SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIAL

Subject Name:OBJECT ORIENTED PROGRAMMING WITH C++ **UNIT 1** **SubjectCode : SBS1102**

Unit I

Introduction to C++

Object Oriented Programming Paradigms - Comparison of Programming Paradigms – Object Oriented Languages- Benefits of Object Oriented Programming - Comparison with C - Structure of C++ program. Characteristics of Object Oriented Programming.

1.1Object Oriented Programming Paradigms

- **Software evolution**

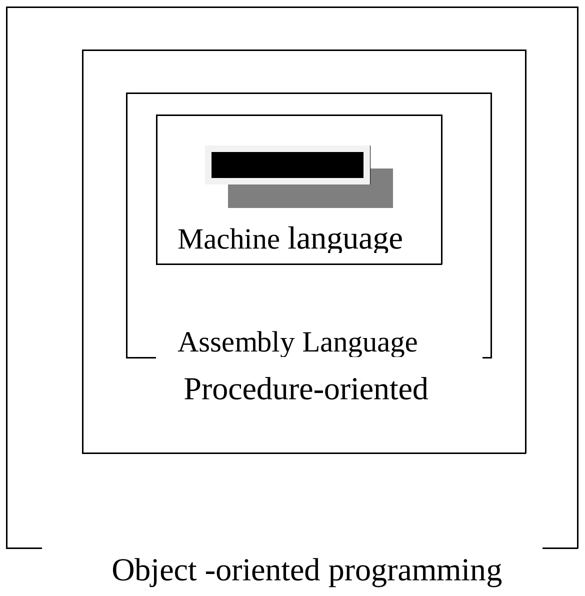


Fig. Layers of computer software

Since the invention of the computer, many programming approaches have been tried. These include techniques such as modular programming and structured programming. The primary motivation in each has been the concern to handle the increasing complexity of programs that are reliable and maintainable. These techniques have been popular among programmers over the last two decades.

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIAL

Subject Name:OBJECT ORIENTED PROGRAMMING WITH C++ **UNIT 1** **SubjectCode : SBS1102**

Machine language

It is the lowest-level programming language. Machine languages are the only languages understood by computers. While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.

Assembly language

Machine language is almost impossible for humans to understand.Programmers, therefore, use either a high-level programming language or an assembly language.

An assembly language contains the same instructions as a machine language, but the instructions and variables have names instead of being just numbers.

Procedure-oriented Programming

The high level languages, such as BASIC, COBOL, C, FORTRAN are commonly known as Procedure Oriented Programming.

Using this approach, the problem is viewed in sequence of things to be done, like reading, processing and displaying or printing. To carry out these tasks the function concepts must be used.

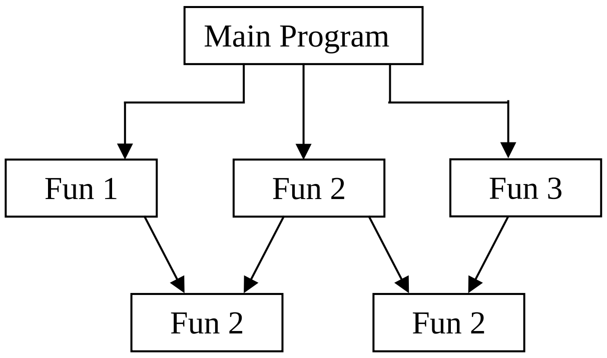


Fig. Typical structure of Procedure Oriented Programs

- This concept basically consists of number of statements and these statements are organized or grouped into functions.
- While developing these functions the programmer must care about the data that is being used in various functions.
- A multi-function program, the data must be declared as global, so that data can be accessed by all the functions within the program & each function can also have its own data called local data.

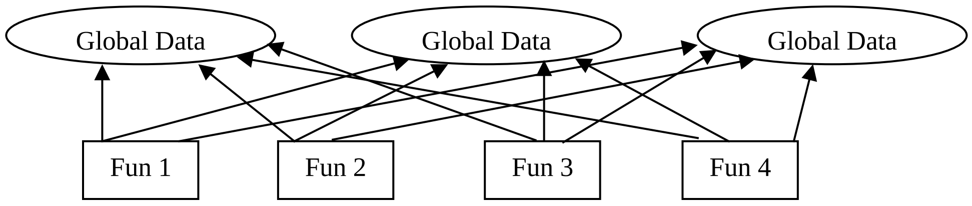


Fig. Relationship of data and functions in procedural programming

- The global data can be accessed anywhere in the program. In large program it is very difficult to identify what data is accessed by which function. In this case we must revised about the external data and as well as the functions that access the global data. At this situation there is so many chances for an error.

Characteristics of Procedure-oriented Programming (C):

- Emphasis is on doing things (algorithms)
- Larger programs are divided into smaller programs known functions.
- Most of the functions share global data.
- Data move openly around the system from function to functions
- Functions transforms data from one form to another

- Employs TOP-DOWN approach in program design.

Object-oriented programming

- This programming approach is developed to reduce the some of the drawbacks encountered in the Procedure Oriented Programming approach.
- The OOProgramming approach treats data as critical element and does not allow the data to flow freely around the program.
- It bundles the data more closely to the functions that operate on it; it also protects data from the accidental modification from outside the function.
- The object oriented programming approach divides the program into number of entities called objects and builds the data and functions that operates on data around the objects.
- The data of an object can only access by the functions associated with that object.However the functions of one object can access the functions of other objects.

