Hearty Welcome
Have A Nice Day

Programming in c++

YEAR II

TERM III

EEE

UNIT I : INTRODUCTION TO OOPS & C++ PROGRAMMING

- **1.1 Introduction to OOPS:** Paradigms of Programming Languages – Basic concept of Object Oriented Programming – Differences between Procedure Oriented Programming & Object Oriented Programming, characteristics of Object Oriented Languages – Objects, Classes, Inheritance, Polymorphism, Dynamic binding, message communication – Benefits of OOP - Application of OOPs.
- **1.2 C++ :** Introduction to C++: Features of C++ - Benefits of C++ - Applications of C++ - Structure of C++ program- Tokens, Comments, Basic data types, User defined data types, Derived data types ,Symbolic constants, Type Compatibility, Declaration Of Variables, Dynamic Inilialization of variables, Reference variables.
- **1.3 Operators in C++:** Scope resolution operator, Member dereferencing operators, Memory management operators ,Manipulators (setw & endl), Type cast operator , Operator precedence, control structures.

INTRODUCTION

- TO WATCH BASIC C++
- https://www.youtube.com/watch?v=gscX6RGYd2k

Features of procedure oriented programming

- Large problems are divided into smaller problems known as functions or procedures.
- It uses top-down programming technique.
- Data moves freely from one function to another.
- Functions can share global variables.
- Data hiding is not possible.
- It is difficult to added new functions and data structures.

Features of Object Oriented Programming

- Problems are divided into objects.
- It is not possible to access data freely.
- Data hiding is possible.
- It is easy to add new data and functions.
- Objects can exchange data through its function.

BENEFITS OF OOP:

- **Reusability**: In OOP's programs functions modules written by a user can be reused by other users without any modification.
- **Code sharing**: In OOP's , the programmer can share the codes (data definition and function coding) which are common to more than one class by defining it in the parent classes in hierarchical order.
- **Data hiding**: Programmer can hide data and function's in a class from other classes.
- **Reduced Complexity of a problem:** In OOP's the given problem is viewed as a collection of different objects. Each object is responsible for a specific task. The problem is solved by interfacing the objects.
- In OOP's, software system can be developed more quickly and easily by using the existing defined components. This technique of development is called prototyping,
- **Message passing technique:** This technique Prototyping reduces the complexity of interfacing two objects.
- **Extendability :** Object oriented software can be easily extended from small to large.

FEATURES OF C++:

- Access right: C++ hass a facility to protect the data and functions in a class This is done by the following **visibility modifiers**
  - Public: Data and functions in a class can be accessed from any class.
  - Protected: Data and functions in a class can be accessed only by the derived classes.
  - Private: Data and functions in a class can be accessed by the class itself.
- Spot Access: The function or a class declared as protected or private can be accessed by a non member class with the help of "friend" declaration. This concept is called spot access.
- Overloading facility: This facility helps the users to give a new meaning to the exiting operators or functions.
- Memory allocation facility**:** C++ supports several memory allocation schemes for objects. They are
  - Statically allocation schemes for objects.
  - Stack based allocation.
  - Run time allocation from a heap.

APPLICATIONS OF OOP:

It is used in

- Computer animation.
- Logical network designing.
- Electrical distribution design systems.
- Simulation & modeling.

It is used in

- Design Compilers.
- Develop expert system software.
- Develop computer games.
- Access relational databases.
- Develop software administrative tools, system tools etc.

Basic concepts of oop:

- OOP development is a new programming style having real world thinking.
- To standardize the thinking, the following concepts are defined:
  - **Object**
  - **Class**
  - **Data Abstraction**
  - **Data Encapsulation**
  - **Inheritance**
  - **Polymorphism**
  - **Dynamic Binding**
  - **Message Passing**

OBJECT:

- An object is defined as an entity that contain data and its related functions. The functions operate on that data. The objects may be either physical or logical.

- While representing objects in computer it occupies some memory space and have an associated address. During execution, objects can exchange information.

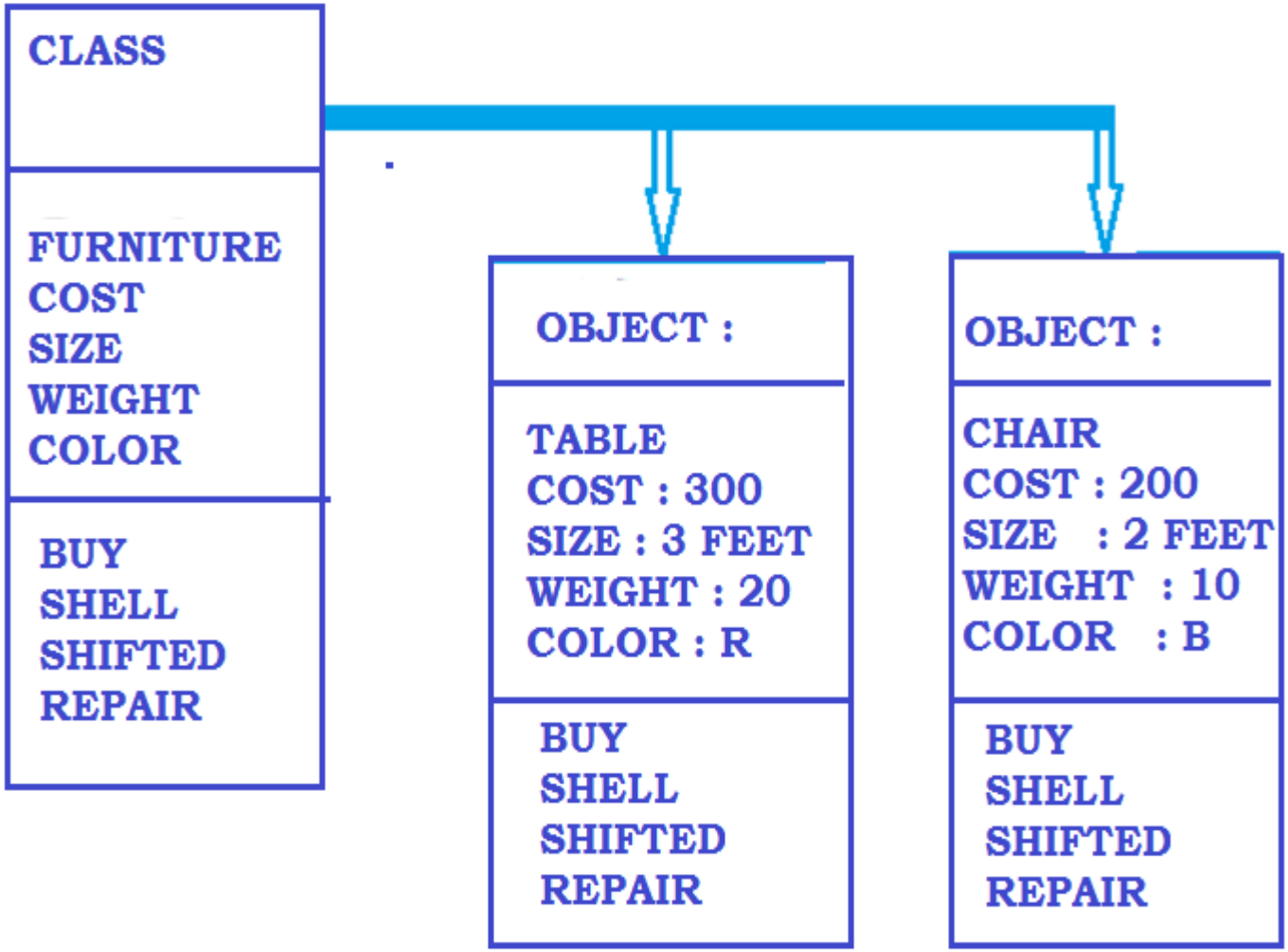
EXAMPLE

OBJECTS	DATA(ATTRIBUTES)	FUNCTIONS
TRIANGLE 	Number of Sides Side a , Side b, Side c Border color , Fillcolor	draw(), fillcolor(), area() , move()
CYCLE 	make , framesize wheelsize, gears material	shift(), move() , repair() , assemble()
TABLE REG | NAME | CLASS 1 | AB | E 2 | BC | F 3 | CA | S	Number of Rows Number of Columns	insert() , delete(), search()

CLASS: A class is defined as a collection of objects with same type of data and functions.

