Cross-compiler
A compiler that runs on platform (A) and is capable of generating executable code for platform (B) is called a cross-compiler.

Source-to-source Compiler
A compiler that takes the source code of one programming language and translates it into the source code of another programming language is called a source-to-source compiler.

Interpreter
An interpreter, like a compiler, translates high-level language into low-level machine language. The difference lies in the way they read the source code or input. A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes. In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence. If an error occurs, an interpreter stops execution and reports it. whereas a compiler reads the whole program even if it encounters several errors.

Assembler
An assembler translates assembly language programs into machine code.The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.

Linker
Linker is a computer program that links and merges various object files together in order to make an executable file. All these files might have been compiled by separate assemblers. The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

Loader

Loader is a part of operating system and is responsible for loading executable files into memory and execute them. It calculates the size of a program (instructions and data) and creates memory space for it. It initializes various registers to initiate execution.

BASIC CONCEPT:

- **What is C**

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and developed by Dennis Ritchie from the existing languages ALGOL, BCPL and B on DEC PDP-11 that used the UNIX operating system. C is often called a *middle-level* computer language because it combines the best elements of high-level languages with the control and flexibility of assembly language. It is also called as a procedural language.

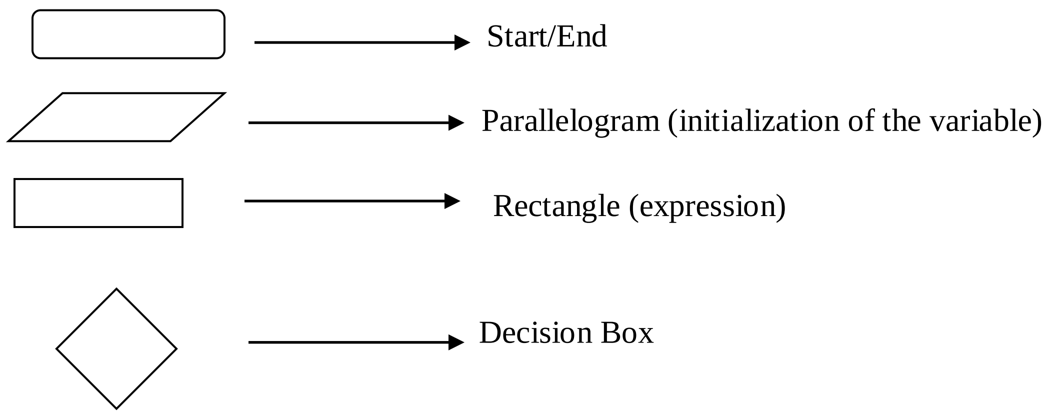
- **Algorithm, Flow Chart, Program**

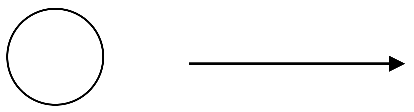
  Algorithm: Algorithm is a finite set of instructions in sequential order that performs the certain type of task to solve a problem. Below is an example for clear understanding an Algorithm

// Addition of two numbers and display of the results //

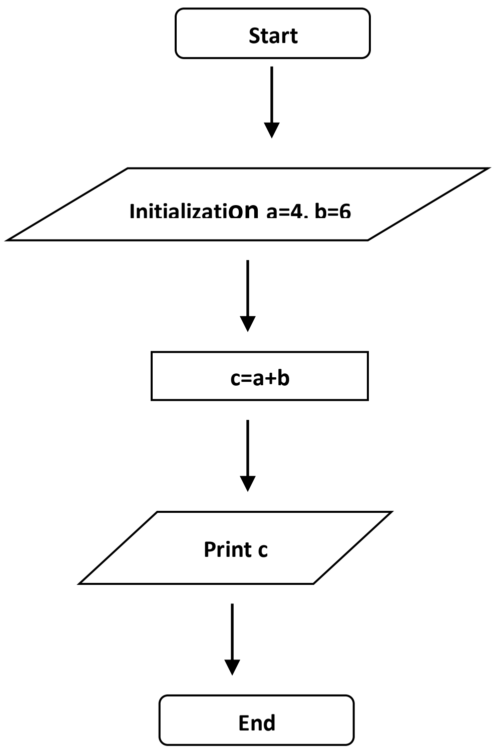
```
Start
    Step1: [Initialization]
            a=4
            b=6
    Step2: [Addition operation]
            c=a+b
    Step3: [Display the result of c]
End
```

Flow Chart: It is a pictorial representation of an algorithm. The following pictures are used to draw a flow chart.





