Unit-3

Classes and Objects

Date.
Page No. 217

⊛ Classes / Specifying a class:

A class is a collection of objects of similar type. Classes are also user-defined data types and behaves like a built-in types of programming language. A class is a way to bind data and its associated functions together.

Class allows the data to be hidden if necessary from external use. A class is an extension of the idea of structure used in C. It is a new way of creating and implementing a user-defined data type.

While defining a class, a class specification has two parts:

i) Class declaration.
ii) Class function definitions.

The class declaration describes the type and scope of its members. The class function definitions describe how the class functions are implemented.

The general form of class declaration is:

```
class class_name
{
    private:
        variable declarations;
        function declarations;
    public:
        variable declarations;
        function declarations;
};
```

The class declaration is similar to a struct declaration. The keyword class is used during declaration of class.

Date.
Page No.

The class body contains the declaration of variables and functions. These functions and variables are collectively called class members. These functions and variables are grouped under two sections namely private and public to denote which of the members are private and which are public. The keywords private and public are known as visibility labels followed by a colon. The body of class is terminated by semicolon.

Data members → The variables declard inside the class are known as data members and the functions are known as member functions. Only the member functions can have access to private data members and private functions. Howeves, the public members (both functions and data) can be accessed from outside the class.

A simple class example:

```
class item {
    int number;          // Variable declaration
    float cost;          // Private by default
public:
    void getdata(int a, float b);
    void putdata(void);
};                       // ends with semicolon
/* function declaration */
/* using prototype */
```

If private or public is not specified then data members are private by default. The data members are usually declared as private and member functions as public.

Date.
Page No.

⊛ Accessing Class Members:

As we know that private data of a class can be accessed only through the member functions of that class. The main() cannot contain statements that access member functions of class directly, so to access member functions of class call of member function is needed in main(). The following is the format for calling member function:

```
object_name.function_name(actual arguments);
```

For example:

```
#include<iostream>
using namespace std;
class item{
    int number;
    float cost;
public:
    void getdata(int a, float b);
    void putdata(void);
};
int main(){
    item x;
    x.getdata(100, 75.5);
    x.putdata();
}
item::putdata{
    int a=100;
    float b=75.5;
    number=a;
    cost =b;
    cout<<"number is"<<number;
    cout<<"cost is"<<cost;
}
```

(this part is not required)

Date.
Page No.

In the example getdata() assigns the value 100 to number and 75.5 to cost with the help of object x.

Similarly x.putdata(); would display the values of data members. int a=100; and float b=75.5; is not necessary in program we can perform directly cout operations.

⊛ Defining member functions:

Member functions can be defined in two places: Outside the class definition and inside the class definition. Member functions that are declared inside a class have to be defined seperately outside the class. We use scope resolution operator(::) for defining member function outside the class. the general form of a member function defination is:

```
return_type class_name::function_name(arg.declaration)
{
    function body
}
```

The membership label class_name:: tells the compiler that the function function_name belongs to the class class_name. That is, the scope of the function is restricted to the class_name specified in the header line.

Creating Objects:

The creating of class does not define any objects but the class variable only specifies what they will contain. Once a class has been declared, we can create variable of that type by using the class name. In C++ the class variables are known as objects.

For example:

```
#include<iostream>
using namespace std;
class item{
    int number;
    float cost;
    public:
        void getdata(int a, float b);
        Void putdata(void);
};
int main(){
    item x;
    x.getdata(100, 75.5);
    x.putdata();
}
item::putdata{
    cout<<"number is"<<number;
    cout<<"cost is"<<cost;
}
```

In the above example `item x;` creates a variable of type item. We know that in C++ the class variables are known as objects. Therefore, x is called an object of type item.

Date.
Page No.

