Lecture 4

Object Oriented Programming

Class and Object

Concept of a data structure: instead of holding only data, it can hold both data and functions.

An *object* is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

```
1.  Class class_name
2.  {
3.    public:
4.         data type  member1;
5.         data type function_name(per)
6.         ….   …..      …..
7.    Private:
8.         data type  member1;
9.         data type function_name(per)
10.        ….   …..      …..
11.   Protected:
12.        data type  member1;
13.        data type function_name(per)
14.        ….   …..      …..
15. };
```

Where class_name is a valid identifier for the class. The body of the declaration can contain members that can be either data or function declarations, or optionally access specifiers. All is very similar to the declaration on data structures, except that we can now include also functions and members, but also this new thing called *access specifier*. An access specifier is one of the following three keywords: private, public or protected. These specifiers modify the access rights that the members following them acquire:

- Private members of a class are accessible only from within other members of the same class or from their *friends*.
- Protected members are accessible from members of their same class and from their friends, but also from members of their derived classes.
- Finally, public members are accessible from anywhere where the object is visible.

By default, all members of a class declared with the class keyword have private access for all its members. Therefore, any member that is declared before one other class specifier automatically has private access.

General steps to write a C++ program using class and object:

1. Header files
2. Class definition
3. Member function definition
4. void main function

ex1:- Write a program to find sum of two integers using class and object.

```
#include <iostream>
#include <conio>
class Add
{
int x, y, z;
```

```
public:
void getdata()
{
cout<<"Enter two numbers";
cin>>x>>y;
}
public:
void calculate(void)
{
z=x+y;
}
public:
void display(void)
{
cout<<z;
}
};

void main()
{
Add a;
a.getdata();
a.calculate();
a.display();
getch();
}
```

Ex2:- Write a program to find the area of rectangle using class

```
#include <iostream>
#include <conio>
class CRectangle
{
int x, y;
public:
void set_values ()
{
cin>>x>>y;
}
int area ()
 {
return (x*y);
}
};
 main ()
 {
CRectangle rect;
rect.set_values ();
cout << "area: " << rect.area();
getch();
}
```

One of the greater advantages of a class is that, as any other type, we can declare several objects of it. For example, following with the previous example of class CRectangle, we could have declared the object rectb in addition to the object rect:

```
#include <iostream>
#include <conio>
class CRectangle
{
int x, y;
public:
void set_values ()
{
cin>>x>>y;
}
int area ()
 {
return (x*y);
}
};
 main ()
 {
CRectangle rect ,rectb;
rect.set_values ();
rectb.set_values ();
cout << "area: " << rect.area();
cout << "area: " << rectb.area();
getch();
}
```

That is the basic concept of *object-oriented programming*: Data and functions are both members of the object. We no longer use sets of global variables that we pass from one function to another as parameters, but instead we handle objects that have

their own data and functions embedded as members. Notice that we have not had to give any parameters in any of the calls to rect.area or rectb.area. Those member functions directly used the data members of their respective objects rect and rectb.

A member function can be defined:

1. Inside the class definition
2. Outside side the class definition using scope resolution operator (::).
3. Here in the above example we are defining the member function getdata() inside the class definition. And we are defining the member functions calculate() and display(), outside the class definition using the scope resolution operator.
4. Here void Add :: calculate() means the scpoe of member function calculate() is inside the class Add or we can say the function calculate() belongs to the class Add. :: is the scope resolution operator which tells the scope of a member function.
5. We cannot directly call a function, we can call it using object (through . operator) of the class in which the function is declared.

Ex3 :- Write a program to add two time objects (in the form hh:mm).

```
#include <iostream>
#include <conio>
class time
{
int hours, minutes;
public:
void gettime(int h, int m)
{
```

```
hours=h;
minutes=m;
}
void sum(time, time);
void display(void);
};
void time :: sum (time t1, time t2)
{
minutes=t1.minutes+t2.minutes;
hours=minutes/60;
minutes=minutes%60;
hours=hours+t1.hours+t2.hours;
}
void time :: display()
{
cout<<hours<<" : "<<minutes<<endl;
}
void main()
{
time T1, T2, T3;
T1.gettime(2,45);
T2.gettime(3,30);
T3.sum(T1, T2);
T1.display();
T2.display();
cout<<"Additionof above two time is ";
T3.display();
```

```
getch();
}
```

Array of object

Collection of similar types of object is known as array of objects.

Ex 4:-Write a program to input name and age of 5 employees and display them.

```
#include <iostream>
#include <conio>

class Employee
{
char name[30];
int age;
public:
void getdata(void);
void putdata(void);
};
void Employee:: getdata(void)
{
cout<<"Enter Name and Age:";
cin>>name>>age;
}
void Employee:: putdata(void)
{
cout<<name<<"\t"<<age<<endl;
}
void main()
```

```
{
Employee e[5];
int i;
for(i=0; i<5; i++)
{
e[i].getdata();
}
for(i=0; i<5; i++)
{
e[i].putdata();
}

getch();
}
```

Constructors and destructors

Objects generally need to initialize variables or assign dynamic memory during their process of creation to become operative and to avoid returning unexpected values during their execution. For example, what would happen if in the example2 we called the member function area() before having called function set_values()? Probably we would have gotten an undetermined result since the members x and y would have never been assigned a value. In order to avoid that, a class can include a special function called constructor, which is automatically called whenever a new object of this class is created. This constructor function must have the same name as the class, and cannot have any return type; not even void. We are going to implement CRectangle including a constructor

example: class constructor

```
#include <iostream>
```

```
class CRectangle
{
int width, height;
public:
CRectangle (int,int);
int area () {
return (width*height);
}
};
CRectangle::CRectangle (int a, int b)
{
width = a;
height = b;
}
int main ()
 {
CRectangle rect (3,4);
CRectangle rectb (5,6);
cout << "rect area: " << rect.area() << endl;
cout << "rectb area: " << rectb.area() << endl;
return 0;
}
```

As you can see, the result of this example is identical to the previous one. But now we have removed the member function set_values(), and have included instead a constructor that performs a similar action: it initializes the values of x and y with the parameters that are passed to it.

Notice how these arguments are passed to the constructor at the moment at which the objects of this class are created:

CRectangle rect (3,4);

CRectangle rectb (5,6);

Constructors cannot be called explicitly as if they were regular member functions. They are only executed when a new object of that class is created.

You can also see how neither the constructor prototype declaration (within the class) nor the latter constructor definition include a return value; not even void.

The *destructor* fulfills the opposite functionality. It is automatically called when an object is destroyed, either because its scope of existence has finished (for example, if it was defined as a local object within a function and the function ends) or because it is an object dynamically assigned and it is released using the operator delete.

The destructor must have the same name as the class, but preceded with a tilde sign (~) and it must also return no value. The use of destructors is especially suitable when an object assigns dynamic memory during its lifetime and at the moment of being destroyed we want to release the memory that the object was allocated