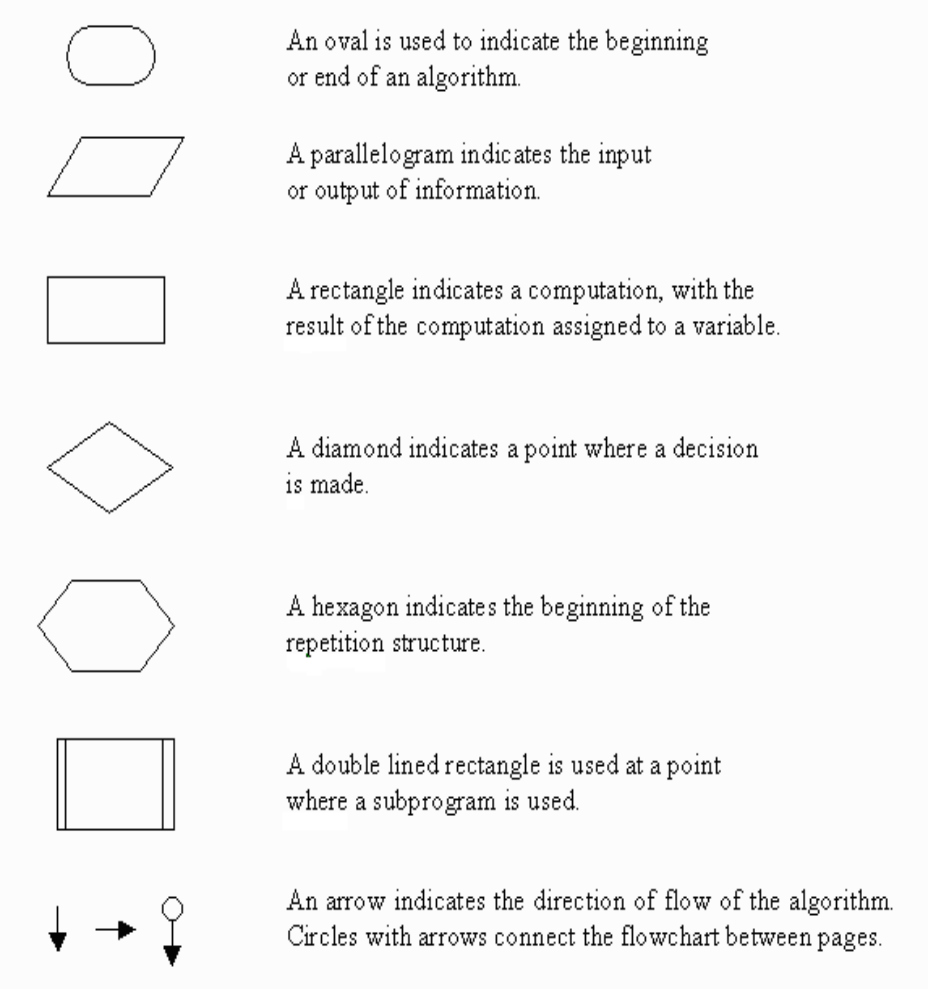
CP FAQS

Q-1) Define flowchart and explain Various symbols of flowchart

ANS. Flowchart:- A diagrametic reperesentation of program is known as flowchart

Symbols



Q-2) Explain basic structure of c language

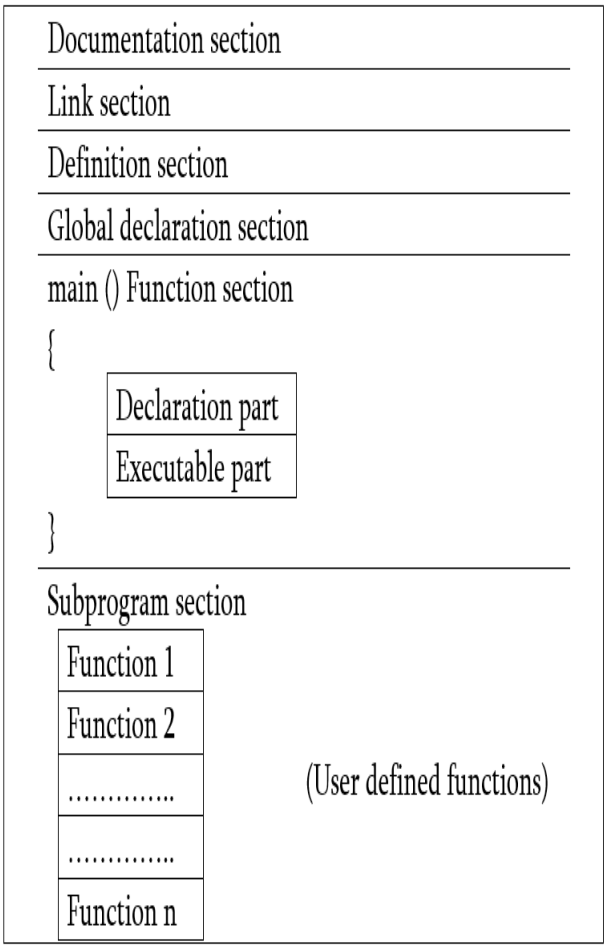
Documentation section : The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.

1. Link section : The link section provides instructions to the compiler to link functions from the system library.

2. Definition section : The definition section defines all symbolic constants.

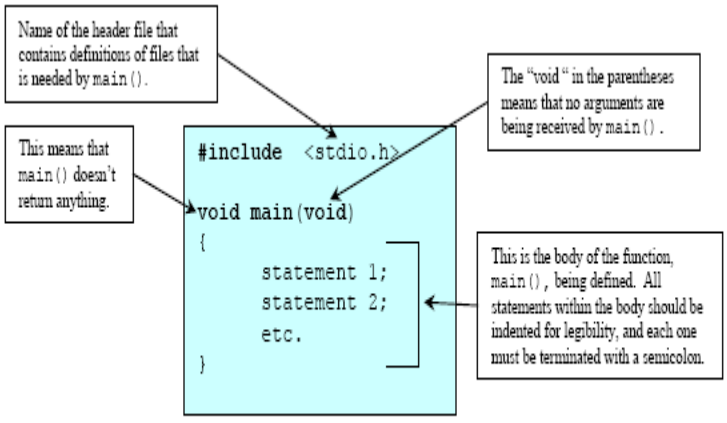
3. Global declaration section : There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.

4. main () function section : Every C program must have one main function section. This section contains two parts; declaration part and executable part.



5. Declaration part : The declaration part declares all the variables used in the executable part.
6. Executable part : There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.
7. Subprogram section : The subprogram section contains all the user-defined functions that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order.

Q-3) Explain structure of C program



Program 1: Basic Structure of the C/C++ program

Q-4) Write short note on

a) Machine level language

Low-level language is a programming language that deals with a computer's hardware components and constraints. It has no or a minute level of abstraction in reference to a computer and works to manage a computer's operational semantics.

Low-level language may also be referred to as a computer’s native language.

b) Assembly level language

An assembly language is a low-level programming language for microprocessors and other programmable devices. It is not just a single language, but rather a group of languages. Assembly language implements a symbolic representation of the machine code needed to program a given CPU architecture.

C) Higher level language

High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture.

High-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

Q-5) What is variables. Explain rules to define variables

The variable represent the quantities which changes with time

Rules to define variables

1) Onl;y uppercase and lowercase leters, digits and underscore can be used.
2) Variable name always begins with lwtter
3) only first 32 character are significant
4) keywords can not be used
5) variable names are case sensitive.

Q-6) Enlist and explain various types of operators in C.

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators
- Increament and decreament operator

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	**Description**	**Example**

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

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

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

<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

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

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, i.e,. 1100 0011 in 2's complement form.

