ALGORITHM

Algorithm and flowcharts are alternative representations of a program and are considered basic to any programming activity. Without algorithm development, programming activity is considered to be uncompleted.

The dictionary meaning of the word algorithm is "a process or a set of rules used for calculation".

There are various definitions of an algorithm, which are following

1) It can be defined as a prescribed sequence of well-defined and precise instructions for the solution of a given problem.
2) A method that can be used by a computer for the solution of a problem or a sequence of computational steps that transform the input into the output.
3) A step-by-step procedure to solve the given problem is known as Algorithm.
4) It is a sequence of unambiguous instructions for solving a problem.

This algorithm is implemented on a computer system.

Why should we study algorithms?

A good algorithm implemented on a slow computer may perform much better than a bad algorithm implemented on a fast computer.

Every algorithm is supposed to have the following five important features:

1) Definiteness

2) Effectiveness

3) Finiteness

4) Input

5) Output

Definiteness: By definiteness it is implied that each step of the algorithm must specify a definite action i.e., the steps should not be vague. Each step or instruction in an algorithm must be precisely defined. The actions to be carried out must be rigorously and unambiguously specified in each case. For example “Add 2 or 3 to x" is not a valid algorithmic step because it is ambiguous, whereas "Add 2 to x" is a valid. As a result of this statement, the value of x is increased by 2.

Effectiveness: An algorithm is generally expected to be effective. This requires that all the operations to be performed in an algorithm must be sufficiently basic, so that a person using a pen and pencil must be able to carry them out, in principle, in a finite amount of time. This saves lots of time.

Finiteness: It implies that the algorithm must have finite number of steps. Also the time taken to execute all the steps of the algorithm should be finite and within a reasonable limit. It means that every algorithm must always terminate after the execution of a finite number of steps. Otherwise the procedure is said to enter an infinite loop that never stops until it is forcibly aborted. Such procedures are referred to as computational procedures, for example, operating systems.

Input: Every algorithm takes zero or more inputs. Inputs are the numeric or non-numeric quantities that are given to an algorithm. The algorithm steps use these initially supplied inputs when the algorithm actually executes. In certain algorithm the input is part of algorithm. For example, in numerical algorithm.

Output: An algorithm produce one or more outputs. If there are no output, the algorithm is considered not to have solved any computational problem. These outputs must have specified relations with the corresponding inputs. The output conforms to the inputs. For example, when 2 and 3 are the inputs for a multiplication algorithm, the output should be 6.

Three main reason for using algorithms are:

- Efficiency
- Abstraction
- Reusability

Advantages

1) It is simple to understand step by step solution of the problem.
2) It is easy to debug i.e., errors can be easily pointed out.
3) It is independent of programming languages.
4) It is compatible in the sense that each step of algorithm can be easily coded into its equivalent in high level languages.

Method for Developing an Algorithm

1. Define the problem: State the problem you are trying to solve in clear and concise terms.

2. List the inputs (information needed to solve the problem) and the outputs (what the algorithm will produce as a result)

3. Describe the steps needed to convert or manipulate the inputs to produce the outputs. Start at a high level first, and keep refining the steps until they are effectively computable operations.

4. Test the algorithm: choose data sets and verify that your algorithm works.

Example: Finding the largest of three unequal positive numbers

Input: Three unequal positive numbers X, Y and Z

Output: Largest number among X, Y and Z.

```
Step 1: Start
Step 2: Read numbers X, Y and Z.
Step 3: If X is greater than Y, go to step 4. Else go to step 5
Step 4: If X is greater than Z, go to step 6. Else go to step 8
 Step 5: If Y is greater than Z, go to step 7. Else go to step 8
 Step 6: Print X is largest. Go to step 9
Step 7: Print Y is largest. Go to step 9
Step 8:Print Z is largest. Go to step 9
 Step 9: Stop
```

Example: Write an algo to find out no. is odd or even

```
Step1: Start
Step 2: Input a,b
Step 3: remainder=a mod 2
Step 4: If rem= 0 then
                Print"even"
        Else
                Print:odd"
        End
Step 5 : Stop
```

Example: Write an algo to subtract, multiple & divide of given 2 nos.

```
Step1: Start
Step 2: Input a,b,c,d,e
Step 3: c=a-b
Step 4: d=a*b
Step 5: e=a/b
Step 6: print c,d,e
Step 7 : Stop
```

Example: Write an algo for computing the avg. no of the default 4 no.

```
Step1: Start
Step 2: Input a,b,c,d
Step 3: average=(a,b,c,d)/4
Step 4: print Average
Step 5 : Stop
```

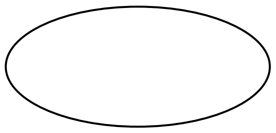
FLOW CHART

There are many techniques for developing and representing the program design. One of the most commonly used methods is the use of flow charts. The flowchart graphically represent the logic needed to solve a programming language. A program flowchart represents the detailed sequence of steps, needed to solve the problem.

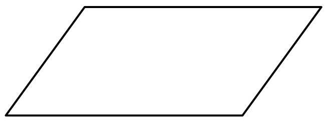
- The graphical or visual representation of algorithm is called as flow chart.
- It is independent of computer languages.
- The flow charts are easier to understand the flow of the solution.
- Help in algorithm design.
- Check the program logic.
- Help in coding.
- Modification becomes easy.
- Flow charts are drawn with the standard symbols accepted worldwide.
- Each standard flow chart symbol represents on action to be performed such as Start or Stop, input operations Read or Write, decision making etc.
- Better documentation provided.