// Flow Chart representation of the above Algorithm is



Program: It is a finite set of instructions in sequential order. The above mentioned *flow chart* may be written in C as given below.

//Addition of two numbers and display of the result using C

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=4,b=6,c;
c=a+b;
printf("The result of sum=%d",c);
getch();
}
```

Output:

```
  The result of sum=10
```

Keywords & Identifier:

- **Header Files**

Each function defined in C standard library has a header file associated with it. The header files are called for work by the term #include <header file name>.
Some Header files and their purposes are given below:

Header File	Purpose
<conio.h>	For console input output extension
<ctype.h>	Character handling
<errno.h>	Error reporting
<float.h>	Defines implementation-dependent floating point limits
<locale.h>	Supports localization
<math.h>	Uses of math library
<signal.h>	Supports signal handling
<stddef.h>	Defines some commonly use constant
<stdio.h>	Supports I/O system
<string.h>	Supports string function
<time.h>	Supports system time function
<complex.h>	Supports complex arithmetic
<stdlib.h>	Miscellaneous declarations

- **Input Statement and Mode Specifications**

A "scanf" statement indicates the Input statement of the "C" program.The general forms of the statement is **scanf ("%<format specifiers>",&<variable name>);**

Format specifiers	Description
%d	Prints an **int** argument in decimal
%ld	Prints a **long** argument in decimal
%c	Prints a character
%s	Prints a string
%f	Prints a **float** or **double** argument in decimal
%e	Same as %f,but uses exponential notation
%g	Uses %e or %f, whichever is better
%o	Prints an **int** argument in octal(base 8)
%x	Prints an **int** argument in hexadecimal(base 16)

Example: scanf (“%d %c %f”, &a, &b, &c);

Input: 10   k   1.20

- **Output Statement and Format Specification**

The statement “printf” indicates the output in “C” program.
The general forms of the statement is available in two forms.

printf(“<String>”);

printf(“%format specifires”, variable name);

Apart from format specification (%d,%f……..) as mentioned above some more symbols (like \n,\r,\t\b) are used. These are explained in the program given below.

```
//Meaning of format specification \n \r \t \b and uses
#include<stdio.h>
#include<conio.h>
void main()
{
printf(“This is a Name List”);
/*    \n   to move the cursor to the next line */
/*    \t    to move the cursor to the next tab(each tab consists of 5 characters) */
/*   \b    to move the cursor to the backspace */
/*   \r     to replace the character */
printf(“\n Swarnendu Kr. Chakraborty”);
printf(“\n Subhasish\tBanerjee”);
printf(“\n Rajat \bGoswami”);
printf(“\n Complete\rJava”);
getch();
}
```

<u>Output</u>