<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

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

Show Examples

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

>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ **sizeof & ternary**

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

Show Examples

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

Increament and Decreament operator

Operator	**Description**	**Example**
++	Adds one to the variable.	i=i+1
--	Subtracts one to the variable.	i=i-1

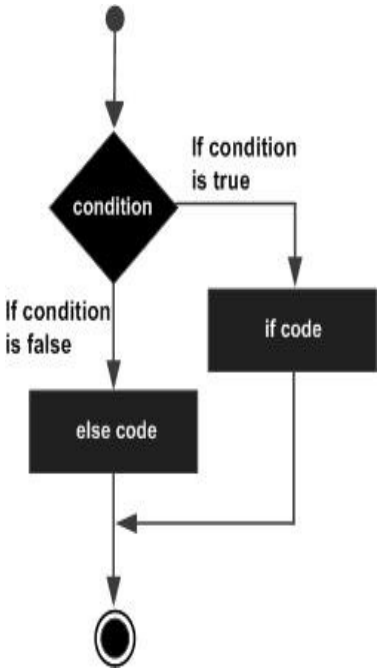
Q-7) Explain if, if..else with example

IF statement

The structure of an if statement is as follows:
if (statement is TRUE)
   Execute this line of code

Here is a simple example that shows the syntax:
if (5 < 10)
   printf("Five is now less than ten, that's a big surprise");

Here, we're just evaluating the statement, "is five less than ten", to see if it is true or not; with any luck, it is! If you want, you can write your own full program including stdio.h and put this in the main function and run it to test.



To have more than one statement execute after an if statement that evaluates to true, use braces, like we did with the body of the main function. Anything inside braces is called a compound statement, or a block. When using if statements, the code that depends on the if statement is called the "body" of the if statement.

For example:
if (TRUE) {
   /* between the braces is the body of the if statement */
   Execute all statements inside the body
}
I recommend always putting braces following if statements. If you do this, you never have to remember to put them in when you want more than one statement to be executed, and you make the body of the if statement more visually clear.

IF.. else statement
An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

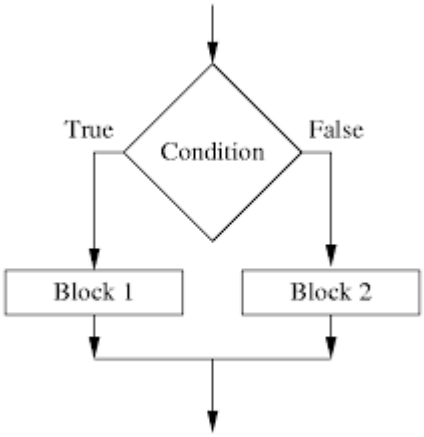
Syntax

The syntax of an if...else statement in C programming language is −

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
}
else {
   /* statement(s) will execute if the boolean expression is false */
}
```

If the Boolean expression evaluates to true, then the if block will be executed, otherwise, the else blockwill be executed. C programming language assumes any non-zero and non-null values as true, and if it is either zero ornull, then it is assumed as false value.

Flow Diagram



EXAMPLE

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a < 20 ) {
      /* if condition is true then print the following */
      printf("a is less than 20\n" );
   }
   else {
      /* if condition is false then print the following */
      printf("a is not less than 20\n" );
   }

   printf("value of a is : %d\n", a);

   return 0;
}
```

Q-9) Explain nested if..else with example

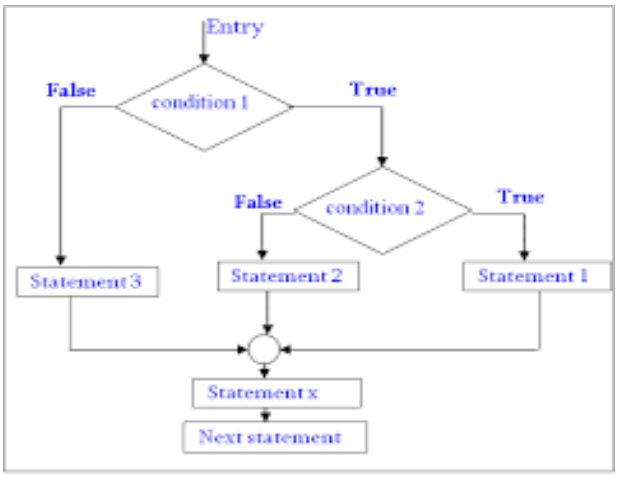
Nested if else statment is same like if else statement, where new block of if else statement is defined in existing if or else block statment.

