Programming for Problem Solving (PPS)

Unit 1: Number system

Number system-The number systems give us different ways to represent numbers.

BASE(Radix)-

- A base of a number system (or notation) is the number of symbols that we use to represent the numbers.
- The notations are usually named after their base – binary, octal, decimal, hexadecimal.
- When a number is written, its base should be written after the number in subscript.
  123_{10}, 110010_2, $1F4_{16}$
  If the base is not provided, the default is that the number is in decimal.
- If our radix is larger than 10, we will need to invent "new digits". We will use the letters of the alphabet: the digit A represents 10, B is 11, K is 20, etc.
- **Positional and non positional number systems**
- **Positional**
- the value of a digit depends on its position

10 – the digit 1 has a value of ten
1000 – the same digit 1, has a value of a thousand, because it is in a different position.

Non-Positional

the value of the digits does not depend on their position

Eg- roman numeral system

X : the symbol 'X' means 10
XXX : each symbol has the value of 10. The number “XXX” is equal to their sum 30

Value representation-

All positional systems that we use, represent the numbers in the same manner. To represent a given number we need to break it down to a sum of exact powers of the base of that notation.

Example-

The number 1234 = 1000 + 200 + 30 + 4, where

1000 = 1 * 103 or in words 1 times 10 to the power of 3

200 = 2 * 102 or .. 2 times ten to the power of 2

30 = 3 * 101 or .. 3 times 10 to the power of 1

4 = 4 *100 or .. 4 times 10 to the power of 0. Remember, each number to the power of 0 is equal to 1. So 4 = 4 * 1 = 4 * 100

1234 = 1 * 103 + 2 * 102 + 3 * 101 + 4 * 100

Some types of number system-

1. Binary Numbers- base 2

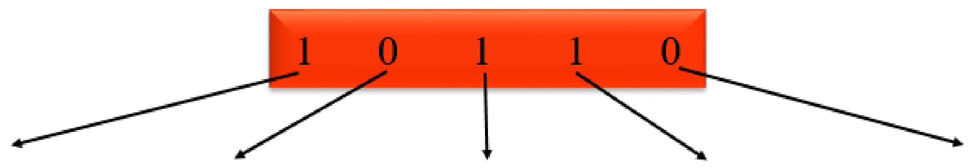
2. octal Numbers- base 8

3. Decimal Numbers- base 10

4. Hexadecimal Numbers- base 16

1. Binary system

- In computer applications, where binary numbers are represented by only two symbols or digits, i.e. 0 (zero) and 1(one).
- The binary numbers here are expressed in the base-2 numeral system.



2^4*1	2^3*0	2^2*1	2^1*1	2^0*0
16*1	8*0	4*1	2*1	1*0
16 +	0 +	4 +	2 +	0 = 22

This is how you convert binary into decimal.

Decimal to Binary conversion

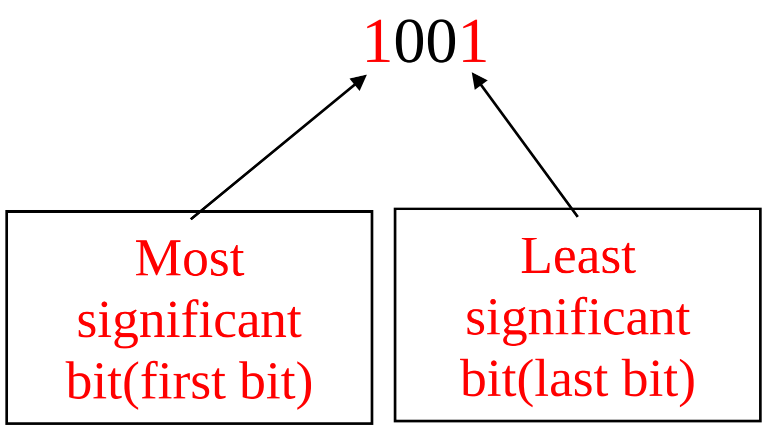
Step 1:-

Divide number by 2 and divide the quotient again by 2. Repeat until quotient becomes zero. Consider 9.

Dividend	Remainder
9/2=4	1
4/2=2	0
2/2=1	0
1/2=0	1

Step 2:-

Write remainder in reverse chronological order(bottom to top).


1001
Most significant bit(first bit)
Least significant bit(last bit)

Bit is single unit. 9 has four bits.

Bit conversion chart-

1 byte (B) = 8 bits

1 Kilobytes (KB) = 1024 bytes

1 Megabyte (MB) = 1024 KB

1 Gigabyte (GB) = 1024 MB

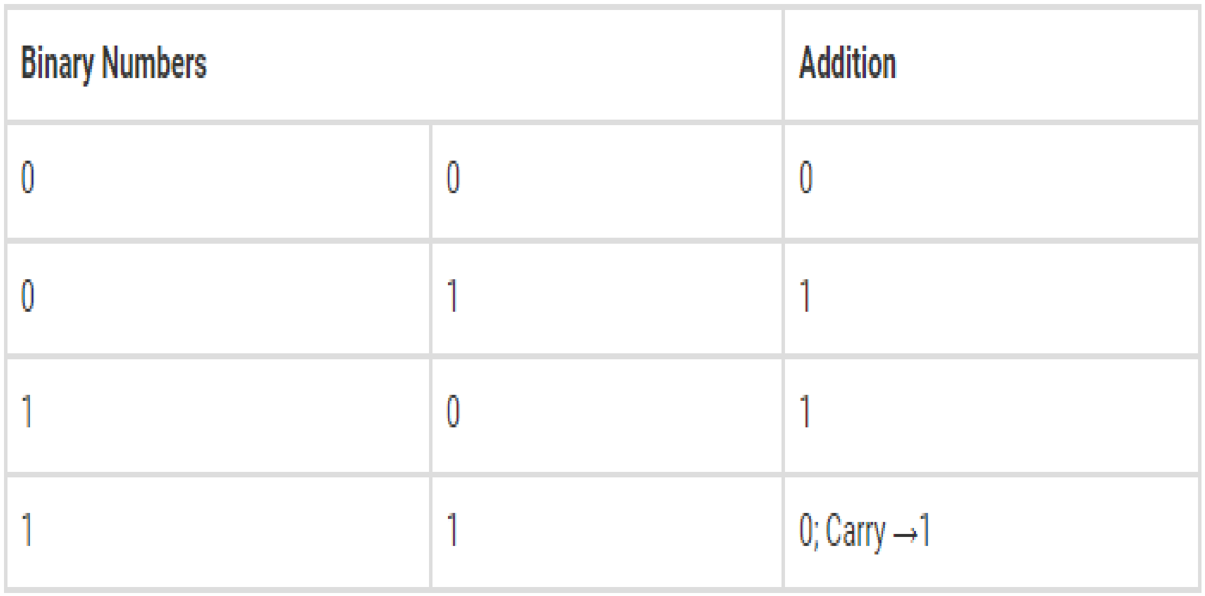
1 Terabyte (TB) = 1024 GB

1 Exabyte (EB) = 1024 PB

1 Zettabyte = 1024 EB

1 Yottabyte (YB) = 1024 ZB

Binary Arithmetic-



Binary Numbers		Addition
0	0	0
0	1	1
1	0	1
1	1	0; Carry $\rightarrow$1

1+1+1= 1 with carry 1

Adding two binary numbers will give us binary-

Eg- lets add 0101 and 1001

0 1 0 1

1 0 0 1

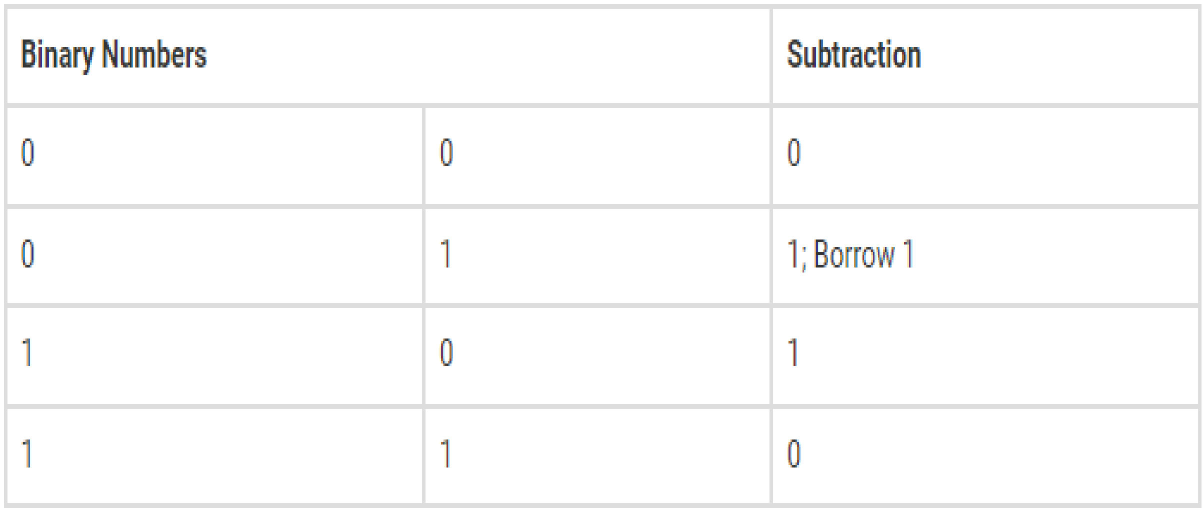
----

1 1 1 0

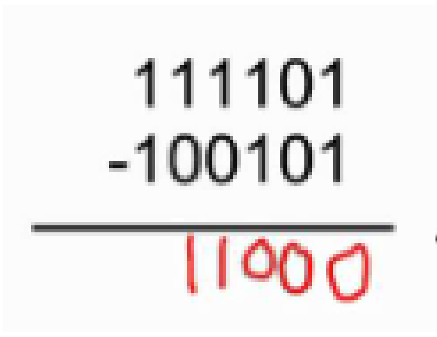
lets add 1011 and 1101

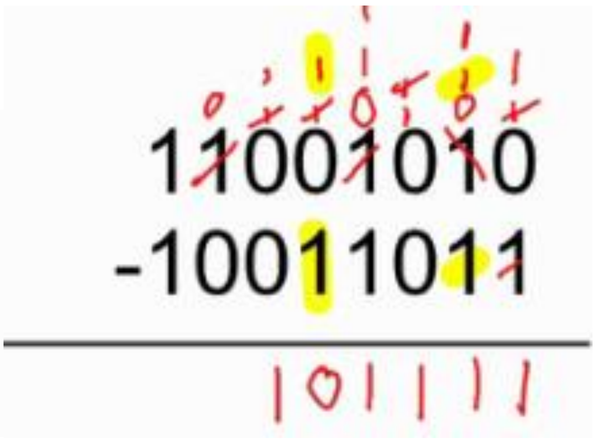
```
          1            0            1          1

          1            1            0          1

-------------------------------------------------

1         1             0            0         0
```

Binary subtraction-



Binary Numbers		Subtraction
0	0	0
0	1	1; Borrow 1
1	0	1
1	1	0





Binary Multiplication-

It is the same as decimal multiplication.

```
      1101
    × 1010
 ------------
      0000
     1101
    0000
   1101
 ------------
  10000010
 ------------
```

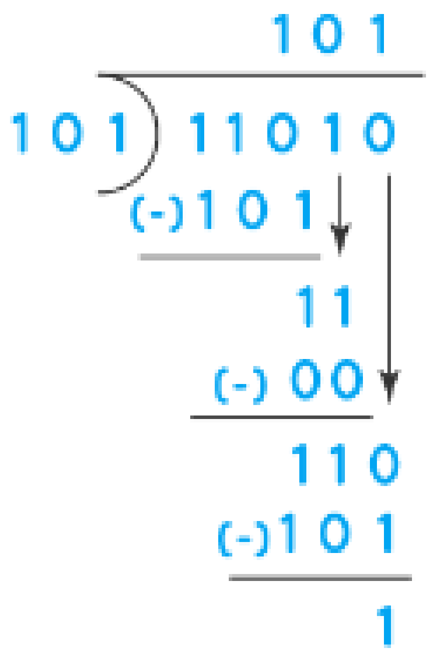
Binary Division-

It is the same as decimal division.

