

# C++

---

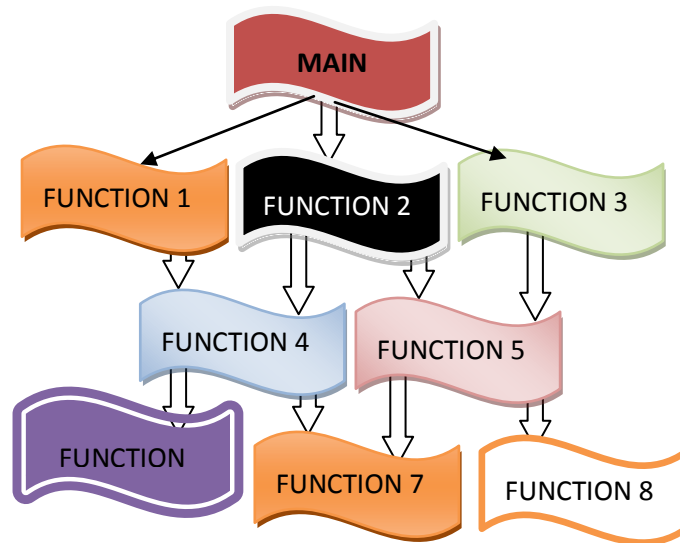
**DARSHAN.M**

**SHIVA CHETHAN.R.K**

**MUKUNDA.C.A**

## PROCEDURE ORIENTED PROGRAMMING

The high level languages such as COBOL, Pascal, FORTRAN, c, are commonly known as Procedure oriented Programming. In procedure oriented approach the problem is viewed as a sequence of things to be done a number of functions are written to a complete specific task. The following diagram shows the program structure of POP. The technique of Hierarchical decomposition has been used to specify the task to be completed for solving a problem.

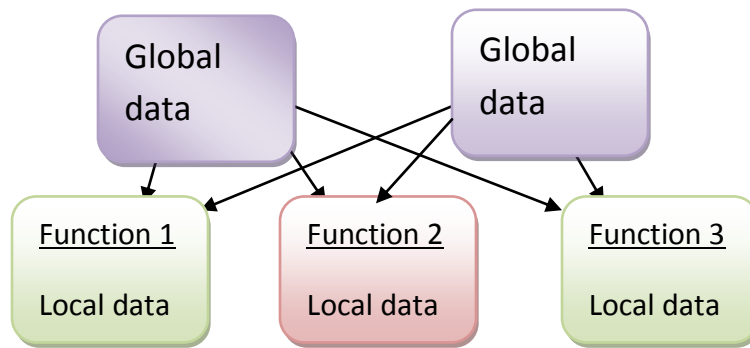


Procedure oriented programming consists of list of instruction for computer to follow and organizing these instruction known as functions.

In a multifunction program many important data items are placed as global so that they may be accessed by all the function. Each function may have its more local data. The global data are shared by all the functions. It is very difficult to identify what data is used by which function. Functions are action oriented.

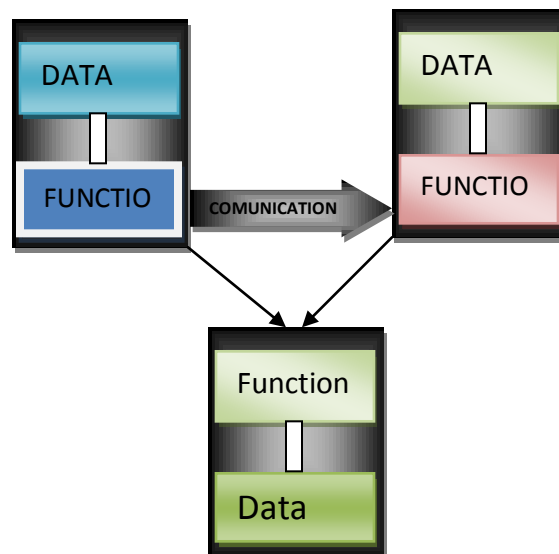
Characteristics:-

1. Large programs are divided into smaller programs known as functions.
2. Most of the functions share global data.
3. Data move openly around the system from function to function.
4. Function transforms data from one form to another.
5. It approaches top down programming



### **OBJECT ORIENTED PROGRAMMING:-**

The major motivation of object oriented approach is to be remove some of the draw backs encountered in POP. In OOPS treats data as a critical element in program the program development and doesn't allow it to flow freely around the system. It binds the data more closely to the function that operate on it and protect it from outside function OOPS allows decomposition of a problem with a number of entities called object. The following diagram shows the organization of data and functions in OOPS. The data of an object can be accessed by the function associated with that object however functions of one object can access the function of other object.



Characteristics:-

1. Programs are divided into object.
2. Data structures are designed such that they characterize the object.
3. Functions that operate on the data of an object are bound together in the data structure.
4. Data is hidden and can't be accessed by external function.
5. Objects may communicate with each other through functions.
6. New data and functions can be easily added whenever necessary.
7. It approaches bottom up programming.

### **BASIC CONCEPT OF OOPS:**

Following are the concepts used extensively in oops.

1. Class
2. Objects
3. Data abstraction and encryption
4. Inheritance
5. Polymorphism
6. Dynamic binding
7. Message passing

Class:-

Entire set of data and code of an object can be made a user defined data type with the help of class. Class is a user defined state and behavior of an object. A class is a way to find the data and associated code of an object. Each object is associated with data type of class.

e.g.: class student

```
{ Private:
```

```
    Int regno;
```

```
    Char name [10];
```

```
Public:
```

```
    Void getch ();    Void showresult(); };
```

### Object:-

Objects are the basic runtime entity in oops system. They may represent person, place, bank account, and set of data item that the program has to handle. Programming problem is analyzed in terms of object and the nature of communicate between them

Objects are the instance of class or variable of class when program is executed object interact by sending message to another. Each object contains data and code to manipulate the data.

e.g.: student s1, s2;

In above example s1 and s2 are the object of student type.

### Data abstraction and encapsulation:-

The wrapping up of data and function in a single unit is known as encapsulation. Data is not accessible to outside world and only those functions which are wrapping in the class can access it. Function provides interface between objects data and the program. This insulation of data from direct access from the program is called data hiding or information hiding.

Abstraction refers to cat of representing essential future without including background. Class use list of attribute and function to operate on these attribute. The attribute are called data member. Because they hold the information. The functions that operate on these data are called member function. The class names are act as built in type. Therefore it is known abstract data type.

### Inheritance:-

It is process by which the objects of one class acquire the properties of another class. It supports the concepts of hierarchical classification. The principle behind this class is that each derived class shares common characteristic from the class which is derived.

Concept of inheritance provides of the idea of reusability. This means that we can additional features to an existing without modification it. This is possible by deriving new class by existing one. The new class will have combined futures of both the class. The new class is called subclass/derived class/child class. Existing class is called base class/super class/parent class.

```
e.g.: class point
{ Public:
    Int x,y;
};
Class point3d: point
{ int z;
};
```

parent class

child class

### Polymorphism:-

It is important oops concept. It means ability to take more than one form. An operation can exhibit different behavior at different instance. There are two form of polymorphism.

- I. Function over loading.
- II. Operator over loading.

Using single function name to perform different type of tasks known function over loading.

e.g.: Void add();

Void add (int a, int b);

Void add(float a, float b);

Void add(char a.char b);

An operator may exhibit different behavior in different instants. It is known as operator over loading. Behavior depends up on types of data used in the operations.

e.g.: 5 + 6

2.5 + 3.5

'a' + 'b'

"aig" + "cta"

### Dynamic binding:-

Binding refers to the linking of procedure call to the code to be executed in response to call. Dynamic binding means that code associated with a given call is not known until the time of call in the run time. It is associated with polymorphism and inheritance.

### Message passing:-

An oops consists set of objects that communicate with each other. The process of programming an object oriented language involves

1. Creating class – i.e. define object.
2. Creating object from class definition.
3. Establishing communication among object.

Objects communication with one another by sending receiving information. A message passing for an object require for execution of a procedure message passing involves specifies name of object, name of function and information to be send.

Eg:-      class point  
{  
    Private;  
int x;  
int y;  
Public:  
Void get\_point ( int a, int b)  
    {  
        x=a,y=b;  
        } };  
point p;  
P. getpoint(10,20)

### **BENEFITS OF OOPS:-**

OOPS after several benefits to both the program designer and the user objects orientation program provide solution of many problems with the development and quality of software products. Following are the benefits of OOPS.

- ♥ Through inheritance we can eliminate redundant code and extend the code of existing class.
- ♥ The principle of data hiding helps the programmer to build secure program.
- ♥ Object oriented systems can be easily upgraded from small to large system.
- ♥ Message passing techniques for communication between objects makes the interface between with external systems.
- ♥ It is easy to partition the work in a project based on object.
- ♥ We can build program from the stand working modules that communicate with one another.
- ♥ It is possible to map objects in the problem in the program.
- ♥ Software complexity can be easily managed.

### **OBJECT ORIENTED LANGUAGE:-**

The programming language includes different techniques such as structured program, modular <fun> program, top down approach and bottom up approach object oriented concept can be implemented by using programming languages. The languages supports several of OOPS concept depending on the features into 2 categories.

- ψ Object based program language.
- ψ Object oriented program language

Object based program is the style of programming that primarily supports encapsulation and program identify major features of OBP are

- ψ Data encapsulation.
- ψ Data hiding and access mechanism.
- ψ Automatic initialization and deleting object.
- ψ Polymorphism



The object based program language are C, PASCAL, COBOL, FORTRAN, ADA ...etc.