Used when user want to check more than one conditions at a time.

Syntax

```
if(condition is true)
{
    if(condition is true)
    {
        statement;
    }
    else
    {
        statement;
    }
}
else
{
    statement;
}
```

Flowgraph




```
#include <stdio.h>
 int main () {
  /* local variable definition */
   int a = 100;
   int b = 200;
    /* check the boolean condition */
   if( a == 100 ) {
        /* if condition is true then check the following */
     if( b == 200 ) {
        /* if condition is true then print the following */
        printf("Value of a is 100 and b is 200\n" );
     }
   }
     printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );
    return 0;
```

}

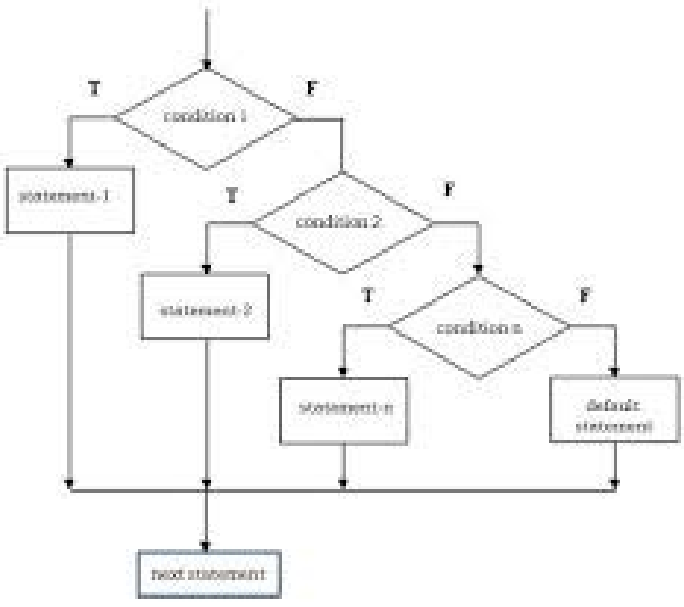
Q-10) Explain if..else if ladder with example

If we are having different - different test conditions with different - different statements, then for these kind of programming we need else if leader. Else if the leader is not interdependent to any other statements or any other test conditions.

Syntax Of Else If Leader:

```
---------------------------------------------------------------------
 if(test_condition1)
    {
      statement 1;
     }
 else if(test_condition2)
    {
      statement 2;
     }
else if(test_condition3)
    {
      statement 3;
     }
 else if(test_condition4)
    {
      statement 4;
     }
 --------------------------------
--------------------------------- // Test Condition And Codes As Per Requirement.
--------------------------------
  else                                  // at last, we use only else.
      {
        statement x;
      }
```

Flowchat



Example

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

```
int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   /* check the boolean condition */
   if( a == 100 ) {

      /* if condition is true then check the following */
      if( b == 200 ) {
         /* if condition is true then print the following */
         printf("Value of a is 100 and b is 200\n" );
      }
   }

   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );

   return 0;
}
```

Q-11) Explain Switch case with example

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Syntax

The syntax for a **switch** statement in C programming language is as follows −

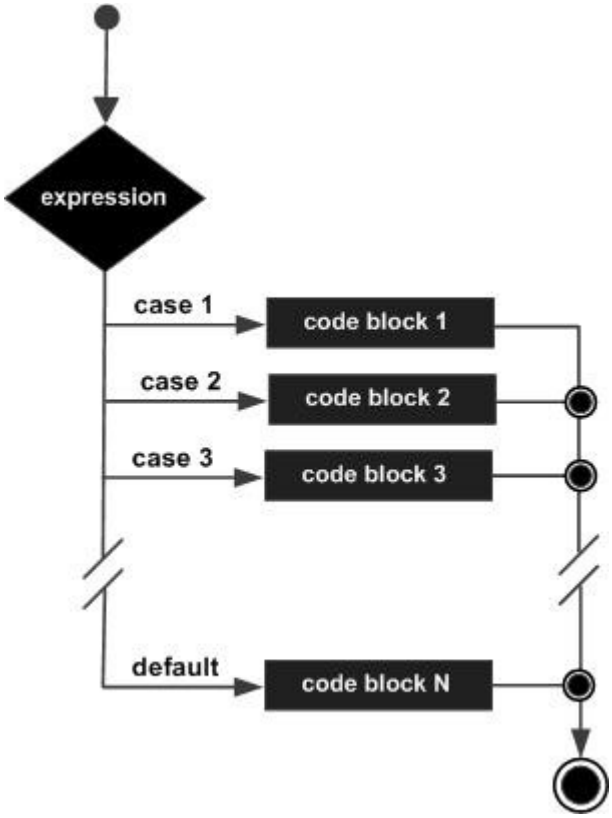
```
switch(expression) {

   case constant-expression  :
      statement(s);
      break; /* optional */

   case constant-expression  :
```

```
   statement(s);
   break; /* optional */

  /* you can have any number of case statements */
  default : /* Optional */
  statement(s);
}
```