```
This is a Name List
Swarnendu Kr. Chakraborty       // gone to next line
Subhasish       Banerjee          // moved the cursor 5 spaces advance after u
RajatGoswami     // moved one space back side
Javalete                    // first four letters comp of complete have been replaced by ‘java’, from the
beginning
```

➢ Constants and Variables

A constant is a quantity that doesn't change. This quantity can be stored at a location in the memory of the computer. A variable can be considered, as a name given to the location in memory that is used to hold a constant.For example in the equation**a=2b-3c+10;**

Since 2,3and 10 cannot change, they are called constants, whereas the quantities a, b and c can vary, hence are called variables. The location created in memory for a,b and c can store any value.

➢ Data Types and Variables

Data types can be subdivided by three categories

- Basic data types
- Derived data types
- User-defined data types

Declaration	Name Type	Range
Char	Character	-128 to 127
unsigned char	Unsigned character	0 to 255
signed char	Signed character (same as char)	-128 to 127
Int	Integer	-32768 to 32768
signed int	Signed integer (same as int)	-32768 to 32768
short int	Short integer	-32768 to 32768
unsigned shot int	Unsigned short integer	0 to 65535
signed short int	Signed short integer (same as short int)	-32768 to 32768
long or long int	Long integer	-2147483647 to 2147483647
signed long int	Signed long integer	-2147483647 to 2147483647
unsigned long int	Unsigned long integer	0 to 4294967295
Float	Floating-point	Six digits of precision
double	Double floating point	Ten digits of precision
long double	Long double floating point	Ten digits of precision
unsigned int	Unsigned integer	0 to 65535

➢ Rules for Naming the Variables

The variables are recognized by the computer through a name. Following are the rules for constructing names of c variables

(i) Names of variables are constructed by any combination of alphabets,digits and underscores. General size is 1 to 8.

(ii) The first character must be an alphabet or under score.
(iii) No comma or blank space or any other specials characters (except underscore) are allowed in constructing the names of variables.
(iv) The name of any library function should not be used as name of variable.
(v) Some compilers allow up to variable length 40 but it is advisable to keep the length up to 8.

- **Data Variables:**

Data variables are declared in two ways

- LOCAL VARIABLE
- GLOBAL VARIABLE

LOCAL VARIABLES

//A program to compute Fahrenheit from centigrade using the C/5=(F-32)/9

```
#include<stdio.h>
#include<conio.h>
void main()
{
 float F,C;
 printf("enter the value of centigrade temperature:");
 scanf("%f",&C);
 F=(9/5.0)*C+32;
 printf("The Fahrenheit temperature %f",F);
 getch();
 }
```

Output

```
  enter the value of centigrade tempeture:40
   The Fahrenheit temperature 104.000000
```

Here C and F are the two float variables declared inside the **main()** function, hence these variables can be called local variables. Local variables are accessed only that function where it is declared .It is unknown to outside of that function.

GLOBAL VARIABLE

C programming allows us to declare the variable globally, in such fashion that any block code or function can access these variables. Global variables must be declared at the beginning of the program (before **main()** function).

// A program to calculate the simple interest(SI)=(Principle Amount*Year*Rate of Interest)/100

```
#include<stdio.h>
#include<conio.h>
float SI;
void main()
```

```
{
 float N,R,P;
 clrscr();
 printf("enter the value of years,interest,amount:");
 scanf("%f%f%f",&N,&R,&P);
 SI=(P*R*N)/100.0;
printf("The Simple Interest %f",SI);
getch();
 }
```

Output

```
enter the value of years, interest, amount:
2
5
7000
The Simple Interest 700.000000
```

The floating-point variables N,R and P are local variables but floating-point variable SI is called Global variable ,since it has been declared before **main()** function .

<u>Operators</u>

An operator is a special type of symbol that tells the computer to perform certain mechanical or logical manipulation on the data.

There are four main classes of operators:

- Arithmetic
- Relational
- Logical
- Bitwise

<u>ARITHMETIC, LOGICAL, BITWISE AND ASSIGNMENT OPERATORS</u>

Arithmetic Operator	**+,-,*,/,%,++,--**
Relational Operator	>,>=,<,<=,==,!=
Logical Operator	**&&,||,!**
Bitwise Operator	**&,|,^,<<,>>,~**
Assignment Operator	=,+=,-=,*=
Works of Arithmetic Operators	
+	Addition
-	Subtraction
*	Multiplication
/	Division

Works of Relational Operators	
>	Greater Than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Equal to
!=	Not Equal to
Works of Logical Operators	
&&	AND
||	OR
!	NOT
Works of Bitwises Operators	
&	AND
|	OR
^	Exclusive OR
~	One’s complement(NOT)
>>	Shift right
<<	Shift left
Works of Assignment Operators	
a=b	Assign the value of b to a
a+=b	Means a=a+b
a-=b	Means a=a-b
a*=b	Means a=a*b

Note that: a=b and a==b are not same. The latter one is a logical operator which compares the value of with b where as former one assigns the value b to a.

- ***<u>TERNARY OPERATOR (?)</u>***

C has a very powerful and convenient operator that replaces certain statements of the if-then-else form, this is called ternary operator. The ternary operator **?** takes the general form

Expression1 ? Expression2 : Expression3;

where **Expression1, Expression2, Expression3** are expressions. The ternaryoperator **?** works like this: First **Expression1** is evaluated. If it is true then **Expression2** is evaluated and its value is recorded. If **Expression1** is false then control goes to evaluate **Expression3** and value of **Expression3** is recorded.