- The function of the class should be defined.
- All objects occupies equal memory size
- Each of Object must be member or instance of any one Class.
- In Class definition, the data are declared and the Functions are defined.
- The Objects contain the value of the data and the action of the current function

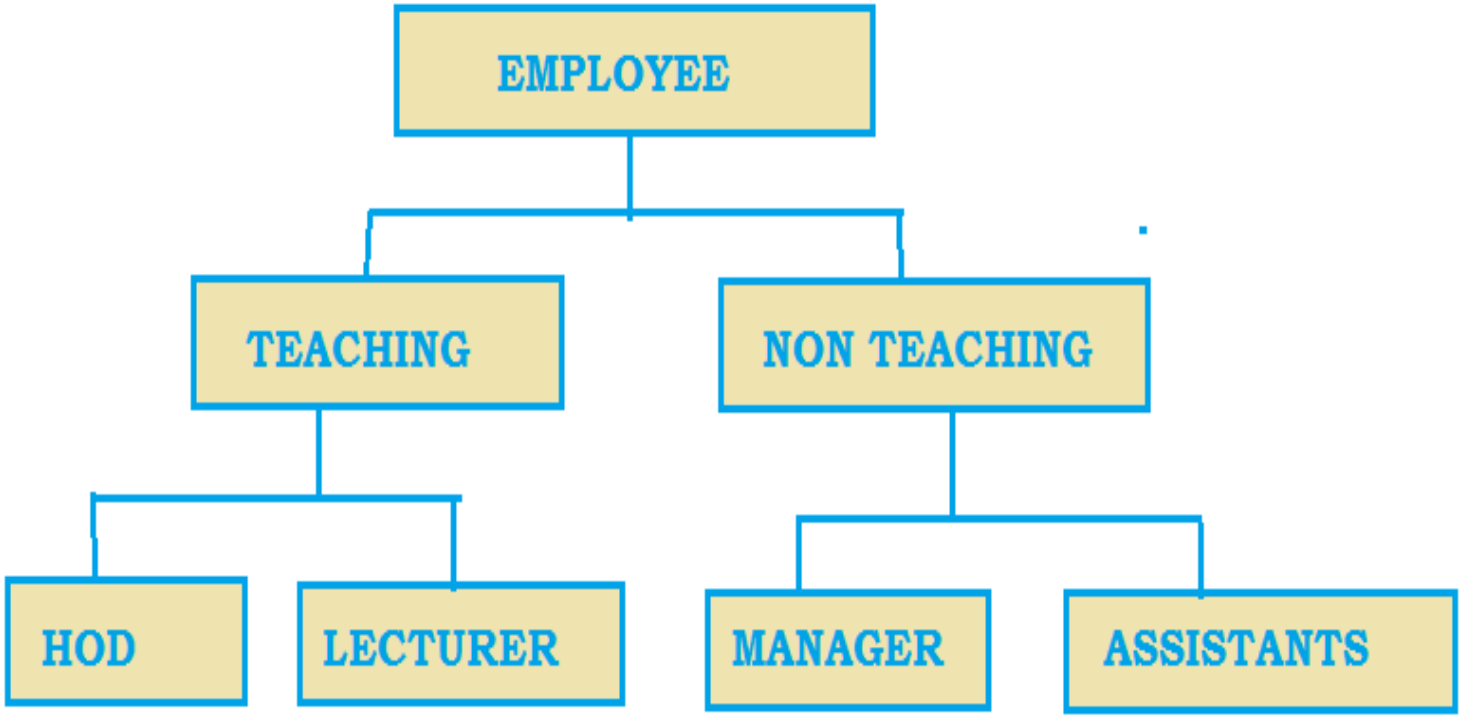
EXAMPLE


CLASS
FURNITURE
COST
SIZE
WEIGHT
COLOR
BUY
SHELL
SHIFTED
REPAIR
OBJECT :
TABLE
COST : 300
SIZE : 3 FEET
WEIGHT : 20
COLOR : R
BUY
SHELL
SHIFTED
REPAIR
OBJECT :
CHAIR
COST : 200
SIZE : 2 FEET
WEIGHT : 10
COLOR : B
BUY
SHELL
SHIFTED
REPAIR

- **Data Abstraction:**
  - Abstraction is defined as a grouping of essential details and ignoring other details,
  - Data abstraction is defined as a named collection of data that describes a data object in a class.
  - Class is abstract data types because classes use the principle of data abstraction.
- **Data Encapsulation**
  - Encapsulation is a technique used to protect the information in an object from other objects.
  - In an object, data and its functions are encapsulated into a single entity.
  - Because of this other objects and programs cannot access the data in an object directly.
  - This concept is called data encapsulation or data hiding.