The following rules apply to a **switch** statement −

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break**statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No**break** is needed in the default case.

Flowchart



Example

#include <stdio.h>

```
int main () {

   /* local variable definition */
   char grade = 'B';

   switch(grade) {
      case 'A' :
         printf("Excellent!\n" );
         break;
      case 'B' :
      case 'C' :
         printf("Well done\n" );
         break;
      case 'D' :
         printf("You passed\n" );
         break;
      case 'F' :
```

```
      printf("Better try again\n" );
      break;
    default :
      printf("Invalid grade\n" );
  }

  printf("Your grade is  %c\n", grade );

  return 0;
}
```

Q-12) Explain for loop in Cwith example

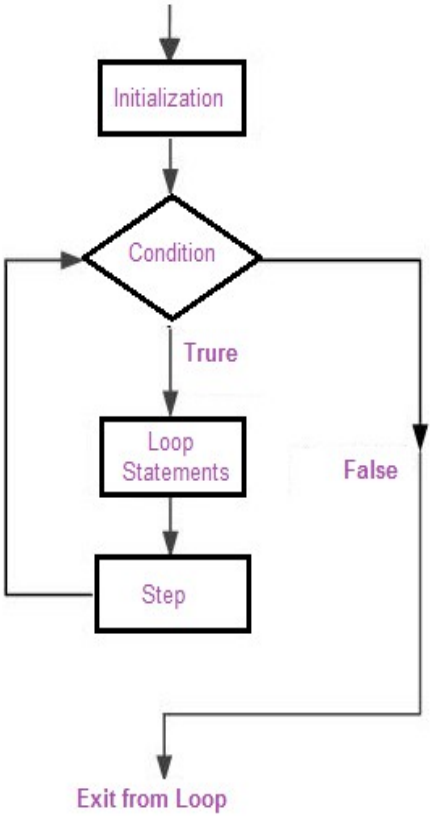
A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for** loop in C programming language is −

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

Here is the flow of control in a 'for' loop −



•

The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

•Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.

•After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

•The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Example

```
#include <stdio.h>

int main () {
   int a;
   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
```

```
    printf("value of a: %d\n", a);
  }
   return 0;
}
```

Q-13) Explain while loop in Cwith example

A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

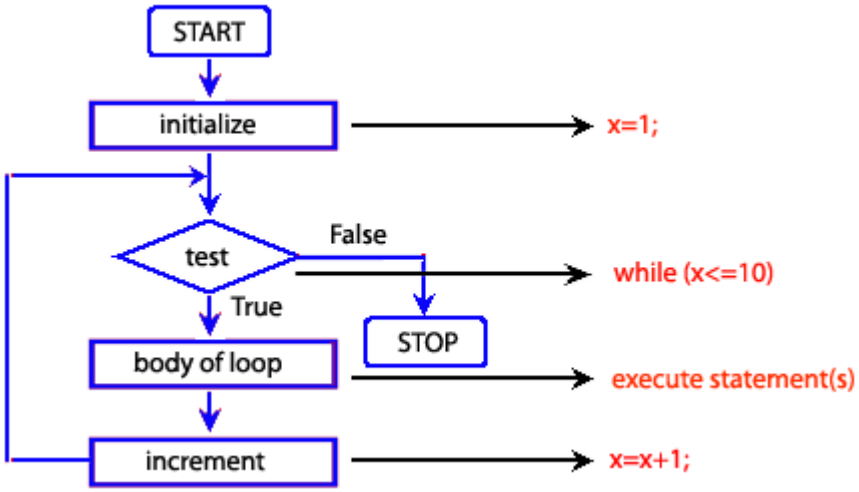
Syntax

The syntax of a **while** loop in C programming language is −

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

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.