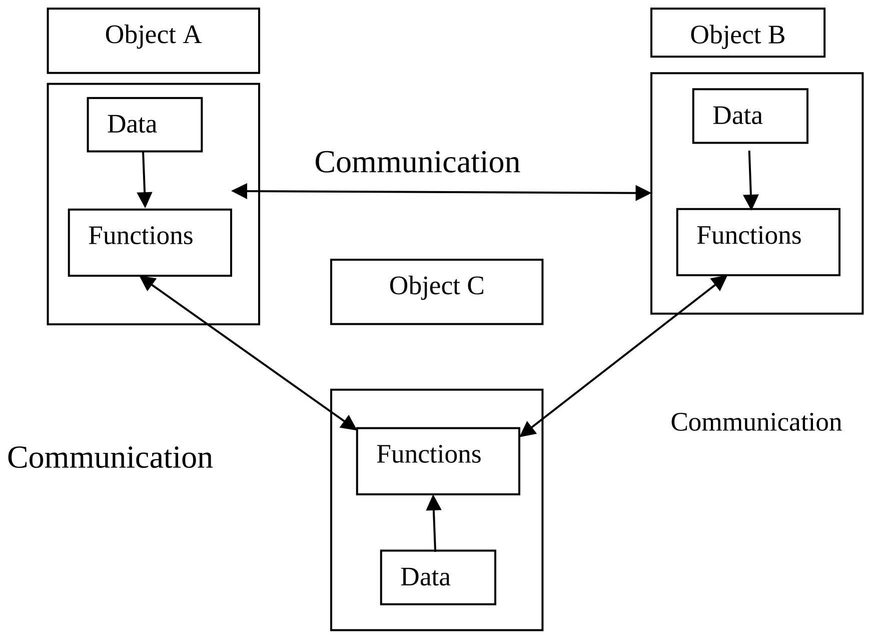


Fig. Organisation of data and functions in OOP

Characteristics of Object-oriented Programming (C++)

- Emphasis is on data rather than procedure.
- Programs are divided into what known as OBJECTS
- Data structures are designed such that they characterize the objects
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions
- New data and functions can be easily added whenever necessary
- Follows BOTTOM-UP approach in program design.

1.2Comparison of Programming Paradigms

❖ **Difference between C & C++**

C is an procedure oriented programming language, whereas C++ is an object oriented programming language .The need for an object oriented environment is because procedure oriented programming language such as C fails to show the desired results for larger programs inters of bug free, complexity & reusable programs. C++ eliminates the pitfalls faced by C language.

There are many differences which makes C++ as an apt alternative to C. There are many concepts in C++ which C language doesn’t support. They are data hiding, data encapsulation, inheritance, polymorphism, classes, objects, operator overloading & message passing.These concepts can eliminate the complexity of C for larger programs.

The main difference between C & C++ is where we use the class (user defined data type), Through class we derive the other concepts. Class is similar to structure in C. In structure only mixed data type can be declared, whereas C++ allows us to declare & define functions in addition to the data's. This makes the main function more simple & grouping all functions into a single class.

Operator overloading concept in C++ makes us to create a new language of or own. Through this we can perform arithmetic operations on variables of class i.e. objects which makes the program easier to understand. In C we can’t perform arithmetic operation on structure variable as we do in C++.

In c for larger programs the presence of redundant codes is unavoidable. This can be eliminated in C++ by following inheritance.

In C there may be accidental invading of codes of one program by another program. This can be eliminated in C++ which supports data hiding which helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.

In C, if we want to add additional features to the program without modifying the function is very difficult. This can be eliminated in C++ which supports inheritance where we can add additional features to an existing class without modifying it. This concept provides the idea of reusability of the programs.

S.No	Procedure oriented Programming (C)	Object Oriented Programming (C++)
1.	Programs are divided into smaller sub-programs known as functions	Programs are divides into objects & classes
2.	Here global data is shared by most of the functions	Objects are easily communicated with each other through functions.
3.	Follows Top-Down Approach	Follows Bottom-Up Approach
4.	Data cannot be secured and available to all the functions.	Data can be secured and can be available in the class in which it is declared.
5.	Here, the reusability is not possible, hence redundant code cannot be	Here, we can reuse the existing one using the Inheritance concept

	avoided.	

1.3 <u>Object oriented Languages:</u>

Object oriented programming languages are divided into two.They are
1.Object based programming languages
2.Object oriented programming languages
<u>Object based programming:</u>
Languages that supports programming with objects are said to be Object based programming languages.They do not inheritance and Dynamic binding.
Object based programming is the style of programming that supports encapsulation and identity.
Major features that are required for this are:
1.Data Encapsulation
2.Data hiding and access mechanism
3.Automatic initialization and clear up of objects
4.Operator overloading
<u>Object oriented programming</u>
_Object oriented programming includes all features of Object based programming with two additional features like inheritance and Dynamic binding.

1.4 <u>Benefits or Advantages of OOP's</u>

- The complexity of software can be managed easily.
- Data hiding concept help the programmer to build secure programs
- Through the class concept we can define the user defined data type
- The inheritance concept can be used to eliminate redundant code
- The message-passing concept helps the programmer to communicate between different objects.
- New data and functions can be easily added whenever necessary.
- OOP's ties data elements more closely to the functions that operates on.

<u>Application of OOP</u>

- Real-time system

- Simulation and modelling
- Object-oriented database
- Hypertext,hypermedia and expertext
- Artificial intelligence and expert system
- Neural network and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD system

1.5 COMPARISON of C with C++

C Language

- Top Down approach
- Procedure oriented
- Variables cannot be dynamically initialised
- Reference variables cocept is not there
- Memory management
  - Allocation: malloc(), calloc()
  - Deallocation: free()
- Type casting:
  Syntax: (typename) expression
- Object Oriented concepts like Inheritanc, polymorphism,dynamic binding, encapsulation etc are not supported
- There is no inline function or friend function
- Input and output:
  - Scanf() and printf() library functions are used for reading input and writing output respectively.
- There is no scope resolution operator to access the global variables in the interior blocks of the functions
- Constructors and destructors are not used
- Object oriented concepts like inheritance, polymorphism, encapsulation, dynamic binding etc are not suppoted

C++ Language

- Bottom up approach
- Object oriented
- Allows dynamic initialisation of variables

Eg:
```
cin>>str;
int len=strlen(str);
```

- Uses reference variables

  Eg:
  ```
  float tot=100
  float & sum=tot;
  // Here the reference variable sum references the variable tot
  ```
- Memory management
  - Allocation: new operator is used
  - Deallocation: delete operator is used
- Type casting:

  Syntax: typename (expression)
- Inline functions : an inline function is a function upon which the compiler has been requested to perform inline expansion

- Eg:
  ```
  #include<iostream.h>
  class inl
  {
  public:
  inline int cube(int a)
  return a*a*a;
  };
   Int main()
  {
  inl o;
  int c=o.cube(3);
  return 0;
  }
  ```
- Input and output:
  - >> Extraction operator for reading input
  - << Insertopm operator for printing the output
    - Eg:
      ```
      cin>> a;
      cout<<a;
      ```
- Scope resolution operator:

The scope resolution operator helps to identify and specify the context to which an identifier refers. Syntax: ::member_name

- Friend functions and virtual functions are used
- Constructors and destructors are not used
- Object oriented concepts like inheritance, polymorphism, encapsulation, dynamic binding etc are suppoted.