⊛ Access Specifiers / Access Modifiers in C++ :

Access specifiers are used to implement an important feature of object oriented programming known as data hiding. There are 3 types of access specifiers available in C++ which are as follows:

i) **Public** → All the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

ii) **Private** → The class members declared as private can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

iii) **Protected** → Protected access specifier is similar to that of private access specifiers, the difference is that the class member ~~det~~ declared as protected are inaccessible outside the class but they can be accessed by any subclass (derived class) of that class.

Date.
Page No.

Some programs related to class and object:

1) Program for calaulating area and perimeter by user input data using concept of class and object.

Soln

```
#include<iostream>
using namespace std;

class arpr{
    int l,b;
    public:
    void setdata();
    void area();
    void perimeter();
};

void arpr::setdata()
{
cout<<"Enter length and breadth of rectangle"<<endl;
cin>>l>>b;
}

void arpr::perimeter()
{ int perimeter;
  perimeter=2*(l+b);
  cout<<"Perimeter of rectangle is"<<perimeter;
}
void arpr::area()
{ int area;
  area = l*b;
  cout<<"Area of rectangle is"<< area<<endl;
}

int main(){
    arpr s;
    s.setdata();
    s.area();
    s.perimeter();
}
```

Date.
Page No.

2> Program to display general information of ~~person~~ student like account no., phone no, roll number using class and objects

Soln

```
#include<iostream.h>
#include<conio.h>
class info{ private:
int accno, ph, roll;
public:
void setinfo(int a, int p, int r)
{ accno = u;
ph = p;
roll = r;
}
void showinfo(){
cout<<endl<<"Account no is"<<accno;
cout<<endl<<"Phone no. is"<<ph;
cout<<endl<<"Roll no is"<<roll;
}
void getinfo();
};
void info::getinfo(){
cout<<endl<<"Enter account number, phone number, roll number";
cin>>accno>>ph>>roll;
}
int main(){
info i1, i2;
clrscr();
i1.setinfo(905 , 984 , 22);
i1.showinfo();
i2.getinfo();
i2.showinfo();
}
```

Date.
Page No.

3) Program to convert indian currency into nepali currency using concept of class and objects.

Soln

```
#include<iostream>
using namespace std;
class conversion{ int a;
public:
void getdata(){
cout<<"Enter Indian currency";
cin a;
}
void setdata(int b){
b=a;
}
void display(){
cout<<"Entered INR "<<b;
}
void conv(){
int c;
c=b*1.6;
cout<<endl<<"The INR in RS "<<c;
};
int main(){
conversion s;
clrscr();
s.getdata();
s.setdata();
s.display();
s.conv();
getch();
return 0;
}
```

Date.
Page No.

```
int main(){
    set a;
    a.input();
    a.display();
    return 0;
}
```

Example 2:

```
#include<iostream>
using namespace std;
class oddeven(){
    int x;
    public:
    void getdata();
    void showdata();
    int check();
};

void oddeven::getdata()
{
cout<<"Enter any number"<<endl;
cin>>x;
}

int oddeven::check(){
    if(x%2==0)
    cout<<"even";
    else
    cout<<"odd";
}

void oddeven::showdata(){
    cout<<"The given number is";
    check();
}

int main(){
    oddeven a;
    a.getdata();
    a.showdata();
    return 0;
}
```

Date.
Page No.

⊛ Arrays within a class:

The arrays can be used as a member variables in a class. The following class definition is valid.

```
const int size=10; //provides value for array size
class array{ int a[size]; //'a' is int type array
public:
  void setval(void);
  void display(void);
};
```

The array variable a[] declared as private member of the class array can be used in the member functions, like any other array variable. We can perform any operation on it.

