Classes

- A class is a user defined data type.
- It contains data and code to manipulate data.
- Class contains a collection of objects.

Syntax

```
class classname
{
        accesss specifier:
                variable declaration;
                method declaration;
        accesss specifier:
                variable declaration;
                method declaration;
};
```

Adding variables:

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables, because they are created whenever an object of the class is instantiated.

```
class student
{
        private:
                int rollno;
                char name[20];
};
```

Adding methods:

Methods are added for manipulating the data contained in the class. Methods are declared inside the body of the class, usually after the declaration of instance variables.

```
class student
{
        private:
                int rollno;
                char name[20];
        public:
                void print()
                {
                        cin>>rollno>>name;
                        cout<<rollno<<name;;
                }
};
```

Example program for classes

```
#include<iostream.h>
#include<conio.h>
class student
{
        private:
                int rollno;
                char name[20];
                int mark1,mark2,total;
                float avg;
        public:
                void getdata()
                {
                        cout<<"Enter your rollno";
                        cin>>rollno;
                        cout<<"Enter your name";
                        cin>>name;
                        cout<<"Enter your marks";
                        cin>>mark1>>mark2;
                }
                void calculate()
```

```
        {
                total=mark1+mark2;
                avg=total/2;
        }
        void print()
        {
                cout<<rollno<<name<<mark1<<mark2;
                cout<<total<<avg;
        }
};
void main()
{
        student ob;
        ob.getdata();
        ob.calculate();
        ob.print();
}
```

Access Specifiers

Access Specifier	Description
Public	Can be accessed from anywhere, including outside the class
Private	Can only be accessed from inside the class to which it belon s
protected	Can be accessed from within the class to which it belongs, or a type derived from that class

Function and Data Members

Definition:

- A function is a set of program statements that can be processed independently.
- A function can be invoked by a function call.
- A function is a set of instructions to perform a specified task.

Syntax:

```
return-type methodname(parameter-list)
{
        //function body
}
```

Example:

```
void swap(intx,int y)
{
        int temp ;
        temp=x ;
        x=y ;
        y=temp ;
}
```

Parameter passing:

Parameter passing is a mechanism for communication of data and information between the calling function(caller) and the called function(callee).

Parameter passing methods:

1. Pass by value.
2. Pass by address.
3. Pass by reference.

1)Pass by value:

While calling the function value is passed.

```
#include<iostream.h>
#include<conio.h>
void swap(int x,int y)
{
        int temp=x ;
        x=y ;
        y=temp ;
        cout<<”Values after swapping ”<<x<<endl<<y;
}
void main()
{
        int a,b;
        cout<<”Enter the values to swap”;
        cin>>a>>b;
        swap(a,b);
        getch();
}
```

2)Pass by value:

While calling the function address is passed.

```
#include<iostream.h>
#include<conio.h>
void swap(int *x,int *y)
{
        int temp=*x ;
        *x=*y ;
        *y=temp ;
        cout<<”Values after swapping ”<<x<<endl<<y;
}
void main()
{
        int a,b;
        cout<<”Enter the values to swap”;
        cin>>a>>b;
        swap(&a,&b);
        getch();
}
```

3)Pass by reference:

This process has the functionality of pass by address and syntax of pass by value.

```
#include<iostream.h>
#include<conio.h>
void swap(int &x,int &y)
{
        int temp=x ;
        x=y ;
        y=temp ;
        cout<<"Values after swapping "<<x<<endl<<y;
}
void main()
{
        int a,b;
        cout<<"Enter the values to swap";
        cin>>a>>b;
        swap(a,b);
        getch();
}
```

Default Arguments

- In a c++, function call, when or more arguments are omitted, the function may be defined to take default values for the omitted arguments by providing the default values in the function prototype.
- To establish a default value, the function prototype or the function definition must be used.
- Parameters without default arguments are placed first, and those with default values are placed later.

Example program for default arguments

```
#include<iostream.h>
#include<conio.h>
void print(char='-',int=70);
void main()
{
        print();
        print('!');
        print('*',40);
}
void print(char ch,int count)
{
        cout<<endl;
        for(i=0;i<count;i++)
        cout<<ch;
}
```

output:

```
----------------------------------------------------------------------------
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
****************************************
```

Function Overloading

Definition

- Using the same function name and performing different process.
- When a function is redefined with different set of arguments,then it is known as overloaded function.This process is known as function overloading.