Overview of C++ Programming

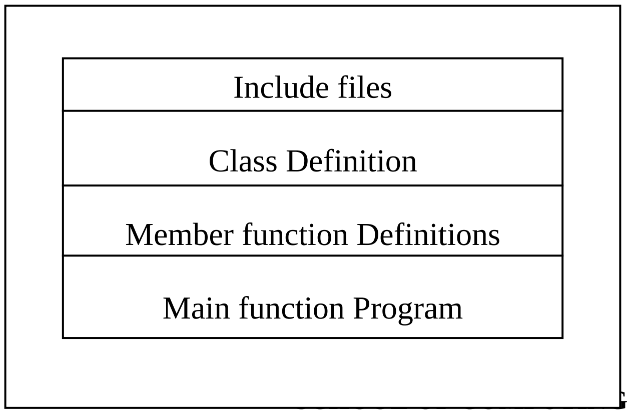
C++ was developed by BJARNE STROUSSTRUP at AT&T BELL Laboratories in Murry Hill, USA in early 1980's.

Striysstryo combines the features of 'C' language and 'SIMULA67' to create more powerful language that support OOPS concepts, and that language was named as "C with CLASSES". In late 1983, the name got changed to C++.

The idea of C++ comes from 'C' language increment operator (++) means more additions.

C++ is the superset of 'C' language, most of the 'C' language features can also applied to C++, but the object oriented features (Classes, Inheritance, Polymorphism, Overloading) makes the C++ truly as Object Oriented Programming language

1.6 Structure of C++ Program



Include files provides instructions to the compiler to link functions from the system library.

Eg: #include <iostream.h>

#include	–	Preprocessor Directive
iostream.h	–	Header File

- ◆ A class is a way to bind and its associated functions together. It is a user defined datatype. It must be declared at class declaration part.
- ◆ Member function definition describes how the class functions are implemented. This must be the next part of the C++ program.
- ◆ Finally main program part, which begins program execution.

```
main( )
{

}
```

Program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical and of the program.

❖ **Input / Output statements**

Input Stream

Syntax:

```
cin >> var1 >> var2 >>;
```

cin – Keyword, it is an object, predefined in C++ to correspond to the standard input stream.

>> - is the extraction or get from operator
Extraction operation (>>) takes the value from the stream object on its left and places it in the variable on its right.

Eg:

```
cin>>x;
cin>>a>>b>>c;
```

- **Output Stream:**

Syntax:

```
cout<<var1<<var2;
```

cout - object of standard output stream
<< - is called the insertion or put to operator
It directs the contents of the variable on its right to the object on its left.
Output stream can be used to display messages on output screen.

Eg:

```
cout<<a<<b;
cout<<”value of x is”<<x;
cout<<”Value of x is”<<x<<”less than”<<y;
```

❖ **Tokens**

The smallest individual units in a program are known as tokens.

C++ has the following tokens:

- Keywords
- Identifiers
- Constants
- Strings
- Operators

  - In C++ program’s written using these tokens, white spaces and the syntax of the language.

- Most of the C++ tokens are basically similar to the C tokens with the exception of some additions and minor modifications.

Keywords

- The keywords implementt specific C++ language features.
- Keywords has a predefined meaning and cannot be changed by the user
- Keywords cannot be used as names for the program variables.
- Additional keywords have been added to ANSI C keywords,in order to enhance its features amd make it an OO language.

Keywords supported by C++ are:

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

The specific C++ Keywords

There are several keywords specific to C++

asn	new	template
catch	operator	this
class	private	throw
delete	protected	try
friend	public	virtual
inline		

Identifiers

Identifiers refer to the names of variables, functions, arrays, classes, etc. created by the programmer.They are the fundamental requirement of any language.

Rules for naming these identifiers are common to both C and C++:

1. Only alphabetic characters, digits and underscores are permitted.
2. The name cannot start with a digit.
3. Uppercase and lowercase letters are distinct.
4. A declared keyword cannot be used as a variable name.

The major difference between C and C++ is the limit on the length of a name i.e., C recognizes only the first 32 characters in a name.Where as C++ has no limit on its length and therefore,all the characters in a name are significant.

Constants

It refers to fixed values that do not change during the execution of a program.C++ recognises all constants available in C.

<u>Types of constants:</u>

- Integer constants - Eg: 123, 25 – without decimal point
- Character constants - Eg: ‘A’, ‘B’, ‘*’, ‘1’
- Real constants - Eg: 12.3, 2.5 - with decimal point

Strings

A sequence of characters is called string. String constants are enclosed in double quotes as follows:

<u>Eg:</u>

“Hello”

Operators

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.

Types of Operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment & decrement Operators
6. Conditional Operators
7. Bitwise Operators
8. Special Operators
9. Manipulators
10. Memory allocate / delete Operators

- **Manipulators**

Manipulators are operators used to format the data display. The commonly used manipulators are endl, setw.

- **endl manipulators**

It is used in output statement causes a line feed to be inserted, it has the same effect as using the newline character “\n” in ‘C’ language.

Eg:

```
#include <iostream.h>
main()
{
        int a=10, b=20;
        cout << “C++ language” << endl;
        cout << “A value: “ << a << endl;
        cout << "B value:" << b << endl;
    }
```

Output:

```
    C++ language
    A value: 10
    B value: 20
```

- **Setw Manipulator**

The setw manipulator is used or specify the field width for printing, the content of the variable.

Syntax: **setw(width);**

where width specifies the field width.

Eg:

```
    int a = 10;
    cout << " A value" << setw(5) << a << endl;
```

Output:

```
    A value      10
```

- **Memory allocate /Delete operators:**
  - **delete** memory release operator
  - **new** memory allocation operator

- **::** **Scope resolution operator**

In C, global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by introducing a new operator ::.

Syntax:

```
        :: variable-name
```

This operatoe allows access to the global version of a variable.