Step 1: Compare the divisor with the dividend. If the divisor is larger, place 0 as the quotient, then bring the second bit of the dividend down. If the divisor is smaller, multiply it with 1 and the result becomes the subtrahend. Then, subtract the subtrahend from the minuend to get the remainder.

Step 2: Then bring down the next number bit from the dividend portion and perform step 1 again.

Step 3: Repeat the same process until the remainder becomes zero or the whole dividend is divided.



Representation of binary-

1. Sign representation-

In sign-magnitude representation, the Most Significant bit of the number is a sign bit and the remaining bit represents the magnitude of the number in a true binary form. For example, if some signed number is represented in the 8-bit sign-magnitude form then MSB is a sign bit and the remaining 7 bits represent the magnitude of the number in a true binary form.

+ 34 = 0 0 1 0 0 0 1 0

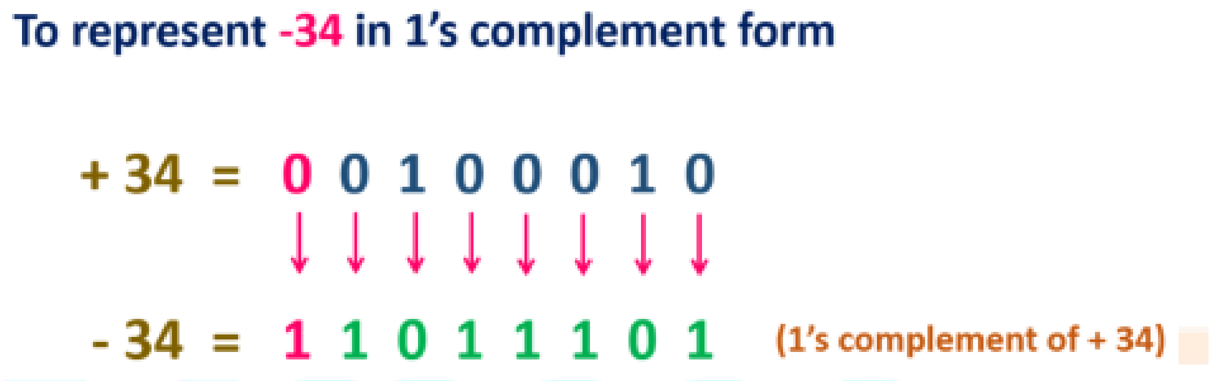
- 34 = 1 0 1 0 0 0 1 0

Since the magnitude of both numbers is the same, the first 7 bits in the representation are the same for both numbers. For +34, the MSB is 0, and for -34, the MSB or sign bit is 1.

2. 1s compliment-

In 1’s complement representation, the representation of the positive number is same as the negative number. But the representation of the negative number is different.

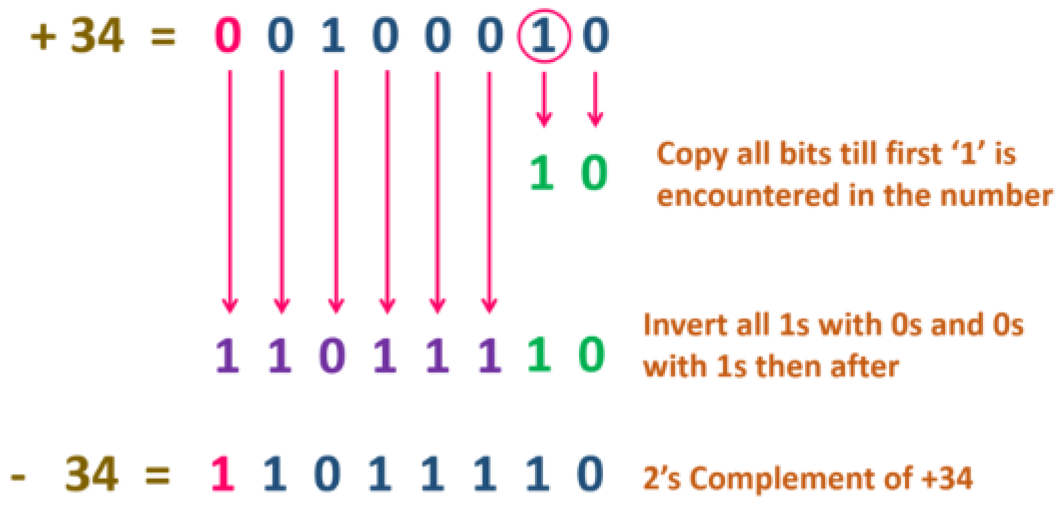
Invert all 1s in that number by 0s and 0s by 1s in that number. The corresponding inverted number represents the -34 in 1’s complement form. It is also called 1s complement of the number +34.



3. 2s compliment-

In 2’s complement representation also, the representation of the positive number is same as1’s complement and sign-magnitude form.

But the representation of the negative number is different. For example, if we want to represent -34 in 2’s complement form then


+ 34 = 0 0 1 0 0 0 1 0
1 0
Copy all bits till first '1' is encountered in the number
1 1 0 1 1 1 1 0
Invert all 1s with 0s and 0s with 1s then after
- 34 = 1 1 0 1 1 1 1 0
2's Complement of +34

2. Octal Numbers

A number system which has its base as 'eight' is called an Octal number system. It uses numbers from 0 to 7.

If we solve an octal number, each place is a power of eight.

(124)8 = 1 × 8^2 + 2 × 8^1 + 4 × 8^0

Suppose 560 is a decimal number, convert it into an octal number.

Solution: If 560 is a decimal number, then,

560/8 = 70 and the remainder is 0

70/8 = 8 and the remainder is 6

8/8 = 1 and the remainder is 0

And 1/8 = 0 and the remainder is 1

So the octal number starts from MSD to LSD, i.e. 1060

Therefore, 56010 = 10608

Convert $(100010)_2$ to an octal number.

Solution: With the help of the table we can write,

100→4

and 010→2

Therefore, $(100010)_2$ = 42

Similarly, we can convert an octal number to a binary number with the help of the table.

Octal Number	Equivalent Binary Number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

4. hexadecimal Numbers

The **hexadecimal number system** is a type of number system, that has a base value equal to 16. It is also pronounced sometimes as **'hex'**. Hexadecimal numbers are represented by only 16 symbols. These symbols or values are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

You have learned how to convert hexadecimal numbers to decimal numbers. Now let us find out how we can convert a decimal number into a hexadecimal number system. Follow the below steps:

Firstly divide the number by 16

Take the quotient and divide again by 16

The remainder left will produce the hex value

Repeats the steps until the quotient has become 0

Example: Convert (242)10 into hexadecimal.

Solution: Divide 242 by 16 and repeat the steps, till the quotient is left as 0.

decimal to hexadecimal number system

Therefore, (242)10 = (F2)16

$$\begin{array}{r|lll} 16 & 242 & & \\ \hline 16 & 15 & 2 & \rightarrow 2 \\ \hline & 0 & 15 & \rightarrow F \end{array}$$

To convert octal to hex, we have to first convert octal number to decimal and then decimal to hexadecimal. Let us understand it with the help of an example;

Example: Convert $(121)_8$ into hexadecimal.

Solution: First convert 121 into decimal number.

$\Rightarrow 1 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$
$\Rightarrow 1 \times 64 + 2 \times 8 + 1 \times 1$
$\Rightarrow 64 + 16 + 1$
$\Rightarrow 81$

$(121)_8 = 81_{10}$

Now converting 81_{10} into a hexadecimal number.

Therefore, $81_{10} = 51_{16}$

$$\begin{array}{r|l l}
16 & 81 & \\
\hline
16 & 5 & 1 \rightarrow 1 \\
\hline
 & 0 & 5 \rightarrow 5
\end{array}$$

Binary to hexadecimal conversion is a simple method to do. You just have to put the values of the binary number to the relevant hexadecimal number.

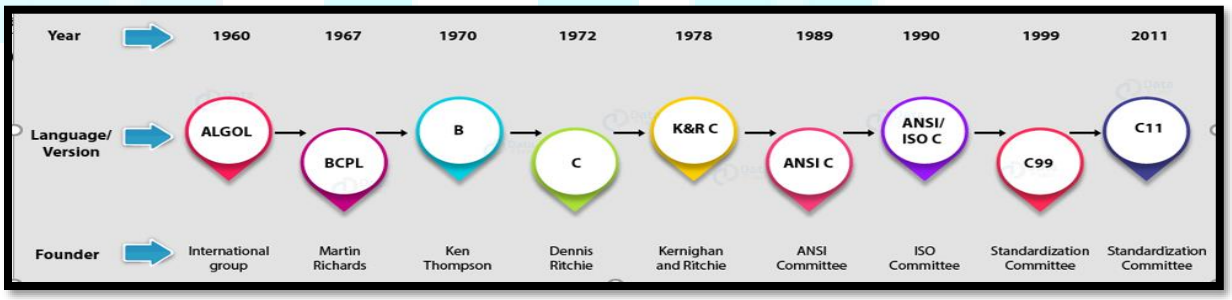
Example: Convert $(11100011)_2$ to hexadecimal.
Solution: From the table, we can write, 11100011 as E3.

