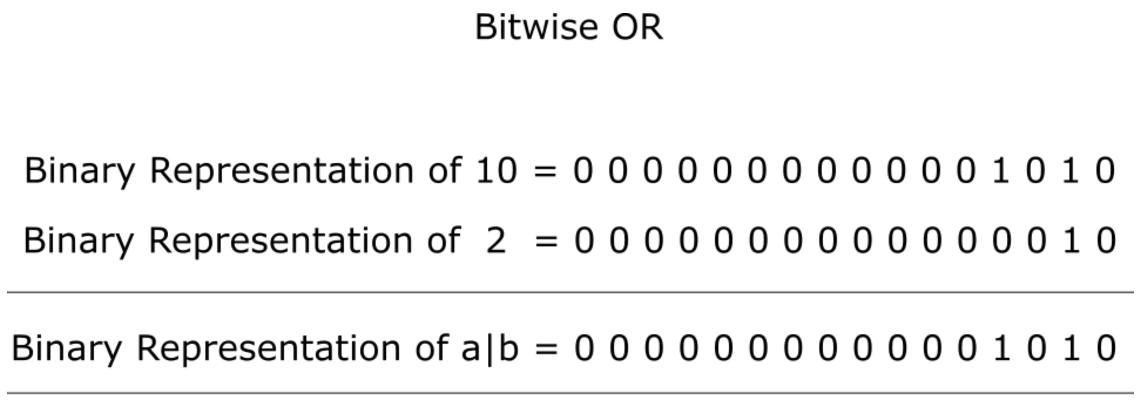
bitwiseor.c

```
#include <stdio.h>
int main()
{
int a = 10, b = 2;
printf("a | b = %d ", a | b);
return 0;
}
```

```
a | b = 10
```

Note:

Binary representation is given below.



Bitwise OR

```
Binary Representation of 10 = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
Binary Representation of  2 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
Binary Representation of a|b = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
```

Bitwise XOR Operator

Bitwise exclusive OR (XOR) operator, ^, combines corresponding bits such that if both bits are the same the result is 0; otherwise, the result is 1.

Bitwise XOR Table

Input		**Output**
X	**Y**	**Z**
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise XOR Program

Let us write a C program to demonstrate **Bitwise XOR operator**

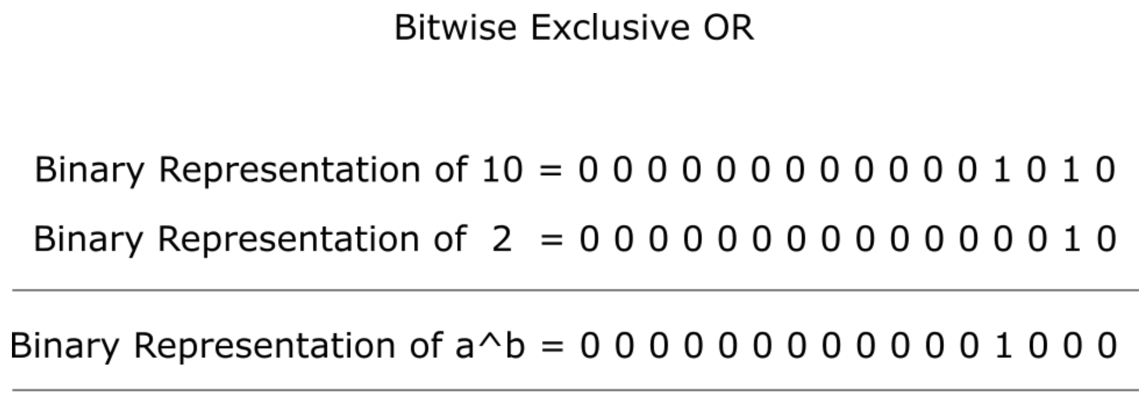
bitwisexor.c

```
#include <stdio.h>
int main()
{
int a = 10, b = 2;
printf("a ^ b = %d ", a ^ b);
return 0;
}
```

```
a ^ b = 8
```

Note:

Binary representation is given below.



Bitwise Exclusive OR

```
Binary Representation of 10 = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
Binary Representation of  2 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
----------------------------------------------------------------
Binary Representation of a^b = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
```

Right shift Operator (>>)

Right shift operator requires two operands. It takes Left hand side operand as bit sequence or bit Pattern to be shifted and right hand side operand as positive integer or unsigned integer that indicates the number of displacements or bit positions to the **right**.

C Program for Right Shift

Let us write a C program to demonstrate **Right Shift operator**

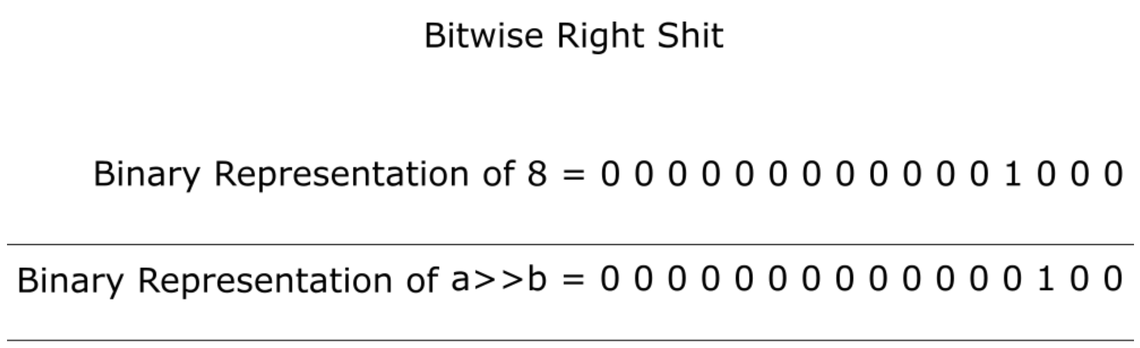
rightshift.c

```
#include <stdio.h>
int main()
{
int a = 8,b;
a>>= 1;
b = a;
printf("The Right shifted data of 8 by 1 = %d ",b);
return 0;
}
```

```
The Right shifted data of 8 by 1 = 4
```

Note:

Binary representation is given below.


Bitwise Right Shit

Binary Representation of 8 = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Binary Representation of a>>b = 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

Left shift Operator (<<)

Left shift operator also requires two operands. It takes Left hand side operand as bit sequence or bit Pattern to be shifted and right hand side operand as positive integer or Unsigned integer that indicates the number of displacement or bit positions to the **left**.

C Program for Left Shift

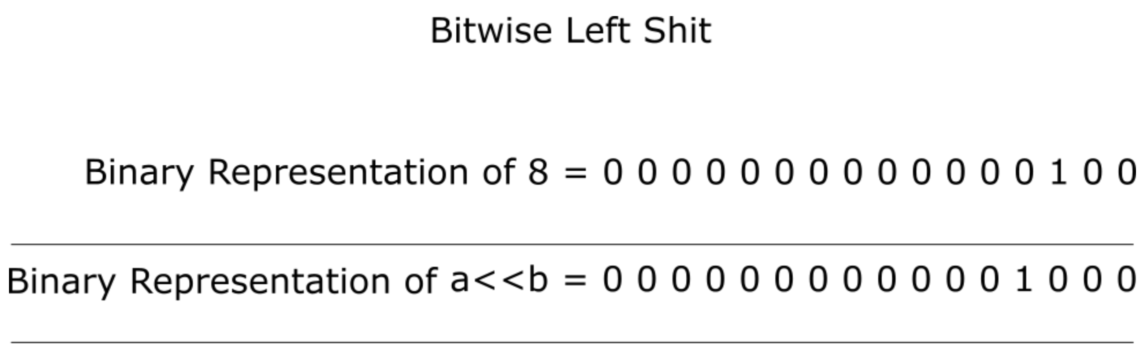
leftshift.c

```
#include <stdio.h>
int main()
{
int a = 4, b;
a<<= 1;
b = a;
printf("The Left shifted data of 4 by 1 = %d ",b);
return 0;
}
```

```
The Left shifted data of 4 by 1 = 8
```

Note:

Binary representation is given below.


Bitwise Left Shit

Binary Representation of 8 = 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

Binary Representation of a<<b = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

One's Complement

One's Complement operator is a unary operator, it always precedes to the variable or an operand. One's Complement operator inverts bits of its operand. So, 1s becomes 0s and 0s becomes 1s.