```
//A simple program touse of ternary operator (?)
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
int A=12, B;
clrscr();
B=A>10?50:25;
printf("\n B=%d",B);
getch();
}
```

Output

```
B=50
```

- ***THE COMPILE-TIME OPERATOR (sizeof)***

The “**sizeof”** operator is a unary compile –time operator that returns the length, in byte, of the variable or parenthesized type-specifier that it precedes.

//A simple program touse of “sizeof” operator

```
#include<stdio.h>
#include<conio.h>
void main()
{
int t;
char c;
float f;
double d;
clrscr();
printf("\n T=%d",sizeof(t));
printf("\n C=%d",sizeof(c));
printf("\n F=%d",sizeof(f));
printf("\n D=%d",sizeof(d));
getch();
}
```

Output

```
 T=2
 C=1
 F=4
 D=8
```

Since t is an integer variable so size is 2 byte. For character variable c size is 1 byte. Floating point variable f size is 4 and double variable d size is 8 byte.

- ***THE COMMA (,) OPERATOR***

The comma operator strings together several expressions. The left side of the comma operator always evaluated as **void.** This means that the expression on the right side becomes the value of the total comma-separated expression.

```
//A simple program touse of comma operator
#include<stdio.h>
#include<conio.h>
void main()
{
int t,x;
clrscr();
x=(t=10,t+1);             /* Use of , operator*/
printf("\n X=%d",x);
getch();
}
```

Output

```
 X=11
```

- ***<u>THE OPERATOR MODULAR (%)</u>***

The operator % is called modular operator. The form is a%b. This gives the remainder when a is divided by b. Example 7%2 will give result 1 and 6%2 will give result 0. There is difference in / and % operator. 7/2 will give 3.5, however if a is in integer mode a=7/2 will give 3.

Increment and decrement Operator

```
#include<stdio.h>
int main()
{
int a = 10, b = 100;
float c = 10.5, d = 100.5;

printf("++a = %d \n", ++a);
printf("--b = %d \n", --b);
printf("++c = %f \n", ++c);
printf("--d = %f \n", --d);

return 0;
}
```

Output

```
++a=11
--b=99
++c=11.500000
--d=99.500000
```

// Working of assignment operators

```
#include <stdio.h>
int main()
{
int a = 5, c;

    c = a;      // c is 5
printf("c = %d\n", c);
    c += a;     // c is 10
printf("c = %d\n", c);
    c -= a;     // c is 5
printf("c = %d\n", c);
c *= a;     // c is 25
printf("c = %d\n", c);
c /= a;     // c is 5
printf("c = %d\n", c);
c %= a;     // c = 0
printf("c = %d\n", c);
return 0;
}
```

OUTPUT

```
C=5
C=10
C=5
C=25
C=5
C=0
```

// **Working of relational operators**

```
#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;
}
```

OUTPUT

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

1. Area and perimeter of circle in C program

```
#include<stdio.h>
#include <math.h>
int main()
{
  float radius, area, perimeter;

  printf("Enter the radius of a circle\n");

  scanf("%f", &radius);

  area = 3.14159*radius*radius;

perimeter=2*3.14*radius;

  printf("Area of the circle = %.2f\n", area);

printf("Perimeter of the circle = %.2f\n", perimeter);

  // printing upto two decimal places

  return 0;
}
```

Output

```
Enter the radius of circle is : 5
Area of the circle =78.54
Perimeter of the circle =31.42
```

2. Write a Program to calculate and display the volume of a CUBE.