Standard flow chart symbols

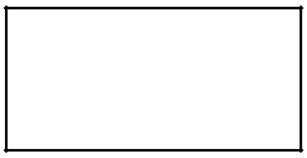
1)Terminal (Start or Stop Symbol)



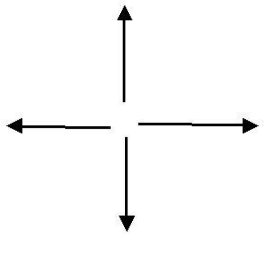
2) Input / Output



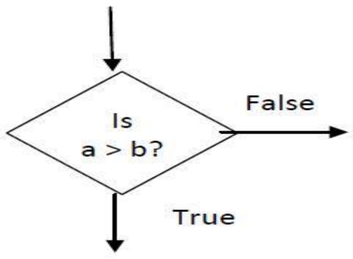
3) Processing



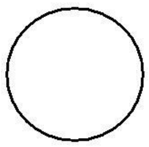
4) Flow Lines: These are arrow mark symbols used to connect two boxes and to indicate the direction of data or processing flow



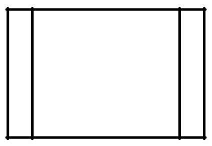
5) Decision Box: This is a diamond shaped box, which is used to indicate logical checking and gives decision after comparing between two or more objects (Eg. Yes or No; True or False, =, >, <, etc.)

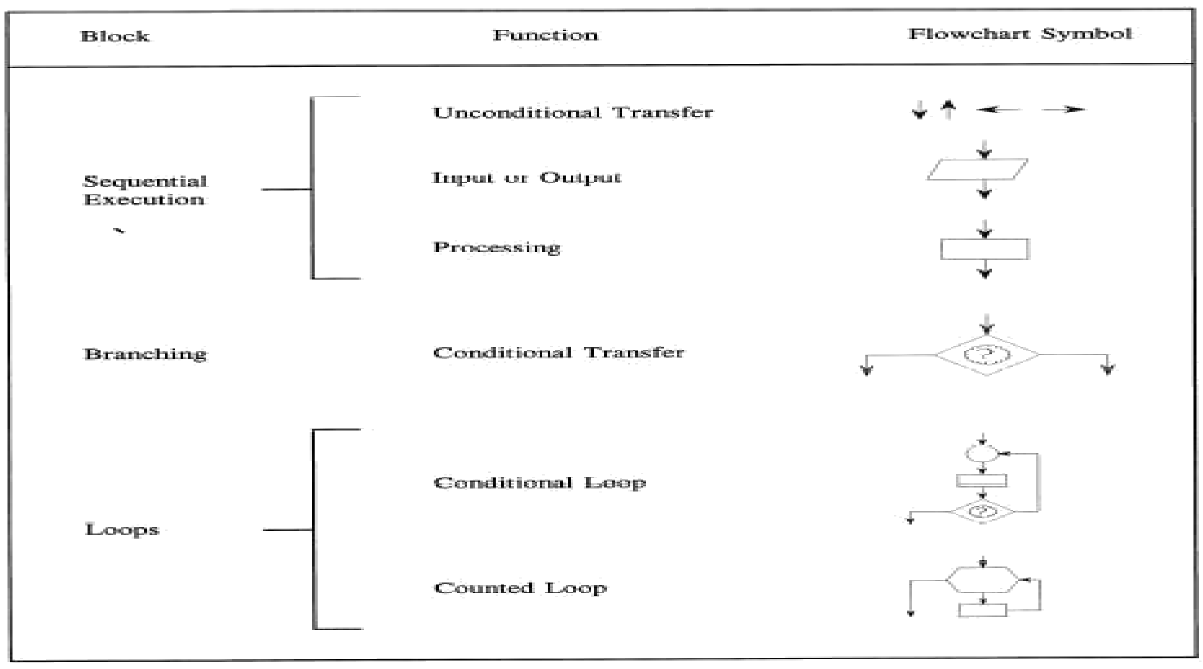


6) Connector: This is a Circular-shaped symbol used to connect different parts of flowchart. When the flow chart is lengthy, it is split into different pages. Then these connectors are used to connect between these pages at the beginning and at the end of each page



7) Predefined: It is a symbol often used to represent a process that is used several times in the same program. This process is defined only once and reference by this block thereafter.





Block	Function	Flowchart Symbol
Sequential Execution	Unconditional Transfer	↓ ↑ ← →
	Input or Output	
	Processing	
Branching	Conditional Transfer	
Loops	Conditional Loop	
	Counted Loop	

General Rules for flowcharting

1. All boxes of the flowchart are connected with Arrows. (Not lines)

2. Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.

3. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.

4. Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.

5. Connectors are used to connect breaks in the flowchart. Examples are:
    - From one page to another page.
    - From the bottom of the page to the top of the same page.
    - An upward flow of more than 3 symbols

6. Subroutines and Interrupt programs have their own and independent flowcharts.

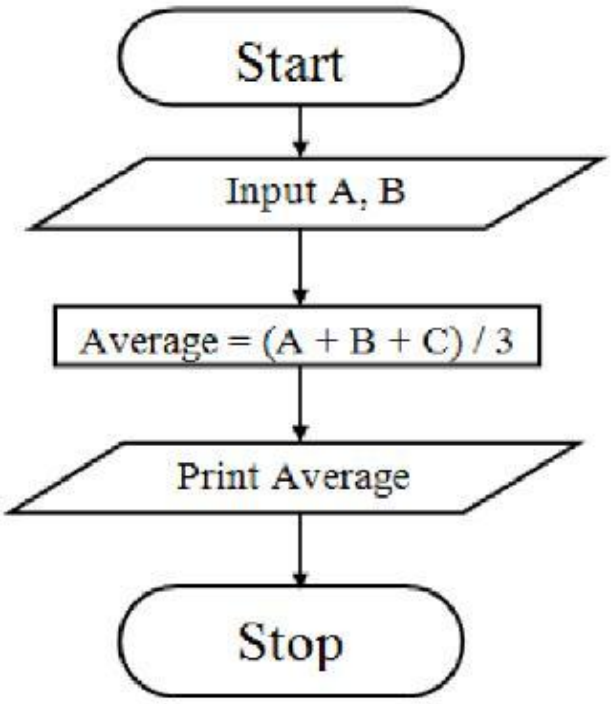
7. All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.

8. All flowcharts end with a terminal or a contentious loop.

Q) Algorithm and flow chart to find the average of three numbers.

Algorithm

Step 1 : Start
Step 2 : Enter Three Numbers A, Band C
Step 3 : Compute Average = (A+B+C)/3
Step 4 : Print Average
Step 5 : Stop



Q) Algorithm and flow chart to find the largest of two numbers.