Table for One's Complement

Input	Output
X	**Y**
0	1
1	0

C Program for One's Complement

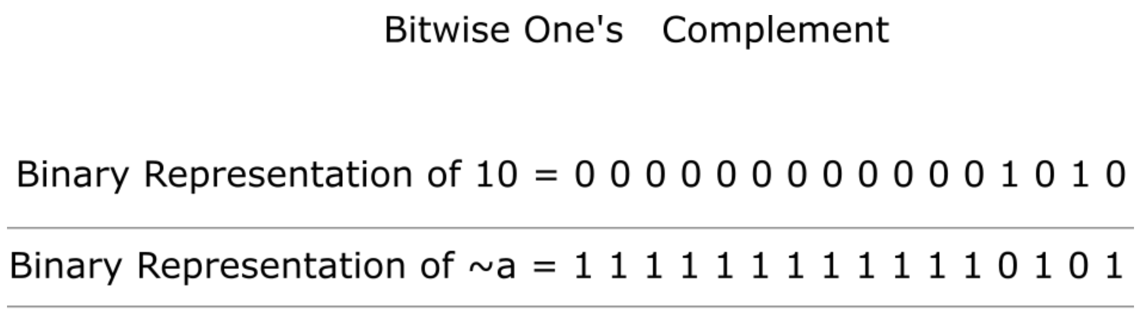
onescomplement.c

```
#include <stdio.h>
int main()
{
unsigned short int a = 10;
a = ~a;
printf("After One's Complement a = %u", a);
return 0;
}
```

```
After One's Complement a = 65525
```

Note:

If a variable 'a' is declared under normal int data type some garbage value will be displayed say -11. Binary representation is given below.



Bitwise One's Complement

Binary Representation of 10 = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

Binary Representation of ~a = 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1

C Conditional or Ternary Operator

The **conditional operator** is sometimes called a ternary operator because it involves three operands. It is best understood by considering the following example
younger = son < father ? 18 : 40;

In the above example, son's age is 18 whereas father's age is 40. But we need the younger age so we make use of conditional operator to extract least age.

Syntax of Conditional Operator in C

- The conditional operator contains a condition followed by two statements or values.
- If the condition is true the first statement is executed, otherwise the second statement or value.

Syntax

```
condition ? (value 1) : (value 2)
```

How Conditional Operator Works

Conditional operator performs nothing more special. It is the abstraction of **if else** statement. Consider the following example how **Condtional Operator** works
younger = true ? **18** : 40;

If the logical expression resulted true, 18 will be stored in a variable younger.
younger = false ? 18 : **40**;

If the logical expression resulted true, 40 will be stored in a variable younger.

Program without Conditional Operator

Here we are to find the least age and store it in a variable **younger_age** without using Conditional operator.

noternary.c

```
#include <stdio.h> //header file section
int main() //main section
{
int son = 18;
int father = 40;
int younger_age;
if(son < father)
younger_age = son;
else
younger_age = father;
printf("%d is the younger age", younger_age);
return 0;
}
```

```
18 is the younger age
```

Note:

Here the code looks more verbose as we make use of if else statement.

Program Using Conditional Operator

Here we are to find the least age and store it in a variable **younger_age** using Conditional operator.

ternary.c

```
#include <stdio.h> //header file section
int main() //main section
{
int son = 18;
int father = 40;
int younger_age;
younger_age = son < father ? son : father;
printf("%d is the younger age", younger_age);
return 0;
}
```

```
18 is the younger age
```

Note:

In line 7, **Conditional Operator** is get used to reduce the size of the program.

Operator Precedence In C

Why Operator Precedence?

Every Operator have their own precedence because without operator precedence a complier will conflict with data when performing mathematical calculations.

Example

Consider the following expression **6 - 4 + 8** without operator precedence compiler is helpless to choose which operator needs to execute first. Thus Operator Precedence helps compiler out there.

The following table lists all C operators and their precedence from higher priority to lower priority

Operator	Operation	Clubbing	Priority
( )	Function call	Left to Right	1st
[ ]	Array	"	"
→	Structure Operator	"	"
.	Structure Operator	"	"
+	Unary plus	Right to Left	2nd
-	Unary minus	"	"
++	Increment	"	"
--	Decrement	"	"
!	Not Operator	"	"
~	One's Complement	"	"
*	Pointer Operation	"	"
&	Address Operator	"	"
sizeof	Size of an data type	"	"
(typecast)	Type Cast	"	"
*	Multipication	Left to Right	3rd
/	Division	"	"
%	Modular Division	"	"
+	Addition	Left to Right	4th
-	Subtraction	"	"
<<	Left shift	Left to Right	5th

Operator	Operation	Clubbing	Priority
>>	Right shift	"	"
<	Less than	Left to Right	6th
<=	Less than or equal to	"	"
>	Greater than	"	"
>=	Greater than or Equalto	"	"
=	Equality	Left to Right	7th
!=	InEquality	"	"
&	Bitwise AND	Left to Right	8th
^	Bitwise XOR	Left to Right	9th
|	Bitwise OR	Left to Right	10th
&&	Logical AND	Left to Right	11th
||	Logical OR	Left to Right	12th
?:	Conditional Operator	Right to Left	13th
^=, !=, <<=, > >=	Assignment Operator	Right to Left	14th
,	comma operator	Right to Left	15th

Operator precedence program(Easy)

operatorprecedence1.c

```
#include <stdio.h> //header file section
int main()
{
int a = 2, b = 6, c = 12, d;
d = a + c / b;
printf("The value of d = %d ", d);
return 0;
}
```

```
The value of d = 4
```

Note:

/ (Division operator) executed first. Now the expression will be **d = a + 2**. Finally **+** (Addition operator) executed and the resultant value 4 is stored in a variable 'd'.

Operator precedence in C(Medium)

operatorprecedence2.c

```
#include <stdio.h> //header file section
int main()
{
int a = 2, b = 6, c = 12, d;
d = a * b + c / b;
printf("The value of d = %d ", d);
return 0;
}
```

```
The value of d = 14
```

Note:

/ (Division operator) executed first. Now the expression is **d = a * b + 2**. Secondly, ***** (Multiplication operator) will be executed, now the expresssion becomes **d = 12 + 2** Finally **+** (Addition operator) will be executed and store the value 14 in a variable d.

Operator Precedence in C(Hard)

operatorprecedence3.c

```
#include <stdio.h> //header file section
int main()
{
int a = 10, b = 10, c = 1, d = 5, r;
r = ++a + (++b) + b-- * (++c) + --d;
printf("The value of r = %d ", r);
return 0;
}
```

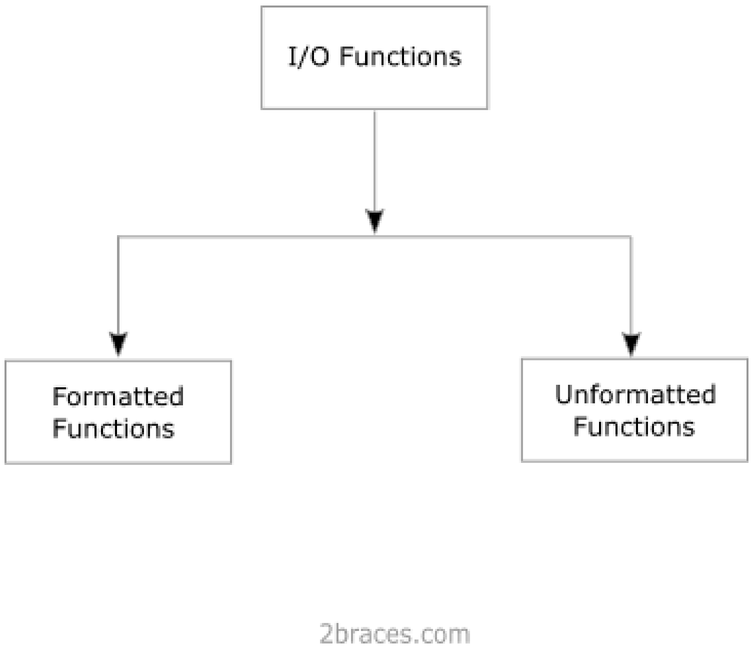
```
The value of r = 48
```

Note:

- **Step 1:** r = 11 + 11 + b-- * 2 + --d
- **Step 2:** r = 11 + 11 + 11 * 2 + 4
- **Step 3:** r = 11 + 11 + 22 + 4
- **Step 4:** r = 48.

C Input and Output

In programming input means reading data from the input devices or from a file. Output means displaying results on the screen. C provides number of input and output functions. These input and output functions are predefined in their respective header files. Header file for standard input and output functions is stdio.h. These are included into program prefixed with #include statement. Input and Output functions are classified into **2** categories. Decision making statements (or) Control statements are used to check the condition, if the condition is true it will executes a block of statements.



C Formatted IO Functions

Input data or output results are formatted as per requirement. Formatted function improves the readability of the input and output. Formatted functions can be used to read and write data of all data type(char, int, float, double). Formatted input and output functions require format specifiers(%c, %d, %f, %lf) to identify the type of data.

scanf() Function

Scanf function reads all types of data value from input devices (or) from a file. The address operator '&' is to indicate the memory location of the variable. This memory location is used to store the data which is read through keyboard.

scanf() Function Program

scanf.c

```
#include <stdio.h> //header file section
int main()
{
int a, b, c;
printf("Enter the value of a = ");
scanf("%d",&a) ; //read value a through keyboard
printf("\nEnter the value of b = ");
scanf("%d",&b); //read value b through keyboard
c = a + b;
printf("\na + b = %d ",c); //Print the sum of two value a and b
return 0;
}
```

- Enter the value of a = 6
- Enter the value of b = 4
- a + b = 10

Width specifier program

widthspecifier.c

```
#include <stdio.h> //header file section
int main()
{
printf("\n%.2s","abcdefg ");
printf("\n%.3s","abcdefg ");
printf("\n%.4s","abcdefg ");
printf("\n%.5s","abcdefg ");
printf("\n%.6s","abcdefg ");
return 0;
}
```

- ab
- abc
- abcd
- abcde
- abcdef

Note:

Output itself shows the process done by %.2s, %.3s, etc..

scanf to Get Input

getinput.c

```
#include <stdio.h> //header file section
int main()
{
float a, b, c, d;
printf("Enter three float numbers:\n ");
scanf("\n %f %f %f ",&a,&b,&c);
d = (a + b + c)/3;
printf("\nAverage of given number is %f ",d);
return 0;
}
```

- Enter three float numbers:
- 4.5
- 4.5
- 5.0
- Average of given number is 4.666667

Note:

Here, three float variables a, b and c read through scanf() function. The resultant average value is stored in a variable d.