Example program:

```
#include<iostream.h>
int m=10;       //global m;
int main()
{
        int m=20;    //m redecalred, local to main
        {
        int k=m;
        int m=30;       //m declared again, local to inner block

        cout<<"we are in the inner block \n";
        cout<<"k =" << k<< "\n";
        cout<<"m="<<m<<"\n";
        cout<<" ::m = "<< ::m << "\n";
        }

        cout<<"\n we are in outer block\n";
        cout <<"m ="<< m<<"\n";
        cout<< "::m" << ::m << "\n";

        return 0;
}
```

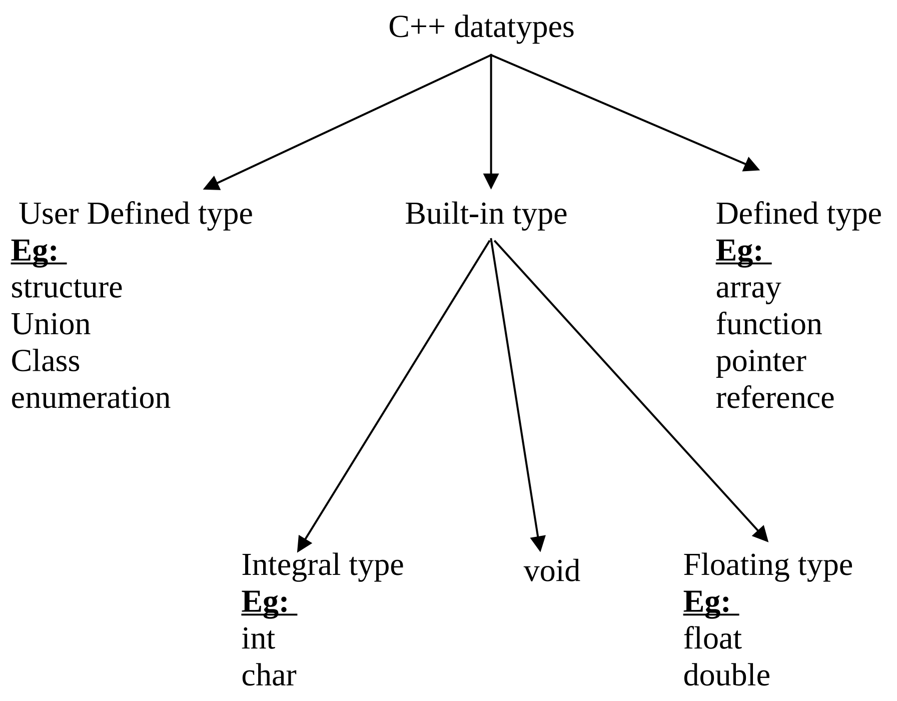
Output:

```
We are in inner block
k=20
m=30
::m=10

We are in outer block
m=20
::m=10
```

Datatypes

It is the type of data, that is going to be processed within the program



Built-in type

Both c and C++ compilers support all the built-in data,with some exceptions in void.

Eg:

```
int x;
float y,z;
char a;
```

Void

Uses of void:

(i) To specify the return type of a function when it is not returning a value.
(ii) To indicate an empty argument list to a function.
Eg:
```
void func(void);
```
(iii) It is used in pointer declaration.

Eg:
```
void *p;     //p becomes pointer
```

Assigning any void pointer to a non-void poiter without using a cast to non-void pointer type is allowed in C.
Eg:
```
void *ptr1;
char *ptr2;
ptr2=ptr1;
```

But this is not allowed in C++.A void pointer cannot be directly assigned to other type pointers in C++.we need to use a cast operator.

For example:
```
void *ptr1;
char *ptr2;

ptr2=(char*)ptr2;
```

- **User Defined type**

- **Structures, Unions and classes**

We have user defined data types such as **struct** and **union** in C. While these data types are legal in C++,some more features have been added to make them suitable for the object-oriented programming. C++also permits us to define another user defined data type known as **class** which can be used just like any other basic data type, to declare variables. The class variables are known as objects, which are the central focus of OOPs.

The only difference between a structure and a class is that, in a class, the member data or function are private by default whereas, in a structure, they are public by default.

The following code segment:

```
Class demo
{
        private:
        int Num1;
        public:
        void func(void);
};
```

can be written as:

```
class demo
{
        int Num1;
        public:
        void func(void);
};
```

The keyword private need not be mentioned because, in a class, the members are private by default.

The code segment can be modified using a structure in the following way:

```
class demo
{
        int Num1;
        public:
```

```
        void func(void);
    };
```

The keyword private need not be mentioned because, in a class, the members are private by default.

The code segment can be modified using a structure in the following way:

```
    struct demo
    {
        void func(void);
        int Num1;
    };
```

The keyword public need not be mentioned because the structure members are public by default.

Enumerated data type

- It provides a way for attaching names to numbers.
- The enum keyword autumatically enumerates a list of words by assigning them values 0,1,2, and so on.
- This facility provides an alternative means for creating symbolic constants (using **#define** preprocessor directive).
- The syntax of enum is similar to that of the **struct** statement.

Syntax:

```
enum enum-type-name { enum-list } enum-variable;
```

- In this form, enum-type-name is optional. However, if you want to use enum type in several places, it is better to use another way of enum declaration:

(Or)

```
enum enum-type-name { enum-list };

//... (and somewhere below)

enum enum-type-name enum-variable;
```

- In the second case enum-type-name cannot be omitted.

Eg:

```
enum shape{circle,triangle,circle};
enum color{yellow,blue,white,red};
enum position{off,on};
```

- The enumerated data types differ slightly in C++ when compared with those in C.In C++,the tag names **shape,color,position** becomes new type names.
- By using these tag names we can declare new variables.

```
shape ellipse;          //ellipse is of type shape
color background;       //background is of type color
```

- C defines the types of **enums** to be ints.
- In C++,each enumerated data type retains its own seperate type i.e., C++ does not permit an int value to be automatically converted to an **enum** value.

Eg:

```
color background = blue;            //allowed
color background = 7;               //error in C++
color background = (color) 7;       //OK
```

Enumerated value can be used in the place of an **int** value.

```
        int c = red;        //valid,color type promoted to int
```

- By default, the enumerators are assigned integer values starting with 0 and each successive enumerator is one larger than the value of the previous one, unless you explicitly specify a value for a particular enumerator i.e., We can over-ride the default by explicitly assigning integer values to the enumerators.