Therefore, $(11100011)_2 = (E3)_{16}$

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

Unit 2: Introduction to C programming

- C is a general purpose language which is very closely associated with UNIX for which it was developed in Bell Laboratories.
- Most of the programs of UNIX are written and run with the help of 'C'.
- Many of the important ideas of 'C' stem are from BCPL by Martin Richards.
- In 1972, Dennis Ritchie at Bell Laboratories wrote C Language which caused a revolution in computing world .
- From beginning C was intended to be useful for busy programmers to get things done easily because C is powerful, dominant and supple language.
- As many features came from B it was named as 'C'.



Year	1960	1967	1970	1972	1978	1989	1990	1999	2011
Language/ Version	ALGOL	BCPL	B	C	K&R C	ANSI C	ANSI/ ISO C	C99	C11
Founder	International group	Martin Richards	Ken Thompson	Dennis Ritchie	Kernighan and Ritchie	ANSI Committee	ISO Committee	Standardization Committee	Standardization Committee

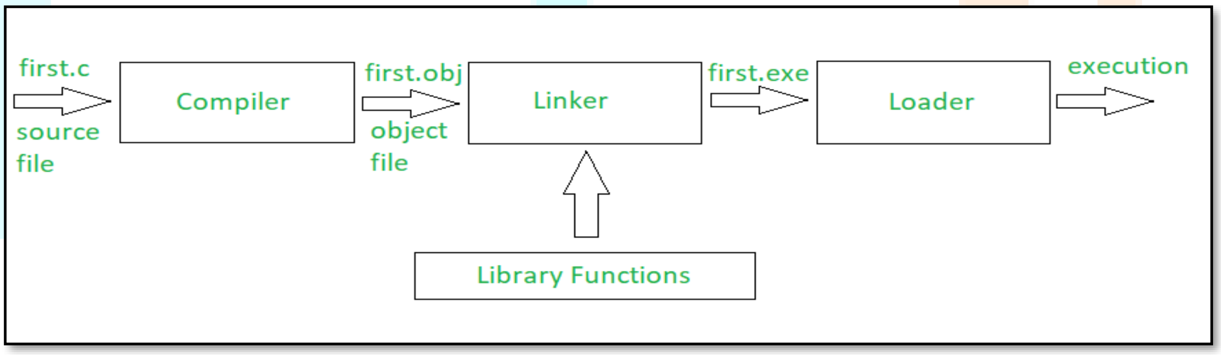
Features of C Programming

- Procedural Language.
- Fast and Efficient.
- Modularity.
- Statically Type.
- General-Purpose Language.
- Rich set of built-in Operators.

- Libraries with rich Functions.
- Middle-Level Language.
- Portability.
- Easy to Extend.

Running a C Program

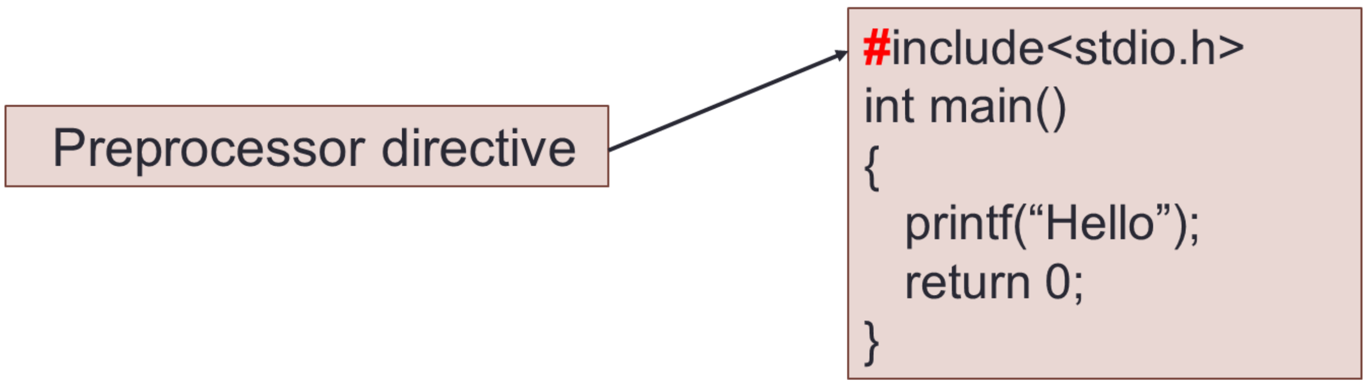
- Compile the program – This will generate an .exe file (executable)
- Run the program (Actually the exe created out of compilation will run and not the .c file)
- In different compiler we have different option for compiling and running.



Structure Of C Program

Preprocessor Directives-

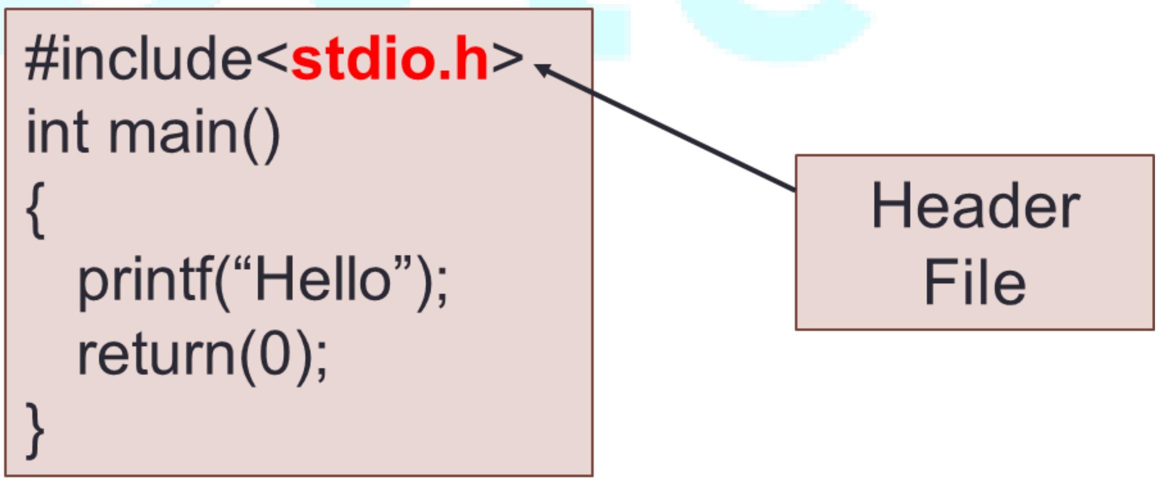
- The preprocessor directives give instruction to the compiler to preprocess the information before actual compilation starts.
- All preprocessor directives begin with #, and only white-space characters may appear before a preprocessor directive on a line. Preprocessor directives are not statements, so they do not end with a semicolon (;).



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

Header Files

- In C language, header files contain the set of predefined standard library functions.
- The "#include" preprocessing directive is used to include the header files with ".h" extension in the program
- Header files generally contain definitions of data types, function prototypes and C preprocessor commands.



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

Following are the few header files that are available in C:

- stdio.h :- Input/Output functions
- conio.h :- Console Input/Output functions
- stdlib.h :- General utility functions

- math.h :- Mathematics functions
- string.h :- String functions
- ctype.h :- Character handling functions
- time.h :- Date and time functions

Main function

- This is the “Entry Point” of a program.
- When a file is executed, the start point is the main function.
- From main function the flow goes as per the programmers choice.
- There may or may not be other functions written by user in a program.

Return statement

- It is used to return a value from the function or stop the execution of the function.
- return 0 in the main function means that the program executed successfully.
- return 1 or any non zero number in the main function means that the program does not execute successfully and there is some error.
- Nowadays even if you don’t use return statement program will not show error as by default it is added by compiler.

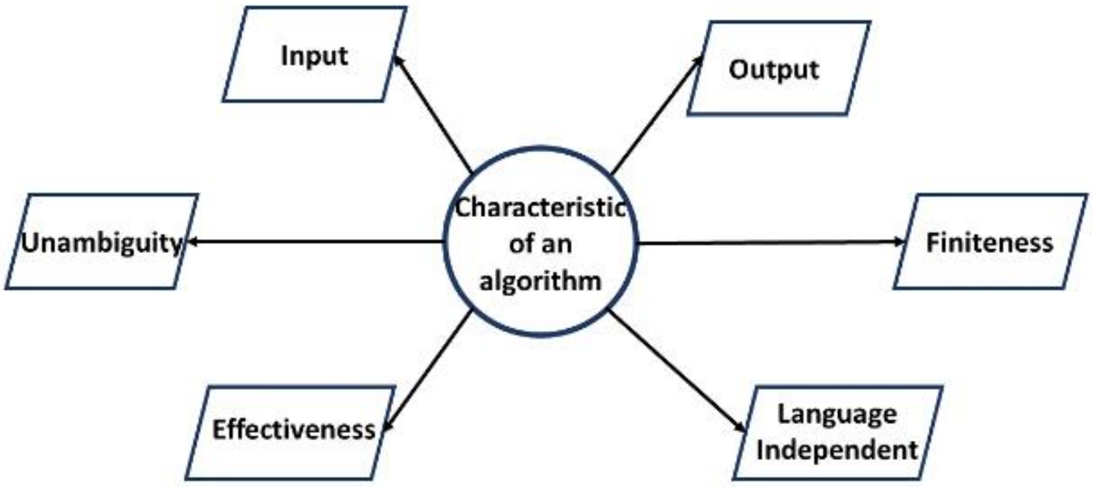
Comments

- Comments are used to make a program more readable.
- Compiler ignores comments and does not execute it.
- We can have single line and multi line comments in c language.
- **Single-line comments** start with two forward slashes (//).