Algorithm

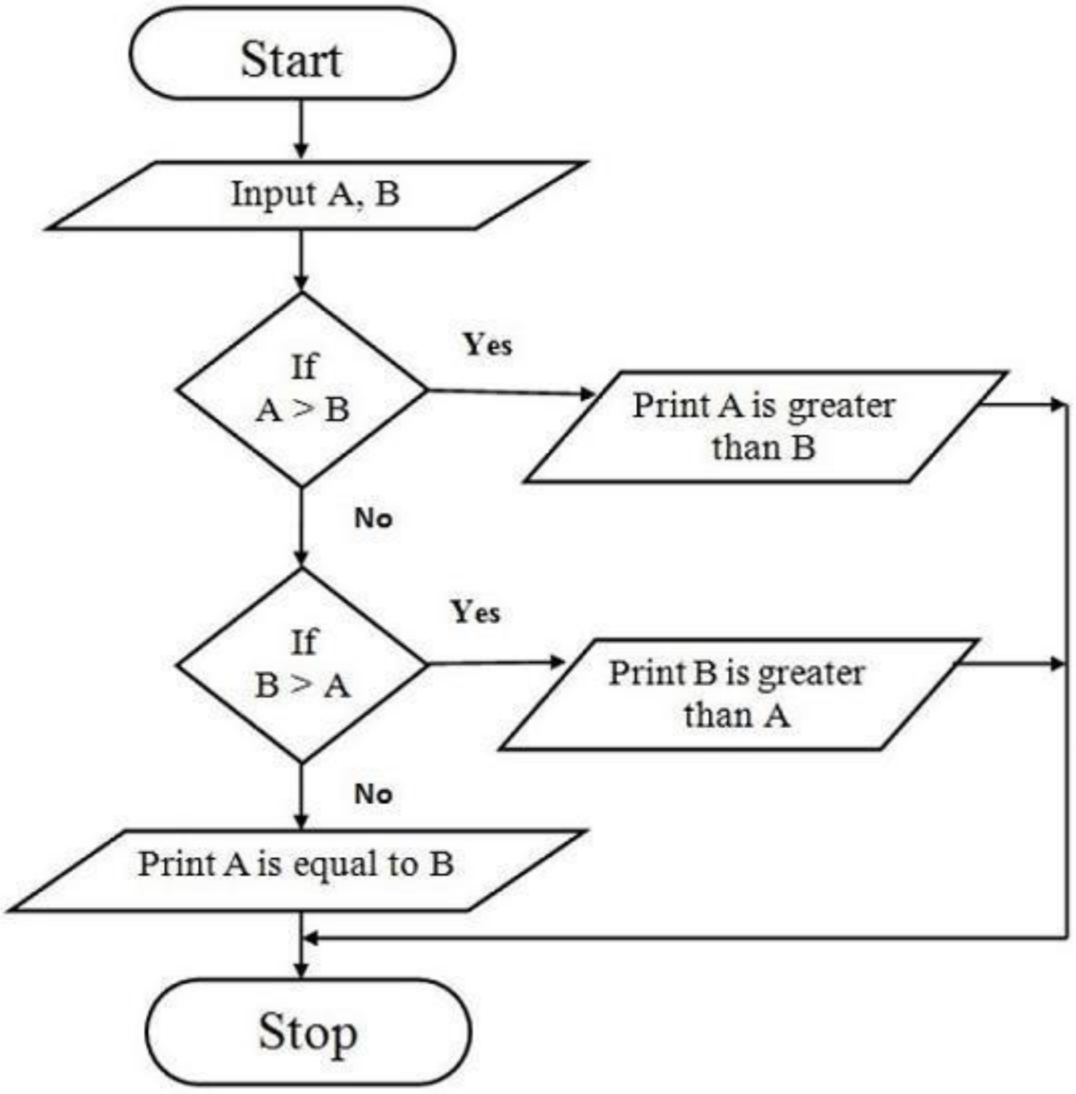
Step1: Start

Step 2: Enter two numbers A and B

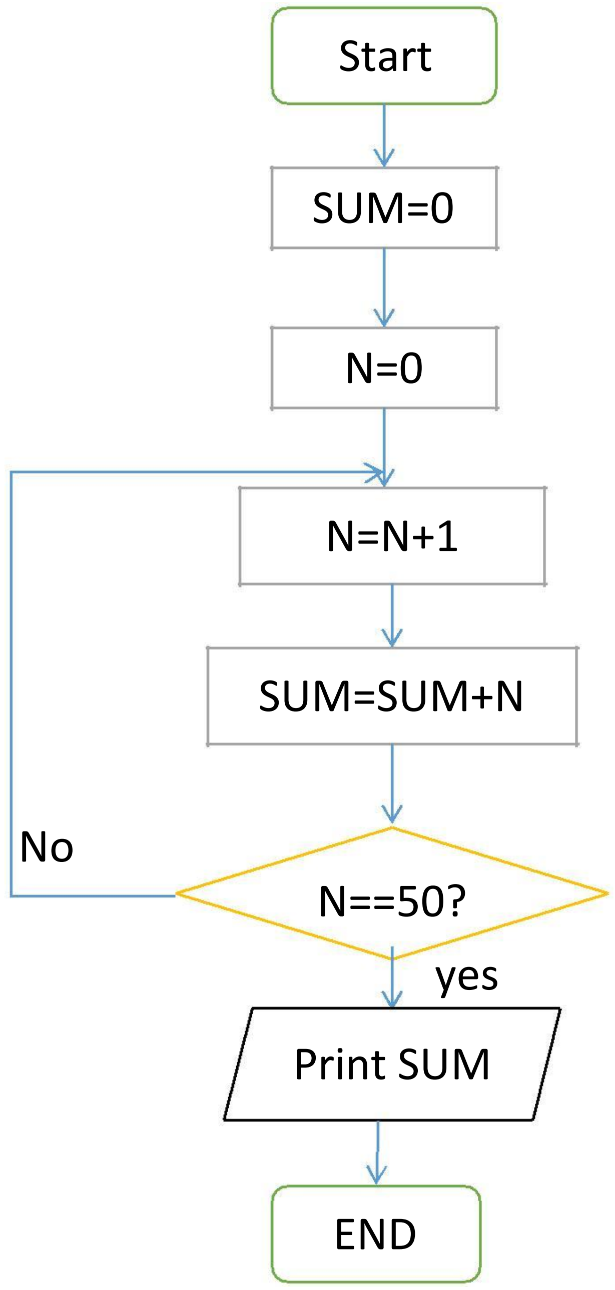
Step 3: Check if A is greater than B if yes go to Step 4 else go to Step
5 Step 4: Print A is greater than B