Here, the key point to note is that a while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example

```
#include <stdio.h>


int main () {
```

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

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

   return 0;
}
```

Q-14) Explain do...while loop in Cwith example

A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

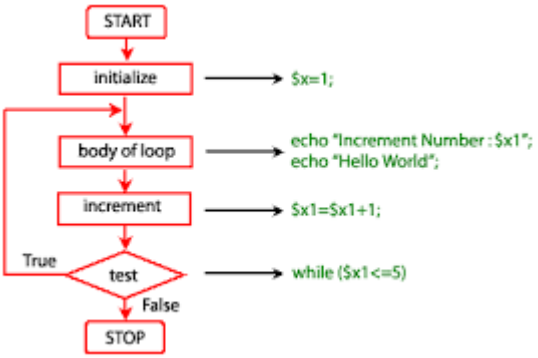
Syntax

The syntax of a **do...while** loop in C programming language is −

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

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.



Example

```
#include <stdio.h>

int main () {

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

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

   return 0;
}
```

Q-15) Explain Goto statement and also define forward jump and backward jump.

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

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

Syntax

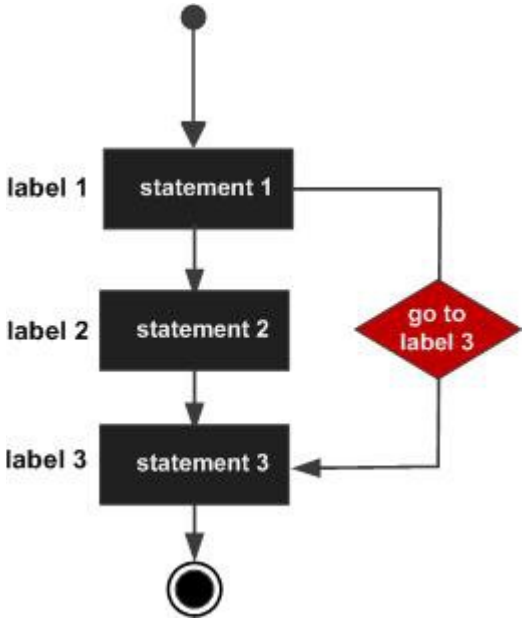
The syntax for a **goto** statement in C is as follows −

```
goto label;
..
```

```
.
label: statement;
```

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

Flow Diagram



Example

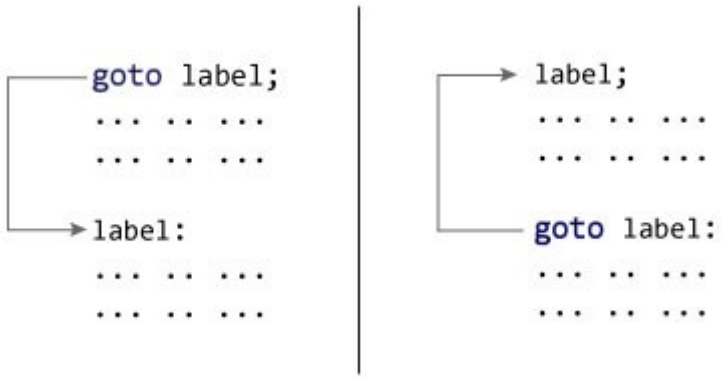
```
#include <stdio.h>

int main () {
   /* local variable definition */
   int a = 10;
   /* do loop execution */
   LOOP:do {
        if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }
      printf("value of a: %d\n", a);
      a++;
```