- Any text between // and the end of the line is ignored by the compiler (will not be executed).
- **Multi-line comments** start with /* and ends with */.
- Any text between /* and */ will be ignored by the compiler

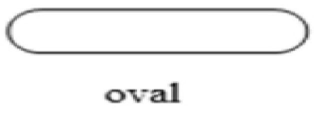
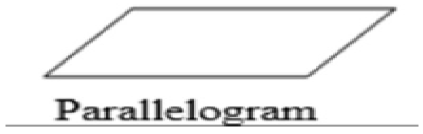
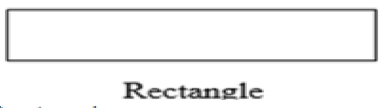
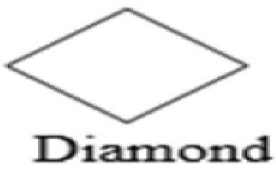
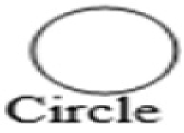
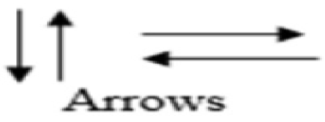
Algorithms

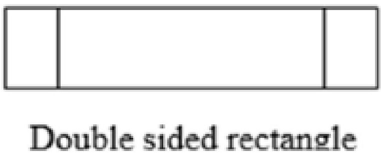
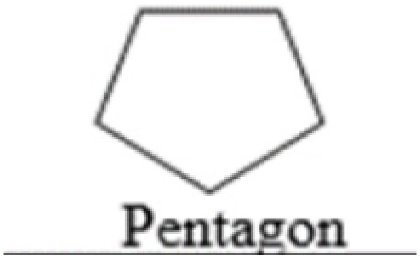
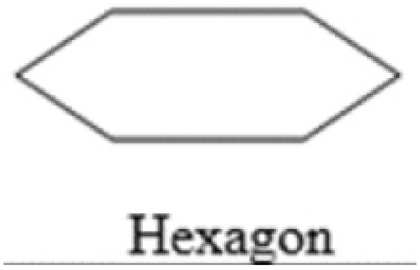
An algorithm is a step-by-step procedure that defines a set of instructions that must be carried out in a specific order to produce the desired result. Algorithms are generally developed independently of underlying languages, which means that an algorithm can be implemented in more than one programming language.



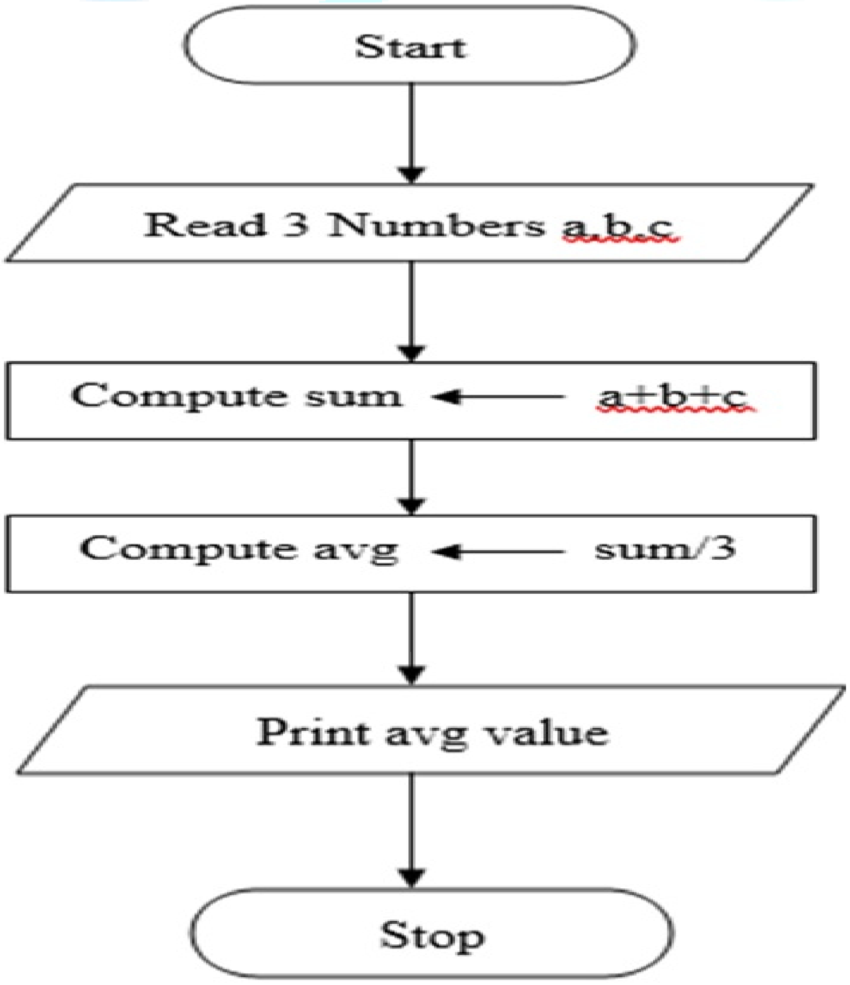
- Input: An algorithm requires some input values. An algorithm can be given a value other than 0 as input.
- Output: At the end of an algorithm, you will have one or more outcomes.
- Unambiguity: A perfect algorithm is defined as unambiguous, which means that its instructions should be clear and straightforward.

- Finiteness: An algorithm must be finite. Finiteness in this context means that the algorithm should have a limited number of instructions, i.e., the instructions should be countable.
- Effectiveness: Because each instruction in an algorithm affects the overall process, it should be adequate.
- Language independence: An algorithm must be language-independent, which means that its instructions can be implemented in any language and produce the same results.
- **Flowcharts**
- Diagrammatic representation of algorithms-

Name	**Symbol**	**Purpose**
Terminal	 Oval	start/stop/begin/end
Input/output	 Parallelogram	Input/output of data
Process	 Rectangle	Any processing to be performaed can be represented
Decision box	 Diamon	Decision operation that determine which of the alternative paths to be followed
Connector	 Circle	Used to connect different parts of flowchart
Flow	 Arrows	Join 2 symbols and also represents flow of execution

Pre defined process	 Double Sided Rectangle	Module (or) subroutines specified else where
Page connector	 Pentagon	Used to connect flowchart in 2 different pages
For loop symbol	 Hexagon	shows initialization, condition and incrementation of loop variable

Example-



Programming errors

Errors are the problems or the faults that occur in the program

There are mainly five types of errors exist in C programming:

Syntax error

Run-time error

Linker error

Logical error

Semantic error

Syntax errors

These errors are mainly occurred due to the mistakes while typing or do not follow the syntax of the specified programming language.

If we want to declare the variable of type integer,

int a; // correct form

Int a; // incorrect form.

Run time errors

When the program is running, and it is not able to perform the operation is the main cause of the run-time error.

For example- division by zero error

Linker errors

Linker errors are mainly generated when the executable file of the program is not created.

Example-

int Main()

Logical errors

These errors produce the incorrect output, but they are error-free, known as logical errors.

Semantic errors

When compiler doesn’t understand the language , these errors occur

The following can be the cases for the semantic error:

Use of a un-initialized variable.

```
i=i+2;
```

Type compatibility

```
int b = “abc";
```

Errors in expressions

```
int a, b, c;
```

```
a+b = c;
```

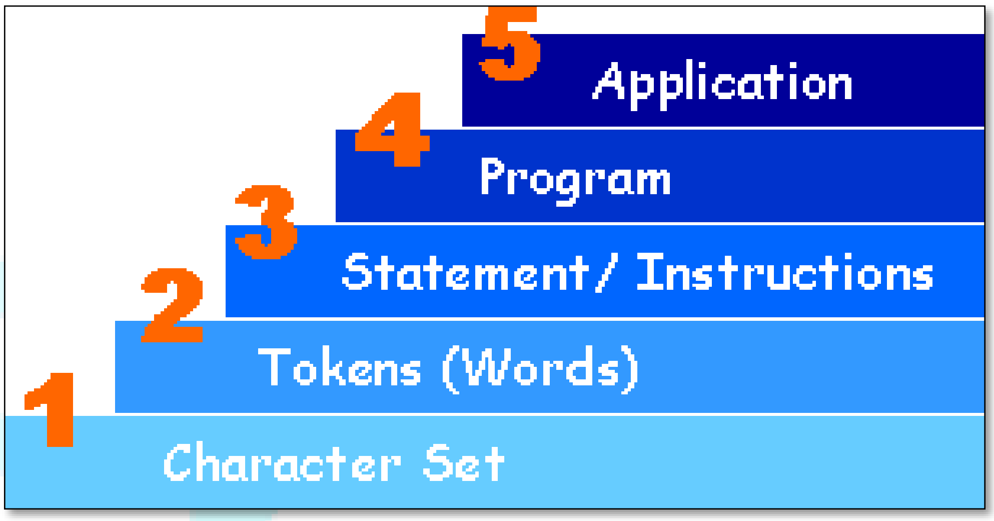
Array index out of bound

```
int a[10];
```

```
a[10] = 34;
```

Unit 3: Constants, variables and data types

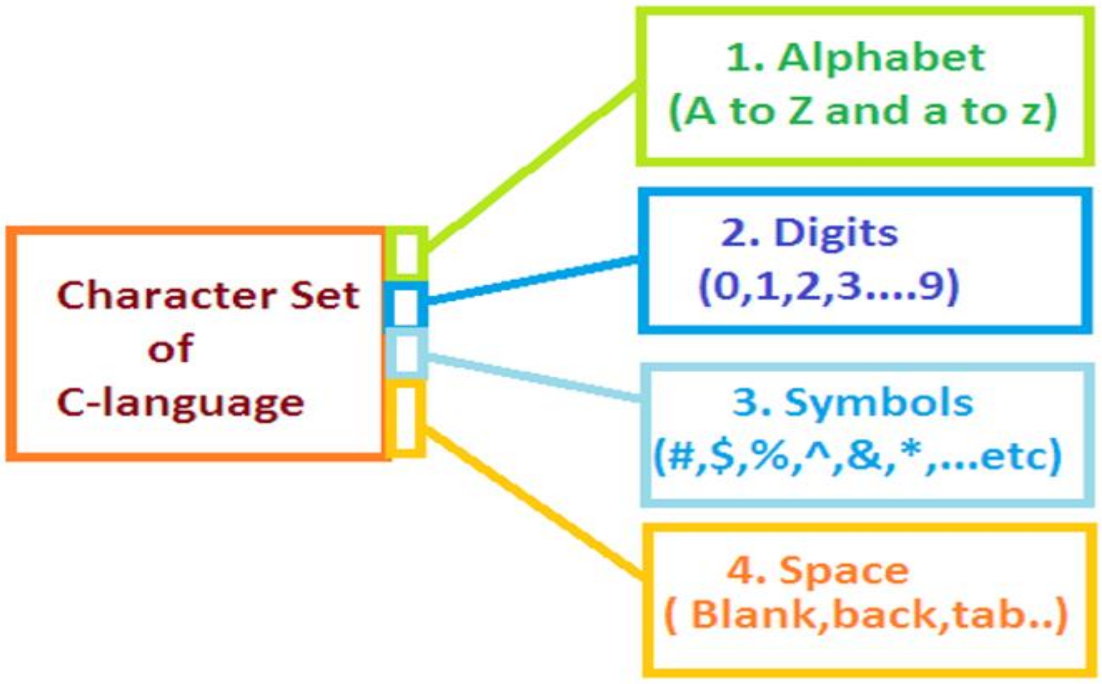
Classical way of learning any language starts with learning letters, numbers and symbols of that language, and then learning words, sentences and finally speaking meaningfully using grammar. Let us follow the same steps in learning C language instead straightaway writing programs.