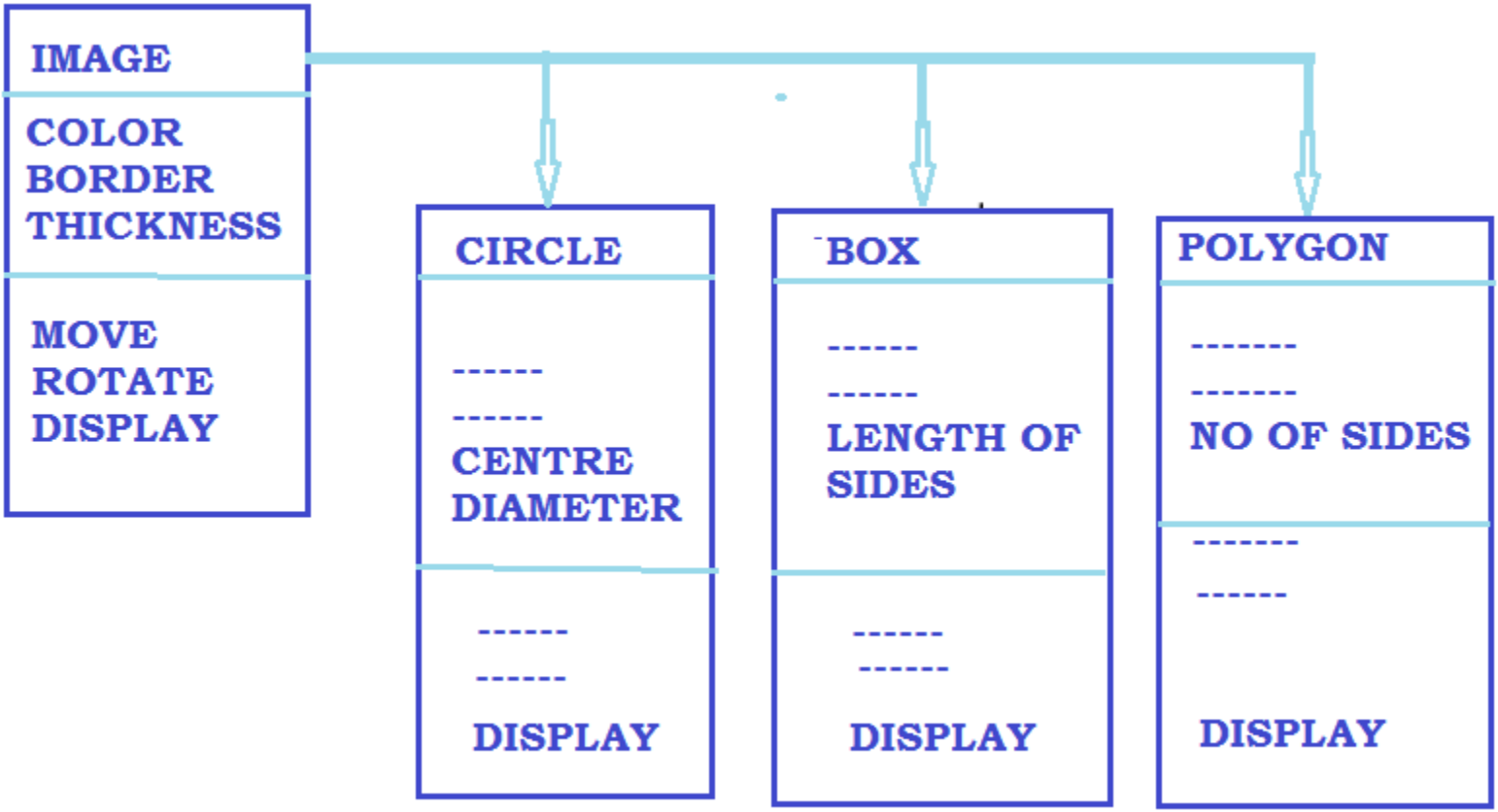
INHERITANCE

- Inheritance is defined as sharing of attributes (data) and functions among classes based on a hierarchical or sequential relationship.



POLYMORPHISM

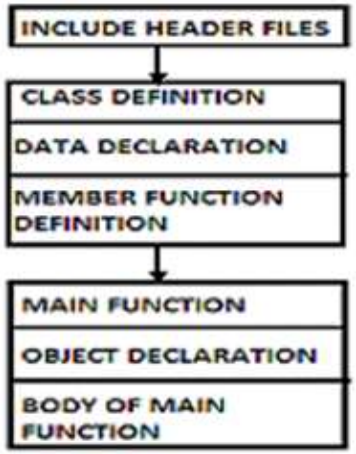
- A function is said to be polymorphic, if is applied to many different classes with different operation.



- In the above classes, the function display is defined in all classes.
- But the operation of the function display is different.
- In circle class, the display function draws a circle with the given centre and diameter.
- In polygon, display function draws a polygon with the given number of sides.

- **DYNAMIC BINDING:**
  - Binding is defined as the connection between the function call and its corresponding program code to be executed.
  - There are two types of binding. They are static and dynamic binding.
- In **static binding**, the binding occurs during compilation time.
- In **dynamic binding**, the binding occurs runtime. This is also called late binding.
- **MESSAGE PASSING:**
  - It is a process of locating and executing a function in response to a message locating means matching the given message with the list of available functions.

STRUCTURE OF C++ PROGRAM:



Include file : In c++, number of functions,classes and variables are defined in a special file called HEADER FILE. There are so many header files are available. Each header file has related functions,classes and data.

General form to include header file is

```
#include <header_file_name.h
```

Class definition : It is a user defined data type having defined functions and data

Example

```
#include <iostream.h> --->header file
class Student ---------->class definition
{
private: ---------------->visibility modifier
int Rno;
char name[25]
public : --------------->visibility modifier
void read() ---------------->member function definition
    { cout<<"\n enter register number and name";
      cin>>Rno>>name; }                          body of the member function
void print() ---------------->member function definition
    { cout<<"\n Register number = " <<Rno;
      Cout<<"\n Name : "<<name; }                body of the member
}; ------------------------------------------> end of class
void main() ------------> main function
{
  Student p,q; ---------->data declaration
  p.read();  --> calling member function read() for the object p
  q.read();--> calling member function read() for the object q
  p.print();  --> calling member function print() for the object p
  q. print();--> calling member function print() for the object q
}
```

cout and cin objects:

- cout is an Object.
  - ❖It is predefined in the header file iostream.h.
  - ❖It is used to display the information on the VDU.
- General form:

```
cout<<list to be printed;
```

Where cout – Object

<< - insertion or put to operator.

- **Rules:**
  - ❖This should terminated with semicolon (;).
  - ❖If the list to be printed contains more than one, each should be separated by <<.
  - ❖List to be printed may be variables (or) constants.

SNO	EXAMPLE	OUTPUT
1	cout<<"\nWelcome";	Welcome

2	int a = 7;
	char S[15] = "APPLE";
	cout<<"\nString = "<<S;
	cout<<"\nvalue = "<<a;
		String = APPLE
		value = 7

cin object:

- ❖cin is an object.
- ❖It is predefined in the header file iostream.h.
- ❖It is used to give values to the variables through keyboard.

- General form:

```
cin>>list of variables;
```

- **Rules:**
  - ❖This should terminate with ;
  - ❖List of variables should be separated by >>.

Sno	Example	Meaning
1	cin>>a;	The value given through keyboard assigned to a;
2	int a,b,c; cin>>a>>b>>c;	User has to give 3 values through keyboard. These values are assigned to the variables a, b and c.

C++ TOKENS :

- A token is an individual element in a program.
- More than one token can appear in single line separated by white spaces.
- White spaces may be blank, carriage return or tab.
- Possible tokens are 1) keywords 2) identifiers 3) constants
  4) operators 5) strings.
- Example :

```
void main()
{
    int a, b ,c ;
    ----- }
```

- In this example, tokens are void main ( ) int a , b , c

Keywords :

- Keywords are words which have standard predefined meaning.
- These words should Be used only for their intended purpose.
- They should be written in lower case.

asm	**auto**	**break**	**case**	**catch**	**char**	**class**	**Const**
continue	default	delete	do	double	else	enum	extern
float	for	friend	goto	if	inline	int	long
new	operator	private	protected	public	register	return	short
signed	sizeof	static	struct	switch	template	this	throw
try	typedef	union	unsigned	virtual	void	volatile	While

Identifiers:

- identifiers are names given to variables, functions, arrays and other user defined objects. These are user defined names.
- **Rules:**
  - ❖1) identifiers are formed with alphabets,digits and a special character underscore(_)
  - ❖2) The first character must be an alphabet.
  - ❖3) No special characters are allowed other than underscore.
  - ❖4) They are case sensitive.

- **Example**: Total, A1, name, B12.

Data Types :

- C++ data types can be classified as

  1) Basic data type :

     (i) int (ii) float (iii) char (iv) double

  2) Derived Data type :

     (i) Functions (ii) Arrays (iii) pointers

  3) User defined data type :

     (i) Enumerations (ii) Structures

     (iii) Unions (iv) Classes.

Special Basic Data type:

- void is a special data type.
- The Compiler will not allocate any memory space for this data type.
- This data type is used to

  (i) define the functions.