C Unformatted Functions

Unformatted input and output functions are only work with character data type. Unformatted input and output functions do not require any format specifiers. Because they only work with character data type.

Character IO Functions

getchar() Function

The getchar() function reads character type data form the input. The getchar() function reads one character at a time till the user presses the enter key.

getchar() C Program

getchar.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c;
printf("Enter a character : ");
c = getchar();
printf("\nEntered character : %c ", c);
return 0;
}
```

- Enter a character : y
- Entered character : y

Note:

Here, getchar() reads the input from the user and display back to the user.

getch() Function

The getch() function reads the alphanumeric character input from the user. But, that the entered character will not be displayed.

getch() C Program

getch.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
printf("\nHello, press any alphanumeric character to exit ");
getch();
return 0;
}
```

```
Hello, press any alphanumeric character to exit
```

Note:

The above program will run until you press one of many alphanumeric characters. The key pressed by you will not be displayed.

getche() Function

getche() function reads the alphanumeric character from the user input. Here, character you entered will be echoed to the user until he/she presses any key.

getche() C Program

getche.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
printf("\nHello, press any alphanumeric character or symbol to exit \n ");
getche();
return 0;
}
```

- Hello, press any alphanumeric character or symbol to exit
- k

Note:

The above program will run until you press one of many alphanumeric characters. The key pressed by you will be echoed.

putchar() Function

putchar() function prints only one character at a time.

putchar() C Program

putchar.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c = 'K';
putchar(c);
return 0;
}
```

K

Note:

Here, variable c is assigned to a character 'K'. The variable c is displayed by the **putchar()**. Use Single quotation mark `''` for a character.

putch() Function

The putch() function prints any alphanumeric character.

putch() C Program

putch.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c;
printf("Press any key to continue\n ");
c = getch();
printf("input : ");
putch(c);
return 0;
}
```

- Press any key to continue
- input : d

Note:

The getch() function will not echo a character. The putch() function displays the input you pressed.

String IO Functions

gets() Function

The gets() function can read a full string even blank spaces presents in a string. But, the scanf() function leave a string after blank space space is detected. The gets() function is used to get any string from the user.

gets() C Program

gets.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c[25];
printf("Enter a string : ");
gets(c);
printf("\n%s is awesome ",c);
return 0;
}
```

- Enter a string : Randy Orton
- Randy Orton is awesome

Note:

The gets() function reads a string from through keyboard and stores it in character array c[25]. The printf() function displays a string on the console.

puts() Function

The puts() function prints the charater array or string on the console. The puts() function is similar to printf() function, but we cannot print other than characters using puts() function.

puts() C Program

puts.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c[25];
printf("Enter your Name : ");
gets(c);
puts(c);
return 0;
}
```

- Enter your Name: john
- john

C Common Library Functions

clrscr() in C

clrscr() is an inbuilt library function which is used to clear the previous output displayed in a screen. **clrscr()** is defined in **#include <conio.h>** header file.

clrscr() Function Program

clrscr.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
printf("Before clrscr");
clrscr();
printf("clrscr() will clear the screen");
return 0;
}
```

```
clrscr() will clear the screen
```

Note:

clrscr(); works in Turbo C/C++ compiler.

exit() in C

exit() is an inbuilt library function which is used to terimate the program irrespective of the statements followed by it. **exit()** is defined in #include <stdlib.h> header file.

exit() Function Program

exit.c

```
#include <stdio.h> //header file section
#include <stdlib.h>
int main()
{
printf("This statement is before exit(); function ");
exit(0);
printf("It will not display ");
return 0;
}
```

```
This statement is before exit(); function
```

Note:

exit (some numeric value); will exit the program.

sleep() in C

sleep() is an inbuilt library function which is used to delay the program's output. **sleep()** is defined in #include <unistd.h> header file.

sleep() Function Program

sleep.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
printf("Countdown... ");
printf("\n 3");
sleep(1);
printf("\n 2");
sleep(1);
printf("\n 1");
sleep(1);
printf("\n Celebration Time ");
return 0;
}
```

- Countdown...
- 3
- 2
- 1
- Celebration Time

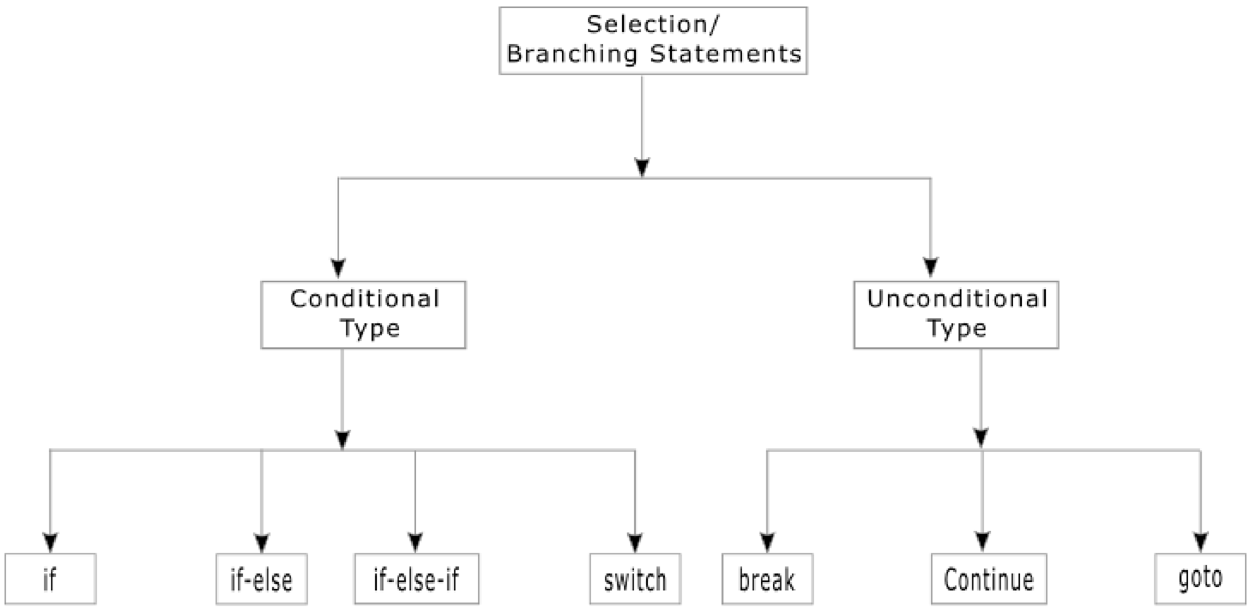
Note:

sleep (seconds); will delay the program's output with respect to seconds which you mentioned.

C Control Statements

C Programs that we encountered up to now were executed in the same order which they appeared in it. In practical applications, there is numerous situations where we have to neglect some parts of program codes. For this purpose, C Programming uses the control statements to control the flow of a program. If the condition is satisfied the code followed by that condition will execute. If not simply that the code will be neglected by the compiler.

Flowchart Control Statements



C if statement

Why if Statement?

All programming languages enable you to make decisions. They enable the program to follow a certain course of action depending on whether a particular **condition** is met. This is what gives programming language their intelligence.

C if Syntax And Definition

- C uses the keyword **if** to execute a set of statements when logical condition is true.
- When the logical condition is false, the compiler simply skips the statement within the block.
- The "if" statement is also known as one way decision statement.

Syntax if statement

Syntax

```
if(condition)
{
here goes statements;
.
.
.
.
here goes statements;
}
```

if Statement Uses

if statements are commonly used in following scenarios

- Is A bigger than B?
- Is X equal to Y?
- Is M not equal to N?

Realtime Time if Statement Uses

Arduino microcontroller make use C programming, where you need to blink a warning light(red light) when certain condition met. Example program is as below:

realtimeif

```
if(x = 123)
digitalWrite(LEDpin, high)
```

Note:

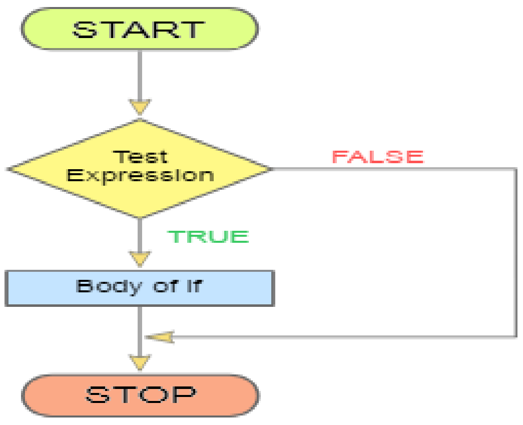
x is a variable which get its input from a sensor continuously.

if Statement Rules

- The expression (or) condition is always enclosed within pair of parenthesis. e.g.) if**(** a > b **)**
- The **if** statement should not be terminated with a semicolon. If it happens, the block of statements will not execute even the condition is true.
- The statements following the if condition is normally enclosed between **2 braces** (in curly braces).

if statement Flow Chart

Following flow chart will clearly explain how **if statement works**



if statement C Program

ifstatement.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
int age = 18;
if(age > 17){
printf("you are eligible for voting ");
}
printf("\nThis is normal flow ");
return 0;
}
```

- you are eligible for voting
- This is normal flow

Note:

If the user input is greater than 17, then the condition will be true and the statements under if condition gets executed. Otherwise, it executes next to the if block.