- Enumerators needn't have unique values within an enumeration.
- The name of each enumerator is treated as a constant and must be unique within the scope where the **enum** is defined.
- An enumerator can be promoted to an integer value.

  Eg:

```
    Enum color{red,blue=5,green=9};
    Enum color{red=5,blue,green};
```

  Note:- The subsequent initialised enumerators are larger by one than their predecessors.

- C++ also permits the creation of anonymous **enums** (i.e., **enums** without tag names)

  Eg:

```
    enum{off,on};
```

Here, **off** is 0 and **on** is 1.These constants may be referenced in the same manner as regular constants.

```
    int switch_1=off;
    int switch_2=on;
```

Characteristics:

- An enumeration set can contain duplicate constant values. For example, you could associate the value 0 with two different identifiers in the same set.
- The identifiers in the enumeration list must be distinct from other identifiers in the same scope with the same visibility, including ordinary variable names and identifiers in other enumeration lists.
- Enumeration tags follow the normal scoping rules. They must be distinct from other enumeration, structure, and union tags with the same visibility.

➢ **Derived data types**

Arrays

Like other normal variables, the array variable must be defined before it use.

Syntax:

datatype Arrayname[array-size];

Arraysize – indicates the maximum number of elements the array can hold.

Example:

```
int marks[100]; // Integer array of size 100
float salary[25]; //Floating print array of size 25
char name[50]; //character array of size 50
```

The main drawback of arrays is that the size of the arrays is fixed and it needs to be specified at compile time. Memory can be allocated and reallocated dynamically by using the new and the delete operators.

Functions

Functions are the building blocks of C++ programs. A function groups a number of program statements into a single unit. This unit can be invoked from other parts of the program.

- It is difficult to implement a large program even if it is algorithm is available.
- To implement such a program in an easy manner, it should be split into a number of independent tasks, which can be easily designed, implemented, and managed.
- This process of splitting a large program into small manageable tasks and designing them independently is called Modular Programming.
- A repeated group of instruction in a program can be organized as a function.
- A function is a set of program statements that can be processed independently.

Pointers

C++ adds the concept of constant pointers and pointer to a constant.

Constant pointer

```
Char * const ptr1="GOOD";    //constant pointer
```

We cannot modify the address of pointer1.

Pointer to constant

```
int const * ptr1=&m;     //pointer to a constant
```

Contents of what it points to cannot be changed.

Reference Variables

C++ provides a new kind of variable known as reference variable. A reference variable provides an **alias** (alternative name) for a previously defined variable.

For example, if we make the variable sum a reference to the variable **total,** then **sum** and **total** can be used interchangeably to represent that variable.

A reference variable is created as follows:

datatype & ref_name = var_name;

Eg:

```
float total = 100;
float & sum = total;
cout << sum << total;
```

Both the variables refer to the same data object in the memory i.e.

total, sum

$\boxed{100}$

A major application of reference variables is in passing arguments to functions.

Example program:

```
void f(int & x)          //uses reference

{

  x=x+10;                //x is incremented; so also m

}
int main( )
{

int m=10;
f(m);                    //function call
..................
..................
}
```

When the function call f(m) is executed, the following initialization occurs:

```
int & x=m;
```

Thus x becomes an **alias** of m after executing the statement

```
f(m);
```

When the function increments x, m is also incremented.

- Such functions are also known as **call by reference**.
- The call by reference mechanism is useful in OO programming because it permits the manipulation of objects by reference and eliminates the copying of object parameters back and forth.
- References can be created not only for buit-in data types, but also for user-defined data types such as structures and classes.

➢ Variables

Like in C, in C++ also all variables must be declared before they are used in executable statements. The only difference is, C requires all the variables to be defined at the beginning of a scope. Where as, C++ allows the declaration of variables anywhere in the scope. This means that a variable can be declared right at the place of its first use. This makes the program much easier to write and reduces the errors that may be caused by having to scan back and forth. It also makes the program easier to understand because the variables are declared in the context of their use.

Declaration of Variables

Syntax:

datatype variablename;

A variable is an entity whose value can be changed during program execution and is known to the program by a name. A variable can hold only one value at a time during program execution.

Eg:

int i;

Example program:

```
int main()
{
        int x;                          //declaration
        float sum=0;

        for(int i=1;i<5;i++)            //declaration
        {
                cout<<"enter a value"<<endl;
                cin>>x;
                sum=sum+x;
        }

        float avg;                      //declaration
        avg=sum/(i-1);
        cout<<avg;

    return 0;
    }
```

1.7<u>Basic concepts of OOP's</u>

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

- **Objects**

  - Objects are the basic run time entities in an object-oriented system.
  - Each entity can be treated as object. Object contains data and code to manipulate these data's.
  - They may represent a person, a place, a bank account, a table of data or any item with which the computer must deal or handle.
  - Some objects may correspond to real-world entities such as students,employees,bank accounts,etc.
  - Every object will have a state called **attributes(data)** and the behavior called **operations(function)**.

When a program is executed, the objects interact by sending messages to one another .For example, if 'customer' and 'account' are the two objects in a program, then the customer object may send a message to the account object. Each object contains data and code to manipulate the data. objects can interact without having to know details of each others data or code. It is sufficient to know the type of message accepted, and the type of response returned by the objects.