```
#include <stdio.h>
#include<conio.h>
void main()
 {
 int h,w,d,vol;
printf("enter the height, weight, depth of cube");
scanf("%d%d%d",&h,&w,&d);
 vol=h*w*d;
 printf("The Volume of the cube is: %d",vol);
 getch();
```

Output

```
Enter the height, weight, depth of cube: 5,10,12
The volume of the cube is: 600
```

3. Write a program to swap values of two variables without using third variable

```
#include <stdio.h>
int main()
{
int a, b, c;
printf("enter the values of a and b");
scanf("%d %d", &a, &b);
printf("before swapping %d %d", a, b);
a=a+b;
b=a-b;
a=a-b;
printf("after swapping values are %d %d ", a, b);
return 0;
}
```

OUTPUT

```
enter the values of a and b 8 5
8 5
before swapping 8 5after swapping values are 5 8
```

4. Write a program to swap values of two variables with using third variable

```
#include <stdio.h>
int main()
{
int a, b, c;
printf("enter the values of a and b");
scanf("%d %d", &a, &b);
printf("before swapping %d %d", a, b);
c=a;
a=b;
b=c;
printf("after swapping values are %d %d ", a, b);
return 0;
}
```

OUTPUT

```
enter the values of a and b 8 5
8 5
before swapping 8 5after swapping values are 5 8
```

Decision Making:

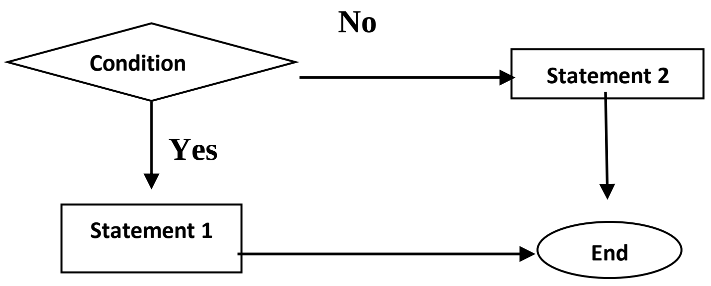
➢ ***if else STATEMENT***

if-else statement is conditional statement. The syntax of if-else statement is

```
if(expression)
     statement1;
else
     statement2;
```

Here statement1 will be executed when expression is true otherwise statement2 will be executed.

if-else Condition may be represented by using Flow-chart as:



i)a.To find the Bigger one between two numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
 clrscr();
 printf("Enter the values of A and B");
 scanf("%d%d",&a,&b);
 printf("\nTwo Numbers A=%d    B=%d",a,b);
 if(a>b)
 printf("\nNumber=%d is Bigger ",a );
 else
 printf("\nNumber=%d is Bigger ",b );
 getch();
 }
```

Output

```
Enter the value of A and B
5
4
```

```
Two Numbers A=5    B=4
Number=5 is Bigger
```

i)b. Identification of Even or Odd Number

```c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,c;
clrscr();
printf("Enter The Number");
scanf("%d",&a);
c=a%2;
if(c==0)
printf("The Numbe %d is Even",a);
else
printf("The Numbe %d is Odd",a);
getch();
}
```

Output

```
Enter The Number
5
The Number 5 is Odd
Enter The Number
6
The Number 6 is Even
```

ii) Calculate Roots of a Quadratic Equation

//If $ax^2+bx+c=0$ is a Quadratic Equation then roots of this equation calculated by the formula $(-b\pm\sqrt{(b^2 -4ac)}/)2a$

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
float a,b,c,d,e;
float root1,root2;
clrscr();
printf("Enter The Vales of A ,B and C");
scanf("%f%f%f",&a,&b,&c);
e=b*b-4*a*c;
if(e>0)
{
d=sqrt(e);  /* sqrt() is a Standard Library function Calculate Square root*/

root1=(-b+d)/(2*a);
root2=(-b-d)/(2*a);
printf("\nThe Root1=%f",root1);
printf("\nThe Root2=%f",root2);
}
else
printf("Roots are imaginary");
getch();
}
```

Output