Object oriented program includes all of object based program features along with 2 additional features such as inheritance and dynamic binding. These features supported by C++, SMALL TALK, SIMULA, OBJECT PASCAL, and JAVA

### **Application of OOPS: -**

The object oriented languages are appeared in all types of software application. The most popular application of objects oriented programs are

- § Real time systems.
- § Object oriented database.
- § Artificial intelligence and expert systems.
- § Hyper text and hyper media.
- § Neural networks.
- § Office automation system.
- § Simulation and modeling system.
- § Cad cam system.

## C++






### History of C++:-

C++ is an object oriented programming language it was developed by BJARNE STROUSTRUP at BELL Laboratories in U.S.A. in 1980, the stroustrup combine the powerful features of c language and object oriented features from simula into the new language. Initially new language is called C with classes, however in 1983 the name was changed to C++. The idea of C++ comes from the c increment operator C++ suggesting that C++ is an extension of C with major addition of class features. In 1990 the number of improvement and changes are made to the new language 1997 ANSI/ISO standard community approve these changes.

The improvement facilities of C++ are classes inheritance function overloading and operator overloading. This features allows the programmer to built large programs with clarity, extensibility and easy of maintenance.

### TOKENS OF C++:-

Tokens are the smallest individual units in a program depends on their function the tokens are classified into 5 types

-  Keywords
-  Identifiers
-  Literals
-  Operators
-  Separators

### Keywords:-

Keywords are the special words having specific meaning those meaning are not alter by the user. These are the predefined words and gives proper instruction to the compiler. There also known as reserve words. All the keywords are specified in lower case only. All the keywords of C are supporte4d in C++. Additionally keywords have been added in C++ for the features of object oriented. The C++ keywords are class, private, protected, inline, friend, operator, new, delete, l templates, this, through, try, catch, virtual,

### Identifiers:-

Identifiers are the name to identify different elements of the program such as variables array's string pointer structures, functions, classes, ..Etc.

The following rules must be followed to mention valid identifiers. It consists of alphabets (A to Z & a to z), number (0-9) and special symbol underscore (\_)

- 🕸 They can't begin with numbers.
- 🕸 Keywords are not allowed.
- 🕸 There is no successive underscore.
- 🕸 Identifiers are case sensitive.
- 🕸 Spaces and dots are not allowed.
- 🕸 Any length.

Ex:-y2k, total\_marks, y2009,\_2009, Rs\_100, \_2009\_

### Literals:-

Literals are the constants. Constants are the data values whose values must be remains unchanged during the program execution. C++ provides 4 types of constants

1. Integers  
Decimal: - (0-9) 36,-78,256,0.  
Octal :- (0-7) 036,0256,00  
Hexadecimal :- Ox36. Ox20,OxBE (0-9.A-F),OxO.
2. Floating point :- 2736,25,-275, .358, 0.0
3. Char :- 'A' '\$' '2009'
4. String :- "AIG", "S", "2009", " " .

### Separator:-

Separators are special type of entities are used to organize the different code of the data. These are the special character separates program elements. The most commonly used separators are semicolon, colon symbol, dot, double quotes, tab, square bracket, parenthesis, brasses.

## **OPERATOR:-**

Operator is an entity which performs operation on the given operands are the data values where the operator acts. It can be either value or expression.

Based on number of operand's must be used in operator the operators are classified into 3 types

1. Unary operator
2. Binary operator
3. Ternary operator

Unary operator:-

This operator requires only one operand to perform an operation.

1. unary minus  $-( )$

This operator negates the give operand value eg:-  $-(5)=-5, -(-9)=9$

2. Unary plus  $+( )$

This operator perform reverse function of unary minus.  $+(-5)=-5$

3. Increment operator  $(++)$ :-

This operator increase the value of an operand by one depends on the operator must be specified on the operand there are two forms.

1. Post increment  $(a++)$
2. Pre increment  $(++a)$

Ex: - 1.  $A=5$   $b=a++$ ,

3. Decrement operator  $(--)$ :-

This operator decreases the value of operand by 1 depends on the operator must be specified on the operand. It is of two forms.

1. Pre decrement operator  $(--a)$
2. Post decrement operator  $(a--)$

Binary operator:-

These operators review at least two operands to perform an operation. The binary operators are:-

 Arithmetic operator

 Relational operator

- 🐞 Logical operator
- 🐞 Assignment operator
- 🐞 Bitwise operator

## 1. Arithmetic operator

These operators are used to perform same basic arithmetic operation on given data. The arithmetic operators are `+`, `-`, `*`, `/`, `%` (only integer values)

## 2. Relational operator:-

The relational operator are used to compare values of 2 operand depends on the comparison gives the Boolean value either true or false. The relational operators are

`>`, `<`, `>=`, `<=`, `==`, `!=`

## 3. Logical operator:-

The logical operator are used to combine multiple condition depends on the condition must be satisfied. Then it gives the result either true or false. The logical operators are

1. Logical AND (`&&`):- It returns true if all the conditions are true, when returns false if any one of the condition is false.

Condition 1	Condition 2	Result
T	T	T
T	F	F
F	T	F
F	F	F

2. Logical OR (`||`):- It returns true when any one of condition is true and returns false when all the conditions are false.

Result	Condition 1	Condition 2
T	T	T
T	T	F
T	F	T
F	F	F

3. Logical NOT (!):- This operator returns true if the condition is false & returns false if the condition is true.

Condition	Result
F	T
T	F

#### 4. Assignment Operator:-

Variable=value/expression

The assignment operator assigns the R.H.S. value to L.H.S. The R.H.S. can be either value or expression but L.H.S must be a variable. The R.H.S. and an expression, then it is evaluated first and is assign to L.H.S.

A=b, A=5, A=a+b,

The assignment operator combines with arithmetic operator to reduce complexity of the expression such operators are known as short hand assignment operator.

A+=b → A=A+b,

a-=10 → a=a-10,

a\*=b → a=a\*b,

a/=10 → a=a/10

Chained Assignment: - a=b=c=0

Compound Assignment: - a= (b=30) +50

#### 5. Bitwise operator:-

The bitwise operator are used to perform the manipulation of data are 0 & 1. These operators' takes the operands must be either integer or character.

The bitwise operators are

1. Compliment operator(~)
2. Bitwise AND(&)
3. Bitwise OR (!)
4. Bitwise XOR(^)
5. Right shift operator(>>)
6. Left shift operator(<<)

Ternary operator: -

This operator requires at least 3 operand's to perform an operation.

C++ supports only one ternary operator called conditional operator (?).

(Condition)? True expression: false expression

This operator checks the given condition whether it is true or false. If the condition is true then it performs true expression else it performs false expression.

(Condition)? True expression: False expression

( a>b) ? a:b →b

( 1)? a:b →a

( -100)? a:b →a

( a-b)? a:b →b

Expression: -

Expression is a combination of operand's & the operator. Operands are the data values it can be either variable or constants. The operator can be either arithmetic, relational, logical, and bitwise or assignment.

The mathematical expression & C++ expression are different. The representation of C expression is also similar in C++.

$A^2 \rightarrow a * a$ ,  $ab \rightarrow a * b$ ,  $(a+b)^2 \rightarrow (a+b)*(a+b)$ ,  $\frac{a+b}{c} \rightarrow (a+b)/c$

**DATA TYPES:-**

Data represents raw facts of an entity. Data types specify the actual presentation of the given data. It specifies storage structures and access methods of the data. C++ provides 3 types of data types.

1. Fundamental data types
2. Derived data types
3. User defined data types

**Fundamental data types:-**

These are the basic data types under operated by the machine instruction directly. It represents the basic data of the program. There also known as primitive data types or intrinsic data type. Every data types having their own size and memory.

	Bytes	Range
Int	2	-32768 to 32767
Short	2	-31760 to +32767
Signed int	2	-32760 to 32767
Unsigned int	2	0 -65535
Long int	4	-2147483648 to 2147483647
Signed long int	4	-2147483648 to 2147483647
Unsigned long int	4	0 to 4294967295
Float	4	34e-38 to 3.4e+38
Double	8	1.7e-308 to



		1.7e+308	
Long double	10	17e-4932 to 1.7e+4932	
Char	10	-128 to +127	
Signed char	10	-128 to +127	
Unsigned char	10	0 – 255	
Void	0		

C++ introduce the special type called void used to represent empty value for the list of arguments to a function & also specifies return type of a function when it is not returning any value.

Derived data types:-

These are the highly sophisticated data types under the derived from fundamental data type. These are homogeneous & heterogonous data elements.

Ex:-array's, string, functions, pointers references

User defined data types:-

These data types are created and constructed by the user these are helpful in case of project based application.

Ex: - structure, union, classes, enumeration.

## **VARIABLE DECLARATION:-**

Variables are the data and it may take any value during execution of the program. Before using any variable. First we must declare the variables. The variable declaration statement specifies name type, size, and scope of the variables.

Syntax: - <data type> var1, var2;

int a, b;

float x;

C language requires whole the variable to be defined at the beginning of a scope. C++ allows the definition of the variables anywhere in the scope this means that a variable can be declare at right place of its 1<sup>st</sup> use this makes the program much easier to write and reduce the errors.

The only disadvantage of this type declaration is that we can't see all the variable used in a scope at glance.

```
For(int i=1; i<=10; ++i)

{   int r;
    }
```

### **VARIABLE INITIALIZATION: -**

Initialization is a process to assign an initial value for the variable at the time of declaration i.e., the compiler will assign the value for the variable at the time of compilation such initialization is known as static initialization

```
<data types> var = variable
int a =75;
float b=3.5
char c = ' a '
```

Where the value must be a constant and same type as given data type.