  (ii) define the parameter passing
- Example

  1) void fun1(); // fun1() will not return any value

  2) int fun2(void); /* function fun2 will not receive any value & returns an Integer type value. */

Basic data type :

- They are already defined in the language.
- They are (i) int (ii) float (iii) float (iv) double
- The keywords (signed, unsigned, long & short) are prefixed with the basic data types to produce new data type.
- The keywords (signed, unsigned, long & short) are called as **modifiers**

SNO	TYPE	SIZE		RANGE
		(BITS)	(BYTES)	
1	Char (or) signed char	8	1	-128 to 127
2	unsigned char	8	1	0 to 255
3	int (or) signed int	16	2	-32768 to 32767
4	unsigned int	16	2	0 to 65535
5	short int (or) signed short int	8	1	-128 to 127
6	unsigned short int	8	1	0 to 255
7	long int (or) signed long int	32	4	-2,147,483,648 to 2,147,483,647
8	unsigned long int	32	4	0 to 4,294,967,295
9	float	32	4	3.4E-38 to 3.4E+38
10	double	64	8	1.7E-308 to 1.7E+308
11	long double	80	10	3.4E-4032 to 1.1E+4932

Declaration of variables:

- **All variables used in the program should be declared before it is used.**
- **The declaration can also be done before the first use of the variable.**
- **Syntax :**

```
Datatype      variable list;
```

- **Where**
  - ❖**datatype : int,float,char,etc**
  - ❖**Variable list : Valid c++ variables separated by comma**

Example No	Statement	Explanation
1	int a,b,c;	Declares a, b , c as integer variables and allocates memory space
2	float avg;	Declares avg as float variables and allocates memory space
3	Char name[50];	Declares an character type array named as name having 50 memory location to store 50 character
4	Int A[3][3];	Declares an integer array named as A having 3 rows and 3 columns. 18 memory location to store 9 integer data

Dynamic Initialization of variables:

- We can assign initial values to the variables during the run time is called Dynamic Initialization of variables
- Syntax :

```
Datatype var = value;
```

- Where
  - ❖ Datatype :- int,float,char,etc
  - ❖ Var :- Valid c++ variable name
  - ❖ value :- Initial values to be given to the variable var

Example no	Statement
1	int I = 1;
2	int k; cin>>k; float avg = k/10;

User defined Data type :

- These are data types defined by user.
- They are

  (i) Enumeration

  (ii) Structures

  (iii) Class

Enumeration :

- This allows us to define our own data type with predefined values
- General form :
-

```
enum userdefined_name { value1,value2, ----- value n } ;
```

- Where
  - ❖**enum** is keyword
  - ❖**userdefined_name –** valid c++ name
  - ❖**value1,value2, …value n →** list of constants are called members.
- Internally these are treated as integers.
- That is value1 =0, value2 =1,value 3 =2,---- value n =n-1.
- The default values are 0,1,2,----,n-1 can be changed by giving in the declaration

❖General form

```
enum userdefined_name { value1=10,value2 = 5, ----- valuen= 75 } ;
```

Derived data types

- Derived data types are data types derived from existing data types such as basic data types or user defined data types.
- They are (1) Arrays
  (2) Pointer
  (3) Function.
- **Arrays :** An array is defined as a group of same type of data.
- These data share a common name with different index values.
- The index value starts from 0 to n-1.
- The array can be classified as
  - ❖ **(a) one dimensional array**
  - ❖ **(b) Two dimensional array**

Declaration and initialization of arrays :

- An array name with only one subscript is known as One Dimensional Array.(1D)
- An array name with two subscripts is known as Two Dimensional Array. (2D)
- Declaration of array.

sno	Description	Syntax
1	1D	Datatype arr_name[size];
2	2D	Datatype arr_name[size1][size2];

Where	
Datatype	- int, float, char, etc
arr_name	- Valid c++ variable name
size	- Number of contiguous location in the memory to be reserved
size1	- Row number
size2	- Column number

examples

Example No	Statement	Explanation
1	int m[25];	Declares an integer array named as m having 50 memory location to store 25 integer data
2	float salary[25];	Declares an floating point array named as salary having 100 memory location to store 25 float data
3	Char name[50];	Declares an icharacter type array named as name having 50 memory location to store 50 character
4	int A[3][3];	Declares an integer array named as A having 3 rows and 3 columns. 18 memory location to store 9 integer data

Initialization of array:

- **We can assign initial values to the array when they are declared**
- **Declaration and initialization of array.**

sno	Description	Syntax
1	1D	Datatype arr_name[size] = { list of values };
2	2D	Datatype arr_name[size1][size2]= { list of values } ;

Where	
Datatype	- int, float, char, etc
arr_name	- Valid c++ variable name
size	- Number of contiguous location in the memory to be reserved
size1	- Row number
size2	- Column number
list of values	- Initial values to be given to the array.(separated by comma)

Example No	Statement	Explanation
1	int M[3]= { 3,6,9);	Declares an integer array named as m having 6 memory location to store 3 integer data. <table><tr><th>M[0]</th><th>M[1]</th><th>M[2]</th></tr><tr><td>3</td><td>6</td><td>6</td></tr></table>
2	Char A[6] = “KALA” ;	Declares an icharacter type array named as name having 50 memory location to store 50 character <table><tr><th>A[0]</th><th>A[1]</th><th>A[2]</th><th>A[3]</th><th>A[4]</th></tr><tr><td>K</td><td>A</td><td>L</td><td>A</td><td>\0</td></tr></table>
3	int X[2][2] = { 3,4,5,7 }; OR int X[2][2] = {{3,4}, { 5,7} };	Declares an integer array named as A having 3 rows and 3 columns. 18 memory location to store 9 integer data <table><tr><td>X[0][0] 3</td><td>X[0][1] 4</td></tr><tr><td>X[1][0] 5</td><td>X[1][1] 7</td></tr></table>

Pointers: Pointers are variables that contain the address of other variable. Variables that hold memory addresses are called pointers.

SYNTAX : datatype *variable;

```
int a = 5;

int *b;   // b is a pointer variable

b = &a;   //address of a is assigned to b.( ie ) b points to a.
```

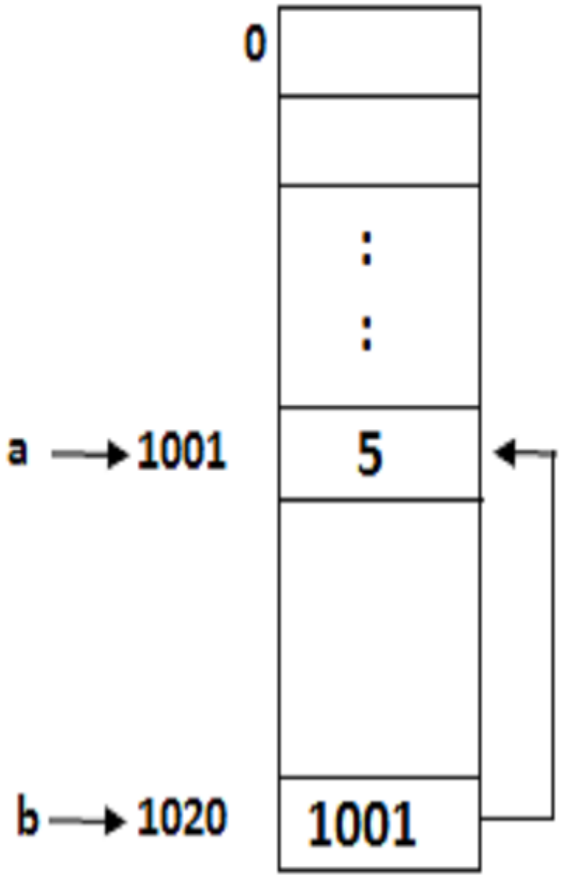
ADVANTAGES

(1) pointers increase the execution speed of the program

(2) pointers reduce the length and complexity of a program

(3) pointers enable us to access a variable and defined Outside the program

(4) pointers are used to pass information back and forth between a function and its reference point.