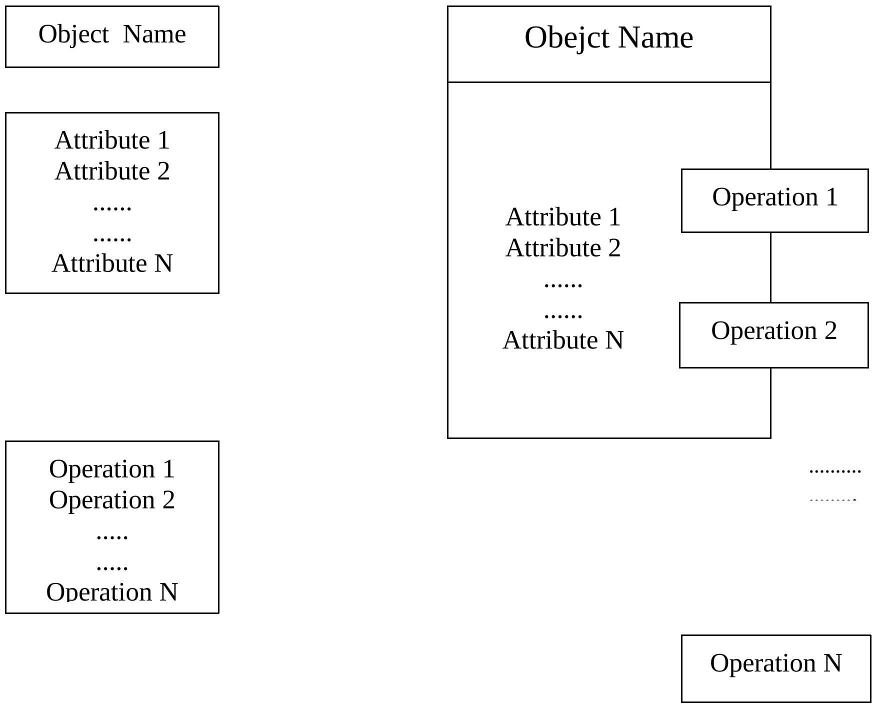
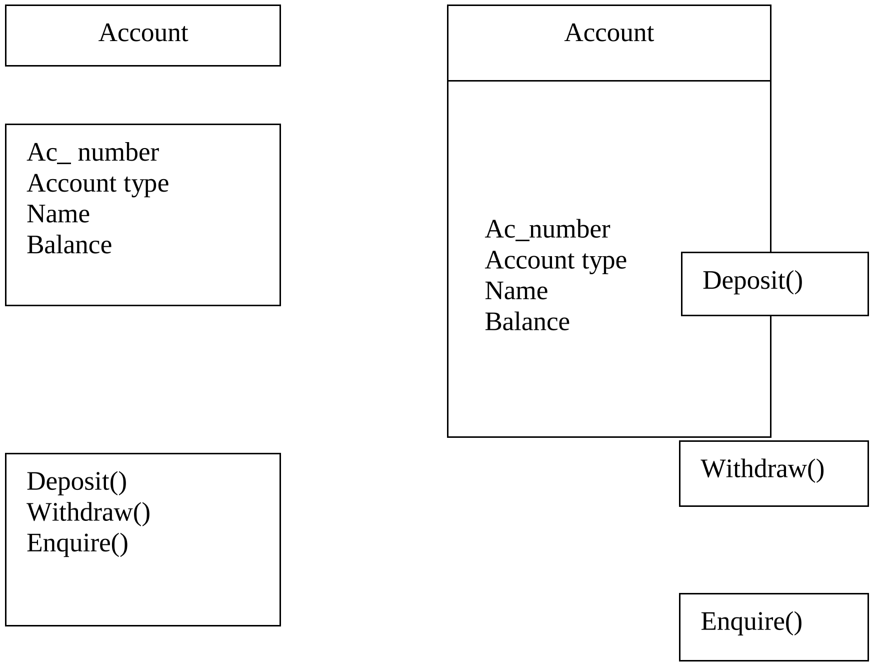


Fig. Two ways of representing an object

Desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off), but your desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune).

Eg:



- Each object will have its own identity though its attributes and operations are same, the objects will never be equal.
- For example in case of a person object, two persons will have same attributes like name,age,sex;but they are not equal.

➢ **Classes**

- We know that objects contain data and function or code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of a class.
- Objects are variables of type class.
- Classes are user defined data types and behaves like the built-in types of a programming language.
- Once a class has been defined, we can create any number of objects associated with that class.
- Thus, a class is collection of objects of similar type i.e., the objects with the same state (attributes) and behavior(operations) are grouped into a class.
- For example, mango, apple and orange are members of class fruit.
- The syntax used to create an object is no different from the syntax used to create an integer object in C.
- If fruit has been defined as a class, then the statement

Eg:

```
fruit mango;
```

will create an object mango belonging to the class fruit.

Examples:

1. Class : person

Object : girl,boy

Attributes : Name,Age

Operations : speak( ),listen( ),walk( )

2. Class : vehicle

Object : car,bus

Attributes : Name,model,color

Operations : start( ),stop( ),accelerate( )

In C++ class can be represented as:

```
Class account
{
Private:
        char name[20];
```

```
            int act_type;
            float balance;
    Public:
            deposit( );
            withdraw( );
    };
```

- It is similar to structure declaration in C.
- It enables the creation of class variables called objects.

Example:

```
            account saving;
            account current;
```

Example programs:

1. /*To find whether given number is even or odd using Class*/

```
#include<iostream.h>
#include<conio.h>
```

```
class even
{
private:
    int a;

public:
    void getdata()
    {
    cout<<"\nenter the number to check: ";
    cin>>a;
    }

    void check()
    {
    if(a%2==0)
    cout<<"\ngiven number is even";
    else
    cout<<"\ngiven number is odd";
    }
};

void main()

{
    clrscr();

    even e;
    e.getdata();
    e.check();

    getch();
}
```

Output:

```
Enter the number to check: 9
Given number is odd
```

2. /*To calculate velocity by using the formula v=u+(a*t) */

```
#include<iostream.h>

#include<conio.h>

class velocity

{

   int v,u,a,t;


   public:

         void getdata( )

         {

         cout<<"Enter the values for initial velocity, acceleration and  time\n";

         cin>>u>>a>>t;

         }

         void calculate( )

         {

         v=u+(a*t);

         cout<<"FINAL VELOCITY= "<<v<<endl;
```

```
        }
};

void main( )
{
        clrscr( );

        velocity ob;
        ob.getdata( );
        ob.calculate( );

        getch( );
}
```

- **Data Encapsulation**
  - Encapsulation is the most basic concept of OOP.

- The wrapping up of data and functions into a single unit(called class) is known as encapsulation.
- Data encapsulation is the most striking feature of a class.
- It is a mechanism that associates the code and the data it manipulates and keeps them safe from any external interface and misuse.
- The data is not accessible to the outside world (entire program), and only those functions which are wrapped in the class can access it.
- These functions provide the interface between the object’s data and the program.
- This is also known as “data hiding “ or “information hiding”.

➢ **Data Abstraction**

- Abstraction refers to the act of representing essential features without including the background details or explanations.

To understand this concept more clearly, take an example of 'switch board'. You only press particular switches as per your requirement. You need know the internal working of these switches. What is happening inside is hidden from you. This is abstraction, where you only know the essential things to operate on switch board without knowing the background details of switch board.