Ex:- int a=75

C++ allows initialization of the variable at run time this is referred as dynamic initialization in C++ a variable can be initialized at run time using expression at the place of declaration.

```
<Data types> var =expression
int a = b+c
float avg =sum/4
```

Dynamic initialization is extensively used in object oriented programming.

## INPUT / OUTPUT STATEMENTS:-

The I/O statements are used to send & receive the data “to” or “from” the program.

C++ object oriented programming language & it's for performs I/O operation through objects

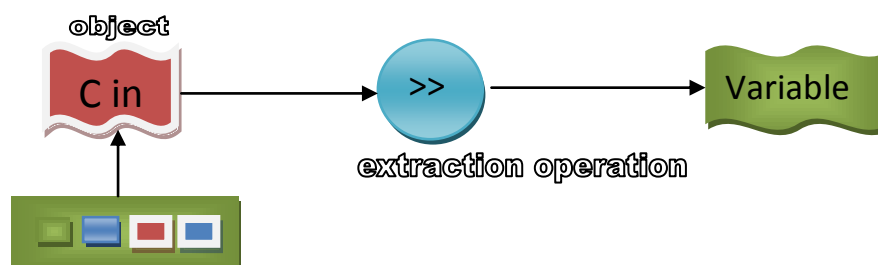
Input statement: -

Syntax:-`Cin>> variable`

Where `Cin` predefined Input strep Object here keyboard represents the standard Input string Object. This statement causes the program to wait for the user Input and that value must be assigned to the right side variable.

The operator right shift is known as extraction or get from operator it extracts the data values from the keyboard and assign the value to the variable

`Cin>>a`



We can use more than one extraction operator in a single statement known as input cascading. The values are assigned from left to right.

Ex:`Cin>>a>>b>>c>>name.`

<code>Cin&gt;&gt;a;</code>	<code>Cin&gt;&gt;a</code>
<code>Cin&gt;&gt;b;</code>	<code>&gt;&gt;b</code>
<code>Cin&gt;&gt;c;</code>	<code>&gt;&gt;c;</code>

Output operator: -

The only output operator is

`Cout<<variable`

This statement displays the result for information on the screen. Here the identifier `Cout` is a predefined object that represents standard output stream here the standard O/P represents the screen. It is also possible to redirect the O/P to other output devices.

The operator left shift (<<) is called the insertion or put to operator it inserts the content of the variable or its right to the object on its left. The object output as simple interface it performs the same operation as printf in C language

```
Cout<<x;
```

```
Cout<<welcome to C++;
```

```
Cout<<a<<b<<c;
```



It is possible to send multiple values in a single Cout object. The multiple use of insertion operator in one statement is called O/P cascading.

```
Cout<<"\nsum="<<s
```

```
<<"\n average="<<avg
```

```
Cout<<a<<b<<c;
```

### **ESCAPE SEQUENCE:**

These are the special character to display the results in proper format. These characters are placed only in output statement. They are begins with back slash. Therefore they are known as back slash character constant

\n-new line character

\t-horizontal tab

\b-back space

\v-vertical tab

\'-single quotation

\"-double quotation

\\- back slash

\a-alert/beep

\r-carriage return

\f-form feed

### **IOSTREAM FILE:-**

The I/O stream is a library file and must be included in a program whenever using stream object using the preprocessor statement.

`#include<iostream.h>` this statement causes the preprocessor to add the contents I/O string file to the program. It contains the declaration of the identifiers `Cout` & `Cin` and the operator insertion & extraction.

The header file I/O stream to be included at the beginning of all programs that we input output statement. We must include the appropriate header file depending on the contents of program and implementation.

Comments:-

Comments are the special type of statement are used to enter some description about the program, author name and other details these statements are never executed by the compiler.

C++ supports two types of comments

1. Single line comment: -

The comments start with double slash i.e., forward slash (`//`) are known as single line comment. The comment is start with double slash & it's terminated at the end of the line. A comment may start anywhere in the line & his follows till the end of the line. Note that there is no closing symbol.

2. Multiple comment: -

A comment start with `/*` & end with `*/` suitable multiline comments. The statements b/w these symbols are ignored by the compiler. This statement is also appearing to make part of the statement as a comment.

Ex:- `for(i=1;i<=10;/*repeat 10 times*/++i)`

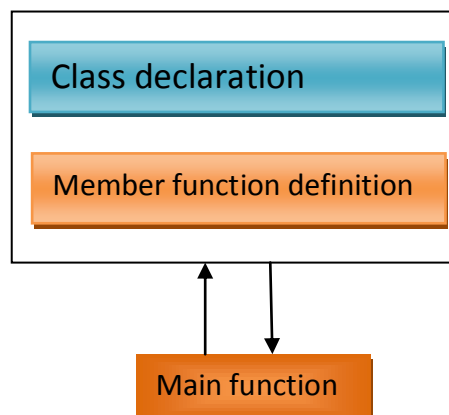
**STRUCTURE OF C++ PROGRAM: -**

C++ program contains 4 sections. These sections are placed in separate code file & then compiled independently.

DOCUMENTATION
PREPROCESSOR STATEMENT
CLASS DEFENATION
MEMBER FUNCTION DEFENATION
MAIN FUNCTION

In general a program to be organized into 3 separate files. The class declaration are placed in the header files & the definition of member function to another file finally the main program that uses the class is placed in 3<sup>rd</sup> file which include previous 2 files.

This approach is based in the concept of client server model the class definition including member function acts as a server that provides services to the main program known as client.

**SYMBOLIC CONSTANT OR CONSTANT QUALIFIER:-**

The variables are the data name & whose values must be changed during the execution of a program. In C++ we must define the symbol constant meanse that the variables are the symbol which represent a fixed value. Any value declared as const can't be modified by the programmer in any way. The modifier const allows to crate name constant and symbolic constant. These are just like a variable except that there values cannot be change.

Syntax:-

Const<type> var=value;

Const float PI=3.14;

Where const is a key word, type specifies the data type of the variable.

C++ requires a named constant to be initialized if no initialized given if initializes the value zero if no type is given by default it take int

Ex: Const float PI=3.14;  
Const int size=10;

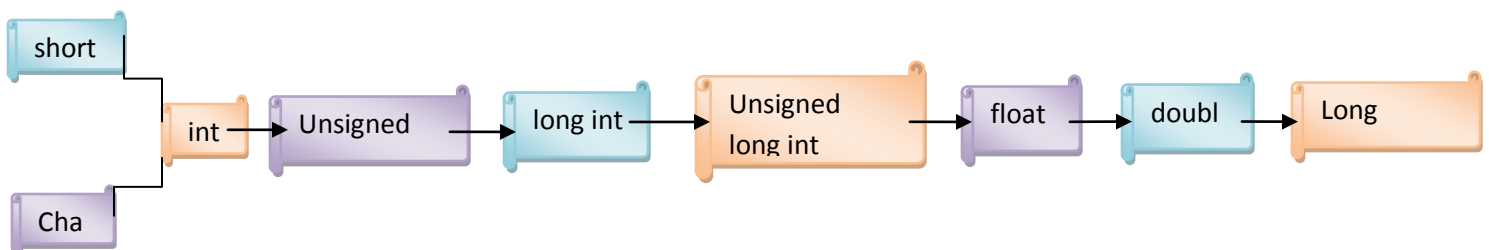
### **TYPE CONVERSION: -**

The type conversion is a process to convert the data from one to another type when evaluating mixed type of expression. C++ provides 2 type of conversion

1. Automatic Or Implicit conversion.
2. Type costing Or Explicit conversion.

#### **1. Automatic Conversion:-**

When evaluating mixed type of expression the C++ performs the conversion of the data automatically. This process is known as automatic or implicit conversion. The compiler converts the data from one type to another using the rule that smaller type is to be converted higher type. This means that the destination is having enough space to store the source values. The following model show the type conversion process.



### **TYPE CASTING:-**

C++ allows the explicit type conversion of a variable or expression by using type cast operator. The process of such conversion is known as casting a value. The general format of type caste is

<type>var=(type cast)value/expression;

where type caste is one of the standard data type & is same has destination type.

The expression may be constant variable some other functions.

C++ supports new caste operator

1. Const\_ caste.
2. Static\_ caste.
3. Dynamic\_ caste.
4. Reinterpret\_ caste.

### **SCOPE RESOLUTION OPERATOR:-**

C++ is a block structured language we can use the same variable name in different block having different meaning. The scope of the variable extends only from the point of its declaration until the end of the block containing the declaration a variable declared inside a block is local to the block.

Note the declaration of inner block hides the declaration of same variable in outer block. In c languages the global version of the variable can't be access with in inner block but C++ provides a new operator called scope resolution operator :: used to uncover the hide variable.

:: variable

This operator is used to the global versions of variable in inner block.

### **OPERATORS IN C++:-**

C++ as rich set of operators it supports all the C operators in addition C++ introduces some new operator along with insertion(<<) exertion(>>)

Scope resolution operator (:: )

Pointer to member declarative operator (::\*)

Pointer to member access operator (→\*)

Pointer to member access operator(.\*)

Delete memory release operator (delete)

memory allocation operator (new)

line feed operator(endle)

field width operator (setw)

### **REFERENCE VARIABLE:-**

C++ provides a new kind of variable known as reference variables reference variable provides an alias name for previously defined variables. Both reference variables and existing variable can use interchangeably. To represent a reference variable in a program by specifying an '&' symbol and is created as follows.