C if-else Statement

Why if-else Statement?

As We've seen, the if statement allows us to run a block of code if an expression evaluates to true. If the expression evalutes to false, the code is skipped. Thus we can enhance this decision-making process by adding an **else** statement to an if construction. This lets us run one block of code if an expression is true, and a different block of code if the expression is false.

C if-else Syntax And Definition

- C uses the keywords **if** and **else** to execute a set of statements either logical condition is true or false.
- If the condition is **true,** the statements under the keyword **if** will be executed.
- If the condition is **false,** the statements under the keyword **else** will be executed.

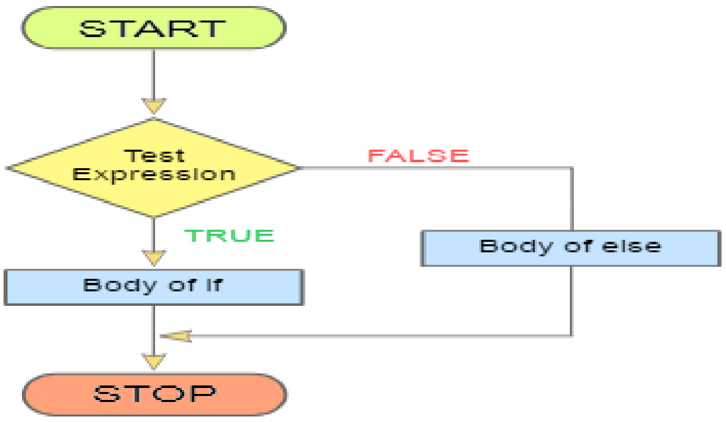
Syntax if-else statement

Syntax

```
if(condition)
{
here go statements....
}
else
{
here go statements....
}
```

if else Statement Flow Chart

Following flow chart will clearly explain how **if else statement works**



Realtime Time **if else** Statement Uses

Arduino microcontroller make use C programming, where you can alarm **if** your sensors output in greater than or lesser than expected value, **else** blink green light always.

```
if(x = 123)
digitalWrite(LEDpin,high)
else
digitalWrite(LEDpin,low)
```

Note:

x is a variable which get its input from a sensor continuously.

if-else C Program

ifelsestatement.c

```
#include <stdio.h> //header file section
int main()
{
int age;
printf("Enter your age : ");
scanf("%d",&age);
if(age > 17)
{
printf("\nyou are eligible for voting ");
}
else
{
printf("\nSorry, you are not eligible for voting ");
}
printf("\nThis is normal flow ");
return 0;
}
```

- Enter your age : 16
- Sorry, you are not eligible for voting
- This is normal flow

Note:

The user input for a variable 'age' is 16, the expression or condition tends to false. So, the else block gets executed.

C Nested if Statement

Why Nested if Statement

The statement that is executed when an **if** expression is true can be another if, as can the statement in an else clause. This enables you to express such convoluted logic as "**if** age of Lingcoln is greater than age of john "and **if** age of Lingcoln is greater than age of renu". Then we decide Lingan is elder of all

Syntax and Definition Nested if statement

- In C programming, control statements like as **if** can be nested, that means we can write one within another.
- If outer **if statement** fails, then the compiler skips the entire block irrespective of their inner if statement.

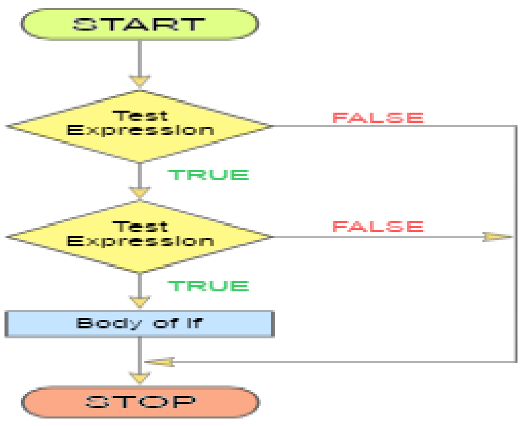
Syntax nested if statement

Syntax

```
if(condition)
{
if(condition)
{
here goes statements;
}
}
```

Nested if Statement Flowchart

The following flowchart will clearly demonstrate how **nested if statement works**



Program Nested if statement

Let us write a C program using **Nested if statement** nestedif.c

```
#include <stdio.h> //header file section
int main()
{
int a = 47, b = 25, c = 3;
if (a > b)
{
if (a > c)
{
printf("The value a is greater than b and c ");
}
}
printf("\nThis is normal flow ");
return 0;
}
```

- The value a is greater than b and c
- This is normal flow

Note:

When the first if statement executed, the value 'a' is compared with value 'b', if the expression is true, then inner if statement executed, the inner expression compares the value a with value c, if the inner if statement also true, the compiler will display the output.

C Nested else if Statement

- In nested else if statement, the number of logical conditions can be checked for executing various statements.
- If any logical condition is true the compiler executes the block under that else if condition, otherwise, it skips and executes else block.
- By default else block will be executed, if all the remaining conditions are false.
- It must be noted in nested else if statement, if more than one else if statement is true then the first else if statement which satisfies the condition in it will alone execute.

C Nested else if Syntax

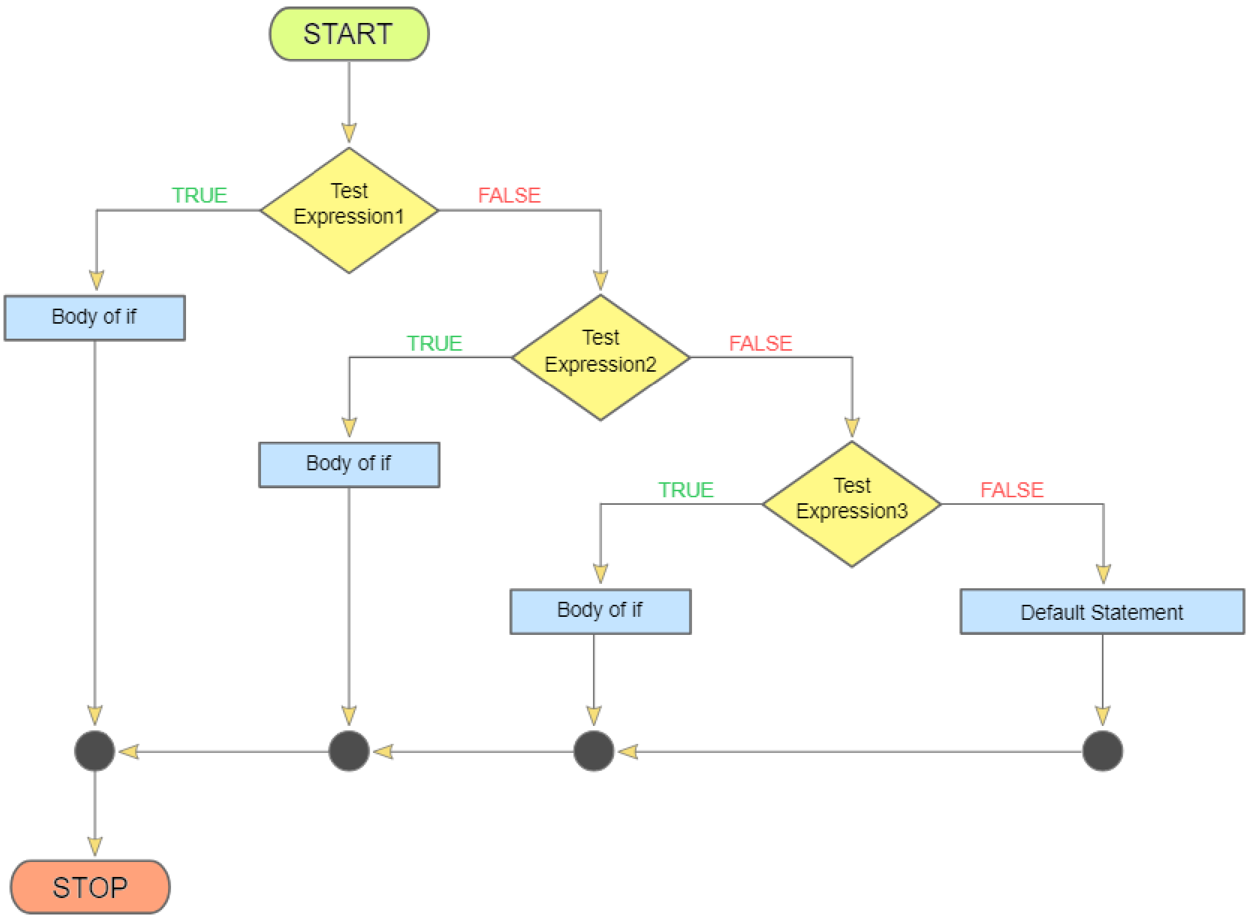
Syntax Nested else if Statement

Syntax

```
if(test_expression 1)
{
here goes statements;
}
else if(test_expression 2)
{
here goes statements;
}
else
{
here goes statements;
}
```

else-if Flowchart

The following flowchart will clearly demonstrate, how **else-if works?**



else if Statement Program

Lets take a look at the **else if statement** in action.