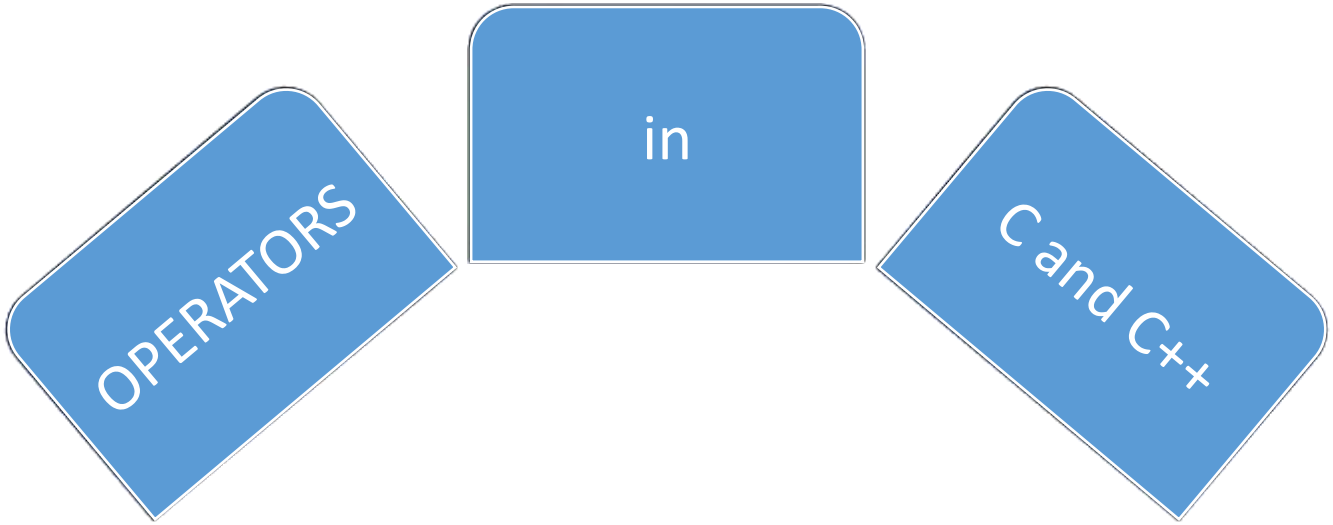
Reference variable:

- A reference variable is a name that acts as an alternative name for a previously defined variable.
- General form:

```
datatype  &  new name = already defined name;
```

- Where
  - ❖datatype - datatype
  - ❖& - reference operator
  - ❖new name - valid C++ variable name

STATEMENT NO	C++ STATEMENT	MEANING
1	int x;	[ ] x
2	x = 20;	[20] x
3	// defining reference variable int &a = x;	a [20] x
4	cout<<"\n value = "<<a;	value = 20
5	a++; cout<<"\n value = "<<x;	a [21] x value = 21


OPERATORS
in
C and C++

Arithmetic Operator

Relational Operator

Logical Operator

Assignment Operator

Increment/Decrement Operator

Conditional Operator

Bitwise Operator

Special Operator

Arithmetic Operators

- **Arithmetic Operators are used to perform numerical operations**

Operator	Meaning	Example
+	Addition	x=10 ; y=5 ; x+y = 15
-	Subtraction	x-y = 5
*	Multiplication	x*y = 50
/	Division	x/y = 2
%	Modulo Division	x%y = 0

Relational Operators

- **Relational operators are used to test the relationship between two variables or constant**

Operator	Meaning	Example	Result	value
<	Less Than	x=10 ; y=5 ; x < y	False	0
>	Greater Than	x > y	True	1
<=	Less than or Equal to	x <= y	False	0
>=	Greater than or equal to	x >= y	True	1
!=	Not Equal to	x != y	False	0
==	Equal To	x == y	False	0

Logical Operators

- Logical operators are used to combine two or more relational expressions.
- This operator is used to test more than one condition at a time.

Operator	Meaning	Example	Result
&&	Logical And	When x=9, y=5 (y>=5) && (x==9)	True
||	Logical Or	(x>=6) || (y=='a')	True
!	Logical Not	!(x>8)	False

Increment / Decrement Operators (++,--)

Operator	Meaning	Syntax	Example	Result
++	Unary Plus	Variablename++; (or) ++ variable name;	X=10; X++;	X=11
--	Unary Minus	Variablename--; (or) -- variable name;	X=10; X--;	X=9

The Assignment Operator

- **In C++ , the assignment operator(=) can be used for assigning a value to a variable**

Shorthand assignment operators

- Syntax

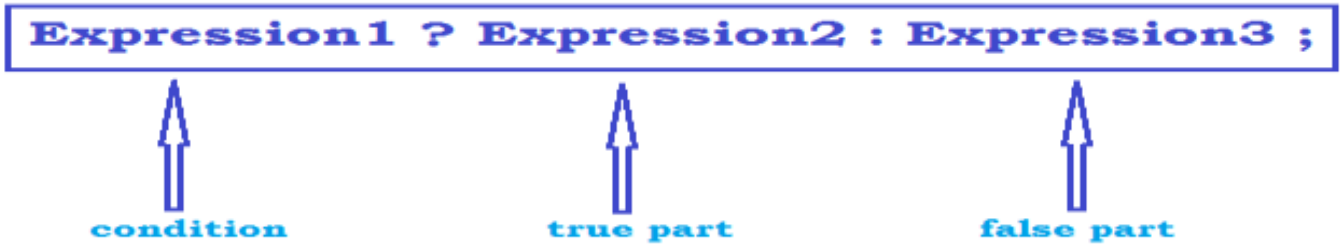
Variablename <arithmetic Operator>=Expression;

Simple Assignment Operators	Equivalent Shorthand Assignment Operators
x=x+1	x += 1
y=y-1	y =- 1
z=z*(x+y)	z *= (x+y)
y=y/(x+y)	y /= (x+y)
x=x%z	x %= z

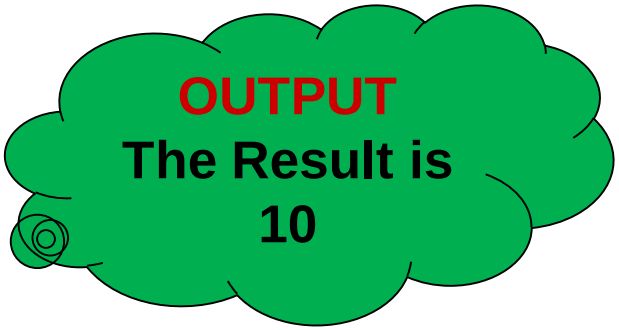
Conditional operator (?:)

Simple conditional operations can be carried out with the conditional operator(?:)

- General format :




```
#include <iostream.h>
void main()
{
int a=10 , b = 5, c;
c = (a>b)?a:c;
cout<<"\n c = "<<c;
}
```



Bitwise operators :

- **Used in applications which require manipulation of individual bits within a word of memory**

Operators	Meaning
~	One's Complement
<<	Left Shift
>>	Right Shift
&	Bitwise AND
!	Bitwise OR
^	Bitwise X-OR

- Special operators

Operators	Meaning	Example
,	Comma Operator	Z=(x=5,y=6,x+y)
*	Pointer indirection Operator	
&	Address Operator	scanf("%d",&no);
->	Arrow Operator in Structure	
.	Dot Operator in Structure	
#	String Sizing Operator (preprocessor)	#include <iostream.h>
##	Token passing Director	

```
/* Bitwise Operator Examples */
#include   <iostream.h>
#include   <conio.h>
void main()
{
int a,b,ans,and;
clrscr();
cout<<"\n Enter A Number";
cin>>a;      b=1;
ans = a&b;
cout<<"\n The Result of AND Operation with 1";
if(ans==0)
cout<<"\n Rightmost bit is OFF";
else
cout<<"\n Rightmost bit is ON";
ans=a/b;
cout<<"\n The Result of OR Operation with 1";
cout<<"\n Rightmost bit is ON and the result =  “<<ans;
getch();
}
```