Syntax:- <type> &Ref-variable=variable;

where reference variable must be initialized at the time of declaration. This establishes the correspondence between the reference and the data objects which it needs. It is important to note that the initialization of referential variable is completely different from assignment to it. Here the '&' symbol is not an address operator. C++ assigns an additional meaning the symbol &.

```
Ex:- int sum=100
      int & total=sum;
      char &line="\n";
      int a[10];
      int &x=a[10];
```

### **DYNAMIC MEMORY ALLOCATION:-**

The Dynamic memory allocation is a process to allocate required amount of memory dynamically & also de allocate a memory at run time. When we use dynamic memory allocation technique, it is not known in advance how much memory space is needed. C++ supports the two unary operator new & delete that performs the task of allocating & de-allocating the memory in efficient manner. They are also known as free store operator.

1. New Operator:- The new operator can be used to create an object of any type dynamically it allocates the memory allocation for required type during execution.

The following format shows the allocation of a single object at run time.

Syntax:- Pointer-var=new<type>;

where pointer variable is a pointer of specified type which holds the address of allocating memory. The new operator allocates memory for a data object of specified type and it returns the address of a very first memory location. The data type may be either basic or user defined.

```
Int *iptr=new int;
int *fptr=new float;
```

It is also possible to initialize the memory using new operator by specifying the value with in parenthesis dynamically.

```
var=new<type>(value)
*iptr=new int(5);
```

Syntax:-    Pointer  
              int

The new operator can also be used to create memory locations for any data types including user defined data type and derived data types such as array's structures and classes.

The following format shows the allocation for one dimensional array

syntax:- `pointer var=new<type>[size];`

where size specifies number element in the array and it must be of type +ve integer or unsigned integer & specified with in square bracket.

`Int *ptr=new int[10];`

when creating multidimensional array with new operator. All the array size must be specified with

`int *ptr= new int[10];`

`int *ptr=new int[10][3];`

### **DELETE OPERATOR:-**

The object is no longer needed it is destroy to remove space for reuse this is done with the help of delete operator. This operator de-allocates the memory for an object which is allocated dynamically.

pointer variable;

pointer variable is a pointer that points data object is created with new operator.

delete  
where  
delete ptr\*  
If we want  
to remove dynamically created array then we must use delete operation with following format.

`delete[size] pointer name ;`

Syntax:-  
when the  
size specifies the number of elements in the array is to be deleted. If the size is not specified then it deletes the compute array.

`delete[5]ptr;`

`delete[ ] ptr;`

Note :-

If sufficient memory is not available for allocating or due to internal problems in such case the new operator returns null pointer.

Advantages:-

1. The new operator offers some of the advantages over malloc function.
2. It automatically computes the size of the data objects we need not use 'sizeof' operator.
3. It automatically returns current pointer type so that there is no need to use a type cast.
4. It is possible to initialize the object by allocating the memory space.
5. The new & delete operator can also be overloaded.

### **MANIPULATORS:-**

Manipulators are the operators that are used to format data output. The manipulators are defined in the library file "iomanip" this file must be included when using the manipulators through preprocessor statements.

```
#include <iomanip.h>
```

The most commonly used manipulators are "endl" & "setw"

The endl manipulator is used to only in O/P statement that causes line feed to be inserted it has the same effect as using new line character "\n"

### **SETW:-**

When O/P the data for number in a proper format & controlling the field width by specifying the manipulator "setw". It specifies a field width for printing the values of a variable once we specified the field width, the values are aligned from right justification. It takes only one argument from integer type specifies the field width.

### **ENUMERATED DATA TYPE:-**

The enumerator data type is a user defined data type provides a way for attaching names to numbers. It also provides the way to defining the constants for created data types the keyword enum declares enumerated list of values by assigning them with 0,1,2, & so on this facility provides an alternate means for creating symbolic constants. The syntax of enum statement is similar to that of declaration of struct statement.

```
enum identifier { val1, val2, val3.....}  
enum color { red, green=4, blue=10, orange};  
enum color bycolor;
```

```
enum{false, true};  
enum color a=3;
```

Where enum is a keyword followed by the identifier known as enumerator & it acts as data type the list of values are specified within braces are treated as integral constant.

The enumerated tag's are acts as data type & retains its own value this means that C++ doesn't permits a int value to be automatically converted to enum value by default the enumerators are assigned integer value starting with zero for first enumerator one for second & so on we can also over write the default value by explicitly assigning integer value to the enumerator.

```
enum color { red, green=4, blue=10,orange};
```

C++ also permits the creation of enumerator without specifying the tag name.

```
enum{false, true};
```

### **CONTROL STATEMENT:-**

Control statement are used to executes the sequence of statements on the basis of conditions generally the statements in a program are executed in a sequential manner such type of flow control known as sequential flow of control. The control statement alters such sequential flow of control. C++ provides 3 types control statement.

### **CONDITIONAL CONTROL STATEMENT:-**

These statement transfer the control from one location to another based on the condition must be satisfied.

Ex:- If, If else , Nested if , switch case.

Unconditional control statement:-

These statements alter the flow of control without any condition it transfer the control anywhere in the program break, continue, go to, return statement.

### **LOOP CONTROL STATEMENT:-**

Looping is a process to execute sequence of statements repeatedly. These statements executes the body of loop for fixed number of times or until the condition must be satisfied. These statements reduce the complexity of the program. There also known as iterative statements.

Ex:- for , while, do while.

### **FUNCTIONS:-**

Functions are the concept of modular programming it plays an important role in the program development. Functions are the subprogram which performs will defined task. It states the argument & returns the result after performing an operation on the arguments. Dividing a program into functions which is the major principle of top down programming. The advantage of using function is to reduce the size of the program by calling & using them at different places in the program. It reduces the complexities of the program. Functions in C++ are classified into two categories

1. Library functions
2. User defined function

### **Library function:-**

These are the predefined functions are also known as built in function. Depends on the functionality & operation to which it performs, they are stored in different files. Such files are known as library files. These files must be included using preprocessor statement.

Ex:- All mathematical statement math.h similarly string functions are defined in string.h.

### **User defined functions: -**

These functions are defined by the user at the time of developing of a program. These are help full in project based applications

The user defined functions having specification

1. Function declaration Or function prototyping
2. Function definition.

### **FUNCTION PROTOTYPING:-**

Function prototyping is one of the improvements added to c++ function the interface to the compiler by giving the details such as number of arguments type of arguments, function name & type of return value.

Syntax

<type> function name(arg\_list);

Where return type indicates the type of the value return by the function it must be any valid basic or user defined data type. A function return no value then

return type must be (no value) void by default the return type must be int. The function name is valid identifier the argument list specifies number of arguments & their type that are passed to the function. To pass more than one argument they are separated by (comma),. The function declaration is terminated by semicolon (;).

Note that each argument variable must be declared independently inside the parenthesis.

```
Ex: int sum ( int, int)
    show();
    void display(void);
    float interest(float,int,int);
    void swap(int *,int *);
```

### **FUNCTION DEFINITION:-**

The body of the function specifies the task to be performed by the function known as function definition.

The entire body of the function is specified within braces. It contains local declaration some executable statements & optionally specifies return statement.

Syntax:-<return type> function name (arg-list)

```
{  local declaration;
   executable statement
   {return[ value/exp]}
}
```

If function definition the name of the arguments are dummy variables which accept values from actual parameter. The return statement returns a control to the calling function along with value.

### **PASSING ARGUMENTS:-**

There are two ways to pass argument the function

1. Pass by value.
2. Pass by reference.

Pass by value:- In this mechanism the actual value must be copied or passed to the function at the time of calling. Here the formal parameters are the dummy arguments which receive the copied value & if made any changes inside the function it doesn't affect the original value because the function must perform

an operation only for copied value in this technique the function doesn't return more than one value.

### **Pass by reference through pointers:-**

This is another mechanism to pass arguments in the function. In this technique at the time of function call we must pass reference or address of the variable so the formal parameter in the function definition must be the variable which holds the address. If we made any changes inside the function it affects the original value & also returns the multiple values.

### **Pass by reference through reference variable:-**

In this technique the formal parameter of the function definition uses reference variables. The reference variables are the alias name for existing variable. These variables are the alternatively for the actual parameter. If we made any changes in the function it affects the original value it also returns multiple parameters even if the return statement is not specified.

### **Return by reference:-**

A function can also return a reference like a value in the return type of a function we must indicate the reference of the variable.

```
Ex:-  int &a=x;
      int & max ( int a, int b)
      { if ( a>b)
        return(a);
        else
        return(b); }
```

In the above example return type of a function is int & the function return the reference of either A or B but not the values this means that above function call can appear left hand side of an argument operator is reference variable.

```
L=max(x,y);
```

### **INLINE FUNCTION:-**

The main objective of using functions in a program is to save some memory space however each & every time a function is called it takes a lot of extra time in a executing series of instructions for tasks. Such as jumping to the function, saving register. Pushing arguments into the stack & returning into the calling function when a function must be called repeatedly the execution time will be increases. In C++ to eliminate the cost of call to small functions by using inline function. An Inline function is a function that is expanded inline when it is invoked that is the compiler replaces the function call with the corresponding function code. The inline function must be defined with the keyword inline.

```
inline<return type> function-name(arg-list)
{    // body
}
```

On inline function must be declared before they are called. The benefit of inline function depending on size of the function

Some of the situation where inline function may not be expanded.