Character Set

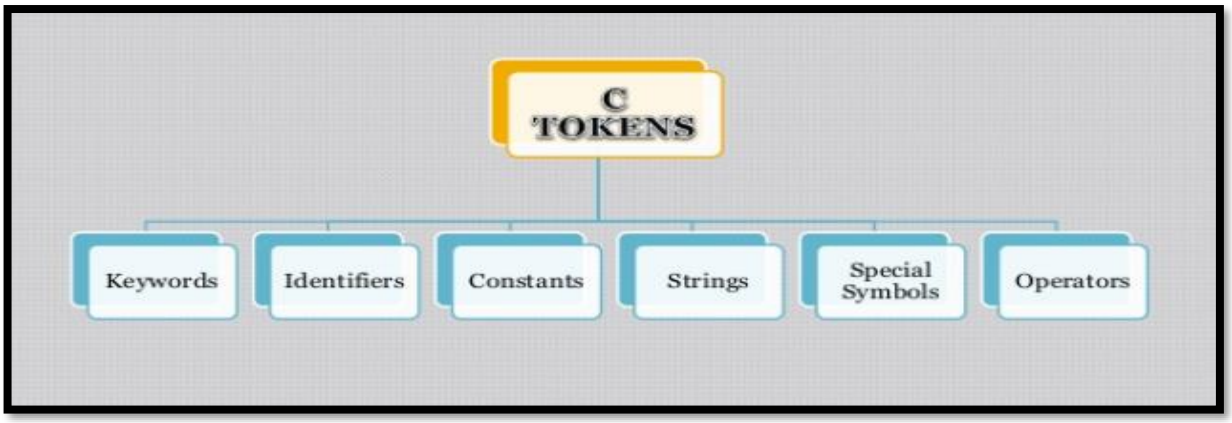
The only characters that C Language uses for its programs are as follows:

- A-Z all alphabets
- a-z all alphabets
- 0-9
- # % & ! _ {} [] () $$$$ &&&& |
- space . , : ; ' $ "
- + - / * =


Character Set of C-language
1. Alphabet (A to Z and a to z)
2. Digits (0,1,2,3....9)
3. Symbols (#,$,%,^,&,*,...etc)
4. Space (Blank,back,tab..)

C language TOKENS

- Tokens in C is the most important element to be used in creating a program in C.
- We can define the token as the smallest individual element in C.
- For example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C.
- Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.



Classification of tokens in C

C language has 6 different types of tokens:

Keywords [ex. float, int, while]

Identifiers [ex. a, b]

Constants [ex. -25.6, 100]

Strings [ex. “hello”, “year”]

Special Symbols [ex. {, }, [,]]

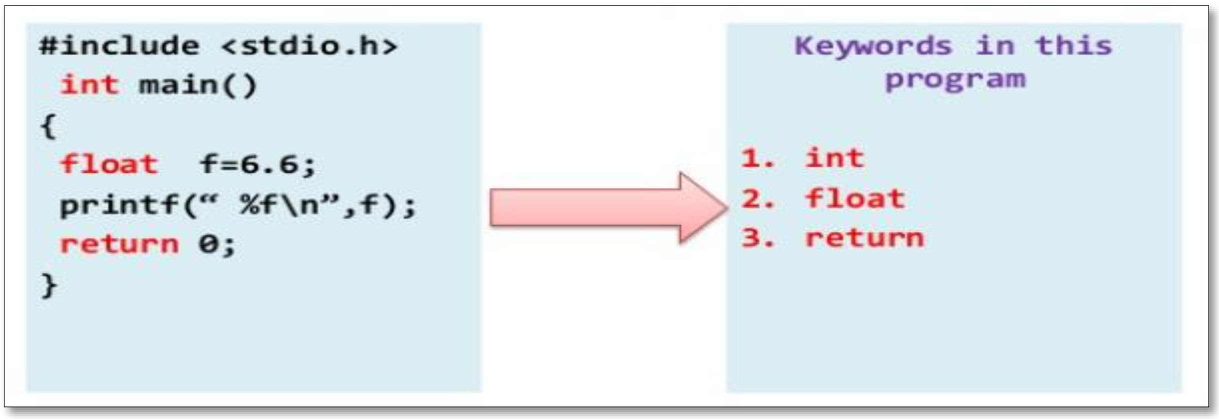
Operators [ex. +, -, *]

C - programs are written using these tokens and the general syntax.

C Keywords

Keywords are the words whose meaning has already been explained to the C compiler.

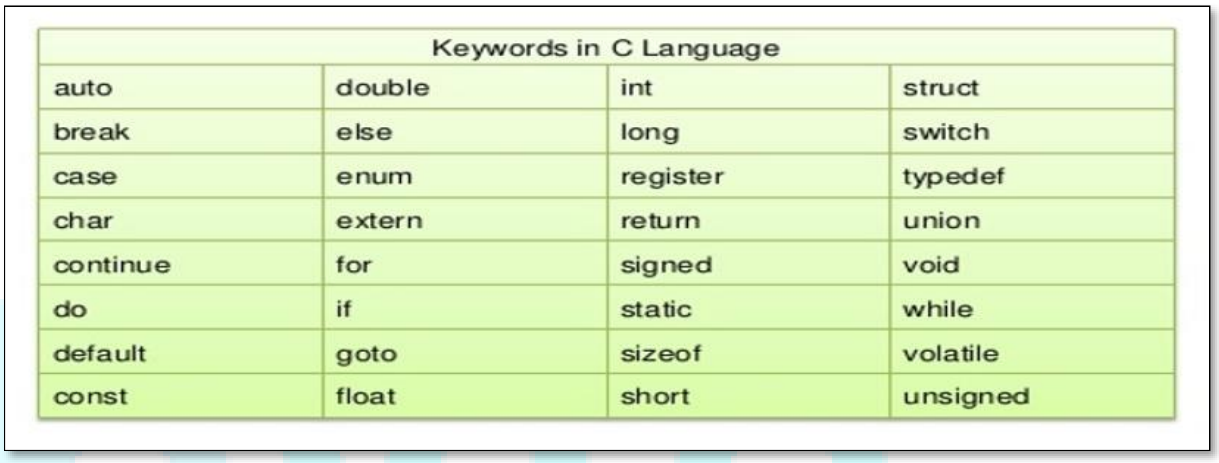
- Keywords are also called as Reserved words.
- Keywords are part of the syntax and they cannot be used as an identifier.
- **C Keywords**



```
#include <stdio.h>
 int main()
{
 float  f=6.6;
 printf(" %f\n",f);
 return 0;
}
```

Keywords in this program

1. int
2. float
3. return



Keywords in C Language			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

C Identifiers

Identifier refers to name given to entities such as variables, functions, structures etc.

- Identifiers must be unique.
- They are created to give a unique name to an entity to identify it during the execution of the program.
- **Rules for naming identifiers:**
- A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
- The first letter of an identifier should be either a letter or an underscore.
- You cannot use keywords like int, while etc. as identifiers.

- There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.
- **C constants**
- When you don't want others (or yourself) to override existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only):

Example

```
const int myNum = 15;
```

#define directive

Macro definitions are not variables and cannot be changed by your program code like variables.

```
#define CNAME value
```

OR

```
#define CNAME (expression)
```

```
#include <stdio.h>
#define PI 3.14
int main() {
  printf("%f",PI);
}


#include <stdio.h>
#define MIN(a,b) ((a)<(b)?(a):(b))
void main() {
   printf("Minimum between 10 and 20 is: %d\n", MIN(10,20));
}
```

C variable

Variable is used to store value.

To create a variable, specify the type and assign it a value:

Syntax

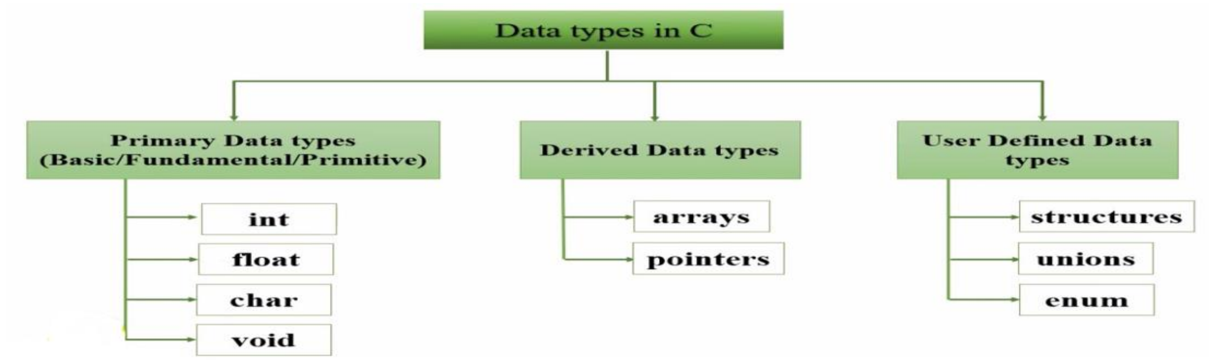
type variableName = value;

Data Types

- In the program, to perform any operation first we must store values. To store values we must allocate memory. Here we need to provide the information about the number of bytes and type of memory that must be allocated to store a value.
- To specify the number of bytes and memory types to the compiler and operating system. we must use some set of keywords.
- These sets of keywords are collectively called data types.

Hence we can say data type is used for allocating memory for storing the value in a program by specifying the required numbers of bytes and memory type.

Classification Of Data Types



Primitive/Basic Data Types

- **Integer Data Type (int)**

Integer data type is used to declare a variable that can store numbers without a decimal. The keyword used to declare a variable of integer type is **int**.

Syntax: int variable_name;

Format Specifier: %d

Example: int a=10;

- **Float data Type (float)**

Float data type declares a variable that can store numbers containing a decimal number.

Syntax: float variable_name;

Format Specifier: %f

Example: float a=10.5;

- **Character Data Type (char)**

Character data type declares a variable that can store a character constant. Thus, the variables declared as char data type can only store one single character.

Syntax: char variable_name;

Format Specifier: %c

Example: char ch='a';

- **Void Data Type (void)**

Unlike other primitive data types in c, void data type does not create any variable but returns an empty set of values. Thus, we can say that it stores null.

Syntax: void variable_name;

Example: void *p;   // pointer ‘p’ is neither pointing to int data type nor char data type.

Data Type Modifiers

In C language Data Type Modifiers are keywords used to change the properties of current properties of data type.

- **long:**

This can be used to increased size of the current data type to 2 more bytes, which can be applied on int or double data types.

Example: 1. long int a;   // 6 bytes

2. long double a; // 10 bytes

- **short:**

It limits user to store small integer values. It can be used only on int data type.