Type Conversion

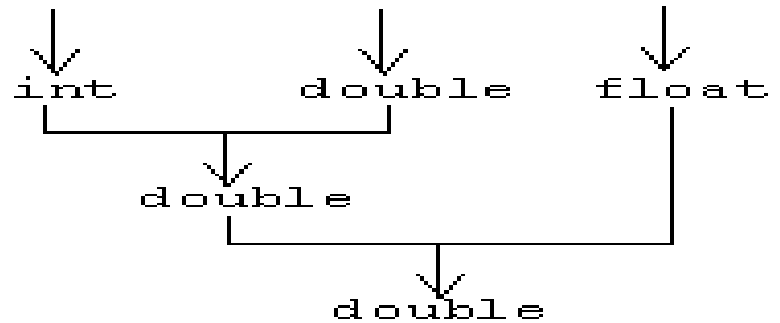
- **This is used to convert one data type to another data type. The automatic type conversions for evaluating an expression are given below -**

The automatic type conversion for evaluating an expression are tabulated below:

a) **char** and **short** are converted to **int** and **float** is converted to **double.**
b) If either operand is **double** , the other is converted to **double** , and the result is **double**.
c) If either operand is **long** , the other is converted to long , and the result is **double**.
d) If either operand is **unsigned** , the other is converted to **unsigned** , and the result is also **unsigned**.
e) Otherwise all that are left are the operands of type **int**, and the result is **int.**

- **For example,**

```
char ch;
int i;
float f;
double d;
result = (ch/i) + (f*d) - (f+i);
```



The sizeof operator

- **sizeof is a unary compile-time operator**

- **The use of the sizeof operator shall be clear from the following example -**

```
#include <iostream.h>
void main()
{
int a=10 ;
float  b;
cout<<"\n size of int  =  "<<sizeof(int);
cout<<"\n sizeof float  =  "<<sizeof(float);
}
```

Scope resolution operator.

- Scope resolution operator - **::**
- It is denoted by a pair of colon.
- It is used to define a member function outside the class.
- Refer a global variable form a function where the global variable is also declared as local.

Syntax : [class name]::variable or function name

Eg : To refer global variable

```
int a=50;
void main()  { int a=30; cout<<"\n local value = "<<a;
                 cout<<"\t global value = "<<::a;  }
```

Output: local value = 30 global value = 50

Purpose of goto statement.

goto statement is used to transfer the program control unconditionally from one point to another.

general form : **goto label;**

label is the name given to the point where the control has to be transferred.

general form : **Label: statements;**

Rules :

(a) Label is a valid C++ identifier. (b) A colon should follow label.

(c) No need to declare labels.

MEMBER DEREFERENCING OPERATOR:

They are used to accessthe member functions in a class through pointers. They are

(i) ::* - used to declare the member function in class as a pointer type.

(ii) * - used to access the member function in class using object name and a pointer .

(iii) ->* - used to access the member in class using pointer to the object and a pointerto that member .

MEMORY MANAGEMENT OPERATORS : They are used to manage the available memory in an efficient manner.

They are (i) new (ii) delete

(i) new : This operator is used to allocate memory space for any object.

SN	General form (or) syntax	Description
1	Pointer_variable = new data_type;	• new - keyword • data_type - valid c++ datatype • Allocates the memory space equal to the size of the data type and assign the starting address of the allocated memory to the pointer variable.
2	Pointer_variable = new data_type(value);	
3	Pointer_variable = new data_type(size);	

SN	EXAMPLE	DESCRIPTION
1	int *p; p = new char[40];	This statement allocates 40 bytes memory space. The starting address of the allocated memory is assigned to the poiter variable p.
2	Teacher *t1; t1 = new teacher;	Teacher is an object having 52 bytes. So when this statement is executed 52 bytes of memory space is allocated and the starting address is assigned to the pointer varible t1.
3	Teacher *t1; t1 = new teacher("Selva",35);	Allocates memory space and initialise it with the given data Selva and 35

(ii) delete: This operator is used to release the allocated memory space by using new operator.

Sno	General form (or) syntax	Description
1	delete Pointer_variable ;	• delete - keyword • Pointer_variable the variable used in new.
Example: int *P = new int[20]; ----		• releases the memory space of 40 bytes allocated to the object int

Decision Making statements :

- These statements are used to skip or execute a group of statements based on the result of some condition(s).
- Decision Making statements are
  - ❖**(1) simple if**
  - ❖**(2) if……else**
  - ❖**(3) switch…..case**

simple if : This statement is used to skip or execute one statement or a group of statements based on a particular condition.

- General form

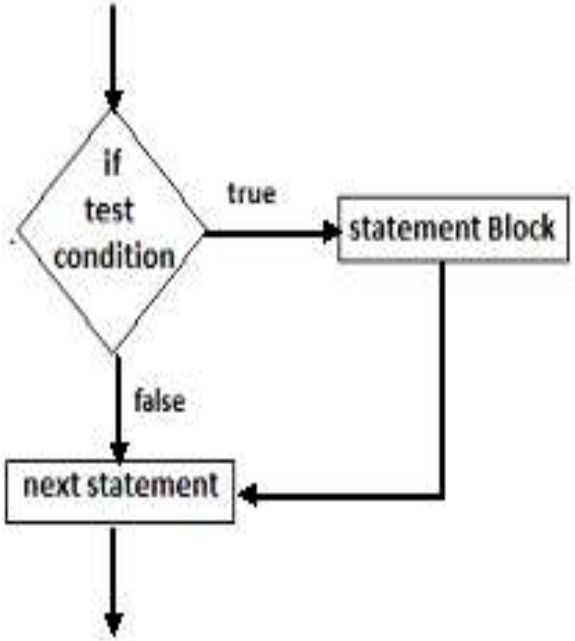
```
if (test condition)
{

 statement block;

}

next statement;
```

- Flowchart



- When this statement is executed , the computer first evaluates the value of the test condition .
- If the value is true, statement block and next statement are executed sequentially.
- If the value is false , statement block is skipped & execution starts from next statement.
- <u>Rules:</u>

  (1) The brackets around the test condition are must.

  (2) The test condition must be relational or logical expression.

  (3) Statement block is called body of the if statement & it contains one or two statements.

  (4) The opening and closing brackets { } are must if the statement block contains more than one statement. [else optional]

Example

```
#include <iostream.h>
void main()
{
 int m;
 cout<<"\n enter value :";
 cin>> m;
 if (m >0)
   { cout<<"\n"<<m<<"is positive"; }
 }
```

Execution Procedure:

Enter value : 10

Output:

10 is positive

Execution Procedure:

Enter value : -7

Output:

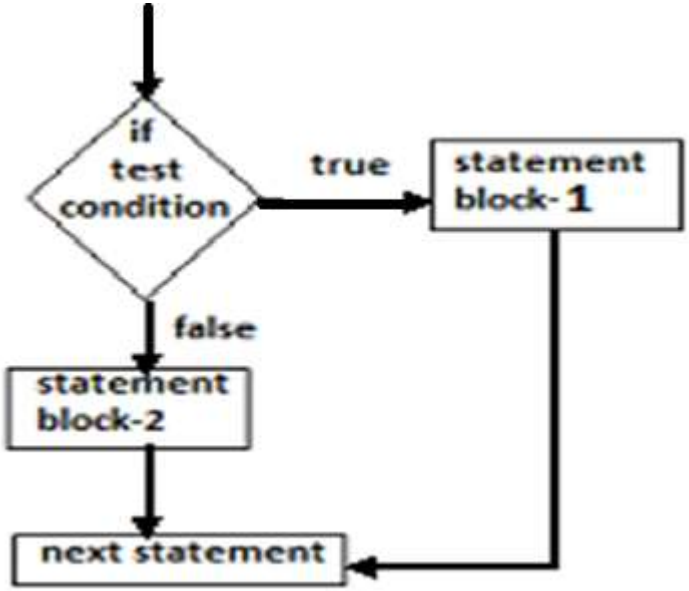
if else : Two way Branching Statement :

This statement is used to skip or execute one statement or A group of statements if the particular condition is true or other group of statements if the particular condition is false.