1. For function returning value, loop switch, break, goto, exist.
2. Function not returning a value if return statement exist, if
3. If function contains static variables
4. If inline function are recursive.

```
Eg:- inline int max(int a, int b)
{ if (a>b)
  return(a);
  else
  return(b); }
```

### **DEFAULT ARGUMENT:-**

C++ allows to call a function without specifying all its arguments. In such cases the function assigns a default value to the parameter which doesn't have matching argument in function call. A default values are specified when the function is declared. The compiler verifies the prototype to see how many arguments a function uses and alert the program for possible default values.

```
int sum ( int a=6,int b=7)
int sum(int, int b=5);
```



```
sum(int a,int b)
```

The default value is assigned in a manner similar to a variable initialization. A default arguments are checked for the type at the time of declaration & evaluation at the time of call note that we must add default values from right to left we can't provide default value to a particular argument in the middle of the argument list.

### **FUNCTION OVERLOADING:-**

The overloading refers to the use of same thing for different purpose. The C++ allows overloading of functions. This means that we can use the same function name to perform variety of different task this known as function overloading or function polymorphism.

Using the concept of function overloading we can design a family of function with one function name but different argument list. The function could perform different operations depending on arguments list in the function call. The correct function is determined by checking number of arguments & this type but not the return type.

```
Sum( int a, int b);  
float sum ( float a, float b, float c);  
float sum ( int, float);
```

Function calls match the prototype having the same number & type of arguments & then cause the appropriate function for execution. A best match function must be unique. The following step be involve for the selection function. The compiler must find an exact matched in which types of actual parameter are the same & use that function. If an exact matche is not found, the compiler user the integral promotion to the actual parameter when either of them fails the compiler use built in conversion of implicit function to the actual parameter and them use the function which is exactly match.

If the all the steps fail when compiler try the user defined conversion & the combination with integral promotions & built in function to find unique match. When either of the steps fails then compiler displays an error message type mismatch.

## **CONST ARGUMENTS:-**

In C++ an argument to a function be declared as const the qualifier const tells the compiler that the function shouldn't modify the argument the compiler will generate an error message when this condition is violated. These types of declaration only when we pass arguments are pointers.

## **-. CLASSES AND OBJECTS:-**

### **CLASSES:-**

A class is a user defined data type it defines the state and behavior of an object. A class is an extension of the structure used in C language. A class represents an object, a class is a way to bind data and its associated member function. The class name must acts as built in data type.

The following general format shows defining the class.

```
Class<class name>
{ private:
    variable declaration;
    function declaration;
public:
    variable declaration;
    function declaration; }
```

The class can be declared with the keyword class that follows an abstract data type called class name. The body of a class is enclosed within braces & terminated by the semi-colon. The class body contains declaration of variables and functions these functions and variables are called class members. There are two sections in class that is private section and public section. The keyword private and public are known as visibility or access specifier and are followed by ( : ) colon symbol. The class members that have been declared as private can be accessed only with in the class. The public members can be accessed outside the class. The data hiding is the key features of OOPS. By default the members of a class are private. The variable declared inside the class are known as data members and the function are known as member functions only the member functions can have access the private members. However public members can be accessed outside the class.

```
Class point
{ private:
    int x,y;
public:
```

```
void getpoint (int a, int b);  
void show point();    };
```

### **CREATING OBJECTS:-**

The class declaration specifies the details of the object but not defining any object once a class has been declared, we can create any number of objects of the type by using class name, the class variables are known as objects. Instances of class are also known as object.

The declaration of object is similar to that of variable of any type, the necessary memory space is allocated to the object at this stage. The objects can be created in 2 ways.

1. The declaration of objects is similar to declaring a variable using a class name as a data type.

Syn:- < class name > object1, object 2.....;

eg:- point p1,p2;

2. The objects can also be created when defining a class by placing object names immediately after the classing braces.

Syn:- class < class-name>

{ private:

variable declaration;

function declaration;

public:

variable declaration;

function declaration ;} object list.....

Eg. Class student

{ int regno;

char name[1];

int age;

public: void getdata();

void showdata();

} s1,s2;

## **ACCESSING CLASS MEMBER:-**

Note that private data of the class can be accessed only through member function of the class. The public members must be accessed outside to store or print the data values by using object with dot operator.

Object.public member.

The private members can be accessed only through member function and not by the object directly. The objects communicate by sending and receiving messages is done through member function. A variable declared as public can also be accessed by the object directly.

Ex. Class point

```
{ private: int x,y;
  public: int z;
        void get(point( int a, int b);
        void showpoint(); } p;
p.getpoint( 10, 20);
p.showpoint();
```

## **DEFINING A MEMBER FUNCTION:-**

The member function can be defined in two places.

1. Outside the class definition.
2. Inside the class definition.

Note that irrespective of the place of function definition should perform the same task. Therefore the code for the function body could be identical in both cases.

## **OUTSIDE THE CLASS DEFINITION:-**

The member functions that are declared inside a class have to define separately outside the class. The definitions are like normal function. An important difference b/w member function & normal function is that member function include member include member to identify label in the header this label take the compiler which the function belongs to

```
<return type> class name::function name( arg list)
{
// body of function
}
```

The membership label class name scope resolution tells the compiler that the function belongs to the particular task that is the scope of the function is restricted to the class name specified in the header line in the function.

### Characteristics:-

Characteristics of this type function definition are

1. Several different classes are same function name. The membership label then specify there scope.
2. Member function can access private data of the class \.
3. The member function can call another function directly without using dot operator.

Sy:-

Class point

```
{ private: int x,y;  
  public: void getpoint( int a, int b); }  
void point :: getpoint( int a, int b)  
{  
  x=a, y=b;  
}
```

### **INSIDE THE CLASS DEFINITION:-**

Another method of defining member function is to replace function declaration by the actual function definition inside the class when the function defined inside the class if is treated as inline function therefore the restriction & limitations that apply to inline function are also applicable to here normal only function are defined inside the class.

Sy:-

Class point

```
{ private: int x,y;  
  void getpoint( int a, int b)  
  { x=a, y=b;  
  } };
```

### **MAKING AN OUTSIDE FUNCTION INLINE:-**

We can define a member function outside the class definition & still make it by the keyword inline in the header line of function definition it is good practice to define the member function outside the class. When a simple function has to be define outside the class always it is defined with the keyword inline.

```
Sy:-Inline void point::getpoint(int a, int b)
    {      x=a, y=b;
          }
```

### **NESTING OF MEMBER FUNCTION:-**

In general member function of a class can be called only by the object of that class by using dot operator a member function can be called by using its name inside another member function of same class is known as nesting of member function.

Private member function:-

In general all the data items are placed in private section and all the function in public section. In sum situations may require sudden function to be hidden from outside the class for handling restricted access in such situations. We can place these functions in private section. A private member function can only be access by another function that is the members of class itself even object can't invoke private function using dot (.) operator.

Sy:-Class point

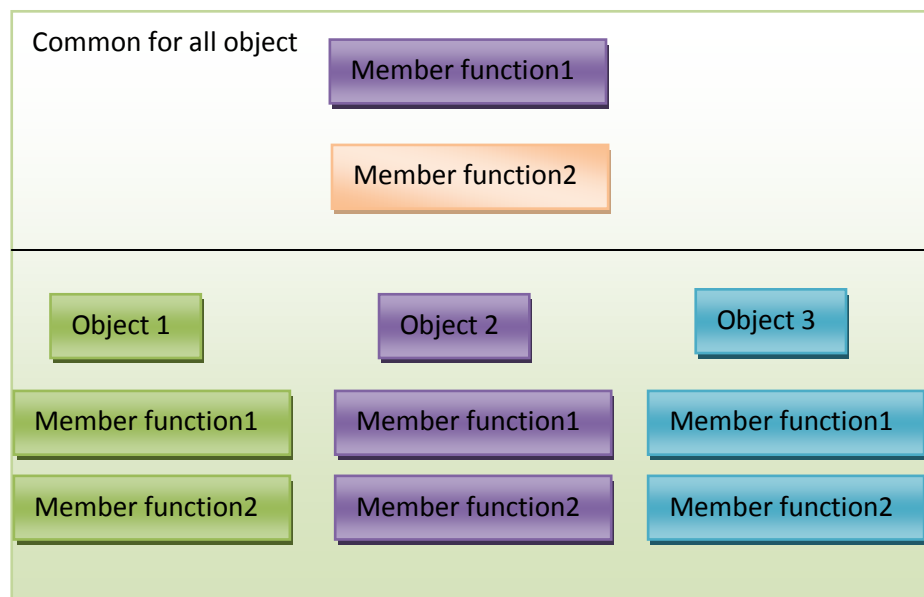
```
{ private:int x,y;
    void show point()
    { cout<<"("<<x<<" "<<y<<"")";
  public:void getpoint(int a, int b)
    { x=a, y=b;
      show point();
      pow p;
      p.getpoint(80,20);
```

### **ARRAY WITH IN A CLASS:-**

Array is a derived data type which holds set of element of some type that is array contains homogenous data elements class is an abstract type which holds anything with in the class. The arrays can be used member variable in class. Like an ordinary array we can declare the array inside the class & it is also treated as data member & can be used in the member function like any other array variable we can perform any operations on it. The array elements must be identified with the index. The array index starts from zero.

**MEMORY ALLOCATION FOR OBJECTS:-**

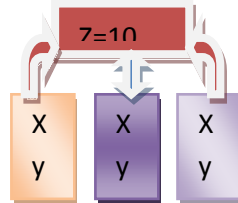
Memory space for the objects is allocated when they are declared but not when the class is specified. This statement is partially true. A member function is created & placed in memory space only which they are defined as a part of class specification since all the objects belong to the class use the same memory function and no separate space for the member function when the objects are created only space for member variables is allocated separately for each object separate memory location for the object because member variable will hold different object.