Example:

```
#include<string.h>
#include<conio.h>
#include<iostream>
  using namespace std;

class student{
    char name[25], address[30];
    int roll;
  public:
    void getdata();
    void showdata();
};
void student::getdata(){
    cout<<"Enter name";
    cin>>name;
    cout<<"Enter address";
    cin>>address;
    cout<<"Enter roll";
    cin>>roll;
}
```

Date.
Page No.

```
void student::showdata(){
        cout<<"Name is:"<<name<<endl;
        cout<<"Address is:"<<address<<endl;
        cout<<"Roll is:"<<roll<<endl;
                }
const int max=3; //Global variable declaration.
int main(){     drscr();
for(int i=0; i<max; i++)
{
cout<<"Info                student:"<<i+1<<endl;
Pri[i].getdata();
}
for(i=0; i<max; i++)
{
cout<<endl<<"Information is"<<i+1<<endl;
Pri[i].putdata();
}
getch();
return 0;
}
```

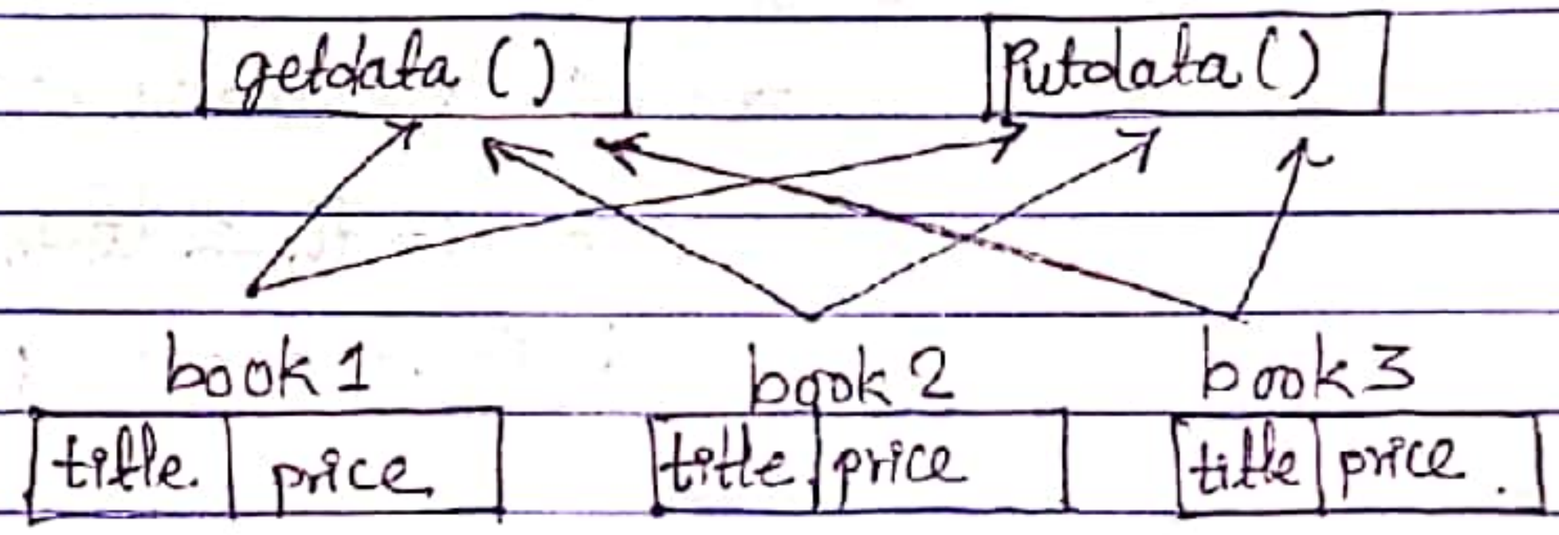
⊛ Memory allocation for Objects:

Before using a member of a class, it is necessary to allocate the required memory space to that member. The memory space is allocated to the data members of a class only when an object of the class is declared, and not when the data members are declared inside the the class. i.e, Objects create space for the data. Space is not created even if we create the class, space is created only when object is declared in main function.

Date.
Page No.

On the other hand, the memory space for the member functions is allocated only once when the class is defined. In other words, there is only a single copy of each member function, which is shared among all the objects.