nestedif.c

```
#include <stdio.h> //header file section
int main()
{
int a;
printf("Enter a number: ");
scanf ("%d ", &a);
if(a > 0)
{
printf("\n%d is a positive number", a);
}
else if(a < 0)
{
printf("\n%d is a negative number", a);
}
else
{
printf("\n%d is a zero",a);
}
return 0;
}
```

- Enter a number: -47
- -47 is a negative number

Note:

Here, the compiler will ask user to enter the number, when user enters the number compiler will check if statement, as if statement resulted in false the compiler will check else if statement, as the condition is true compiler execute the block followed by **else if** statement and then exit.

C Break Statement

nt

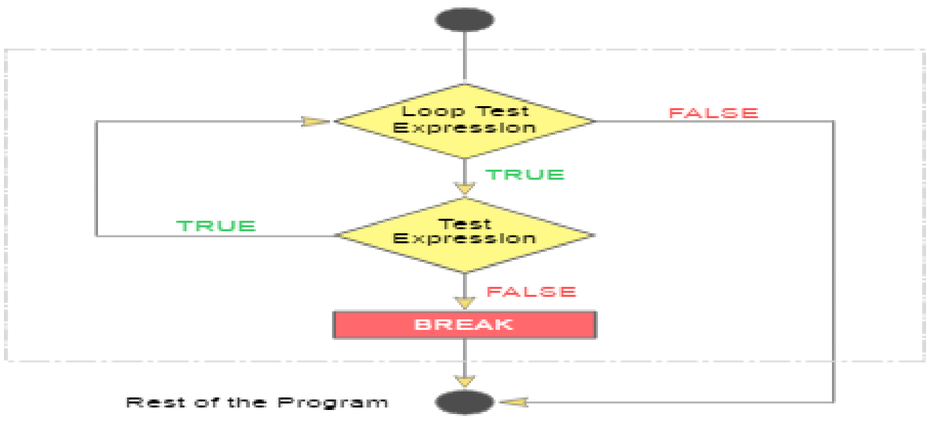
Why Break Statement?

Though **break statement** comes under decision-making statement, its most oftenly using in looping. One can use the break statement to break out from a loop or switch statement. When break is executed within a loop, the loop ends immediately, and execution continues with the first statement following the loop.

- The word **break** is a keyword that terminates the execution of the loop or the control statement and the control is transferred to the statement immediately following it.
- The break statement is mostly used in switch control statement.

Break Statement Flowchart

The following flowchart will clearly demonstrate how **break statement** works



Break Statement Program

Let us write a C program to demonstrate **break statement**.

break.c

```
#include <stdio.h> //header file section
int main()
{
int a;
for(a = 1;a <= 5;a++)
{
if(a == 4)
{
break;
}
printf("%d ", a);
}
return 0;
}
```

```
1 2 3
```

Note:

The loop will be terminated immediately, when a variable a = 4 is encountered.

C Continue Statement

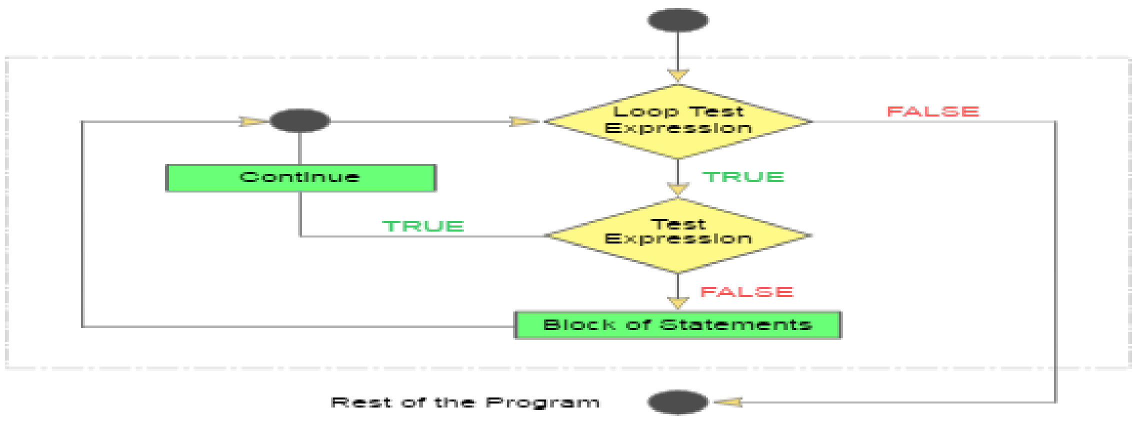
Why Continue Statement?

The **continue statement** enables you to skip to the next iteration in the loop containing the continue statement. The labeled continue statement enables you to skip to the next iteration in an outer loop enclosing the labeled continue that is identified by the label. The labeled loop need not be the loop immediately enclosing the labeled **continue.**

- The **continue** statement is exactly opposite to the break statement.
- The loop does not terminate when a **continue** statement is encountered.
- Instead, the **continue** statement skips that particular iteration.
- The **continue** statement is also called as bypass statement.

Continue Statement Flowchart

The following flowchart will help you to understand **how continue statement works**



Continue Statement Program

Let us write a C program to demonstrate **continue statement**

Continue Statement Program

continue.c

```
#include <stdio.h> //header file section
int main()
{
int a;
for(a = 1;a <= 5; a++)
{
if(a == 4)
{
continue;
}
printf("%d", a);
}
return 0;
}
```

```
1 2 3 5
```

Note:

When the variable reaches the value 4, the loop skips the statements within a block, only for that particular iteration.

C Switch Statement

We use the **switch statement** to select from multiple choices that are identified by a set of fixed values for a given expression. The expression that selects a choice must produce a result of an integer type other than long, or a string, or a value of an enumeration type. Thus, the expression that controls a **switch statement** can result in a value of type char, int, short, long, string, or an enumeation constant.

C switch Syntax And Facts

- A switch statement is an alternative to if-else-if statement.
- The switch statement has four important elements.
- The **test** expression
- The **case** statements
- The **break** statements
- The **default** statement

C Switch Syntax

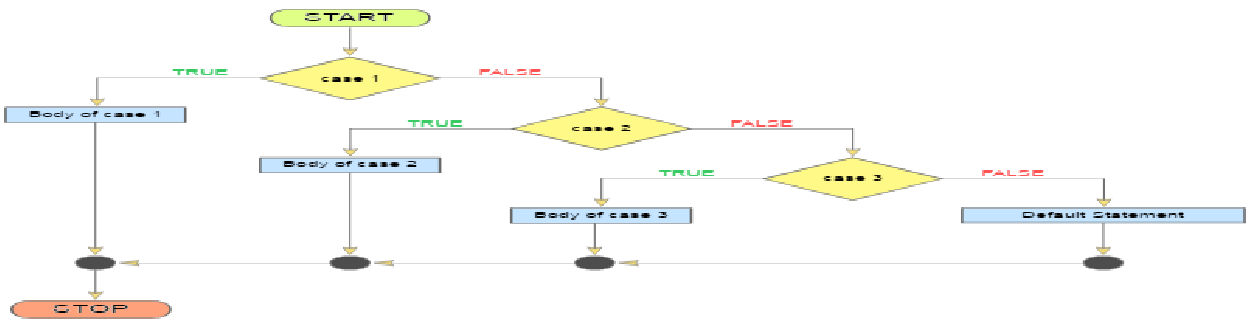
Syntax Switch Statement

Syntax

```
switch ( variable )
{
case value1:
statement;
break;
case value2:
statement;
break;
default:
statement;
}
```

Switch Statement Flowchart

The following flowchart clearly illustrate how **switch statement works**



C Switch Statement Program

Let's take a look at the **switch statement** in action. In the following program we will greet either dad or mom or yourself.

switch.c

```
#include <stdio.h>
int main()
{
int choice;
printf("Press 1 to greet your dad\n ");
printf("Press 2 to greet your mom\n ");
printf("Press 3 to greet yourself\n ");
scanf("%d ",&choice);
switch (choice)
{
case 1:
printf("\nHello dad,How are you? ");
break;
case 2:
printf("\nHello mom,How are you? ");
break;
case 3:
printf("\nHello awesome,How are you? ");
break;
default:
printf("\nInvalid choice ");
}
return 0;
}
```

- Press 1 to greet your dad
- Press 2 to greet your mom
- Press 3 to greet yourself
- 1
- Hello dad,How are you?