**STATIC DATA MEMBER: -**

we can also declare a static variable within the class. The properties of static member variable are similar to static storage class. A static variable member has certain special characteristics.

1. It is initialized to zero when the first object is created & no other initialization is permitted.
2. Only one copy of that member is created for the entire class & is shared by all the objects of the class.
3. It is visible only within the class but the life time is the entire program.
4. Static variables are normally used to maintain common values for the entire class. The static data members are stored separately rather than as a part of an object since they are associated with the class rather than any class object so that they are also known as class variables.

```
Class point
{ private: int x,y;
    static int z;
    public:
        };
point p1,p2,p3;
```



## **STATIC MEMBER FUNCTION:-**

Like static member variable we can also use static member function a member function that is declared as static as the following property.

1. A static function can have access only other static members (functions or variable) declare within the same class.
2. A static member function can be called using class name instead of object with scope resolution operator.

Syn:- Class name :: function name();

## **ARRAY OF OBJECTS:-**

Array is a collection of data elements of same type. An array can be declared like any data type similarly we can also declare an array of particular class type such variables are called array of objects.

The declaration of array of objects is similar to ordinary type.

Class-name array-name[size];

The array of objects behaves like any other objects we can access the array elements individually by specifying index. Array index start from zero therefore the reference of the 1<sup>st</sup> object is array [0], 2<sup>nd</sup> object is array[1] & so on. To access to its members by specifying array name, index, with dot operator.

Student s[0]



### **OBJECT AS FUNCTIONAL ARGUMENT:-**

Any other data type the object can be viewed as function argument. This can be done in two ways' .

1. Pass by value.
2. Pass by reference.

The first method copy the entire object is passed to the function since the copy of the object is passed to the function any changes made to the object inside the function do not affect the object used to call the function. In this technique the function doesn't return multiple values.

The second method is the pass by reference method when address of the object is passed to the function works directly to the actual object used in the call. This means that any changes made to the object inside the function will reflect the actual object that is it affects the contents of original object.

The pass by reference method is more efficient since it requires to pass only the address of the object but not the entire object

### **FRIEND FUNCTION:-**

Note that private members cannot be access from outside the class that is non member function can't have access the private data of a class however in certun situations where two classes share particular function in such situation C++ allows a common function to be made friendly with both the class. There by allowing the function to have access to the private data of these classes such function need not be member of any of these class.To make outside function friendly to a class declaration must be begins with the keyword friend.

Syntax:-

```
friend<return type> function-name( arg-list);
```

The function definition doesn't use the keyword friend are member label ( class scope resolution operator). A friend function is not a member function but has full excess rights to the private member of the class.

Characteristics:-

1. It is not in the scope of the class in which it has been declared as friend.
2. It can't be called using the object of the class
3. It can't invoked like normal function without the help of any object.

4. It can't access members directly & it has to use object name & dot operator with each member name.
5. We can be declare either in public or private section without affecting its meaning usually it has object as argument.

We can declare all the member function of one class the friend function of another class such class is called friend class.

```
Class x
{ int z
  };
class y
{
  friend class x; }
```

### **RETURNING OBJECT:-**

A function not only receives objects as argument but also can return them. This means that the member function of a class can return object of that class is such cases the return type of function must be class name

```
class time
{
  int hh,mm;
  public: void get time(int h, int m);
  friend time sum( time, time)
  void show time(); }
```

In the above example sum is friend function take two object of time type and also return the time object.

### **CONSTANT MEMBER FUNCTION:-**

If a member function doesn't alter any data in the class then we made declared as constant member functions. The keyword const is appended ( add) to the function prototype. The compiler will generated an error message if such function right to be alter to the data values of the class

```
class time
{
  int hh,mm;
  public: void get time(int h, int m);
  void show time() const; }
```

### **LOCAL CLASS:-**

Class can be defined & used inside a function or block such classes are called local classes.

Local classes can use global variables & static variable declared inside the class but cannot use automatic variables. The global variable can be used with scope resolution operator.

There are some restrictions in structure local classes. They cannot have static member function must be defined inside the local classes enclosing function cannot access private member of a local class.

Ex:-

```
void test()
{
    class time
    {
        int hh,mm;
    public: void get time(int h, int m);
           void show time() const;    }; }
```

### **CONSTRUCTOR & DESTRUCTORS:-**

A constructor is a special member function its task is to construct the object and also initialize the object when it is created. It is a special task its name is same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs various data members of the class.

Constructor functions have some special characteristics.

1. They should be declared in public section.
2. They are invoked automatically when the objects are created.
3. They don't have any return type not even void therefore they can't return any value. They can't be inherited through derived class can call base class constructor.
4. They can have default arguments.
5. The constructor can't be virtual.
6. We can't refer to their address.
7. An object with constructor can't be used as member of union.

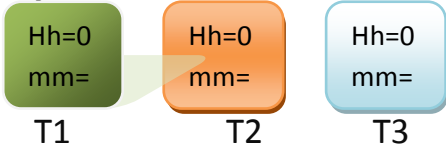
There are different types of constructor

1. Default constructor.
2. Parameterized constructor
3. Copy constructor
4. Implicit constructor
5. Dynamic constructor.

## **DEFAULT CONSTRUCTOR:-**

A constructor that accesses no parameter & initializes the default values for its members is called default constructor. It initializes the same value for the data members of all objects. It will be executed automatically when the object has been declared.

```
Class time
{ int hh, mm;
public: time()
{   hh=0,mm=0; default constructor
```



The diagram illustrates three objects, T1, T2, and T3, each represented by a colored rounded rectangle. T1 is green, T2 is orange, and T3 is light blue. Each object contains the text 'Hh=0' and 'mm=' on two lines. A light green arrow points from the 'mm=' of T1 to the 'mm=' of T2, indicating that all objects initialized by the default constructor have the same values for their data members.

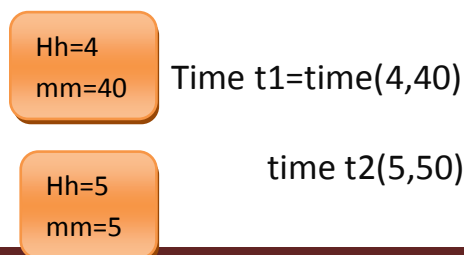
T1                      T2                      T3

## **PARAMETERIZED CONSTRUCTOR:-**

It may be necessary to initialize the various data elements of different object with different values when they are created C++ allows to achieve this concept by passing arguments to the constructor function when the objects are created. The constructor that can take arguments are called parameterized constructor.

We must pass the inside values as arguments to the constructor function when the object is declared.

```
Class time
{ int hh, mm;
public:time( int h, int m)
{   hh=h,mm=m; } };
```



The diagram shows two objects. The first object is an orange rounded rectangle labeled 'Time t1=time(4,40)' to its right. Inside the rectangle, it contains 'Hh=4' and 'mm=40' on two lines. The second object is another orange rounded rectangle labeled 'time t2(5,50)' to its right. Inside this rectangle, it contains 'Hh=5' and 'mm=5' on two lines.

Time t1=time(4,40)

time t2(5,50)

We must pass the arguments to the constructor in two ways.

1. By calling the constructor explicitly

time t1=time(4,40)

In this method we must specify the constructor name explicitly & also specifying the arguments list at the time of call

2. By calling the constructor implicitly.

Time t2(5,50)

this is also known as short hand method is easy to implement by simply specifying in front of the object within parenthesis.

Note that when the constructor is parameterized we must provide appropriate arguments for the constructor.

### **COPY CONSTRUCTOR:-**

The copy is used to declare and initialize an object from another object. The process of initializing through copy const is known as copy initialization. A copy constructor is also same name as class name and it takes reference object of the same class itself as an argument. In the copy constructor simply assign values of one object to another object of some type member by member. The task is done by assignment operator (=). A reference variable can be used to copy constructors we cannot pass an argument by value to a copy constructors.

```
Class time
{ int hh,mm;
public: time(time &x)
    { hh=x.hh,mm=x.mm; }    };
```

### **IMPLICIT CONSTRUCTOR:-**

The implicit constructor is a special type of constructor it also same name has class name & it contain no arguments & empty & it does not do anything it is used create object without initializing the values for its data members if no constructor must be specified the compiler will implicitly creates the constructor which creates the object. The constructor will not do anything & is defined just satisfy the compiler.

```
Class time
{ Int hh,mm;
public: time() };
time t1,t2;
```

## **DYNAMIC CONSTRUCTOR:-**

A constructor is used to allocate memory while creating objects this will enable the system allocate right amount of memory for each object when object are not of same type so its save the allocation of memory to an object at the time of constructor is known as dynamic constructor of objects. The memory is allocated with the help of new operator the dynamic constructor allocates right amount of memory for same object.

## **DESTRUCTOR**

A destructor is a special type of member function ( method) the name of self indicates it is used to destroy the object that have been created by the constructor like constructor. The constructor is also member function its name as same has class name but is precedent by tilled symbol(~).

A destructor never takes any argument and also does not return any value it will be invoked implicitly when the object will be deleted that is the compiler will exist from program or block or function to cleanup storage space. It is a good practice to declare destructor in a program since its releases memory space for feature use.

When ever new operator is used to allocate the memory in the constructor we should use delete operator to free the memory. Note that objects are destroyed in the reverse order of their creation.

```
newchar(#)

delete[] name;
};

Sy:-Class string
{ int len;
  char *name;
public; string()
{ len=0;
  name:
                                }
  ~string ()
  {
                                }
}
```

### **CONSTRUCTOR WITH DEFAULT ARGUMENT:-**

It possible to define constructor with default arguments. The formal parameters of the constructor are initialized with default value and the parameterized constructor called the actual parameter when specified overwrite the default value note that the default arguments of the constructor are specified from right to left.