Example: short int a; // 2 bytes

- **signed:**

It is default modifier of int and char data type if no modifier is specified. It says that user can store negative and positive values.

Example: 1. char a;

2. int a;

3. signed char a;

4. signed int a=10;

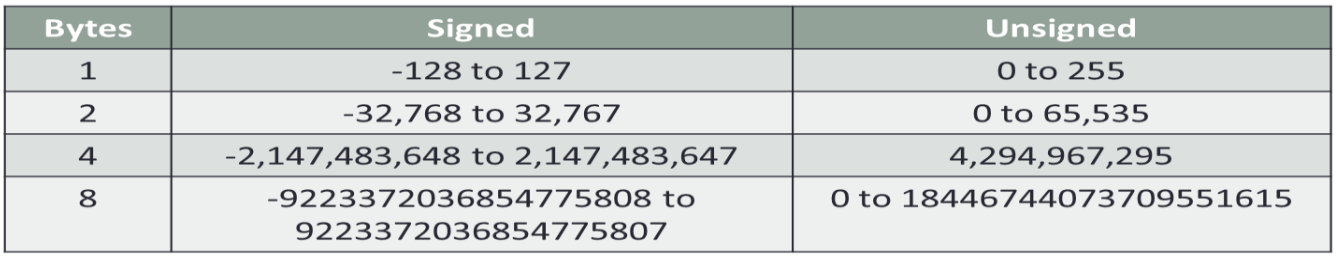
5. signed int a=-10;

- **unsigned:**

When user intends to store only positive values in the given data type (int and char). **Example:** 1. unsigned char a;

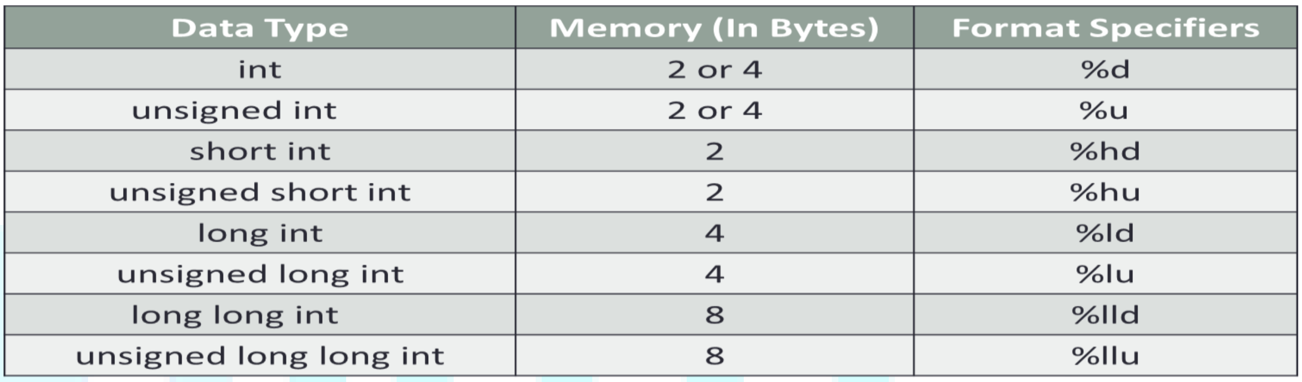
2. unsigned int a=10;

Value range:



Bytes	Signed	Unsigned
1	-128 to 127	0 to 255
2	-32,768 to 32,767	0 to 65,535
4	-2,147,483,648 to 2,147,483,647	4,294,967,295
8	-9223372036854775808 to 9223372036854775807	0 to 18446744073709551615

Classification Integer Data Types:



Data Type	Memory (In Bytes)	Format Specifiers
int	2 or 4	%d
unsigned int	2 or 4	%u
short int	2	%hd
unsigned short int	2	%hu
long int	4	%ld
unsigned long int	4	%lu
long long int	8	%lld
unsigned long long int	8	%llu

Classification Float Data Types:

Data Type	Memory (In Bytes)	Format Specifiers
float	4	%f
double	8	%lf
long double	10	%Lf

Classification Character Data Types:

Data Type	Memory (In Bytes)	Format Specifiers
char	1	%c
Signed char	1	%c
Unsigned char	1	%c

Unit 4: Operators and expressions

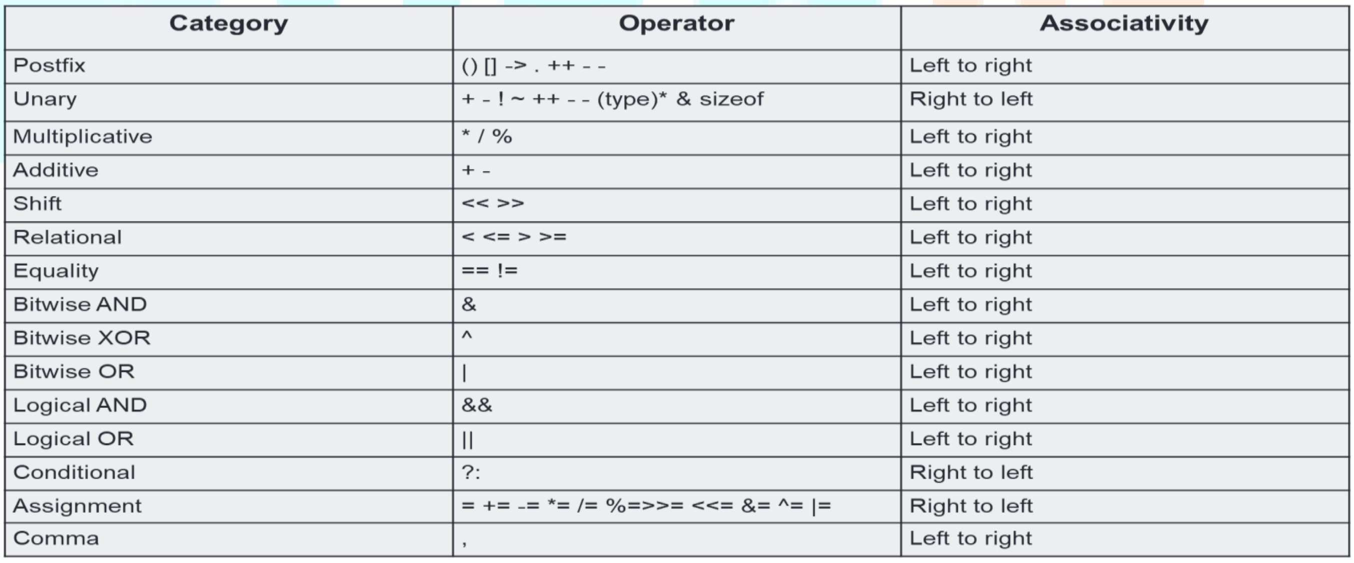
- ❑ **What is Precedence?**

➢ C operator has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated.

- ❑ **What is Associativity?**

➢ The operators at higher level of precedence are evaluated first.

➢ The operators of the same precedence are evaluated either from ‘left to right’ or from ‘right to left’ depending on the level. This is known as the associativity property of an operator.



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

What is an operator?

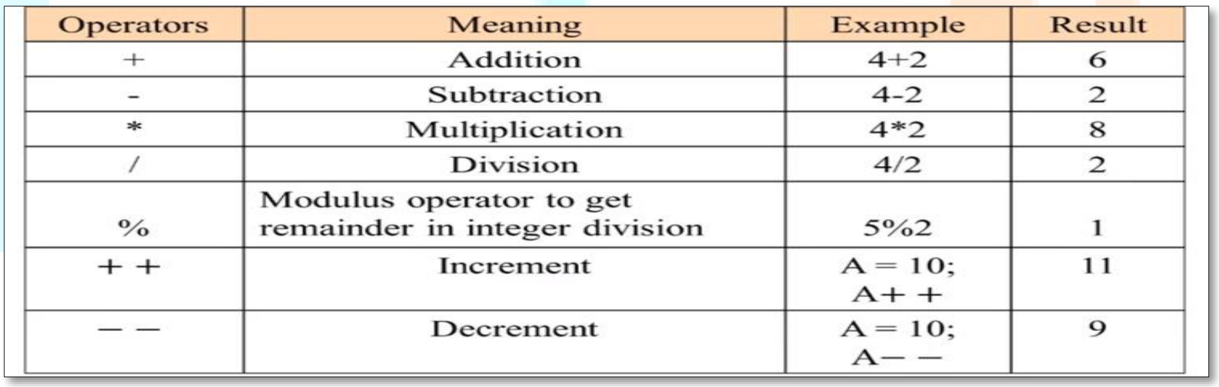
- **An Operator** is a symbol that tells the computer to perform certain mathematical or logic manipulations. Operators are used in programs to manipulate data and variables.
- These operators are used in programs to manipulate data and variables.

- C Operators can be classified as

Arithmetic Operators	+ – * / %
Logical Operators.	&& || !
Relational Operators	== != < <= > >=
Bit wise Operators	& | ^ ~
Assignment Operators.	= += -=
Increment & Decrement Operators	++ --
Conditional Operators	? :

Arithmetic operators-

➢ Arithmetic operators:Arithmetic operators are used to perform numerical calculations among the values.



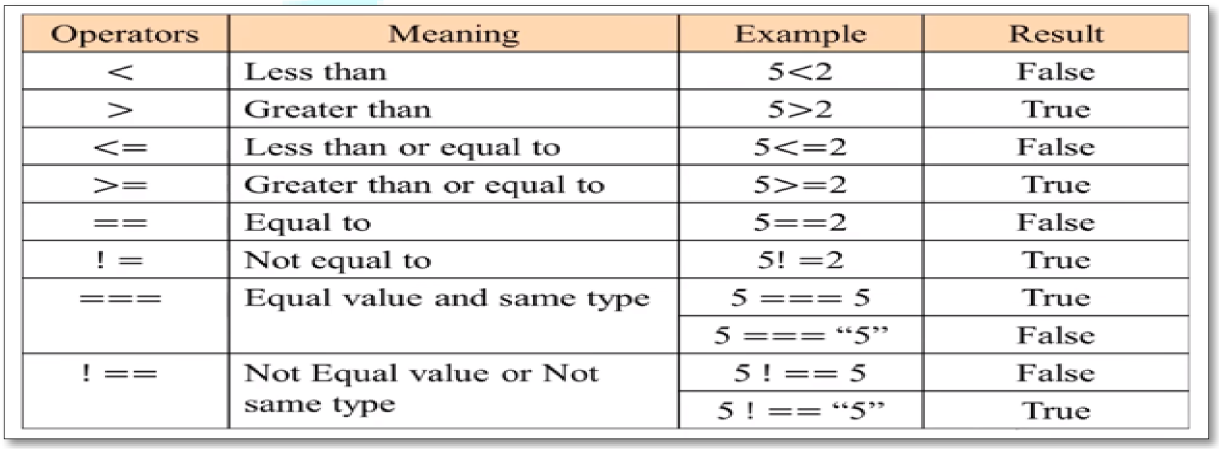
Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
— —	Decrement	A = 10; A— —	9

Example-

```
#include <stdio.h>
int main()
 {
        int a = 9,b = 4, c;
        c = a+b;
        printf("a+b = %d \n",c);
        c = a-b;
        printf("a-b = %d \n",c);
        c = a*b;
        printf("a*b = %d \n",c);
        c = a/b;
        printf("a/b = %d \n",c);
        c = a%b;
        printf("Remainder when a divided by b = %d \n",c);
        return 0;
 }
```

Equality and Relational Operators

- Relational Operators are used to compare two quantities and take certain decision depending on their relation.
- If the specified relation is true it returns one.
- If the specified relation is false it returns zero.