Note:

The above program offers three choices to a user. A choice(number) entered by the user is stored in a variable choice. In switch() statement the value is checked with all the case constants. The matched case statement is executed in which the line is printed according to the user's choice. If user's choice doesn't match with any case statement, then a statement followed by default will be executed.

C goto statement

goto statement in c sometimes called as goto operation in other languages like php.

Why goto Statement

C programming introduces a goto statement by which you can jump directly to a line of code within the same file.

Syntax And Facts

- The goto statement does not require any condition.
- The goto statement simply passes control anywhere in a program without testing any condition.

C goto Syntax

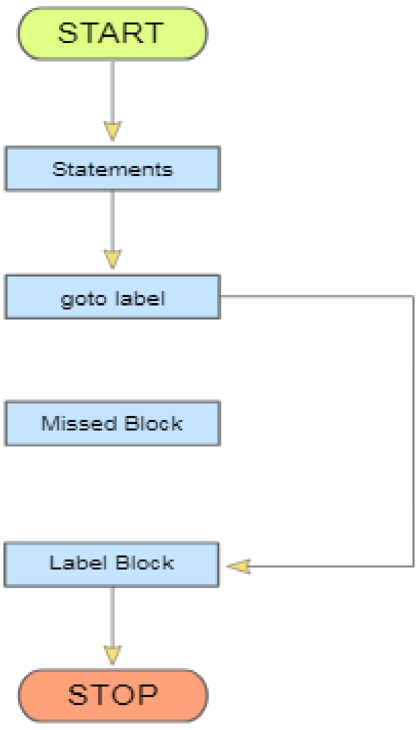
Syntax goto Statement

Syntax

```
goto Label;
.
.
Label:
{
statement n;
}
```

goto statement Flowchart

The following flowchart will clearly demonstrate, **how goto statement works**



goto statement Program

Let's take a look at the **goto statement** in action. In the following program we will print whether the number is positive or negative using **goto statement**

goto.c

```
#include <stdio.h>
int main()
{
int num;
printf("Enter a positive or negative integer number:\n ");
scanf("%d",&num);
if(num >= 0)
goto pos;
else
goto neg;
pos:
{
printf("%d is a positive number",num);
return 0;
}
neg:
{
printf("%d is a negative number",num);
return 0;
}
}
```

- Enter a positive or negative integer number:
- 2
- 2 is a positive number

Note:

In this program, the user entered number is checked for positive or negative number. If the user entered number is positive, the control is passes to pos : by goto statement. If the user entered number is negative, the control is passes to neg : by goto statement.

Control Loops In C

What Is Loop

A loop enables a programmer to execute a statement or block of statement repeatedly. The need to repeat a block of code arise in almost every program.

Varieties Of Loop

There are three kinds of loop statements you can use. They are

- **for** loop
- **while** loop
- **do while** loop

For Loop Program

In for loop control statement, initialization, condition and increment are all given in the same loop.

for.c

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

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

Note:

Here, for loop will iterate until the conditional statement at the center of for-loop fails.

While Loop Program

In While loop, initialization, condition and increment are all done in different lines.

while.c

```
#include <stdio.h>
int main()
{
int i = 0;
while(i<10)
{
printf("%d ",i);
i++;
}
return 0;
}
```

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

Note:

Here, while condition iterate until the condition inside it become false.

Do While Program

Here, it is same as while loop, but it will execute a block of code once irrespective of condition.

dowhile.c

```
#include <stdio.h>
int main()
{
int i = 0;
do
{
printf("%d ",i);
i++;
}while(i>10);
return 0;
}
```

```
0
```

Note:

Though 0 > 10 tends to false block of code following do was executed successfully.

C for loop

what is **for loop**?

Conceptually, **for loop** is a bit more complex than while loop and do while loop, though syntax is neat and compact way to write certain types of loops. Typically, a programmer need to use a **for loop** when you exactly know how many times you want a block of statement to be executed repeatedly.

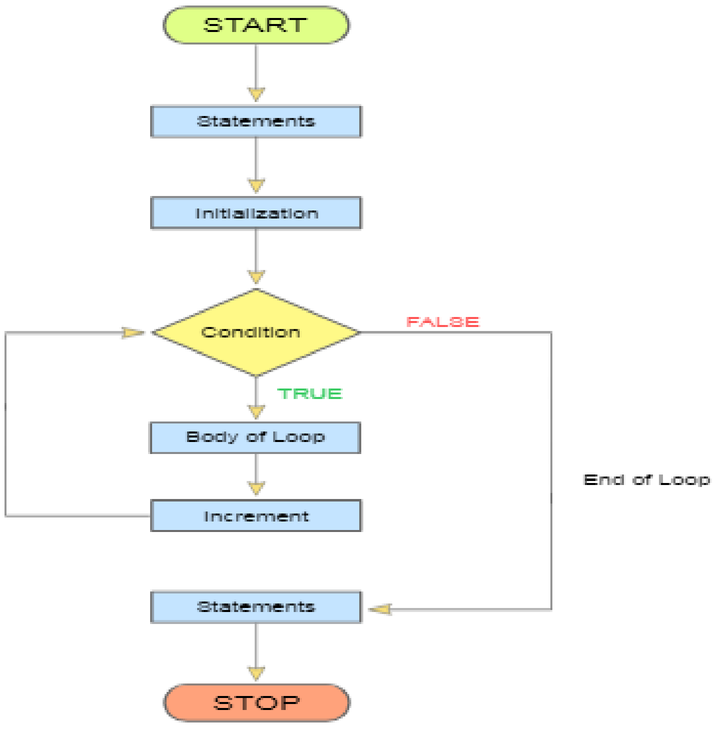
Syntax **for loop**

Let us see how neat a syntax of**for loop** is

Syntax

```
for (initializeCounter; testCondition; ++ or -- )
{
.
.
}
```

for loop FlowChart



for loop Realtime

Program **for loop**

The following example program will clearly explain the concept of **for loop**

for.c

```
#include <stdio.h>
int main()
{
int a;
for(a = 1;a <= 5;a++)
{
printf("%d ",a);
}
return 0;
}
```

```
1 2 3 4 5
```

Note:

The variable a is initialized to 1 for the first time when the program execution starts in. The condition a <= 5 is a test condition, which is tested for every iteration. The statements under a loop will be executed repeatedly until the test condition is true.

Infinite **for loop**

When the for statement contains no test condition between the **;**(semicolon) then the loop is said to be an infinite for loop.

Program Infinite **for loop**

The following C program will clearly demonstrate **infinite for loop**

goto.c

```
#include <stdio.h>
int main()
{
int a = 1;
for(;;)
{
printf("%d ", a++);
}
return 0;
}
```

```
1 2 3 4 5 ...........
```

Note:

for(;;) statement allows execution of a loop continued for infinite times.

C nested for Loop

Using a for loop within another for loop is said to be**nested for loop**. In nested for loop one or more statements can be included in the body of the loop. In nested for loop, the number of iterations will be equal to the number of iterations in the outer loop multiplies by the number of iterations in the inner loop.