Let we take the three objects namely book1, book2 and book3 of the class book having individual copies of the data members title and price. There is only one copy of the member functions getdata() and putdata() but that is shared by all the three objects, as illustrated in the diagram below:



Memory allocation for the objects of the class book.

Arrays of Objects:

We know that an array can be of any data type including struct. Similarly we can also have arrays of variables that are of the type class. Such variables are called arrays of objects.

Consider the following class definition:

```
class employee {
        char name[30];
        float age;
    public:
        void getdata(void);
        void putdata(void);
};
```

Date.
Page No.

The identifies employee is a user-defined data type and can be used to create objects that relate to different categories of the employees.

```
int main() {
    employee manager[3]; //array of manager
    employee foreman[15]; //array of foreman
    employee worker[75]; //array of worker
}
```

Here, in above example the array manager contains three objects namely, manager[0], manager[1] and manager[2], of type employee class. Similarly the foreman array contains 15 objects (foreman) and the worker array contains 75 objects (workers).

Since an array of objects behave like any other array, we can use the usual array-accessing methods to access individual elements, and then the dot member operator to access the member functions. For example, the statement;

```
manager[i].putdata();
```

will display the data of the ith element of the array manager. That is this statement requests the object manager[i] to invoke the member function putdata().

Date.
Page No.

Static data members:

When we declare a normal variable (data member) in a class, different copies of those data members create with the associated objects.

In some cases when we need a common data member that should be same for all objects, we can not do this using normal data members. To fulfill such cases, we need static data members.

Definition → It is variable which is declared with the static keyword, it is also known as class member, thus only single copy of the variable creates for all objects.

Declaration →

```
static data_type member_name;
```

Definition → It should be defined outside of the class following this syntax:

```
data_type class_name :: member_name = value;
```

If we are calling a static data member within a member function, member function should be declared as static.

Example:

```
#include<iostream>
using namespace std;
class demo { private:
        static int x;
public:
        static void fun()
{
cout<<"Value of x:"<<x<<endl;
}
};
```

Date. _____
Page No. _____

```
int demo::x=10; // defining
int main(){
    demo x;
    x.fun();
    return 0;
}
```

Output:
Value of x: 10

Accessing static data member without static member function

A static ~~mem~~ data member can also be accessed through the class name without using the static member function using scope resolution operator (::) as in the following example:

```
#include <iostream>
using namespace std;
class demo{ public:
    static int abc;
};
int demo::abc=10; //defining
int main(){
    cout<<"\n Value of abc:"<<demo::abc;
    return 0;
}
```

Output:
Value of abc: 10

Note: The const data member of class is static by default.

Date.
Page No.

Objects as function arguments:

As we know that, we can pass any type of arguments within the member function and there are any number of arguments. In C++ programming language, we can also pass an object as an argument within the member function of class.

This is useful, when we want to initialize all data members of an object with another object, we can pass objects and assign the values of supplied object to the current object. For complex or large projects, we need to use objects as an argument or parameter.

Considers the program:

```
#include <iostream>
using namespace std;

class demo { private:
                int a;
             public:
                void set (int x)
                { a=x;
                }
             void sum(demo ob1,demo ob2)
             { a= ob1.a +ob2.a;
             }
             void print( ){
             cout<<" Value of A:"<<a<<endl;
                }
};
```

Date. _____
Page No. _____

```
int main()
{   // object declarations
    demo d1;
    demo d2;
    demo d3;

// assigning values to the data member of objects
    d1.set(10);
    d2.set(20);
// passing object d1 and d2
    d3.sum(d1, d2);
// printing the values
    d1.print();
    d2.print();
    d3.print();
    return 0;
}
```

Output:

```
Value of A : 10
Value of A : 20
Value of A : 30
```

Above example demonstrate the use of object as a parameta. We are passing d1 and d2 objects as arguments to the sum member function and adding the value of data members 'a' of both objects and assigning to the current objects (that will call the function, which is d3) data member 'a'.