- General form

```
if (test condition)
{
statement block-1;
 }
else
{
statement block-2;
}
next statement;
```

- Flowchart



- When this statement is executed , the computer first evaluates the value of the test condition .
- If the value is true, statement block-1 and next statement are executed sequentially.
- If the value is false , statement block-2 and next statement are executed sequentially.

<u>Rules:</u>

(1) The brackets around the test condition are must.

(2) The test condition must be relational or logical expression.

(3) Statement block is called body of the if statement & it contains one or two statements.

(4) The opening and closing brackets { } are must if the statement block contains more than one statement between if…else

Example

```
#include <iostream.h>
void main()
{
 int m;
 cout<<"\n enter value :";
 cin>> m;
 if (m >0)
   { cout<<"\n"<<m<<"is positive"; }
 else
   { cout<<"\n"<<m<<"is negative"; }
}
```

Execution Procedure:

Enter value : 10

Output:

10 is positive

Execution Procedure:

Enter value : -7

Output:

-7 is negative

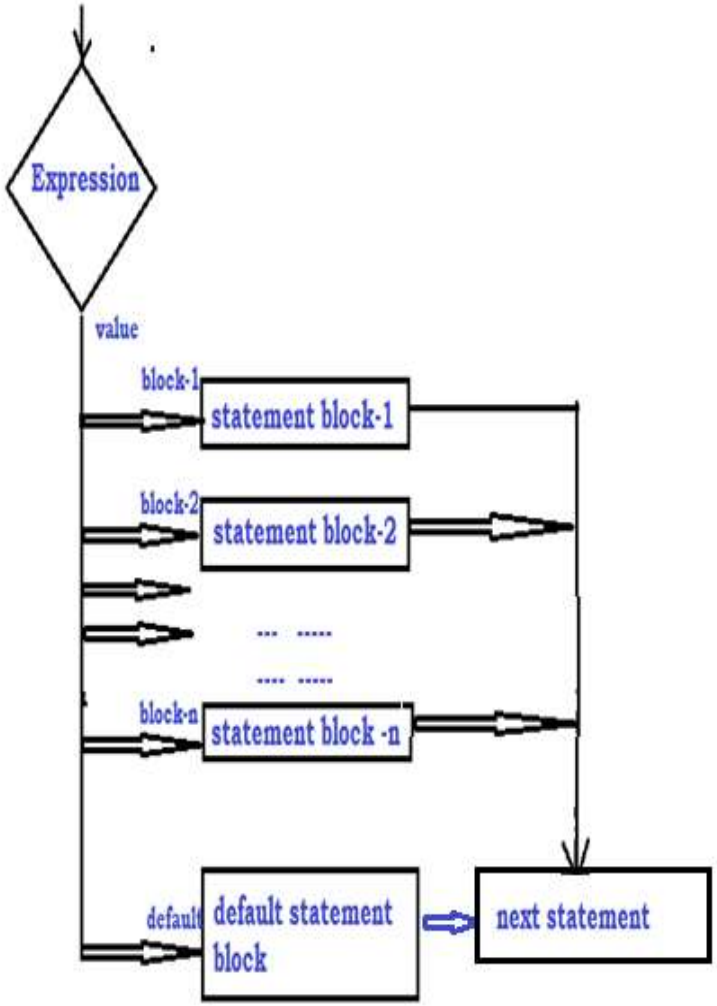
Switch...case : Multiway Branching statement :

This statement is an extension of if ....else statement. This permits any number of branches.

General form:

```
switch  (expression)
{
case label-1 :
            statement block-1;
            break;
case label-2 :
            statement block-2;
            break;
-----
-----
case label-n :
            statement block-n;
            break;
default :
            default statement;
            break;
}
next statement;
```

flow chart



- When this statement is executed , the computer first evaluates the value of the expression .
- If the value is successively compared with the case label-1,label-2,---label-n.
- If a case label matches with the value, the statement block associated with the case label is executed.
- Then the control is transferred to the next statement.
- If none of the case matches with the value, the default statement block is executed.
- Then the control is transferred to the next statement.

Rules:

(1) The brackets around the expression are must.

(2) The value of the expression should be an integer or a character constant.

(3) The body of the switch—case) should be enclosed within { } brackets.

Example:

```
#include <iostream.h>
void main()
{
  int m;
  cout<<"\n enter value :"; cin>> m;
  cout<<"\n m = "<<m;
  switch (m)
  {
  case 1: cout<<"\n One ";          break;
  case 2: cout<<"\n Two ";          break;
  case 3: cout<<"\n Three ";        break;
  case 4: cout<<"\n Four";          break;
  default: cout<<"\n Not match";
}
```

Execution Procedure: Enter value : 3	Output: Three
Execution Procedure: Enter value : 7	Output: Not match
Execution Procedure: Enter value : 1	Output: One
Execution Procedure: Enter value : 2	Output: Two

Example:

```
#include <iostream.h>
void main()
{
 int m;
 cout<<"\n enter value :"; cin>> m;
 cout<<"\n m = "<<m;
 switch (m)
 {
 case 1: cout<<"\n One ";          break;
 case 2: cout<<"\n Two ";          break;
 case 3: cout<<"\n Three ";        break;
 case 4: cout<<"\n Four";          break;
 default: cout<<"\n Not match";
}
```

Execution Procedure:	Output:
Enter value : 3	Three
Execution Procedure:	Output:
Enter value : 7	Not match
Execution Procedure:	Output:
Enter value : 1	One
Execution Procedure:	Output:
Enter value : 2	Two

Looping Statements:

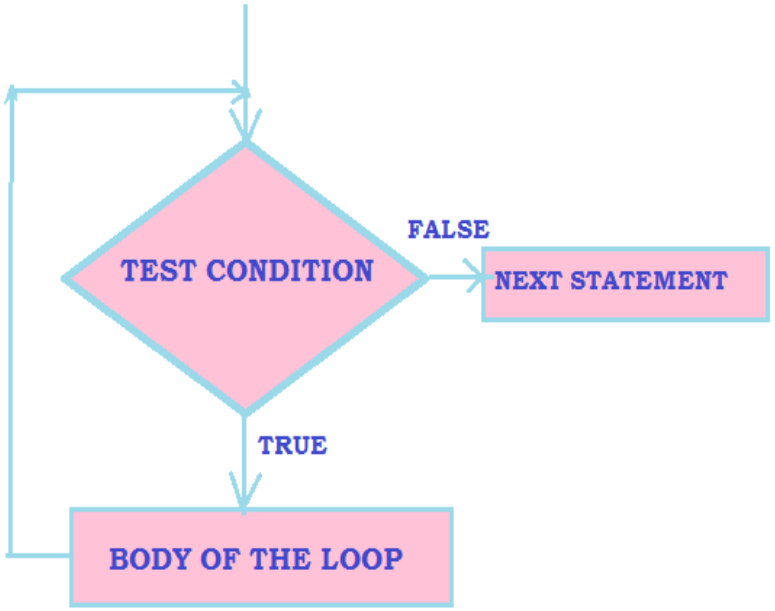
- Looping Statements are used to execute a group of statements repeatedly until some condition is satisfied.
- The looping statements are
  - ❖(1) while statement
  - ❖(2) do ... while statement
  - ❖(3) for statement

Entry Controlled Loop :

- **while :** This statement is simple loop statement.