Step 5: Check if B is greater than A if yes go to Step 6 else go to Step
7 Step 6: Print B is greater than A

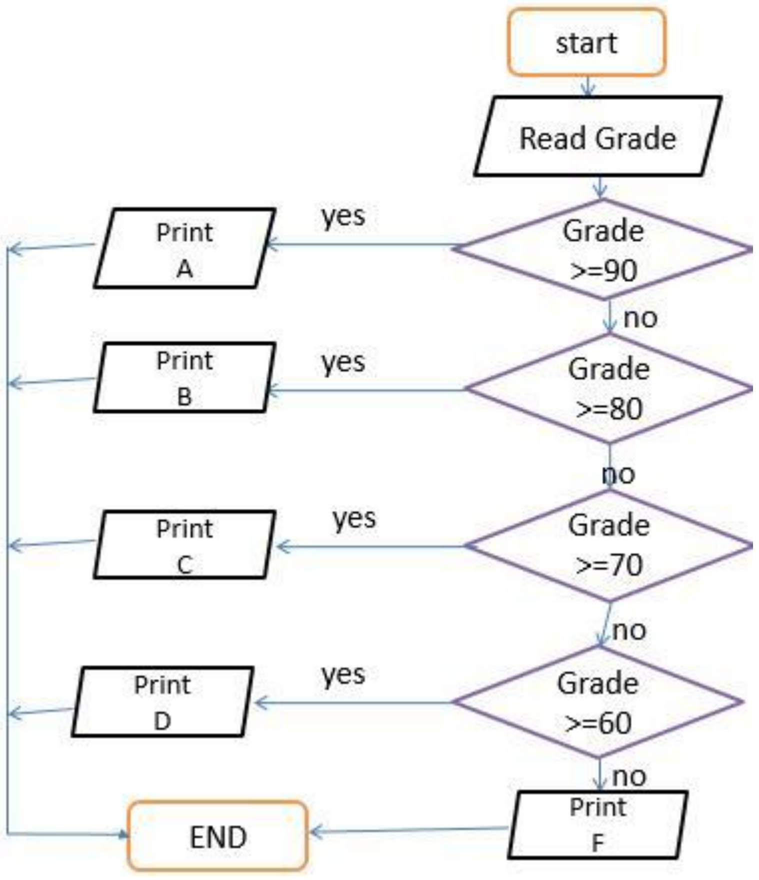
Step 7: Print A is equal to
B Step 8: Stop



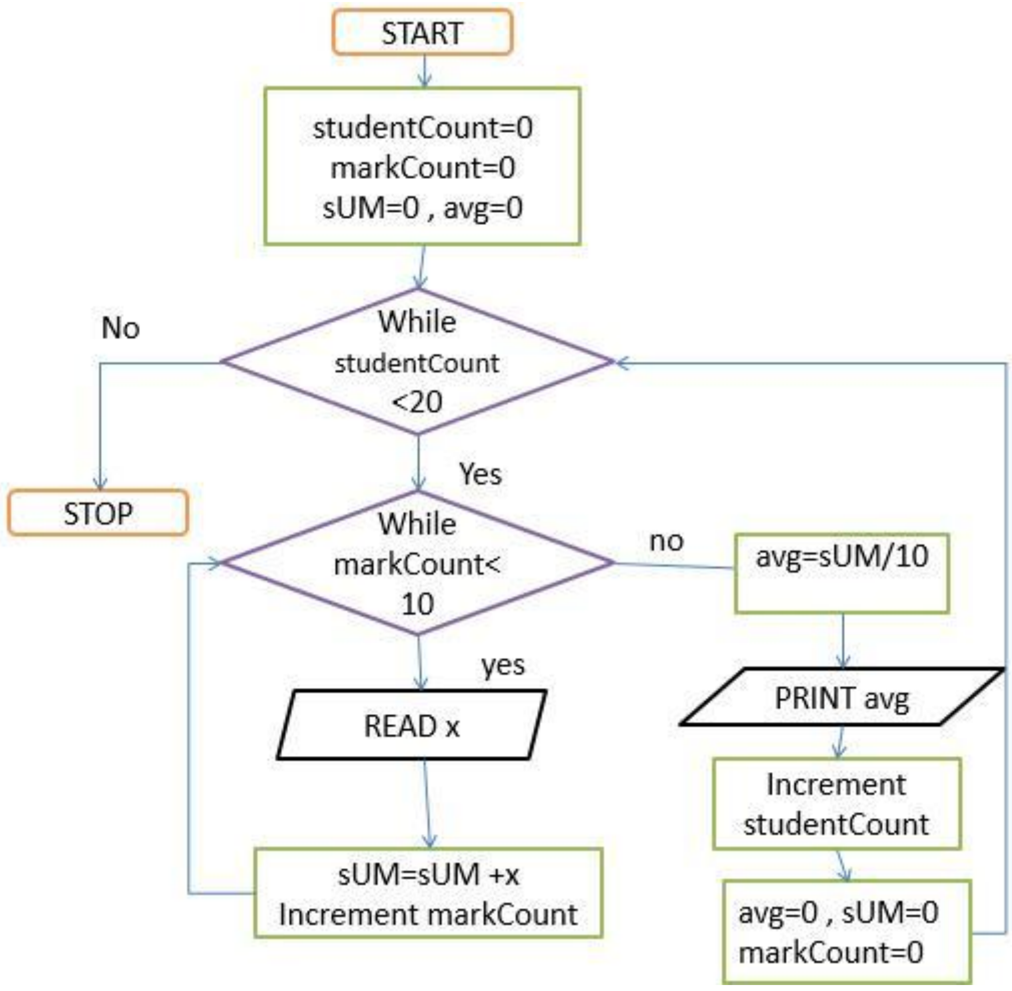
Q) Draw a flowchart to find the sum of first 50 natural numbers