Date.
Page No.

⊕ Returning object from function in C++:

A function can also return objects either by value or by refrence. When an object is returned by value from a function, a temporary object is created within the function, which holds the return value. This value is further assigned to another object in the calling function.

Returning an object by value:

The syntax for defining a function that returns an object by value is:

```
class_name function_name(parameter_list)
{
    //body of the function
}
```

To understand the concept of returing an object by value from a function we consider following example:

```
#include<iostream.h>
class weight{
    int kilogram, gram;
public:
    void getdata();
    void putdata();
    void sum_weight (weight, weight);
    Weight sum_weight (weight);
};
void weight::getdata()
{
    cout<<" Enter kilograms, grams"<<endl;
    cin >> kilogram>> grams;
}
```

Date.
Page No.

Returning an object by refrence:

In case of returning an object by refrence, no new object is created, rather a refrence to the original object in the called function is returned to the calling function.

The syntax for defining a function that returns an object by refrence is:

```
class_name &function_name (parameter_list)
{ // body of function
}
```

To understand this concept, ~~condi~~ consider the function sum weight() of previous program that is modified which will return an object by refrence.

```
weight &weight::sum_weight (weight &w2)
{
  weight temp; //object local to function.
  temp.gram = gram + w2.gram;
  temp.gram = temp.gram % 1000;
  temp.kilogram += kilogram + w2.kilogram;
  return temp; // invalid
}
```

In this code, an attempt has been made to return the refrence to an object of type weight (that is temp). However, it is not possible as the object temp is local to the sum_weight() function and a refrence to this object remains effective only within the function. Thus returning the refrence to temp object outside the function generates a compile-time error.

Date.
Page No.

Imp ⊗. **Constructor:**

A constructor is a member function of a class which initializes objects of a class. ~~In constructor~~ In C++, Constructor is automatically called when object is created. It is special member function of the class. A constructor will have exact same name as the class and it does not have any return type at all, not even void.

Need / Use of Constructor in C++

Suppose we are working on 100's of objects (say Person) and the default value of a data member (say age) is 0. Initializing all objects manually will be a difficult task.

Instead, we can define a constructor that initialises age to 0, for this we have to create a Person object and the constructor will automatically initialise the age. These situations arise frequently while handling array of objects.

Also, if we want to execute some code immediately after an object is created, we can place the code inside the body of the constructor.

Types of Constructors:

There are three types of constructors which are described below with examples.

1) **Default Constructors:** Default constructor is the constructor which does not take any argument. (i.e. It has no parameters).

Date.
Page No.

Example:

```
#include<iostream>
using namespace std;
class construct{ public:          // No parameters
    int a,b;
    construct(){
        a=20;
        b=10;
    }
};
```

ii) Parameterized Constructors: It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way we would to any other function. When we define the constructors body we use the parameters to initialize the object.

Example.

```
Class Point{ private:                    // two parameters
    int x,y;
    public:
    Point(int x1, int y1){
        x=x1;
        y=y1;
    }
    int getX(){
        return x;
    }
    int getY(){
        return y;
    }
};
```

Note: The normal way of object declaration may not work for this. The constructors can be called explicitly or implicitly.

Example

```
e=Example(0,50); //Explicit call
e(0,50);         //Implicit call
```

Use → This constructor is used to initialize element of different objects with different values & to overload constructors.

Date.
Page No.

iii) **Copy Constructor:** A copy constructor is a member function which initializes an object using another object of same class. A copy constructor has the following general function prototype:

```
class_name (const class_name &old_obj);
```

Example:

```
class Point {
    private:
        int x, y;
    public:
        Point (int x1, int y1) {
            x = x1;
            y = y1;
        }
        Point (const Point &p2) {
            x = p2.x;
            y = p2.y;
        }
        int getX() {return x;}
        int getY() {return y;}
};
```

Characteristics of constructors:

i) Constructor name is as same as the name of class.
ii) Constructors should be declared within the public section.
iii) They are called automatically when the objects are created.
iv) They do not have return type not even void and therefore they can't return any values.
v) Like other C++ functions they can have default arguments.
vi) We can not refer to their address.