- GENERAL FORM FLOWCHART

```
While (test condition)
{
 Body of the loop;
 }
next statement;
```



- When this statement is executed , the computer first evaluates the value of the test condition .
- If the value is false , statement block is skipped & execution starts from next statement.
- If the value is true, then the body of the loop is executed repeatedly until the test condition becomes false.
- <u>Rules:</u>

  (1) The test condition must be relational or logical expression enclosed within brackets.

  (2) Statement block is called body of the loop & it contains one or two statements.

  (3) The opening and closing brackets { } are must if the loop body contains more than one statement.

Example

```
#include <iostream.h>
void main()
{
  int m;  int i = 1;
 cout<<"\n enter value :"; cin>> m;
  while (i <= m)
  {  cout<<"\nBest wishes "; i++; }
}
```

Execution Procedure:

Enter value : 4

Output:

Best wishes
Best wishes
Best wishes
Best wishes

Execution Procedure:

Enter value : 2

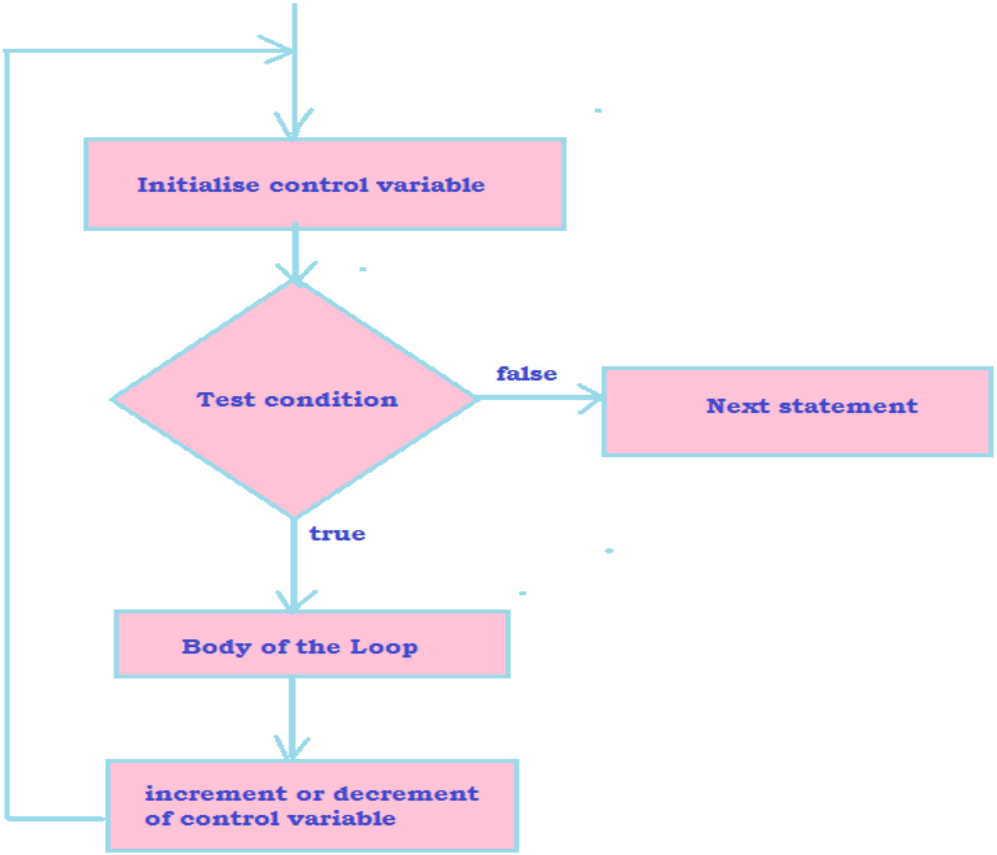
Output:

Best wishes
Best wishes

Entry Controlled Loop :

- **<u>for</u>** :
- This statement is simple loop statement.
- This loop is used to execute a statement or a group of statements repeatedly for a known number of times.

```
for(initialize; test condition; step value)
{
Body of the loop;
}
next statement;
```



- When this statement is executed , the value of the control variable is initialised and tested with the test condition.
- If the test condition is true, then the body of the loop is executed. Then the value of the control variable is incremented or decremented.
- when the test condition becomes false ,the control is transferred to the next statement.
- <u>Rules:</u>

  (1) Initialise is used to set initial values to loop control variable.

  (2) The test condition is used to check the control variable. According to this, the loop is executed or not.

  (3) Step value (Increment / decrement) is used to change the value of the control variable.

  (4) The opening and closing brackets { } are must if the loop body contains more than one statement.

Example

```
#include <iostream.h>
void main()
{
  int m;  int i ;
 cout<<"\n enter value :"; cin>> m;
  for( i=1; i <= m ; i++)
  {  cout<<"\nBest wishes "; }
}
```

Execution Procedure:

Enter value : 4

Output:

Best wishes
Best wishes
Best wishes
Best wishes

Execution Procedure:

Enter value : 2

Output:

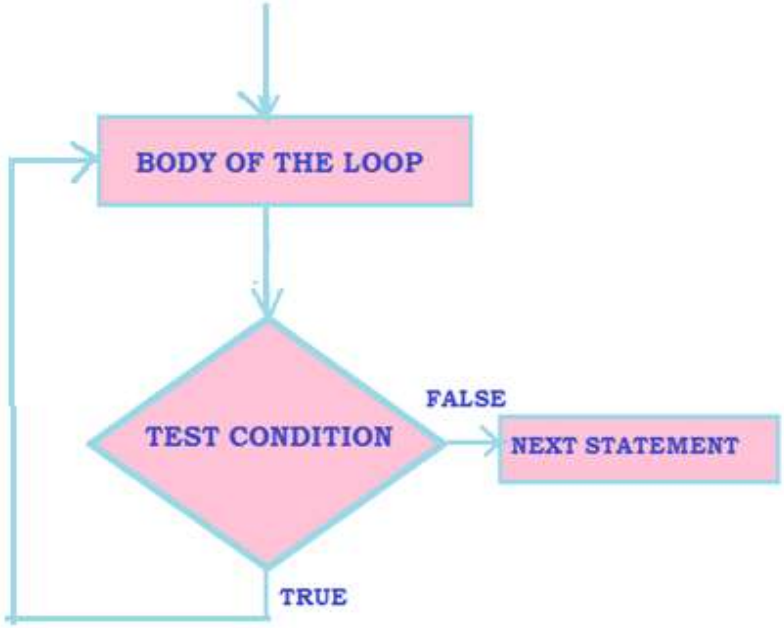
Best wishes
Best wishes

Exit Controlled Loop :

- **do … while :**

- GENERAL FORM FLOWCHART

```
do
{
 Body of the loop;
 } while (test condition);
next statement;
```



- When this statement is executed , the body of the loop is executed first.
- Then the test condition is evaluated.
- If the value is false , statement block is skipped & execution starts from next statement.
- If the value is true, then the body of the loop is executed repeatedly until the test condition becomes false.
- <u>Rules:</u>

  (1) The test condition must be relational or logical expression enclosed within brackets.

  (2) Statement block is called body of the loop & it contains one or two statements.

  (3) The opening and closing brackets {} are must if the loop body contains more than one statement.

  (4) while statement is placed after the body of the loop.

  (5) Whether the test condition is true or false, the body of the loop is executed atleast once.

Example

```
#include <iostream.h>
void main()
{
  int m;  int i = 1;
  cout<<"\n enter value :";
  cin>> m;
  do
  {
   cout<<"\nBest wishes ";
   i++;
  }  while (i <= m) ;
}
```

Execution Procedure:
Enter value : 4

Output:
Best wishes
Best wishes
Best wishes
Best wishes

Execution Procedure:
Enter value : 2

Output:
Best wishes
Best wishes

THANK YOU

STAFF : Mrs.M.ESWARI