Function name is same, so functions invoked on the basis of,

**based on number of arguments*

**based on data-type of the arguments*

Example program for function overloading

```
#include<iostream.h>
#include<conio.h>
class example
{
        public:
                        void area(int side)
                        {
                                int result1=side*side;
                                cout<<"Area of square"<<result1;
                        }
                        void area(int length,int breadth)
                        {
                                int result2=length*breadth;
                                cout<<"Area of rectangle"<<result2;
                        }
                        void area(float radius)
                        {
                                int result3=3.14*radius*radius;
                                cout<<"Area of circle"<<result3;
                        }
};
void main()
{
        clrscr();
        example ob;
        ob.area(2);
        ob.area(4,8);
        ob.area(2.0);
        getch();
}
```

Friend Functions

- The Concept of encapsulation and data hiding dictate that non member functions should not be allowed to access an object's private and protected members.
- Using friend function or friend class, a non member function can able to access private data members.
- A function declaration must be prefixed by the keyword friend where as the function definition must not.
- A function can be friend to multiple classes.
-

Friend Function Characteristics

- The scope of a friend function is not limited to the class in which it has been declared as a friend.
- A friend function cannot be called using the object of that class .It can be invoked like a normal function with out the use of any object.
- It can be declared in the private part or public part of a class without affecting it's meaning.

Friend Function Example Program

```
#include<iostream.h>
#include<conio.h>
class sample
{
        private:
                int a,b;
        public:
                void setdata()
                {       a=10;
                        b=10;
                }
                friend float mean(sample s);
};
float mean(sample s)
{
        return float(s.a+s.b)/2.0;
}
void main()
{
        clrscr();
        sample ob;
        ob.setdata();
        float result= mean(x);
        cout<<result;
        getch();
}
```

Friend Function common to two classes

```
#include<iostream.h>
#include<conio.h>
class first
{
        private:
                int a;
        public:
                void getdata()
                {       cin>>a;
                }
                friend void add(first ob1,second ob2);
};
class second
{
        private:
                int b;
        public:
                void getdata()
                {       cin>>b;
                }
                friend void add(first ob1,second ob2);
};
void add(first ob1,second ob2)
{
        int c= ob1.a+ob2.b;
        cout<<c;
}
void main()
{
        clrscr();
        first ob1;
        second ob2;
        ob1.getdata();
        ob2.getdata();
        add(ob1,ob2);
        getch();
}
```

Const Functions

Definition: If a function does not alter any data in the class, then we may declare it as a const member function.

Syntax:

```
return-type function-name(arg1,arg2….)const
{
    function-body
}
```

Example:

```
void print()const
{
    cout<<rollno;
}
```

Const Functions

```
#include<iostream.h>
#include<conio.h>
class example
{
    public:
        void print(int rollno)const
        {
            cout<<rollno;
            rollno=4;
        }
}
void main()
{
    example ob;
    ob.print(2);
}
```

Volatile Functions

Definition:

- A member function can also be declared as volatile if it is invoked by a volatile object.
- A volatile object's value can be changed by external parameters which are not under the control of the program.

Example:

An object taking an input from a Network Interface card does not take input from our program.

Syntax:

```
class example
{   public:
        void print() volatile
        {
            //function body
        }
}
```

Static Members

Definition:

Static variables have a specific property that they are initialized only once (when the function is called for the first time).They retain their values between function calls.

- All static members are stored with the global variables at a permanent location.
- A static member defined inside a class is also stored at a permanent location.
- Static members of class are a little different from global data members, differences are
  - ✓ Static data members are stored at a location where they are retained throughout the execution of the program and are not stored with class objects.
  - ✓ Static data members are stored only as a single copy. This single copy of the static data member is shared between all the objects of the class.
  - ✓ All the static members are initialized to zero at the time of declaration by the c++ compiler.