Date.
Page No.

Program to understand constructor:

```
#include<iostream>
using namespace std;
class volume {  int l,b,h,v;
public:
volume(){} //default constructor
(//parametrized constructor)  volume(int a, int c, int d)
                                { l=a;
                                  b=c;
                                  h=d;
                                }
void display(){
                    V=l*b*h;
cout<<"volume is"<<V;
                }
int main(){
volume v1, v2(2,3,4);
v1.display(); <- for default const displays 0.
v2.display();
return 0;
}
```

Overloaded Constructor:

Overloaded constructor or constructor overloading is similar to function overloading. Overloaded() constructors essentially have same name (as name of the class) and different number of arguments. More than one constructor defined in a class is called constuctor overloading.

Date.
Page No.

Example:

```
#include<iostream>
using namespace std;
class abc{ private:
            int x,y;
        public:
            abc() //constructor 1 with no arguments.
            {
                x=y=0;
            }
            abc(int a) //constructor 2 with one argument.
            {
                x=y=a;
            }
            abc(int a, int b) //constructor 3 with two arguments.
            {
                x=a;
                y=b;
            }

            void display(){
                cout<<"x="<<x<<endl;
                cout<<"y="<<y<<endl;
            }
};
int main(){
    abc p; //constructor 1.
    abc q(10); //constructor 2.
    abc r(10,20); //constructor 3
    p.display();
    q.display();
    r.display();
    return 0;
}
```

Date.
Page No.

Friend function:

If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function. The compiler knows a given function is friend function by the use of keyword friend.

For accessing the data, the declaration of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section) starting with keyword friend.

Declaration

```
class class_name {
... ... ...
friend return_type function_name (argument/s);
... ... ...
};
```

Definition → No friend keyword is used in definition

```
return_type function_name (argument/s) {
... ... ...
/* Private and protected data of
class name can be accessed from this
function because it is a friend function of
class name */
... ... ...
}
```

Characteristics of friend function:

i) It takes objects as arguments.
ii) It is called like a normal function without help of any object.
iii) It is not in the scope of ~~call~~ class to which it has been declared.
iv) It can be declared either in the private or public part of class.
v) It has full access right to private member of class.

Date.
Page No.

Example:

```
// Program to understand friend function.

#include<iostream>
using namespace std;

class Box{    double width;
    public:
friend void printwidth(Box box);
  void setwidth(double wid);
};

void Box::setwidth(double wid){
                width=wid;
}

void printwidth(Box box){
                cout<<"Width of box:"<<box.width;
/* Note-> printwidth() is not a member function of any
class. printwidth() is a friend of Box, it can directly
access any member of this class */

int main(){
                Box box;
box.setwidth(10.0);
// Using friend function to print the width.
                printwidth(box);

                return 0;
}
```

Date.
Page No.

Destructor:

Defⁿ → Destructor is a member function which destructs or deletes an object.

When is destruction called?

A destructor function is called automatically when the object goes out of the scope:

i) the function ends.
ii) the program ends.
iii) a block containing local variables ends.
iv) a delete operator is called.

Note:

Destructors have same name as the class but destructors don't take any argument and don't return anything. There can only one destructor in a class with classname preceded by ~.

Example:

```
class string { private:
                char *s;
                int size;
               public:
                string(char *); // constructor
                ~string(); // destructor.
               };
string::string(char*c){
                size = strlen(c);
                s = new char [size+1];
                strcpy(s,c);
               }
string::~string() {
                delete[]s;
               }
```