```
Enter The Vales of A ,B and C
5
4
3
Roots are imaginary
Enter The Vale of A ,B and C
 1
-1
-6

The Root1=3.000000
The Root2=-2.000000
```

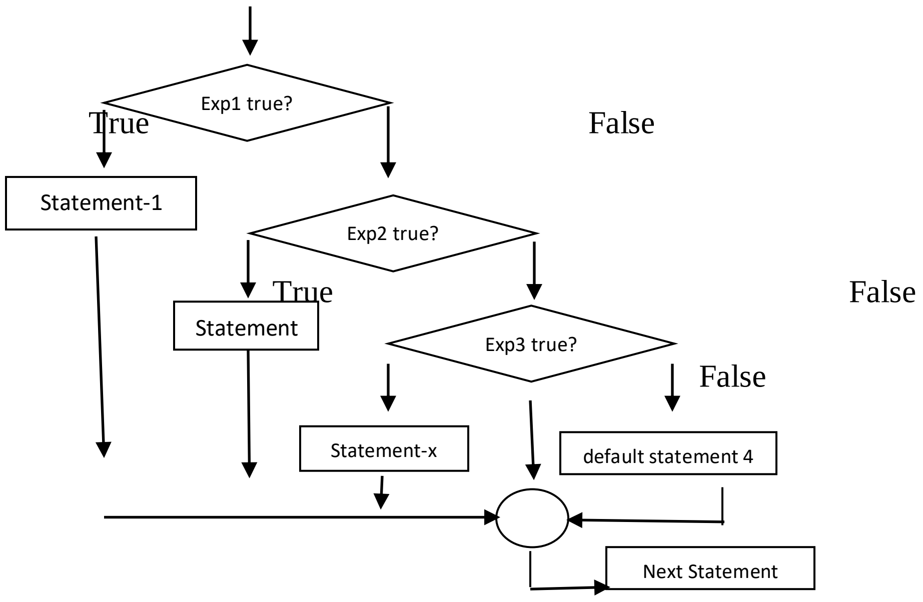
- ***NESTING if else STATEMENT***

When a series of decisions are involved, nested if-else statement is used

The construction

```
        if(expression1)
            statement1
        else if(expression2)
            statement2
        else if(expression3)
            statement3
        else
            statement4
```

Flow chart of *nested if else* statement



iii) Calculate the Biggest Number out of three given numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b, c;
clrscr();
printf("Enter the value of A ,B and C--------");
scanf("%d%d%d",&a,&b,&c);
if(a>b&&b>c)
printf("\nThe Number %d is Biggest ",a);
else
if(b>a&&a>c)
printf("\nThe Number %d is Biggest ",b);
else
printf("\nThe Number %d is Biggest ",c);
getch();
}
```

Output

```
Enter the value of A ,B and C--------
5
3
2
The Number 5 is Biggest
```

iv) Write a program to compute grade of students using **if else adder**. The grades are assigned as followed:

a.	**Marks**	**Grade**
b.	marks<50	F
c.	50≤marks< 60	C
d.	60≤marks<70	B
e.	70≤marks<80	A
f.	80≤marks<90	E
g.	90≤marks≤ 100	O

```
#include<stdio.h>
int main()
{
float marks;
char grade;
printf("Enter marks: ");
scanf("%f", &marks);
if(marks >= 90&& marks<=100)
{
printf("grade is %c =" , 'O');
}
else if(marks >= 80 && marks < 90)
{
printf("grade is %c =" , 'E');
}
else if(marks >= 70 && marks < 80)
{
printf("grade is %c =" , 'A');
}
else if(marks >= 60 && marks < 70)
{
printf("grade is %c =" , 'B');;
}
else if(marks >= 50 && marks < 60)
{
printf("grade is %c =" , 'C');;
}
else
{
printf("grade is %c =" , 'D');;
}
return 0;
}
```