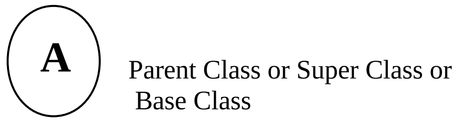
- Classes use the concept of abstraction.
- Since the classes use the concept of abstraction, they are also known as Abstract Data Types (ADT).

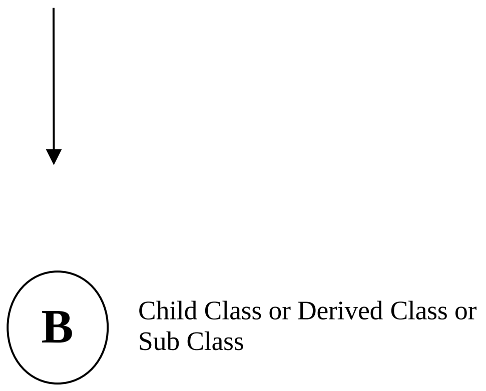
➢ **Inheritance**

- It is the process by which objects of one class acquire the properties of objects of another class.
- It provides **reusability** i.e., we can add additional features to an existing class without modifying it.
- It is the capability to define a new class in terms of an existing class.
- This is possible by deriving a new class from the existing one.
- An existing class is known as a **base** class and the new class is known as **derived** class.

Consider an **example,**

- When the class child inherits the class parent - the class child is referred to as derived class (**sub**-class), class parent is referred to as base class (**super** class).





- Child has two parts:
  - a derived part and
  - Incremental part (new code written specifically for child class).
- Child has its own properties, in addition to those acquired from its parent.

Example:

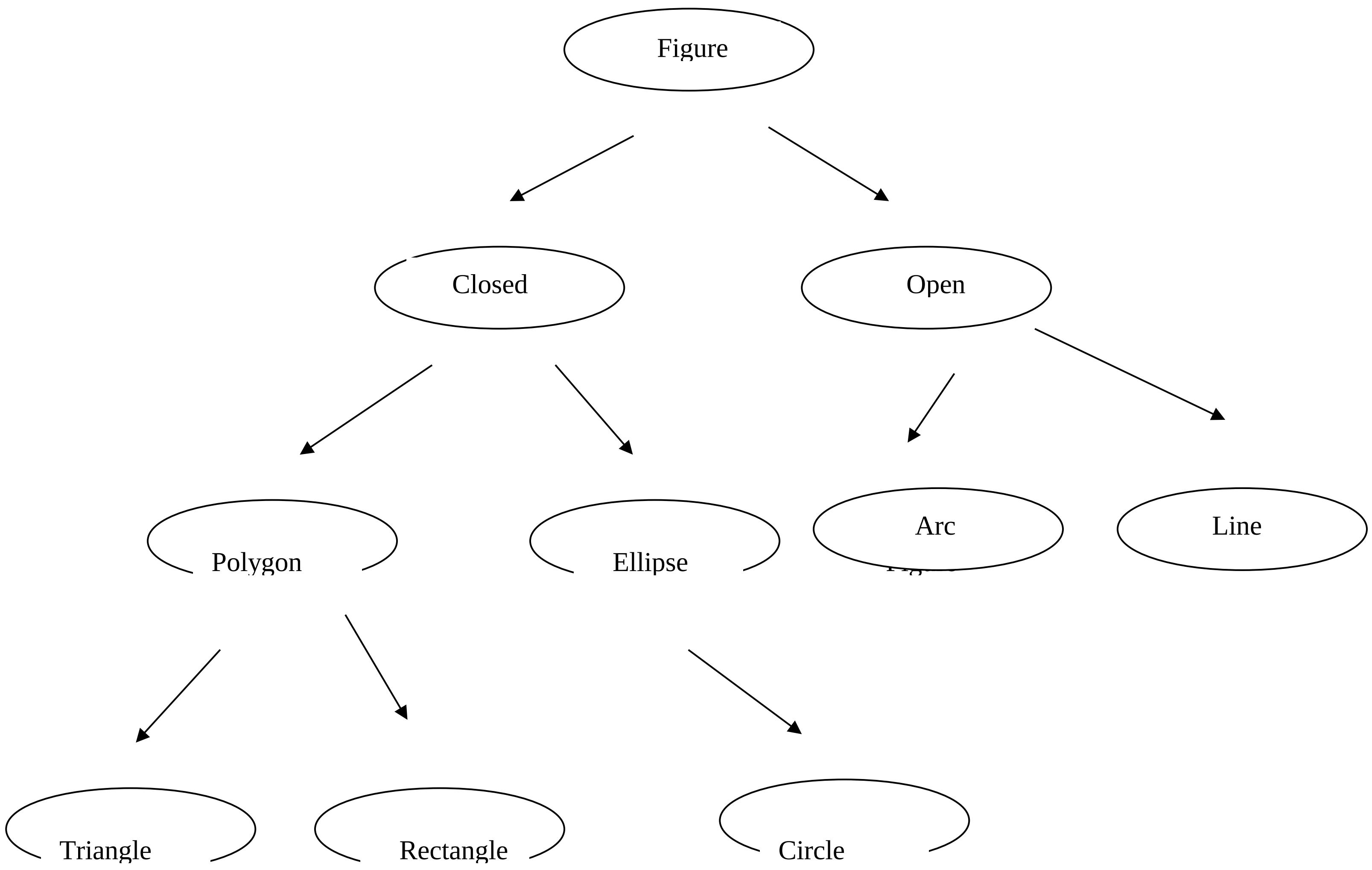


Figure
Closed
Open
Polygon
Ellipse
Arc
Line
Triangle
Rectangle
Circle

C++ supports such hierarchical classification of classes:

- Single inheritance
- Multiple inheritance
- Multi-level inheritance
- Hierarchical inheritance

➢ **Polymorphism**

- Polymorphism is a key to the power of OOPs.
- It is a Greek word, means the ability to take more than one form(*many forms)*.
- It is the concept that supports the capability of data to be processed in more than one form i.e., It allows a single name to be used for more than one related purpose, which are technically different.
- For **example**, an operation may exhibit different behavior in different instances.
- The behavior depends upon the types of data used in the operation.