It is important to distinguish b/w default constructor and default argument constructor. The default argument constructor can be called with either one or more arguments or no arguments when called with no arguments it becomes the default constructor. When both of these forms are used in a class it causes an ambiguity

### **-.:OPERATOR OVERLOADING:-**

Operator overloading is the one of feature of c++ it is an important technique that has enhance the power of extensibility behaves in much the same way has built in types. The user defined data types must be used in the same syntax that is applied to the basic type. This means that C++ has an ability to provide operator with special meanings for a data type. The mechanism of giving such special meaning to an operator overloading provides flexible option for the creation new definition for the most of C++ operator. They can overload all the operator except.

1. Scope resolution operator.
2. Class member access operator.]
3. Size of operator.
4. Conditional operator.

### **DEFINING OPERATOR FUNCTION:-**

To define additional task to an operator, we must specify the relation to the class to which the operator is applied. This is done with the help of special function called operator function which describe the task of function. The general form of operator function is as follows.

```
<return type> class name:: operator <op> ( arg-list)
{
    // body of loop
}
```

Where return type is a of a value return by the specified function. The OP is the operator being overloaded it is preceded by the keyword operator. The operator OP is preceded by the keyword operator. The operator OP is the function name.

The operator function must be either member function Or friend function. The basic difference b/w them is that the friend function will have only one arguments for unary operator & two arguments for binary operator while a member function has no argument for unary & only one argument for binary operator. This is because the object used to invoke the member function is passed implicitly and is available for member function. This is not in case of friend function. The argument may be passed either by value or by reference. Note that the semantics of the operators can be extended but we can't change the syntax that is number of operand precedence and associability. Note that when an operator is overloaded its original meaning is not loss.

### **Rules for overloading operator:-**

There are certain restriction & limitations on overloading operator

1. Only the existing operator can overloaded , new operator can't be created.
2. Overloaded operators must be having at least one operand.
3. It can't change the meaning of an operator.
4. Overloaded operator follow the syntax rules of original operator.
5. Some operators can't be overloaded such as scope resolution dot conditional & size of.
6. We can't use friend function to overloaded square, bracket, paranthesis & equal.
7. Unary operators can be overloaded. The member function takes no argument but friend function takes one arguments.
8. The binary operators overloaded through member function one argument & friend function takes 2 argument.
9. The arithmetic operator must explicitly return a value.



## Types conversion:-

It is the process of converting the data from one type to another type. Then evaluating different types or mixed in a expression. C++ is an object oriented programming language it defines user defined data types i.e., class type which is user defined data types are designed by us according user requirement. The compiler doesn't supports atomic conversion for such type. The 3 types of situation arises the data conversion.

1. Conversion from basic type to class type.
2. The conversion from class to basic type.
3. Conversion from class to class type

## Conversion from basic to class type:-

The conversion from basic to class type is easy to note that the use of constructor is used to build the object & also initialize the constructor used for type conversion takes single argument whose type is to be converted. The `/**` 5 is always class object therefore we can also accomplish his conversion using overloaded assignment operator.

```
Class-name(arg-list)
{
}
Time T1=
```

2333

## Conversion from class to basic type:-

Constructor is not used for type conversion from class to basic type C++ allows to define overloading casting operator that could to be converting a class type data to basic type. The general form of overloaded costing operator usually referred to as conversion function

```
Operator(type.name>())
{
    // body of function
}
```

This function converts a class type data to a type name. The costing operator function must satisfy the following condition .

1. It must be a class member.
2. It must not specify a return type.
3. It must not have any argument since it is a member function it is invoked by an object therefore the values used for conversion inside the function. Belong to the object that invokes the function. This means that the function doesn't used an argument.

### **Conversion from one class to another class:-**

The data conversion between from basic to class type & class to basic type is different from one class to another class such conversions b/w object of different classes can be carried but by either constructor or by conversion function. The compiler treats them in a same manner it depends upon where we want the type conversion function to be located in the source class not in the destination classes.

To convert object from one type to another by using operator type() function. This function Converts the class objects of which it is a member of type name the type name may be built in for user defined in case of conversion b/w object the type name refers to destination class. The conversion take place in the source class & the result is given to destination class object.

```
Operator<type-name>()  
{ //body of the loop  
}
```

## **:-INHERITANCE:-**

Reusability is one of the important feature of OOP'S it is always nice if we would reuse something that already exist rather than create the same all over again. It is not only save the time & also reduce complexity of the program.

Inheritance is the process by which object of one class acquire the properties of objects of another class. The concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. It is possible by deriving new class from existing one. The new class will have combined features of both the classes.

The mechanism of deriving the new class from old one is called Inheritance (or) derivation. The old class is referred as waste class parent class (or) super class. The new class is referred as child class, sub class.

C++ supports different types of Inheritance.

1. Single inheritance.
2. Multilevel inheritance.
3. Multiple inheritance.
4. Hierarchical inheritance.
5. Hybrid inheritance.

### **Defining Sub Class:-**

A derive class can be defined by specifying its relationship with the base class in addition to its own details.

The general form of defining subclass is as follows:

```
Class<sub class name>:<visibility mode><Base class name>
{
    //body of sub class
}
```

The colon indicated that the subclass name is derived from the super class name. The visibility mode indicated the behavior of super class properties is subclass. It is optional if it is present may be either private or public. The default

mode is private the visibility mode specify whether the features of the base class are privately derived or publically derived.

When a base class is privately inherited by a derived class. The public members of base class become private members of subclass & therefore public member of base class can only accessed by the member function of the derived class. They are not accessible to the object of derived class.

When the base class is publically inherited public members of base class becomes public member of derived class & therefore they are accessible to the object of the derived class. In both cases private members are not inherited & therefore private members of base class will never become the member of it's derived class.

In inherited some of base class data member & member function are inherited into the derived class we can add new data & member function to the newly created class

### **SINGLE INHERITANCE:-**

A class which derives only one subclass or a subclass which is derived from only one base class such form of inheritance is known as single inheritance it is simplest form of inheritance.

Class B

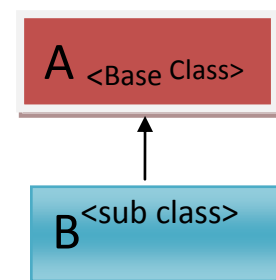
```
{
```

```
};
```

Class C:Public

```
{
```

```
};
```



In above diagram the base class A continues private & public data members. The public members are extended a subclass B that is the derived class B as only one base class A

Making private members Inherited:-

Note that private members of a base class can't be inherited & therefore it is not available for derived class directly however to increases the capability on

existing classes without modifying it we can inherit private data in derive class. This can be accomplished by modifying the visibility limit of the private member by making it public. This make it accessible to all other functions of the program so taking aware the advantage of data hiding.

C++ provide another visibility modifier called protected with serve the limited program in inheritance a member declared has protected is accessible by the member function with in its class and any class immediately derived from it. It can't accessible by the functions outside these two classes when a protected member is inherited in public mode it becomes protected in the derive class & therefore is accessible by the member function of the derive class.

It is also ready for further inheritance. A protected member inherited in a private made derivation becomes private in the derive class. It is also available to the member function of the derive class but is not available for further inheritance.

It is also possible to inherit base class in protected.

In protected derivation both public & protected member of the base class become protected member of the derive class.

Base class visibility	Derived class visibility		
	Private	Protected	Public
Private	Not inheritance	Not inheritance	Not inheritance
Protected	Private	Protected	Protected
Public	Private	Protected	Public

### **MULTILEVEL INHERITANCE:-**

The mechanism of deriving a class from another derived class is known as multilevel Inheritance. It is not uncommon that a class is derived from another derived class.

```
Class A  
{
```

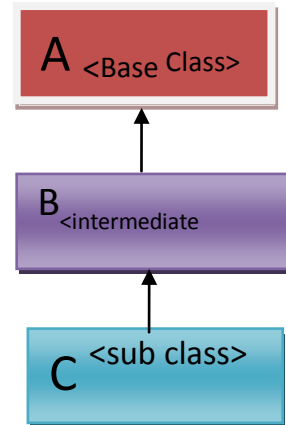
```
};
```

```
Class B  
{
```

```
};
```

```
Class C:Public  
{
```

```
};
```



In above format class A act as base class for derived Class B which in term act as base class for the derived class C. The branch class since it provides a link b/w A&C. The chain ABC is known as Inheritance path.

### **MULTIPLE INHERITANCES:**

The class can inherit the properties of more than one base class. This form of inheritance is called multiple inheritances. Multiple Inheritances allows combining the features of several existing classes as a starting point for defining new classes.

```
Class B
```

```
{
```

```
};
```

```
class B2
```

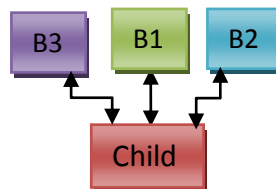
```
{
```

```
};
```

```
Class c:<mode> B, <mode> B2
```

```
{
```

```
};
```

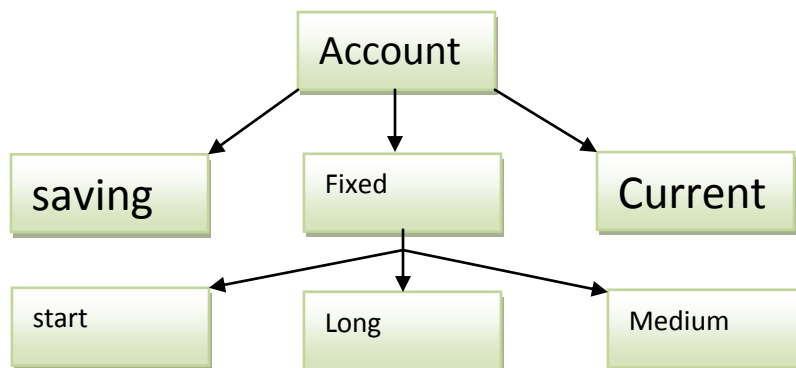
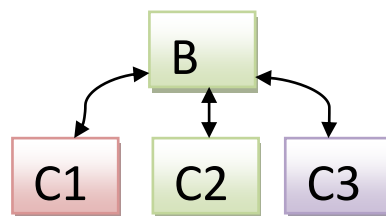


## HIERARCHICAL INHERITANCE

The Inheritance can be used to modify a class. When it did not satisfy the requirements of Particular Problem Additional members are added through inheritance to extended the capability of class.