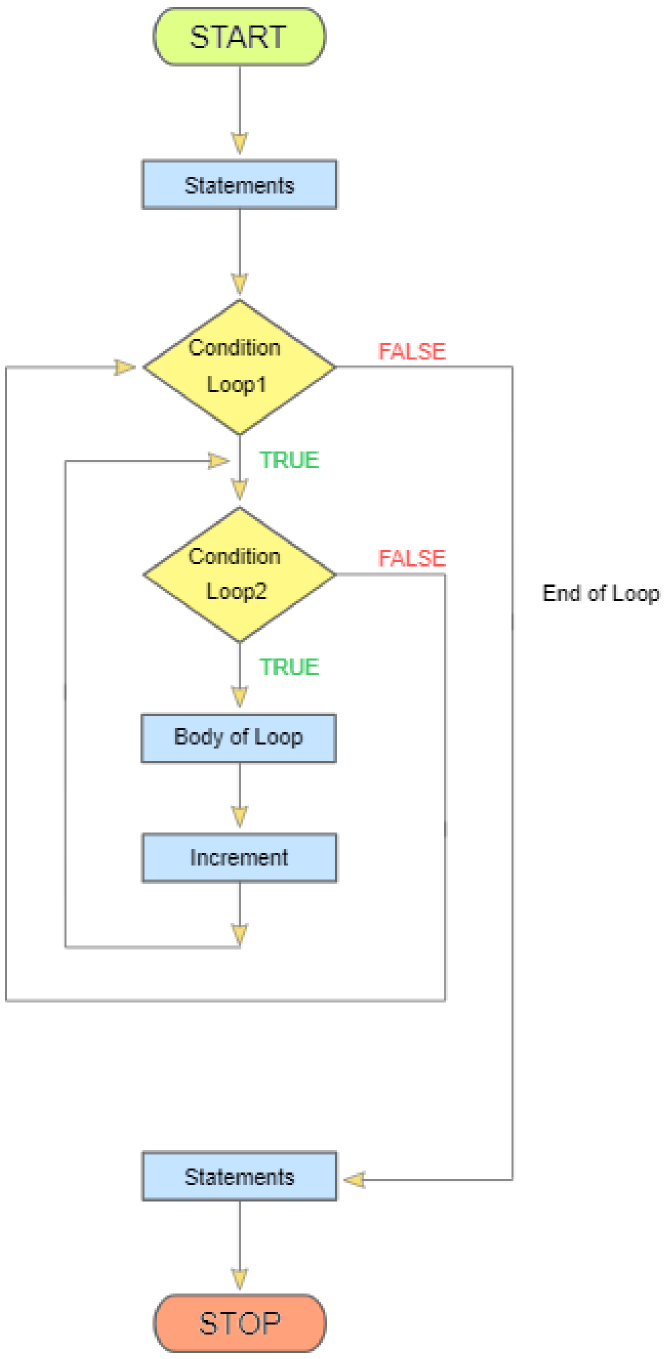
nested for Loop Syntax

Let us see how neat a syntax of**nested for loop** is

Syntax

```
for (initialize counter; test condition; ++ or --)
{
for (initialize counter; test condition; ++ or --)
{
.// inner for loop
}
.// outer for loop
}
```

nested for loop Flowchart



C program - **nested for** loop

The following example program will clearly explain the concept of **nested for loop**

nestedfor.c

```
#include <stdio.h>
int main()
{
int a, b;
for(a = 1; a <= 5; a++)
{
for(b = 1; b <= 5; b++)
{
printf("%d ", b);
}
printf("\n");
}
return 0;
}
```

- 1 2 3 4 5
- 1 2 3 4 5
- 1 2 3 4 5
- 1 2 3 4 5
- 1 2 3 4 5

Note:

The variable a and b is initialized to 1 for the first time when the program execution starts in the for loop. The variable a belongs to outer for loop and a variable b belongs to inner for loop. for a = 1, inner for loop will be executed 5 times. Thus, total number of iterations is 25.

C while loop

The **while** loop, otherwise said to be a splitted for loop. Any program which are written using for loop can be written using **while** loop. Here, a block followed by the **while** loop will be executed repeatedly until the test condition is true.

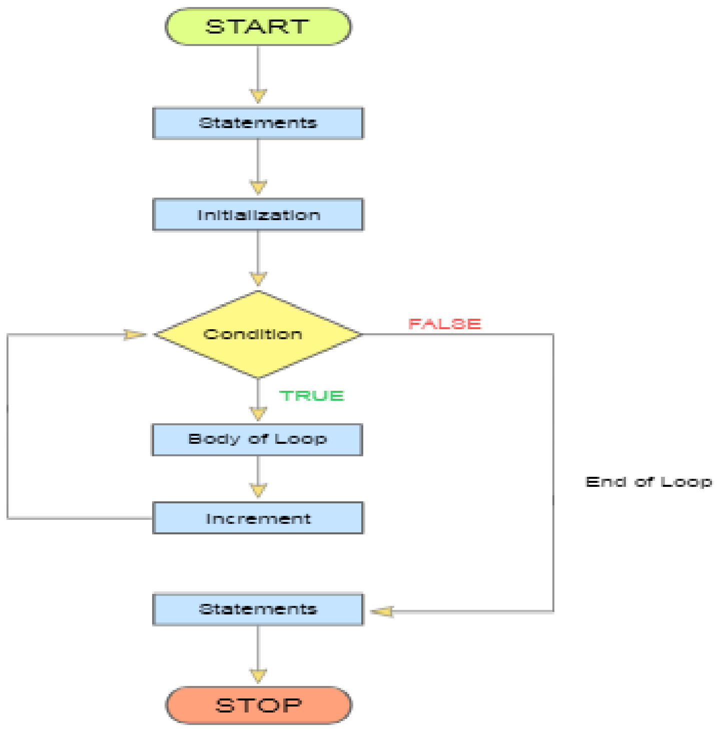
while loop Syntax

Let us see how neat a syntax of **while loop** is

Syntax

```
while(test condition)
{
Body of the loop
.
}
```

while loop Flowchart



while loop Realtime

C program - **while** loop

The following example program will clearly explain the concept of **while loop**

while.c

```
#include <stdio.h>
int main()
{
int a = 1;
while(a <= 5)
{
printf("%d ", a);
a++;
}
return 0;
}
```

```
1 2 3 4 5
```

Note:

A variable a is initialized to 1 at the time of initializing a variable. The condition a = 5 is the test condition, which is tested for every iteration. A block followed by a while loop will be executed repeatedly until the test condition is true. a++ is used to increment the value of a every time after printing the value of a.

C nested while loop

Using While loop within while loops is said to be**nested while loop**. In nested while loop one or more statements are included in the body of the loop. In nested while loop, the number of iterations will be equal to the number of iterations in the outer loop multiplies by the number of iterations in the inner loop which is most same as nested for loop.

nested while loop Syntax

Let us see how neat a syntax of**nested while loop** is

Syntax

```
while (test condition)
{
while (test condition)
{
.// inner while loop
}
.// outer while loop
}
```

C program - **nested while** loop

The following example program will clearly explain the concept of**nested while loop**

nestedwhile.c

```
#include <stdio.h>
int main()
{
int a = 1, b = 1;
while(a <= 5)
{
b = 1;
while(b <= 5)
{
printf("%d ", b);
b++;
}
printf("\n");
a++;
}
return 0;
}
```

- 1 2 3 4 5
- 1 2 3 4 5
- 1 2 3 4 5

- 1 2 3 4 5
- 1 2 3 4 5

Note:

The variable a and b is initialized to 1 at the time of initializing variable. The condition a <= 5 is the outer test condition and b <= 5 is the inner test condition which is tested for every iteration. Here, the inner while loop will be executed for 5 times for every time the test condition in the outer loop is true.

C do while loop

The do while loop evaluates the condition only after the execution of the statements in its body. The statements within the do-while loop executed at least once. A do while loop is also called as bottom tested loop.

do while loop Syntax

Let us see how neat a syntax of **do while loop** is

Syntax

```
do
{
statement 1;
statement n;
}
while(test expression);
```

C program - **do while** loop

The following example program will clearly explain the concept of **do while loop**

dowhile.c

```
#include <stdio.h>
int main()
{
do
{
printf("I will display once ");
}
while(2 > 10);
return 0;
}
```

```
I will display once
```

Note:

Although the test condition in the while loop may false for the very first time. The statements of do-while loop will be executed at least once.

C Functions

What Is a Function?

In general terms, a function is also called a subroutine in some other languages like php, python and so on. Function is a self contained block of code that performs a specific task. A programmer define a function using its predefined syntax, a programmer can then call that function from elsewhere in your program.

Why Function?

- They avoid duplicating codes.
- They make it easier to eliminate errors.
- Function help you break down a big project.

Function and Arguments

A function often accepts none, one or more arguments, which are values passed to the function by the code that calls it. The function is capable of read and work on those arguments. A function may also optionally return a value(non void type) that can then be read by the calling code. in this way, the calling code will communicate with the function.

How Program thinks?

Program think of a function as a black box. The code that calls a function doesn't need to know what's inside the function, it just uses the function to get the job done.

Syntax

Syntax

```
return-type function-name (arguments);
main()
{
.
.
statements within main function;
printf("%d", function-name (arguments));    //function call
.
}
return-type function-name (arguments)      //function definition
{
.
.
statements within user-defined function;
.
return variablename;
}
```

Types Of Function

There are two types of functions they are,

- Library Functions
- User Defined functions

Library Functions

C provides many pre-defined functions to perform a task, those functions are defined in an appropriate header files.

- printf()
- scanf()
- strlen()
- sqrt() and so on

User Defined Functions

C allows users or programmers to define a function according to their requirement. The main() function is also a user-defined function because the statements inside the main function is defined by the user, only the function name is defined in a library.

Function Declaration

- The function declaration statement informs the compiler about a function return type, function name and parameters or arguments type.
- Syntax : return-type function-name (arguments);
- **Return type :** return type is the data type of the value which is given back to the calling function.
- **function name :** function name is the name of a function. A function is called by using the function name. The naming rules are same as variable naming.

- **parameters or arguments type :** C allows programmers to pass information to the called function from the calling function by using parameters. These parameters are variables of data type.

C program - **Functions**

The following example program will clearly explain the concept of**functions**

function.c

```
#include <stdio.h>
int main()
{
//main function definition
int a = 5, b = 10;
int sum;
printf("The value of a and b : %d %d ", a, b);
sum = add(a, b);    //function call
printf("\nsum = %d ", sum);
}
//function definition
int add(int a, int b)
{
int c;
c = a + b;
return c;    //returns a integer value to the calling function
}
```

- The value of a and b : 5 10
- sum = 15

Note:

The above C program illustrates that a function declaration, function definition and function call in a program.