The following are the different ways of achieving polymorphism in a C++ program:

- Function overloading
- Operator overloading
- Dynamic binding

Example:

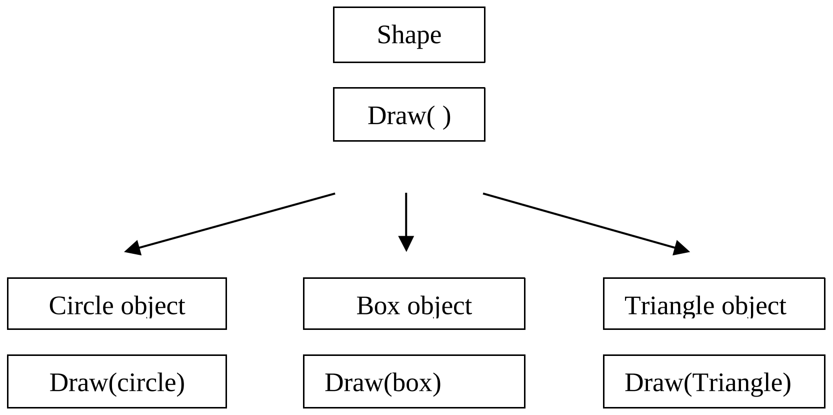


Fig. Polymorphism

Fig. illustrates that a single function name can be used to handle different number and different types of arguments. This is similar to a particular word having several different meanings depending on the context. Using a single function name to perform different types of tasks is known as *function overloading*.

- Let us consider the operation of addition.
    - For two numbers, the operation (+) will generate a sum.
    - If the operands are strings then the operation (+) would produce a third string by concatenation. This is known as *operator overloading*.

➢ **Dynamic binding**

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

<u>Example:</u>

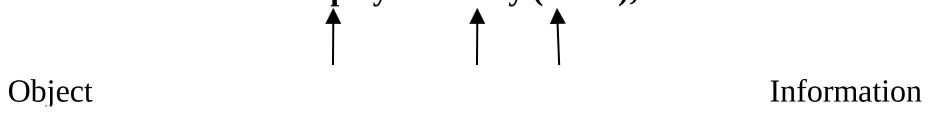
- Consider the procedure "draw" in the above **fig.** By inheritance every object will have this procedure.
- Its algorithm is however, unique to each object and so the draw procedure will be redefined in each class that defines the object.
- At run-time, the code matching the object under current reference will be called.

- **Message passing**

- An OO program consists of a set of objects that communicate with each other.
- The process of programming in an OO language, therefore involves the following steps:
    - Creating classes that define objects and their behavior.
    - Creating objects from class definitions and
    - Establishing communication among objects.
- A message for an object is a request for execution of a procedure and therefore will invoke a function (procedure) in the receiving object that generates the desired result.
- Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another.
- Message passing involves specifying the name of the objects, the name of the function (message) and the information to be sent.

<u>Example:</u>

employee . salary (name);



Message

Objects have a life cycle. They can be created and destroyed. Communication with the object is feasible as long as it is alive.

Structure:

Structure is the collection of elements of different datatypes.

Structure Initialization:

```
  #include<iostream.h>
Struct date
{
  int day;
  int month;
  int year;
};
Void main()
{
date d1={12,3,2013};
date d2={14,3,2013};
cout<<"bday";
cout<<d1.day<<"-"<<d1.month<<"-"<<d1.year;
cout<<"today's date";
cout<<d2.day<<"-"<<d2.month<<"-"<<d2.year;
getch();
}
```

Array of structure

```
#include<iostream.h>
Struct student
{
  char name[10];
  int mark[12];
};
Void main()
```

```
{
Student s[10];
int n;
cout<<"enter no of students";
cin>>n;
for(i=0;i<n;i++)
{
  Cout<<"enter the name";
  Cin>>s[i].name;
  Cout<<"enter mark";
  Cin>>s[i].mark;
}
for(i=0;i<n;i++)
{
  Cout<<"name" <<s[i].name;
  Cout<<"mark">> s[i].mark;
  }
}
```

FUNCTIONS

- Member functions
- Main function
- Function prototyping
- Call by reference
- Return by reference
- Inline functions
- Default arguments
- Constant arguments
- Function overloading
- Friend and virtual functions

POINTERS AND REFERENCES EXAMPLE

/* CALL BY REFERENCE USING POINTER CONCEPT*/

```
#include<iostream.h>
#include<conio.h>
class ptr
{

public:
void swap(int *a,int *b)
{
int t;
t=*a;
*a=*b;
*b=t;
}
void show(int *x,int *y)
{
cout<<*x<<" .."<<*y;
}
};
void main()
{
ptr o;
clrscr();
int x,y;
cin>>x>>y;
o.swap(&x,&y);
o.show(&x,&y);
getch();
}
```

/*RETURN BY REFERENCE*/

```
#include<iostream.h>
#include<conio.h>
class abc
{
public:
int & max(int &x,int &y)
{
if(x>y)
return x;
else
return y;
}
};
void main()
{
int a,b;
clrscr();
abc o;
a=10;
b=20;
int &ar=a;
int &br=b;
int & u=o.max(a,b);
if(u==ar)
cout<<"a is big";
else if(u==br)
cout<<"b is big";
getch();
}
```

//DEFAULT ARGUMENTS

```
#include<iostream.h>
#include<conio.h>
class abc
{
public:
void sum(int a,int b,int c=20);
};
void abc::sum(int a,int b,int c)
{
int d=a+b+c;
cout<<d;
}

void main()
{
clrscr();
abc o;
o.sum(3,4);
getch();
}
```

INLINE FUNCTIONS:

```
#include<iostream.h>
#include<conio.h>
class inl
{
public:
inline int cube(int  a)
{
return a*a*a;
};
int main()
{
inl  o;
o.cube(3);
return 0;
}
```

CONSTANT ARGUMENTS

Syntax:

Returntype function_name(**const** type variable**1**,... **const** type variable**n**)

The qualifier const tells the compiler the the function should not modify the argument

```
#include<iostream.h>
#include<conio.h>
```

```
class abc
{
public:
void sum(int a,int b,const int c=20);
};
void abc::sum(int a,int b,const int c=20)
{
        cout<<enter a and b values;
        cin>>a>>b;
int d=a+b+c;
cout<<d;
}

void main()
{
clrscr();
abc o;
o.sum;
getch();
}
```