```
  }while( a < 20 );
   return 0;
}
```

Forward jump and Backward jump

In forward jump the execution skips few lines and jumps forward.

In backward jump the execution repeats few lines and jumps backward.

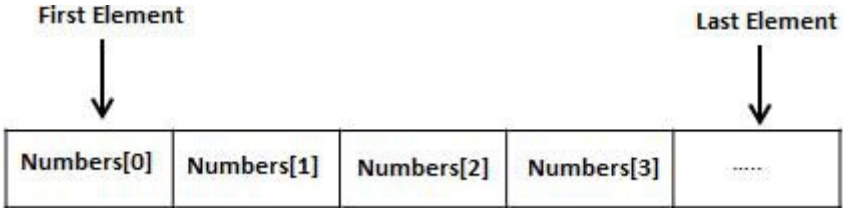


Q-16) What is array? Explain why we have to use arrays in C

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



When we have to store many variables of same datatype it is preferable to use array.

Q-17) Explain one dimensional array with example

The declaration form of one-dimensional array is
Data_type array_name [size];

The following declares an array called ‘numbers’ to hold 5 integers and sets the first and last elements. C arrays are always indexed from 0. So the first integer in ‘numbers’ array is numbers[0] and the last is numbers[4].

int numbers [5];

numbers [0] = 1; // set first element
numbers [4] = 5; // set last element

This array contains 5 elements. Any one of these elements may be referred to by giving the name of the array followed by the position number of the particular element in square brackets ([]). The first element in every array is the zeroth element.Thus, the first element of array ‘numbers’is referred to as numbers[0], the second element of array ‘numbers’is referred to as numbers[1], the fifth element of array‘numbers’is referred to as numbers[4], and, in

Example of One Dimensional Array

```
/*Write  a program to get n numbers and find out sum and average of numbers*/
#include<stdio.h>
#include<conio.h>
void main()
{
  int a[10]; //array of size 10
  int i,n;
  float avg,sum=0;
  clrscr();
  printf("Give the values of n");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
     printf("Give number\n");
     scanf("%d",&a[i]);
     sum=sum+a[i];
  }
avg=sum/n;
printf("Array elements are :\n);
for(i=0;i<n;i++)
  printf("\nSum=%f Average= %6.2f\n",sum, avg);
}
```

general, the n-th element of array ‘numbers’is referred to as numbers[n - 1].

Q-18) Write syntax of nested for loop also explain with example

C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax

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

```
for ( init; condition; increment ) {

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

   statement(s);
}
```

Example

The following program uses a nested for loop to find the prime numbers from 2 to 100 −

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int i, j;

   for(i = 2; i<100; i++) {

      for(j = 2; j <= (i/j); j++)
         if(!(i%j)) break; // if factor found, not prime
         if(j > (i/j)) printf("%d is prime\n", i);
   }

   return 0;
}
```

Q-19) Write advantages and Disadvantages of flow chart

Advantages and Disadvantages of Flowchart

Advantages Of Using FLOWCHARTS:

• Communication: Flowcharts are better way of communicating the logic of a system to all concerned or involved.

• Effective analysis: With the help of flowchart, problem can be analysed in more effective way therefore reducing cost and wastage of time.

• Proper documentation: Program flowcharts serve as a good program documentation, which is needed for various purposes, making things more efficient.

• Efficient Coding: The flowcharts act as a guide or blueprint during the systems analysis and program development phase.

• Proper Debugging: The flowchart helps in debugging process.

• Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part

Disadvantages Of Using FLOWCHARTS:

• Complex logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy. This will become a pain for the user, resulting in a waste of time and money trying to correct the problem

• Alterations and Modifications: If alterations are required the flowchart may require re-drawing completely. This will usually waste valuable time.

• Reproduction: As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.

Q-20) Write advantages and Disadvantages of Algorithms

Advantages:

1) Simple to implement.
2) State changes are stored in stack, meaning we do not need to concern ourselves about them.
3) Intuitive approach of trial and error.
4) Code size is usually small.

Disadvantages:

1) Multiple function calls are expensive.
2) Inefficient when there is lots of branching from one state.
3) Requires large amount of space as the each function state needs to be stored on system stack.