Q) Draw a flowchart for program Calculate the student grade (A,B,C,D,F)



Q) Draw a flowchart for program read (10) marks for (20) students, then print the average for each student



Types of Flowchart

There are three types of flowchart, which are following:-

1) System flowchart
2) Modular flowchart
3) Detail flowchart

Level of Flowchart

There are two level of flowcharts:

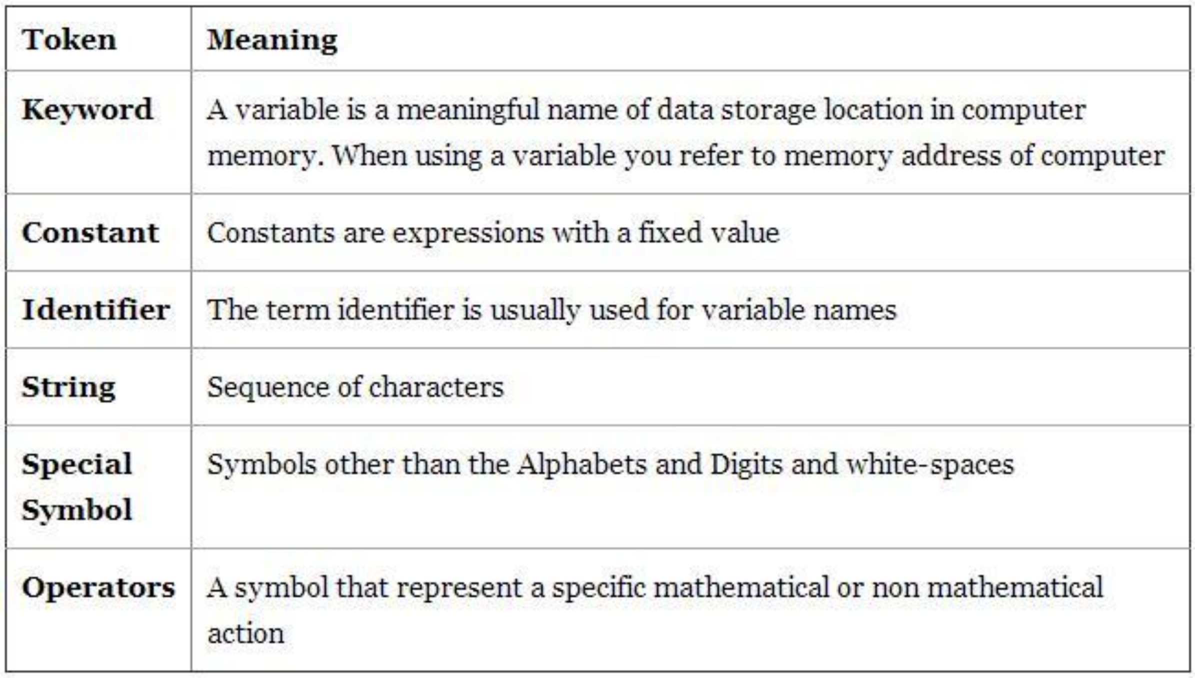
1) Macro flowchart: It shows the main segment of a program and shows lesser details
2) Micro flowchart: It shows more detail of any part of the flowchart

TOKEN

- A token is an atomic unit (smallest indivisible units) in a program.
- The most basic elements in a C program recognized by the compiler are a single character or a group of characters called C tokens.
- The compiler cannot breakdown the token any further. For example, the words main, '{' (brace), '(' (parenthesis) are all tokens of C program.

Types of token

1. Keywords Examples: float, int, double, while, for.
2. Identifiers Examples: main, amount
3. Constants Examples: 12.4, 7894
4. Strings Examples: "CSM", "Thursday"
5. Special Symbols Examples: [,], {, }, (,)
6. Operators Examples: +, *, /



Token	**Meaning**
Keyword	A variable is a meaningful name of data storage location in computer memory. When using a variable you refer to memory address of computer
Constant	Constants are expressions with a fixed value
Identifier	The term identifier is usually used for variable names
String	Sequence of characters
Special Symbol	Symbols other than the Alphabets and Digits and white-spaces
Operators	A symbol that represent a specific mathematical or non mathematical action

Keywords

1. Keywords are predefined tokens in C. These are also called reserved words.
2. Keywords have special meaning to the C compiler. These key words can be used only for their intended action; they cannot be used for any other purpose.
3. C has 32 keywords.

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

Identifier

Identifiers are distinct names given to program elements such as constants, variables, etc. An Identifier is a sequence of letters, digits, and the special character '_' "underscore".

1. It must start with either a letters, underscore. '_'
2. No commas or blanks are allowed within a variable name.
3. The upper case and lower case letters are treated as distinct, i.e., identifiers are case-sensitive.
4. An identifier can be of any length.
5. No special symbol can be used in a variable name.

Constants

Constants are the terms that can't be changed during the execution of a program. For example: 1, 2.5, "Programming is easy." etc. In C, constants can be classified as:

Integer constants

Integer constants are the numeric constants(constant associated with number) without any fractional part or exponential part. There are three types of integer constants in C language: decimal constant(base 10), octal constant(base 8) and hexadecimal constant(base 16) .

Decimal digits: 0 1 2 3 4 5 6 7 8 9

Octal digits: 0 1 2 3 4 5 6 7

Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F.

For example:

Decimal constants: 0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

Notes:

- You can use small caps a, b, c, d, e, f instead of uppercase letters while writing a hexadecimal constant.
- Every octal constant starts with 0 and hexadecimal constant starts with 0x in C programming.

Floating-point constants
Floating point constants are the numeric constants that has either fractional form or exponent form. For example:

-2.0
0.0000234
-0.22E-5

Note:Here, E-5 represents 10-5. Thus, -0.22E-5 = -0.0000022.