C User Defined Functions

What Is User Defined Function

" What you have to do when I call you"
yes, the above quotes clearly define the purpose of user defined function. By speaking conceptually, a programmer can define their own sets of code inside a function and use it for n number of times using its function name.

Types

The user-defined functions are classified into four types, according to parameters and return value.

- Functions **without arguments** but **with return values**
- Functions **without arguments** and **without return values**
- Functions **with arguments** but **without return values**
- Functions **with arguments** and **without return values**

#1 Functions Without Arguments Without Return Values

The calling function will not send parameters to the called function and called function will not pass the return value to the calling function.

C program - **Functions Without Arguments Without Return Values**

function1.c

```
#include <stdio.h>
void add();     //function declaration
int main()
{
//main function definition
add();    //function call without passing arguments
}
//function definition
void add()
{
int a = 5, b = 10;
int c;
printf(" The values of a and b : %d %d ", a, b);
c = a + b;
printf(" \nsum : %d ", c);
//no return values to the calling function
}
```

- The values of a and b : 5 10
- sum = 15

Note:

The program illustrates that, there are no parameters passed through the calling function and no return value to the calling function main().

#2 Functions With Arguments Without Return Values

The calling function will pass parameters to the called function but called function will not pass the return value to the calling function.

C program - **Functions With Arguments Without Return Values**

function2.c

```
#include <stdio.h>
void add(int,int);     //function declaration
int main()
{
//main function definition
int sum;
int a = 5, b = 10;
printf(" The values of a and b : %d %d ", a, b);
add( a, b);    //function call with passing arguments to called function
}
// function definition
void add(int a, int b)
{
int c;
c = a + b;
printf(" \nsum : %d ", c);
//no return values to the calling function main
}
```

- The values of a and b : 5 10
- sum = 15

Note:

The program illustrates that, parameters are passed to the called function(add) from calling function(main) but no return values are passed from called function(add) to the calling function (main).

#3 Functions Without Arguments With Return Values

The calling function will not pass parameters to the called function but called function will pass the return value to the calling function.

C program - **Functions Without Arguments With Return Values**

function3.c

```
#include <stdio.h>
int add();     //function declaration
int main()
{
//main function definition
int sum;
sum = add();    //function call without passing arguments to called function
printf(" \nsum = %d ", sum);
}
// function definition
int add()
{
int c;
int a = 5, b = 10;
printf(" The values of a and b : %d %d ", a, b);
c = a + b;
return c;    //passing return values to the calling function main
}
```

- The values of a and b : 5 10
- sum = 15

Note:

The program illustrates that, no parameters are passed to the called function(add) from calling function(main) but return value is passed from called function(add) to the calling function (main). The return values is assigned to the variable sum.

#4 Functions With Arguments With Return Values

The calling function will pass parameters to the called function and called function also pass the return value to the calling function.

C program - **Functions With Arguments With Return Values**

function4.c

```
#include <stdio.h>
int add(int, int);     //function declaration
int main()
{
//main function definition
int sum;
int a = 5, b = 10;
printf(" The values of a and b : %d %d ", a, b);
sum = add( a, b);    //function call with passing arguments
printf(" \nsum = %d ", sum);
}
// function definition
int add(int a, int b)
{
int c;
c = a + b;
return c;    //passing return values to the calling function main
}
```

- The values of a and b : 5 10
- sum = 15

Note:

The program illustrates that, parameters are passed to the called function (add) from calling function (main) and return value is passed from called function (add) to the calling function (main). The return values is assigned to the variable sum.

C Call By Value And Call By Reference

Accessing Function

There are two different way a function can be accessed. Eitheir by passing arguments (parameters) to a function or by passing nothing to a function. Function which doesn't accept any parameters are mostly void type.

Accessing function using parameters are of two types. They are

- **Call by value**
- **Call by Reference or address**

Before getting into the play of call by value and call by reference, one need to know what is actual parametes and formal parameters.

What Is Actual Parameters

Actual parameters are specified in the function call. In simple terms, value or reference passing at the time of function call.

What Is Formal Parameters

Formal parameters are specified in the function declaration and function definition. In simple terms, variables with datatype at the function declaration and definition.

Call By Value

- Call by value is the method of passing values as the parameters to the function.
- In this method, the values of actual parameters are passed to the formal arguments.
- In call by value method, the values of actual parameters are not changed even they are changed in the called function.
- Because the actual parameters pass the copy of original parameters to the called function.

C program - **Call By Value**

Let us write a c program to demonstrate how to use call by value?

callbyvalue.c

```
#include <stdio.h>
int pass(int, int);     //function declaration
int main()
{
//main function definition
int a = 5, b = 10;
printf("The values of a and b in main function before function call : %d %d ", a,
b);
pass(a, b);    //function call with passing arguments as values
printf("\nThe values of a and b in main function after function call : %d %d ", a,
b);
}
// function definition
int pass(int a, int b)
{
a = 10; b = 5;     //This change will not affect the actual parameters
printf("\nThe values of a and b in called function : %d %d ", a, b);
}
```

- The values of a and b in main function before function call : 5 10
- The values of a and b in called function : 10 5
- The values of a and b in main function after function call : 5 10

Note:

The program illustrates that, the actual parameters are passed as copy of original parameters to the called function. So, the changes of values inside the called function does not affect the parameters of calling function.

Call by Reference

- Call by reference is the method of passing variable addresses as the parameters to the function.
- In call by reference method, changes made in called function will affect the parameters inside the main function.

C program - **Call By Reference**

Let us write a c program to demonstrate how to use call by value?

callbyreference.c

```
#include <stdio.h>
int pass(int*, int*);     //function declaration
int main()
{
//main function definition
int a = 5, b = 10;
printf("The values of a and b in main function before function call : %d %d ", a,
b);
pass(&a, &b);    //function call with passing arguments as addresses
printf("\nThe values of a and b in main function after function call : %d %d ", a,
b);
}
// function definition
int pass(int * a, int *b)
{
*a = 10; *b = 5;     //This change will affect the actual parameters
printf("\nThe values of a and b in called function : %d %d ",*a, *b);
}
```

- The values of a and b in main function before function call : 5 10
- The values of a and b in called function : 10 5
- The values of a and b in main function after function call : 10 5

Note:

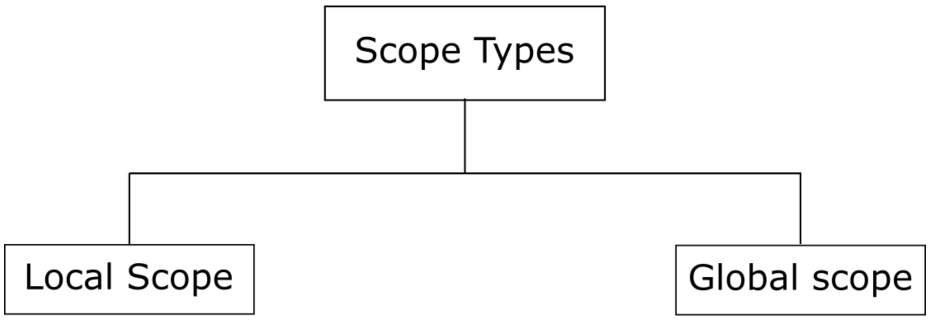
The program illustrates that, changes made in called function affect the parameters inside the main function. So, the changes made in the parameters are permanent.

Scope rules in C

What Is Scope

- A scope is a region of the program where the variables can be accessed.
- A scope contains a group of statements and variables. The variables declared within a block can be accessed only within that block.
- Programmers can declare the variables both inside and outside of block.

Types of Scope



Local Scope In C

{ } **2braces.com**/c-programming/c-local-variables

- The variables declared inside a function or a block is known as local variables.
- Local variables can be accessed only within function or block.
- Local variables can not be accessed outside a function or a block.

C program - **Local Scope**

localscope.c

```
#include <stdio.h>
int main()
{
int m = 5;
void display(){
int m = 10; //Local variable declaration
printf("Value of M inside a function is %d",m);
}
display();
printf("\nValue of M outside a function is %d",m);
return 0;
}
```

- Value of M inside a function is 10
- Value of M outside a function is 5

Note:

m is a variable used locally and globally.

Did You Know?

If there is no declaration of variable in a subfunction. Then the value of a variable will be fetched from global declaration. Let us check the same program that we used early.

localscope.c

Note:

m is a variable used locally, but fetches a values globally.

C Global Scope

{ } **2braces.com**/c-programming/c-global-variables

- The variables declared right before the main function is called as global variables.
- Global variables can be accessed through out the program.

C program - **Global Scope**

globalscope.c

```
#include <stdio.h>
int m1 = 5, m2 = 10;//global variable declaration
void add();
int main()
{
int mul;
mul = m1 * m2;
printf("\nMultiplication of %d and %d : %d ", m1, m2, mul);
add();
return 0;
}
void add()
{
int sum;
sum = m1 + m2;
printf("\nSum of %d and %d : %d ", m1, m2, sum);
}
```

- Multiplication of 5 and 10 : 50
- Sum of 5 and 10 : 15

Note:

The variables m1 and m2 are declared outside all functions. So, those can be accessed in any function within a program.