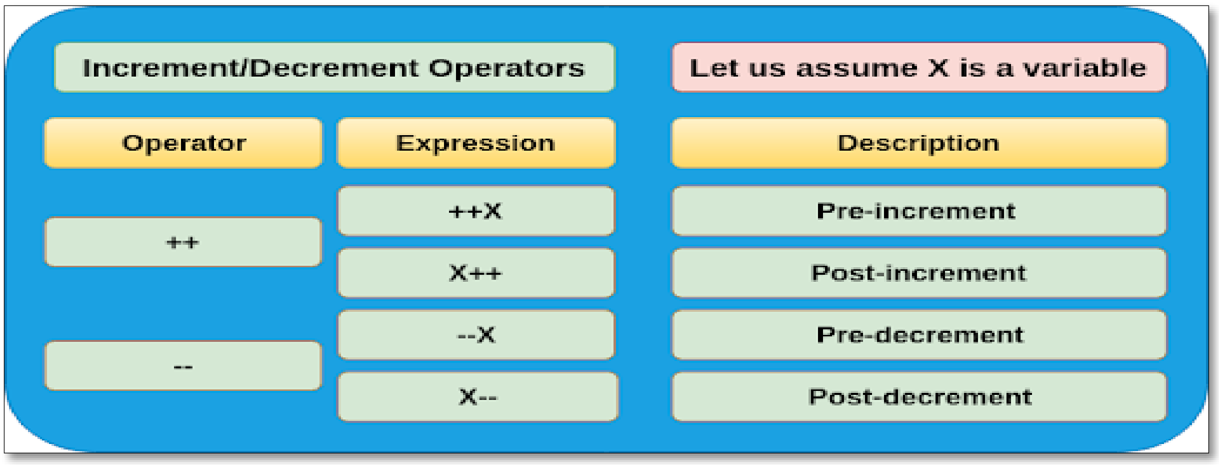
Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === "5"	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == "5"	True

Example-

```
#include <stdio.h>
 int main()
 {
        int a = 5, b = 5, c = 10;
        printf("%d == %d is %d \n", a, b, a == b);
        printf("%d == %d is %d \n", a, c, a == c);
        printf("%d > %d is %d \n", a, b, a > b);
        printf("%d > %d is %d \n", a, c, a > c);
        printf("%d < %d is %d \n", a, b, a < b);
        printf("%d < %d is %d \n", a, c, a < c);
        printf("%d != %d is %d \n", a, b, a != b);
        printf("%d != %d is %d \n", a, c, a != c);
        printf("%d >= %d is %d \n", a, b, a >= b);
        printf("%d >= %d is %d \n", a, c, a >= c);
        printf("%d <= %d is %d \n", a, b, a <= b);
        printf("%d <= %d is %d \n", a, c, a <= c);
        return 0;
}
```

Increment & Decrement Operators

- Two most useful operators which are present in 'c' are increment and decrement operators.
- Operators: ++ and – .
- The operator ++ adds one to the operand .
- The operator -- subtracts one from the operand.
- Both are unary operators and can be used as pre or post increment/decrement.



Increment/Decrement Operators

Let us assume X is a variable

Operator	Expression	Description
++	++X	Pre-increment
	X++	Post-increment
--	--X	Pre-decrement
	X--	Post-decrement

Example-

A=2

B=++A

A=3

B=3

A=2

B=A++

A=3

B=2

A=2

B=--A

A=1

B=1

A=2

B=A--

A=1

B=2

Conditional/ Ternary Operator

- A Ternary operator pair " **? :**"is available in C to construct conditional expression of the form

  condition ? true : false

Eg:

```
a=10;
b=15;
x=(a>b) ? a : b;
```

x will be assigned to the value of b.

Ternary Operator: Example

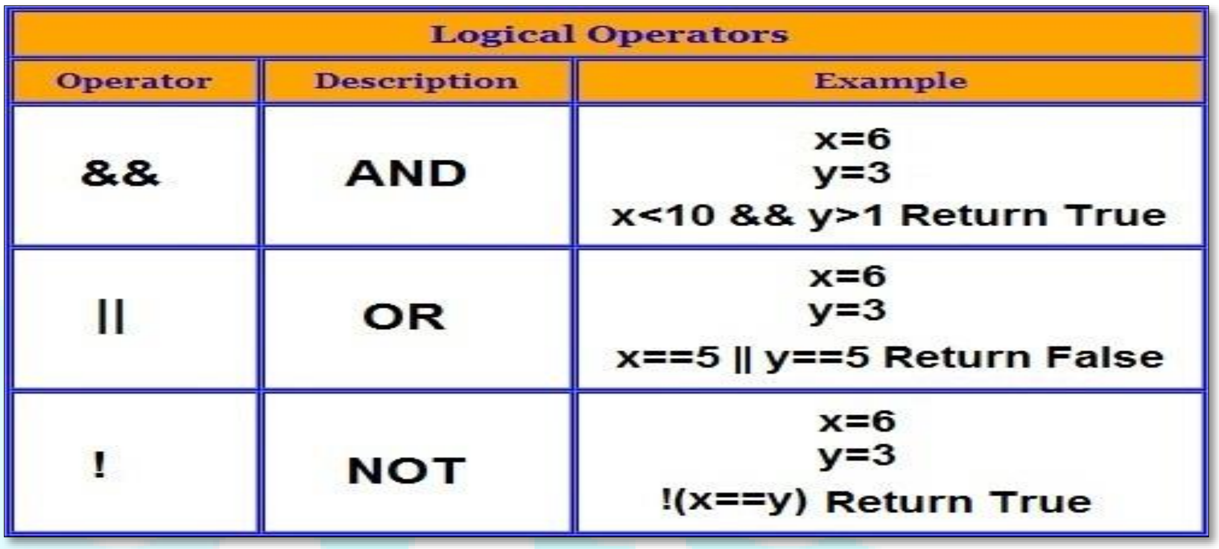
Write a program to display the greatest of two numbers.

```
#include <stdio.h>
main()
{
    int number1, number2, greater;
    printf("\nEnter the first number ");
    scanf("%d",&number1);
    printf("\nEnter the second number");
    scanf("%d",&number2);
    greater=number1 > number2?number1 : number2;
    printf("\nThe greater number is %d",greater);
```

}

Logical Operators

- These operators are used for testing more than one condition and making decisions.
- 'c' has three logical operators they are:



Logical Operators		
Operator	Description	Example

&&	AND	x=6
		y=3
		x<10 && y>1 Return True

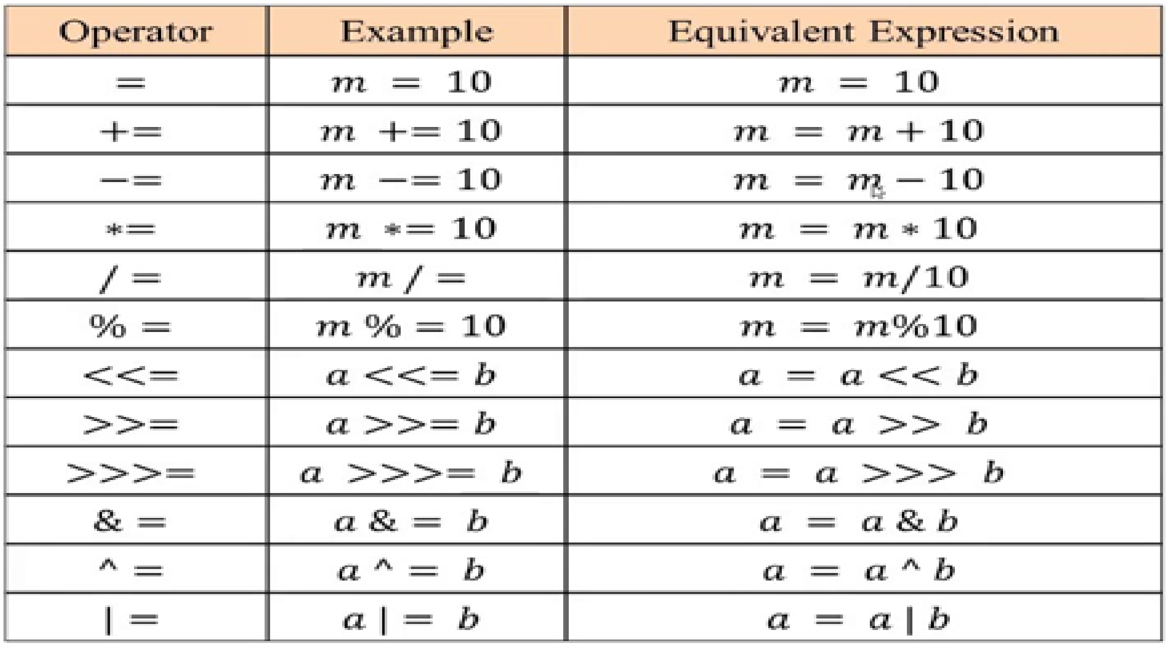
||	OR	x=6
		y=3
		x==5 || y==5 Return False

!	NOT	x=6
		y=3
		!(x==y) Return True

Assignment Operators

- These operators are used for assigning the result of an expression to a variable.

b=a;



Operator	Example	Equivalent Expression
$=$	$m = 10$	$m = 10$
$+=$	$m += 10$	$m = m + 10$
$-=$	$m -= 10$	$m = m - 10$
$*=$	$m *= 10$	$m = m * 10$
$/=$	$m /=$	$m = m/10$
$\% =$	$m \% = 10$	$m = m\%10$
$<<=$	$a <<= b$	$a = a << b$
$>>=$	$a >>= b$	$a = a >> b$
$>>>=$	$a >>>= b$	$a = a >>> b$
$\& =$	$a \& = b$	$a = a \& b$
$\wedge =$	$a \wedge = b$	$a = a \wedge b$
$\vert =$	$a \vert = b$	$a = a \vert b$

Bit-wise Operators

These operators works on bit level .