Character constants
Character constants are the constant which use single quotation around characters. For example: 'a', 'l', 'm', 'F' etc.

Escape Sequences
Sometimes, it is necessary to use newline(enter), tab, quotation mark etc. in the program which either cannot be typed or has special meaning in C programming. In such cases, escape sequence are used. For example: \n is used for newline. The backslash(\ ) causes "escape" from the normal way the characters are interpreted by the compiler.

Escape Sequences

Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

```
"good"                    //string constant
""                        //null string constant
"      "                  //string constant of six white space
"x"                       //string constant having single character.
"Earth is round\n"        //prints string with newline
```

Enumeration constants
Keyword enum is used to declare enumeration types. For example:

enum color {yellow, green, black, white};
Here, the variable name is color and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively by default. For more information about enumeration, visit page: Enumeration Types.

Operators

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in program to manipulate data and variables. The data items that operators act upon are called operands.
- Some operators require two operands, while others act upon only one operand.
- The operators are classified into unary, binary and ternary depending on whether they operate on one, two or three operands respectively.

Types of operators

C has four classes of operators

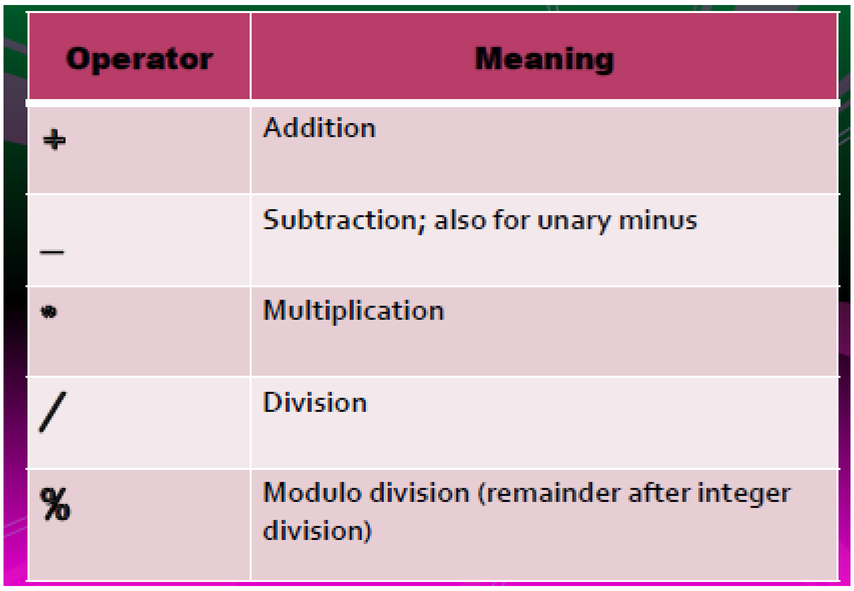
1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bit-wise Operators

In addition, C has some special operators, which are unique to C, they are

1. Increment & Decrement Operators
2. Conditional Operators
3. Assignment Operators, etc.

Arithmetic operators

There are five arithmetic operators in C.
The following table lists the arithmetic operators allowed in C:



Operator	Meaning
+	Addition
–	Subtraction; also for unary minus
*	Multiplication
/	Division
%	Modulo division (remainder after integer division)

```
/* Program to demonstrate the working of arithmetic operators in C.  */
#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;
}
```

```
a+b=13
a-b=5
a*b=36
a/b=2
Remainder when a divided by b=1
```

Relational operators

Relational Operators are symbols that are used to test the relationship between two variables or between a variable and a constant. We often compare two quantities, and depending on their relation takes certain decisions. These comparisons can be done with the help of relational operators.

C has six relational operators as shown below.

Operator	Meaning of Operator	Example
==	Equal to	5==3 returns false (0)
>	Greater than	5>3 returns true (1)
<	Less than	5<3 returns false (0)
!=	Not equal to	5!=3 returns true(1)
>=	Greater than or equal to	5>=3 returns true (1)
<=	Less than or equal to	5<=3 return false (0)

Logical Operators

Logical Operators are symbols that are used to combine or negate expressions containing relational operators. It has three logical operators as defined below.

Operator	Meaning of Operator	Example
&&	Logial AND	If c=5 and d=2 then,((c==5) && (d>5)) returns false.
||	Logical OR	If c=5 and d=2 then, ((c==5) || (d>5)) returns true.
!	Logical NOT	If c=5 then, !(c==5) returns false.

Bitwise operators

- The lowest logical element in the memory is bit.
- C allows the programmer to interact directly with the hardware of a particular system through bitwise operators and expression.
- These operators work only with int and char data types and cannot be used with float and double type.

The following table shows the bitwise operators that are available in C.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

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

  unsigned int a = 60;        /* 60 = 0011 1100 */
  unsigned int b = 13;        /* 13 = 0000 1101 */
  int c = 0;

  c = a & b;      /* 12 = 0000 1100 */
  printf("Line 1 - Value of c is %d\n", c );

  c = a | b;      /* 61 = 0011 1101 */
  printf("Line 2 - Value of c is %d\n", c );

  c = a ^ b;      /* 49 = 0011 0001 */
  printf("Line 3 - Value of c is %d\n", c );

  c = ~a;         /*-61 = 1100 0011 */
  printf("Line 4 - Value of c is %d\n", c );

  c = a << 2;     /* 240 = 1111 0000 */
  printf("Line 5 - Value of c is %d\n", c );

  c = a >> 2;     /* 15 = 0000 1111 */
  printf("Line 6 - Value of c is %d\n", c );
}
```

Output

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

Increment & decrement operators

- C has two very useful operators for adding and subtracting a variable. These are the increment and decrement operators, ++ and -- . These two operators are unary operators.
- The increment operator ++ adds 1 to its operand, and the decrement operator -- subtracts 1 from its operand. Therefore, the following are equivalent operations.
  - ++i is equivalent to i = i + 1;
  - --i is equivalent to i = i – 1;
  - These operators are very useful in loops.

  Let a=5 and b=10
  ```
  a++;  //a becomes 6
   a--;  //a becomes 5
  ++a;  //a becomes 6
  ```

--a; //a becomes 5

Difference between ++ and -- operator as postfix and prefix

When i++ is used as prefix(like: ++var), ++var will increment the value of var and then return it but, if ++ is used as postfix(like: var++), operator will return the value of operand first and then only increment it. This can be demonstrated by an example:

```
#include <stdio.h>
int main()
{
   int c=2,d=2;
   printf("%d\n",c++); //this statement displays 2 then, only c incremented by  1 to
3.
   printf("%d",++c);   //this statement increments 1 to c then, only c is displayed.
   return 0;
}
```

Output
```
2
4
```

Assignment operator

In addition to usual assignment operator =, C has a set of shorthand operators, that simplifies the coding of a certain type of assignment statement. It is of the form var op = exp where var is a variable, op is a C binary arithmetic operator and exp is an expression.

Operator	Example	Same as
=	a=b	a=b
+=	a+=b	a=a+b
-=	a-=b	a=a-b
*=	a*=b	a=a*b
/=	a/=b	a=a/b
%=	a%=b	a=a%b

Conditional operators

C provides a peculiar operator ? : which is useful in reducing the code. It is ternary operator requiring three operands.
The general format is
exp1 ? exp2 : exp3;
where exp1, exp2 and exp3 are expressions.
In the above conditional expression, *exp1* is evaluated first. If the value of *exp1* is non zero (true), then the value returned will be *exp2*. if the value of *exp1* is zero (false), then the value returned will be *exp3*.

Other Operators

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.

Comma Operator

Comma operators are used to link related expressions together. For example:
in
int a,c=5,d;

The sizeof operator

It is a unary operator which is used in finding the size of data type, constant, arrays, structure etc. For example:

```
#include <stdio.h>
int main(){
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%d bytes\n",sizeof(a));
    printf("Size of float=%d bytes\n",sizeof(b));
    printf("Size of double=%d bytes\n",sizeof(c));
    printf("Size of char=%d byte\n",sizeof(d));
    return 0;
}
```

Operator precedence

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

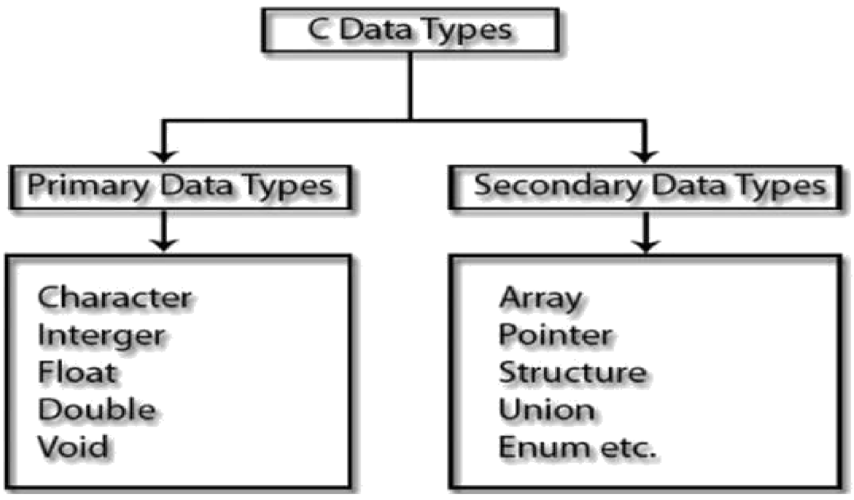
Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

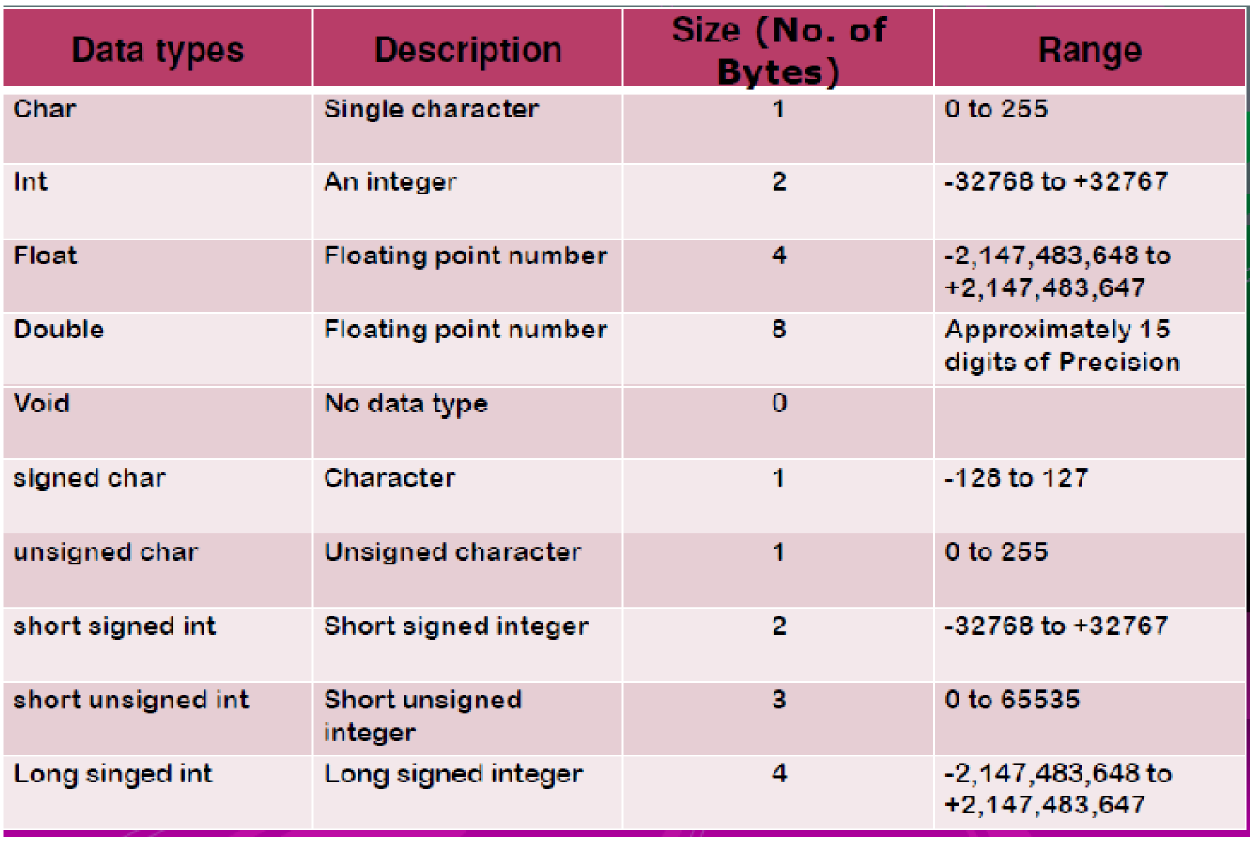
Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left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

Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Data Type

- A data type defines a set of values and the operations that can be performed on them.
- Every datatype item (constant, variable etc.) in a C program has a datatype associated with it.
- C also has a special datatype called void, which, indicates that any data type, i.e., no data type, does not describe the data items.





Data types	Description	Size (No. of Bytes)	Range
Char	Single character	1	0 to 255
Int	An integer	2	-32768 to +32767
Float	Floating point number	4	-2,147,483,648 to +2,147,483,647
Double	Floating point number	8	Approximately 15 digits of Precision
Void	No data type	0	
signed char	Character	1	-128 to 127
unsigned char	Unsigned character	1	0 to 255
short signed int	Short signed integer	2	-32768 to +32767
short unsigned int	Short unsigned integer	3	0 to 65535
Long singed int	Long signed integer	4	-2,147,483,648 to +2,147,483,647

Sample program illustrating each data type