OUTPUT

```
Enter marks : 74
Grade is: A
```

- ***switch STATEMENT***

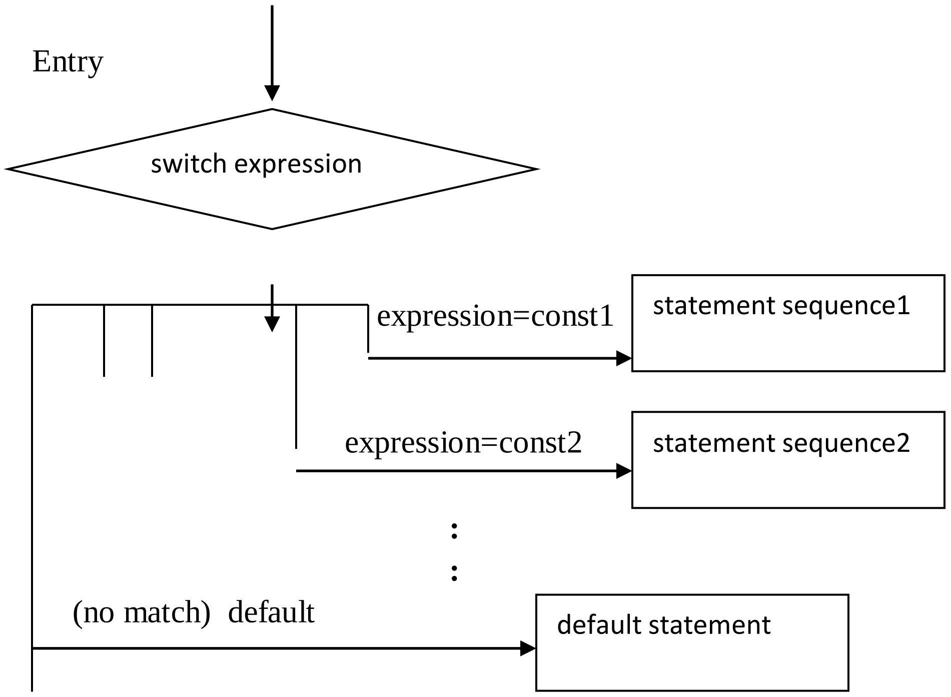
If there are a number of decisions statements, each decision statement is identified by a keyword **case**. When we want to execute a particular block of code (against **case)** then **switch** statement may be used.

The **switch** Statement construction

```
switch (expression)
{
case const1:
                statement sequence1;
                break;
case const2:
                statement sequence2 ;
                break;
case const3:
                statement sequence3;
                break;
default:
        statement sequence4;
}
```

A **default** is optional keyword. If **default statement** is not declared and your choice does not match with any declared **case statement**, no action will be taken. The **break statement** causes an immediate exit from **case**.

Flow chart of *switch* statement



//A program that reads a number from 1 to 7 and accordingly display MONDAY to SUNDAY

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
clrscr();
printf("Enter the number between 1 to 7--------");
scanf("%d",&n);
switch(n)
{
case 1: printf("MONDAY");
        break;
case 2: printf("TUESDAY");
        break;
case 3: printf("WEDNESDAY");
```

```
            break;
case 4: printf("THURSDAY");
            break;
case 5: printf("FRIDAY");
            break;
case 6: printf("SATURDAY");
            break;
case 7: printf("SUNDAY");
            break;
}
getch();
}
```

Output

```
Enter the number between 1 to 7--------5
FRIDAY
```

2. Write a program to find whether a character is consonant or vowel using switch statement**.**

```
// Online C compiler to run C program online
#include <stdio.h>
int main()
{
char ch;
printf("enter any alphabet/ character");
scanf("%c", &ch);
switch(ch)
{
case 'a':
case 'A':
printf("vowel");
break;
case 'e':
case 'E':
printf("vowel");
break;
case 'i':
case 'I':
printf("vowel");
break;
case 'o':
case 'O':
printf("vowel");
break;
case 'u':
case 'U':
printf("vowel");
break;
default: printf("Consonant");
}
return 0;
}
```

OUTPUT

```
enter any alphabet/ character p
p
Consonant
enter any alphabet/ character E
E
vowel
```