Example program for static member

```
#include<iostream.h>
#include<conio.h>
class example
{
        private:
                        static int a;
                        int b;
        public:
                        void getdata(int x,int y)
                        {
                                a=x;
                                b=y;
                        }
                        void print()
                        {
                                cout<<"Static member value is"<<a;
                                cout<<"Nonstatic member value is"<<b;
                        }
};
void main()
{
        example ob1,ob2;
        ob1.getdata(5,5);
        ob1.print();
        ob2.getdata(10,10);
        ob2.print();
        ob1.print();
        getch();
}
```

output:

```
Static member value is 5
Nonstatic member value is 5
Static member value is 10
Nonstatic member value is 10
Static member value is 10
Nonstatic member value is 5
```

Objects

- Instance of a class.
- Real world entities that may represent a place, a person etc.

Example program for Array of objects

```
#include<iostream.h>
#include<conio.h>
class student
{
        private:
                        int rollno;
                        char name[20];
                        float total;
        public:
                        void getdata()
                        {
                                cout<<”Enter your rollno”;
                                cin>>rollno;
                                cout<<”Enter your name”;
                                cin>>name;
                                cout<<”Enter your mark”;
                                cin>>mark;
                        }
                        void print()
                        {
                                cout<<rollno<<endl<<name<<endl<<mark;
                        }
};
void main()
{
        clrscr();
        student ob[5];
        for(int i=0;i<5;i++)
        {
                ob[i].getdata();
                ob[i].print();
        }
        getch();
}
```

Pointers and Objects

- Pointers are variables that contains addresses.
- A pointer to an object contains an address of that object.
- Pointer increment will increment the address by the size of the object.
- The size of the object is determined by the size of all of it's non-static data elements.

Example program for Pointer to objects

```
#include<iostream.h>
#include<conio.h>
class student
{
        private:
                        int rollno;
                        char name[20];
        public:
                        void print()
                        {
                                cout<<rollno<<endl<<name<<endl;
                        }
};
void main()
{
        clrscr();
        student *ob;
        (*ob).rollno=5;
        (*ob).name="Gowri Shankar";
        (*ob).print();
        ob->rollno=5;
        ob->name="Arun Kumar";
        ob->print();
        getch();
}
```

Constant Objects

- Const objects are objects that are not modifiable.
- Functions that are defined as const can be accessed by const objects. Even public variables of the object are not modifiable.
- A normal member function cannot be invoked by const objects.

Example program for Constant Objects

```
#include<iostream.h>
#include<conio.h>
class student
{       private:
                        int rollno;
                        char name[20];
                        float total;
        public:
                        void getdata()const
                        {
```

```
                cout<<"Enter your rollno";
                cin>>rollno;
                cout<<"Enter your name";
                cin>>name;
                cout<<"Enter your mark";
                cin>>mark;
        }
        void print()
        {
                cout<<rollno<<endl<<name<<endl<<mark;
        }
};
void main()
{
        clrscr();
        student ob1;
        ob1.getdata();
        ob1.print();
        const student ob2= ob1;
        ob2.print();
        ob2.getdata();
        getch();
}
```

Nested Classes

- Nested classes are nothing but defining a class inside another class.

Example program for Nested Classes

```
#include<iostream.h>
#include<conio.h>
class a
{
    private:
                class b
                {
                        private:
                                int x;
                        public:
                                void insideprint(int temp)
                                {
                                        x=temp;
                                        cout<<x;
                                }
                };
    public:
                b ob1;
                void outsideprint(int y)
                {
                        ob1.insideprint(y);
                }
};
void main()
{
    a ob2;
    ob2.outsideprint(5);}
```

Local Classes

- For the sake of completeness, c++ allows classes to defined inside functionsas well. These classes are called local classes.
- The scope of the class is confined to the function body.
- The function in which the class is defined is not a member function of the class. It cannot access private variables of the class. One way to access the private variables of the class is to become a friend of it.

Example program for Local Classes

```
#include<iostream.h>
#include<conio.h>
void main()
{
        clrscr()
        int a=100;
        void example(a);
        getch();
}
void example(int roll)
{
        int b=200;
        static int c=300;
        class student
        {
                public:
                        int rollno;
                        char name[20];
                        void print()
                        {
                                cout<<b;
                                cout<<::c;
                                cout<<rollno<<endl<<name;
                        }
        };
        student ob;
        ob.rollno=roll;
        ob.name=”shankar”;
        ob.print();
}
```