```
#include<stdio.h>
main()
{
        int sum;
        float money;
        char letter;
        double pi;
        sum = 10; /* assign integer value */

        money = 2.21; /* assign float value*/
        letter = 'A'; /* assign character value */
        pi = 2.01E6; /* assign a double value */
         printf("value of sum = %d\n", sum );
        printf("value of money = %f\n", money );
         printf("value of letter = %c\n", letter );
        printf("value of pi = %e \n", pi );
}
```

Output

```
value of sum = 10
value of money=2.210000
value of letter = A
value of pi = 2.010000e+06
```

Arithmetic Statement

There are three types of arithmetic statement which are following;

i) Integer mode arithmetic statement – In this all the operands are either integer constants or integer variables.

   Eg int
   a,b,c; a =
   a+5;
   b=a*c-8;

ii) Real mode arithmetic statement - In this all the operands are either real constants or real variables.

   Eg float
   a,b,c;
   a=a+5.5;
   b= a*b-3.8/9.7

iii) Mixed mode arithmetic statement – In this some operand are real and some are integers

   float a,b,c;
   int d,e;
   a=b*c/9.5;
   d=e/9;

<u>Assignment Statement</u>

Once we've declared a variable we can use it, but not until it has been declared - attempts to use a variable that has not been defined will cause a compiler error. Using a variable means storing something in it. You can store a value in a variable using:

name = value;
For example: a=10;

stores the value 10 in the int variable a. What could be simpler? Not much, but it isn't actually very useful! Who wants to store a known value like 10 in a variable so you can use it later? It is 10, always was 10 and always will be 10. What makes variables useful is that you can use them to store the result of some arithmetic.

Consider four very simple mathematical operations: add, subtract, multiply and divide. Let us see how C would use these operations on two float variables a and b.

Add	a+b
Subtract	a-b
Multiply	a*b
Divide	a/b

Note that we have used the following characters from C's

character set: + for add,,- for subtract ,* for multiply

/ for divide

BE CAREFUL WITH ARITHMETIC!!! What is the answer to this simple calculation? a=10/3

The answer depends upon how a was declared. If it was declared as type int the answer will be 3; if a is of type float then the answer will be 3.333.

Two points to note from the above calculation:

- C ignores fractions when doing integer division
- When doing float calculations integers will be converted into float.

Input /Output statement

ANSI standard has defined many library functions for input and output in C language. Functions printf() and scanf() are the most commonly used to display out and take input respectively. Let us consider an example:

```
#include <stdio.h>      //This is needed to run printf() function.
int main()
{
   printf("C Programming");  //displays the content inside quotation
   return 0;
}
```

Output

```
C Programming
```

- Inside quotation of printf() there, is a conversion format string "%d" (for integer). If this conversion format string matches with remaining argument,i.e, c in this case, value of c is displayed and “%f” for float.

```
#include<stdio.h>
int main()
{
   int c;
   printf("Enter a number\n");
   scanf("%d",&c);
   printf("Number=%d",c);
   return 0;
}
```

The following is a simple C program that prints a message on the screen.

```
/* A simple program for printing a message */

# include <stdio.h>
# include <conio.h>
void main( )
{
        clrscr( );
        printf(“Welcome to C”);
        getch( );
}
```

<u>The first line</u>

/* A simple program for printing a message */
is a comment line. Comments in the c program are optional and may appear anywhere in a C program. Comments are enclosed between /* and */.

<u>The second line</u>

# include <stdio.h>
tells the compiler to read the file stdio.h and include its contents in this file. stdio.h, one of header files, contain the information about input and output functions. stdio.h means Standard Input Output Header file. This file contains the information about printf() function.

<u>The third line</u>

# include <conio.h>
tells the compiler to read the file conio.h and include its contents in this file. conio.h means Consoled Input Output Header file. This file contains the information about clrscr() and getch() functions.

<u>The fourth line</u>

void main( ) is the stat of the main program. The word main is followed by a pair of ordinary parenthesis ( ), which indicates that main is also a function.

<u>The fifth line</u>
{
the left brace represents the beginning of the program.

<u>The sixth line</u>
clrscr( );
tells the compiler to clear the screen and kept the cursor at left side corner.

<u>The seventh line</u>
Printf("Welcome to C");
this function causes its arguments to be printed on the screen on the computer.

<u>The eight line</u>
getch( );
is reads the single character directly from the keyboard without printing
on the screen.

<u>The ninth line</u>
}
the right brace represents the ending of the program.