Applied to Integers only .

All these operators work only on integer type operands.

- Bit-Wise logical operators:
- Bit-Wise AND (&)
- Bit-Wise OR (|)
- Bit-Wise exclusive OR (^)

These are binary operators and require two integer-type operands.

These operators work on their operands bit by bit starting from the

least significant bit

Size of operator

- Size of is an operator used to return the number of bytes the operand occupies.
- **Syntax:**

m=sizeof(sum);

k=sizeof(2351);

Type casting

- Changing variable datatype is called type casting
- We have two types of type casting in c language

1. Implicit type casting

When the type casting is done by compiler. Example-

int x=97;

printf(“%c”,x);

2. Explicit type casting

When we do the type casting

Int x=4,y=3;

Float c;

c=(float)x/(float)y;

Unit 5- Management Input and Output

printf() statement

printf( ) is predefined, standard C function for printing output.

- **Syntax:** printf("<format string>",<list of variables>);

<format string> could be,

%f for printing real values.

%d for printing integer values.

%c for printing character values.

- **Example:**

```
      printf("%f",si);
   printf("%d",i);
```

scanf() statement

scanf( ) is a predefined, standard C function used to input data from keyboard.

- **Syntax:** scanf("<format string>",<address of variables>);
- **Example:**

```
      printf("%f",&si);
   printf("%d",&i);
```

- **&** is pointer operator which specifies the address of that variable.

Escape Sequence

Escape Sequence	Meaning

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

Differences between Difference between getc(), getchar(), getch() and getche() functions

All of these functions are used to get character from input and each function returns an integer signifying the status code as well.

Following are the important differences between getc(), getchar(), getch() and getche() functions.

getc()

getc() can read characters from any stream. Returns EOF on failure.

Syntax

int getc(FILE *stream);

getchar()

getchar() can read characters from standard input only.

Syntax

int getchar();

getch()

getch() can read characters from standard input but it does not use any buffer and returns immidately without waiting for enter key pressed.

Syntax

int getch();

getche()

getche() behaves similar to getch() as it can read characters from standard input and it does not use any buffer and returns immidately without waiting for enter key pressed. Only differnce is that it prints the character as well.

Syntax

int getch();

Example

```
#include <stdio.h>
#include <conio.h>
int main() {
  printf("%c", getc(stdin));
  printf("%c", getchar());
  printf("%c", getch());
  printf("%c", getche());
  return 0;
}
```

Output

```
A
B
C
D
EE
```

Unit 6- Control statements

Control statements enable us to specify the flow of program control — that is, the order in which the instructions in a program must be executed.

There are four types of control statements in C:

Decision making statements (if, if-else)

Selection statements (switch-case)

Iteration statements (for, while, do-while)

Jump statements (break, continue, goto)

1. Decision making statement-

1. if statement

It takes an expression in parenthesis and an statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

Syntax-

```
if(condition)
{
Statements
}
```

Example-

```
#include<stdio.h>
int main()
{
int a=8;
if(a>10)
printf("a is greater than 10");
}
```

2. if else

The if-else statement is used to carry out a logical test and then take one of two possible actions depending on the outcome of the test (ie, whether the outcome is true or false).

Syntax-

```
if (condition)
{
  // statements
}
  else
{
  // statements
}
```

if the condition specified in the if statement evaluates to true, the statements inside the if-block are executed and then the control gets transferred to the statement immediately after the if-block. Even if the condition is false and no else-block is present, control gets transferred to the statement immediately after the if-block.

3. else if ladder-

```
if(condition1)
{
// statement(s);
}
else if(condition2)
{
//statement(s);
}
.
.
else if (conditionN)
{
//statement(s);
}
else
{
//statement(s);
}
```

Example- finding greatest of three numbers

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

```
{
int a=6,b=7,c=9;
if(a>b && a>c)
printf("a is greater");
else if(b>c && b>a)
printf("b is greater");
else
printf("c is greater");
```

switch case-

Switch statement is one of the decision control statements of C language, which is primarily used in a scenario where the user has to make a decision between multiple alternatives.

Syntax-

```
switch (Expression or var) {
   // Expression evaluates to a single value.

 case <Value1> : //Case is picked when expression gives Value1.
<Statement1>;
<Statement2>;
              break;
 case <Value2> :
<Statement1>;
<Statement2>;
```

```
                break;
 .
 . //So on.
 default : // When value of expression didn't match with any
case
<Statement1>;
<Statement2>;
                break;
}
```

Example-

```
int a = 9;
switch (a) {
 case 1:
         printf("One");
        break;
 case 2:
         printf("Two");
        break;
 case 3:
         printf("Three");
        break;
 case 4:
         printf("Four");
        break;
 case 5:
          printf("Five");
         break;
 case 6:
          printf("Six");
         break;
 case 7:
          printf("Seven");
         break;
 case 8:
          printf("Eight");
          break;
 case 9:
          printf("Nine");
          break;
default:
          printf("I am default\n");
          break;
}
```

Go to statements-

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

Syntax-

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

Example-

```
#include <stdio.h>

int main() {
        int num = 26;
        if (num % 2 == 0)
                // jump to even
                goto even;
        else
                // jump to odd
                goto odd;
        even:
        printf("%d is even", num);
        odd:
        printf("%d is odd", num);
        return 0;
}
```

Loops-

for loop-

Looping Statements in C execute the sequence of statements many times until the stated condition becomes false.

Types of loops-

1. entry controlled- First condition is checked and then statements are executed. Example- while and for

2. exit controlled- First statements are executed and then condition is checked. Example-do while

1. while loop-

In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After execution of statements condition is checked repeatedly and once it gets false the loop terminates.

Syntax-

```
while(condition)

{

//statements

}
```

Example-

```
#include<stdio.h>

int main()

{

        int num=1;//initializing the variable

        while(num<=10)//while loop with condition

        {

                printf("%d\n",num);
                num++;              //incrementing operation
        }
        return 0;
}
```

2. do while loop-

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. In do while loop first statements are executed and then condition is checked.

Syntax-

```
do {
  statements
} while (expression);
```

Example-

```
#include<stdio.h>
int main()
{
        int num=11;       //initializing the variable
        do
        {
                printf("%d\n",num);
        }while(num<=10);
        return 0;
}
```

In the above example even though the condition is false still 11 will be printed on the output screen as the loop got executed one time.

3. for loop-

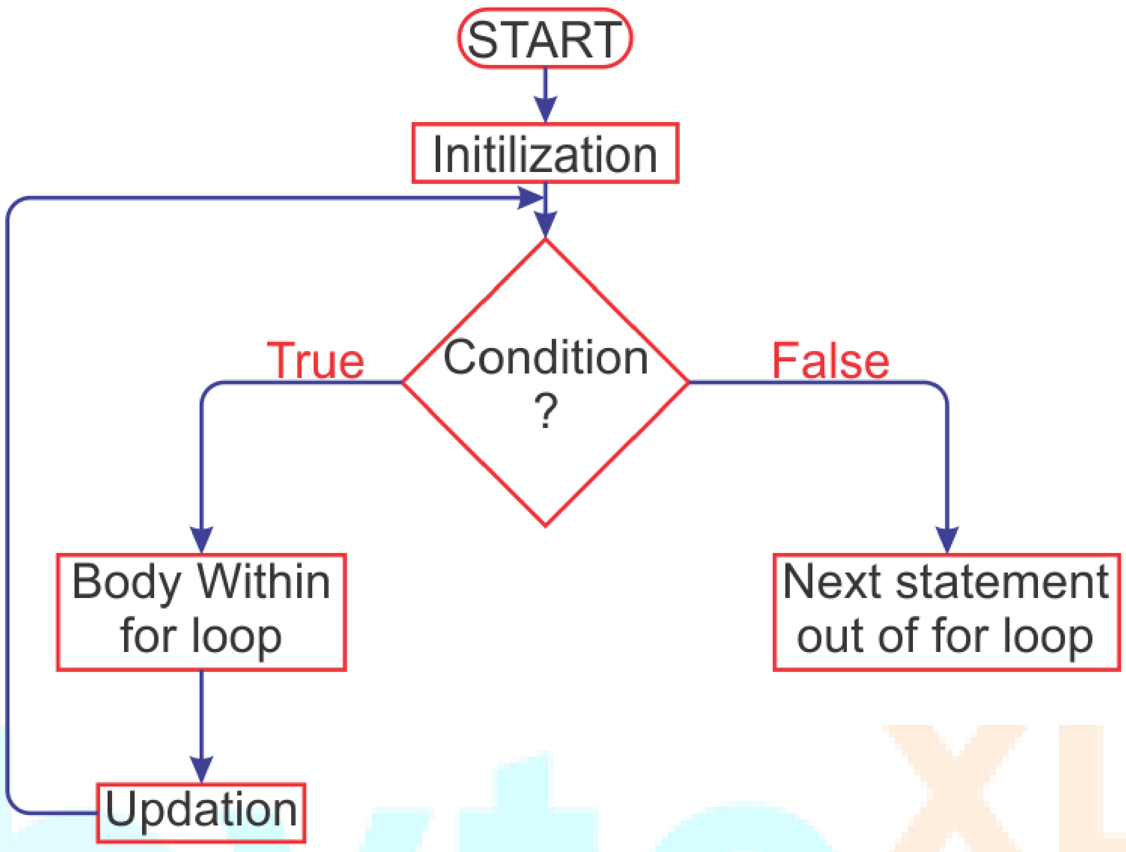
The initial value of the for loop is performed only once.

The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.

The incrementation/decrementation increases (or decreases) the counter by a set value.

Syntax-

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```



Flow Chart of for loop

Example-

```
#include<stdio.h>
int main()
{
        int number;
        for(number=1;number<=10;number++)       // print 1-10
        {
                printf("%d\n",number);              //to print the number
        }
        return 0;
}
```

Control statements-

1. break statement-

Break statement is used to terminate the loop.

Example-

```
#include<stdio.h>
int main()
{
        int number;
        for(number=1;number<=10;number++)        // print 1-10
        {
                break;
                printf("%d\n",number);               //to print the number
        }
        return 0;
}
```

Op- no output

2. continue-

Continue statement forces the compiler to move to next iteration.

```
#include<stdio.h>
int main()
{
        int number;
        for(number=1;number<=10;number++)        // print 1-10
        {
```

```
        If(number%2==0)
        Continue;
        printf("%d\n",number);          //to print the number
    }
    return 0;
}
```

```
Op- 1
3
5
7
9
```