The base class derived a number of classes (or) multiple sub class share the properties of only one base class such form of inheritance is known as Hierarchical Inheritance many programming problem can be cost into hierarchy where sudden features of one level shared by many others below that level.

In C++ the problem can be converted into class hierarchy. The base class will include all the features that are common to subclass. A subclass can be constructed by inheriting the properties of the base class. A subclass can serve as base class for the low level classes and so on



In above example account class contains common features of all account in next level class share their features & also include addition features.

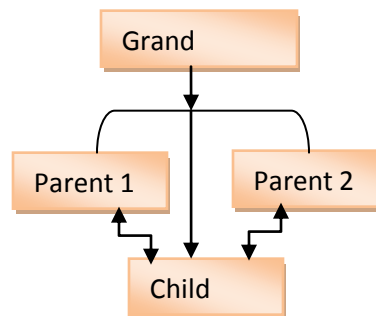
## **HYBRID INHERITANCE:**

In certain situation where we need apply 2 Or more types of Inheritance to design a program.

The hybrid inheritance is a combination of multiple, multilevel, Hierarchical Inheritance depend on the problem different types of Inheritance must be included in a single program it is known also as multiple inheritance.

## **VIRTUAL BASE CLASS:-**

In certain situation we would require the use of all kinds of Inheritance such as multiple, multilevel, Hierarchical Inheritance are involved. This situation forms multipath Inheritance.



The child has two direct base classes that is parent & parent 2, which themselves have common base class grand parent. The child class inherit the grandparent property into separate path it also inherits directly as shown by dotted line. The grandparent sometime referred inherits base class.

In above situation create some problem all public & protected members of grandparent are inherited into child class twice that is through parent & parent 2 this means that the child have duplicate sets of members inherited from grandparent. This introduces ambiguity & it should be avoided.

The duplication of inherited members due to these multiple paths can be avoided by making common base class as virtual base shile declaring the direct or intermediate base classes.

When a class in made as virtual base class C++ takes care to see that only one copy of that class is inherited regardless how any inheritance path exists b/w virtual base class & derived class.



Syntax:- class grandparent

```
{  
};  
class p1:virtual public grandparent  
{ };
```

Note that the keywords virtual & public may be used in either order.

### **ABSTRACT CLASS:-**

The abstract class is one it is not used to create any object an abstract class is designed only to act as base class that is the members of abstract class is to be inherited by other class. It is a designed concept in program development provide a base upon when other class can be built. It always acts as base class so that their properties must be accessed by the object of subclass.

POINTERS and Virtual functions and polymorphism this pointer

C++ uses a unique keyword called this to represent an object that invoke a member function this is a pointer that pointes to the object for which this function was called unique pointer is automatically passed to member function when it is called. The pointer this acts as implicit argument to all the member function.

The starting address is same as address of first variable in a class structure

class time

```
{  
    int hh,mm;  
    public: void get time(int h, int m)  
        {time hh=h, time mm=m;}}; time t1,t2;  
t1.gettime(10,20) t2.gettime(10,20)
```

However we have an implicitly using the pointer this when overloading operation using the member function

### **:-VIRTUAL FUNCTION:-**

Polymorphism refers to the property by which object belonging to different classes are able to respond to message but in different form in an essential requirement of polymorphism is the ability to refer to the objects without any regard to their classes. This necessitates the use of single pointer variable to refer to the object of different classes. Here we use the pointer to base class to refer to all the derived objects but the base class pointer even when it is made to contain the address of the derived class always executes the function in base class the compiler simply ignores the contents of the pointer & chooses the member function that matches the type of the pointer. The runtime polymorphism is achieved using virtual function when we use the same function in both base & derived classes the function in base class is declared as virtual using the keyword virtual. C++ determines which function to use at run time based on type of object pointed by the base pointer rather than the type of the pointer so making the base pointer to point different object we can execute different versions of virtual functions.

Rules for virtual functions:-

1. Virtual functions must be a member of subclass.
2. They can't be a static member
3. They are accessed by using object pointer
4. Virtual functions can be made friend to another class.
5. Virtual base class must be defined even though it may not be used.
6. Prototype of the base class of a virtual function & all the derived class versions must be identical.
7. We can't have virtual constructor but we can have virtual destructor
8. While the base class pointer can point to any type of derived object but the reverse is not true

9. If the virtual function is defined in the base class is need not be nessessary redefine in the derive class in such cases call will invoke base class function.

**Pure virtual** :- In general declare a function virtual inside a base class & redefined in the derived class. The function inside the base class performs no task that is we have not defined any object of the class & therefore the function in base class have been defined empty such functions are called do nothing functions

A do nothing functions may be defined as follows

```
virtual void show()=0;
```

Such functions are called pure virtual functions A pure virtual functions is a function declared in a base class that has no definition in the base class in such cases the compiler required each derived class to either defined or redeclared it has pure virtual function.

## **:-TEMPLATES:-**

Templates is one of the feature is added to C++ programming. It is new concept which enables to define generic classes & functions which provide the support for generic programming.

Generic programming is an approach where generic types are used as parameter in algorithm so that they work for a variety of suitable data types & data structures .

A template can be considered as a kind of macro when an object of specific type is defined for actual use. The template definition for that class is substituted with required data type since a template is defined with a parameter that can be replaced by a specified data type at the time of actual use of the class are function. The templates are some times called parameterized class.

A template can be used to create two types of family

1. Function template or generic functions
2. Class template or generic functions

**Function template or generic functions:-**Templates allows to define function. A function template that could be used to create family of function with different arguments. The generic form of function template is

```
Template<class T>
    <return type> function-name(arg of type)
    { //body of function    }
```

The function definition is very similar to an ordinary function definition except that the prefix with the keyword

template <class T>

And use of type T as a parameter & this statement hence the compiler that we declare a template & use T as a type name in the declaration we must use

template parameter T as and when necessary with the function & body & its argument list.

```
Template <class T>
T sum( T a, T b)
{ T c; c=a+b; return( c) ;}
```

### **Function template with multiple parameters:-**

We can use more than one generic data type in a function template using comma separated list in that situation we must a different value for generic type.

```
Template< Class T1, class T2>
<return type> function name(arg of type1, type2)
{ //body of function
}
```

The declaration of function template with multiple parameter is exactly similar to the function template but the parameters must be different type under separated by comma.

```
Template< class T1, class T2)
void show (T1 a, T2 b)
{ // body of function
}
```

The declaration of function template with multiple parameter is exactly similar to the function template but the parameters must be different type under separated by coma

```
template<class T1, Class T2)
void show(T1 a, T2 b)
{ // body of function
}
```

## Over loading template function

A template function may be overloaded either by template function or ordinary function of its name in such classes, the overloading resolution is accomplished as follows

1. Call an ordinary function that as exact match.
2. Call a template function that could be created with an exact match.
3. Try normal overloading resolution to overloading function & call the one that matches .
4. An error is generated with no match is found note that no automatic conversion are applied to arguments on the template function.

### **NON TYPE TEMPLATE ARGUMENT :-**

Note that a template can have multiple arguments it is also possible to use non-type arguments that is in addition to the type argument T we can also use other arguments such as built in type constant experiment Or string.

```
Template <class T>
void show(T b,int A)
{// body of function
}
```

In above example the function show as an argument of template type & integer type this implies that the second argument of the function must always of type integer.

## **CLASS TEMPLATES:-**

Template allows to define generic class. It is simple process to create generic class using a template with T type. The general format of class template is as follows

```
template<class T>
class<template-name>
{ T*a;
    public:
    };
```

The class template define definition is similar to ordinary class definition except its prefix with the statement

```
template<class T>
```

And use of type T as a member this prefix tells the compiler that a class is declared as template class and use T as type name in the declaration It may be substituted by any data type including user defined data type.

A class created from class template is called template class. The syntax for defining template class as follows

```
class name<type> object-name;
```

This process of creating object of specific class form class templates is called instantiation. The compiler will perform the error analysis only when object is created therefore to create & debug an ordinary class before converting into template

```
template<class T>
class vector
{    T a[10];
    public:        };
```

Class name<type>object name

```
vector<int>v1;
vector<float>v2;
```

### **CLASS TEMPLATE WITH MULTIPLE PARAMETER:-**

We can use more than one generic type in a class template they are declared as comma separated list within template specification. The body contains the set of variable of type T1 & T2

```
template<class T1, class T2>
class<class name>
{ private:
public:
};
```

While creating objects of template class we must specify the type of T1 & T2 as follows

```
class name<type1 type2>object name
vector<int, float>V1;
vector<float,char>V2;
```

```
Template<class T1, class T2>
class vector
{ private : T1 a[10];
            T2 b[10];
public:
}
```