Lab 12
Object Oriented Programming
Dr. John Abraham

We humans are very good recognizing and working with **objects**, such as a pen, a dog, or a human being. We learned to categorize them in such a way that make sense to us. We may categorize them as animate object, inanimate objects, pets, friends, etc. We some times classify objects based on their **attributes**, for example, green apples or red apples, fat or slim people, etc. If you think about it each object has many attributes. If I ask you list the attributes of an orange, you probably could list many things such as color, shape, weight, smell, etc.

In addition to attributes, all objects exhibit **behaviors**. A dog eats, barks, wags its tail, plays, and begs. A dog exhibits many more other behaviors than this short list. It is a good idea to practice listing attributes and behaviors of many of the objects you come across each day. Another thing we need to remember about objects is that objects interact between each other.

Programmers, for over thirty years programmed using functions and procedures. Each function and procedure was called to carry out certain tasks on the data that were given to it and to return (or not return) certain results back. With this type of **procedure oriented programming**, we had to adapt our thinking procedurally. Working with objects is a more normal and suitable approach for human beings. Such programming approach is called **Object Oriented Programming** (OOP).

In Object Oriented Programming, **objects** are packages that contain **data** and functions (**methods**) that can be performed on the data. Data could be considered to be **attributes** and functions are considered to be **behaviors** of the object. We can say that the attributes and behaviors are **encapsulated** into an object. The objects interact between each other through their **interfaces**. As an example a date **object** may have a set of **data** consisting of month, day and year, and **methods** consisting of assign date, display date, yesterday and tomorrow.

Data abstraction is another important feature of Object oriented programming. OOP hides the details of data structures from the users. For example, to set a date in the aforementioned example, values of month, day and year are passed to the assign-date method. The actual representation of the date is hidden from the user.

There can be various **objects** made of a particular **class** (recall that many variables can be made from type). In OOP the general type with which objects can be made is called a **class**. An object is an **instance** of a class. Each class contains data (**data members**) as well as set of functions (**member functions**) that manipulate the data. Classes have the following features:

1. the capability to control access
2. Constructors
3. Destructors
4. Data Members
5. Member functions
6. A hidden, special pointer called this

Program 5-1 is a program that incorporates a c++ object. The **class Grade** contains three data members and three **member functions** (methods) that manipulate the data. The class has a **public:** section that is accessible to those using the class, and a **private:** section that only accessible to member functions and not accessible to the class users.

Program 12-1

```
/*****************************************
Program Grades - class
By Dr. John Abraham
Written for CSCI 1370 students
Objective: introduce object oriented programming
*****************************************/

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

class Grade {
public:
  Grade();              //constructor
  void setGrades(int, int, int);
  void printGrades();
  void printLetterGrade();
private:
  int g1;
  int g2;
  int g3;
};
//remember that if you do not indicate private or public, members of a class are private by
default.

Grade::Grade()
 {
  g1=g2=g3=0;
 }

 void Grade:: setGrades (int a, int b,int c)
 {
  g1=a; g2=b; g3=c;
}
void Grade:: printGrades()
{
  cout << "\n\n--------------------------------";
  cout << "\nHere are the Grades you entered: " <<g1<<setw(4)<<g2<<setw(4)<<g3;
}
void Grade::printLetterGrade()

{
  float av;
  av = (g1+g2+g3)/3.0;
  cout << "\nYour Average and Letter Grade -> "<<av;
  if (av >= 90) cout <<"  A\n";
        else if (av >=80) cout << " B\n";
           else if (av >=70) cout <<" C\n";
                  else if (av >= 60) cout << " D\n";
                    else cout << " F\n";

}

int main (void)
{
        int a,b,c;
        Grade n;      //n is an object of class Grade

        cout <<"\nEnter three grades separated by spaces  ";
        cin >> a >> b >> c;

        n.setGrades(a,b,c);
        n.printGrades();
        n.printLetterGrade();
return (0);
}
```

Program Run 12-1

```
Enter three grades separated by spaces  81 78 82



--------------------------------
Here are the Grades you entered: 81  78  82
Your Average and Letter Grade -> 80.3333 B
Press any key to continue
```

Let us discuss this program in detail. We have declared a **class** named **Grade**. We have an **object** made up of this class, namely **n**. We could have made other objects of class Grade. An object encapsulates the data and functions that operate on that data. In this case the data used are three integers and three functions (methods) are getGrades, printGrades, and printLetter. The public part of the class is visible and accessible to all users of the class, the private part is not. The public part contains a **constructor**; a constructor is a function that is automatically called when an instance of a class is created. A constructor is used to initialize any class member variables, and allocate memory that the class will need. The member functions are similar to the functions we have used so far except in the declaration the scope operator :: is used. For example, void Grade::getGrades (int a, int b, intc), clearly indicates that getGrades is a member function of class Grade.

Now let us examine the main. Here we have three integer variables a, b, c. and an object n. These values of these variables are read from the keyboard and then the three member functions of the object n are called. Imagine creating an include file of the class Grade. Instead of imagining it, let us do it. Make sure to write the include program first and save it. Remember the subdirectory where it was saved and use the appropriate path in the main program below.

```
/****************************************
Program Grades - class
By Dr. John Abraham
Written for CSCI 1370 students
Objective: introduce object oriented programming
****************************************/

#include <iostream>
#include <c:\tempc\classgrade.h>
using namespace std;

 int main (void)
 {

        int a,b,c;
        Grade n;

        cout <<"\nEnter three grades separated by spaces  ";
        cin >> a >> b >> c;


        n.setGrades(a,b,c);
```

```
            n.printGrades();
            n.printLetterGrade();

 return (0);
 }
```

The include file is given below.

```
/*****************************************
Program Grades - class
By Dr. John Abraham
Written for CSCI 1370 students
Objective: introduce object oriented programming
*****************************************/


#include <iomanip>
using namespace std;

class Grade {
public:
  Grade();                    //constructor
  void setGrades(int, int, int);
  void printGrades();
  void printLetterGrade();
private:
  int g1;
  int g2;
  int g3;
};
//remember that if you do not indicate private or public, members of a class are private by
default.

Grade::Grade()
 {
  g1=g2=g3=0;
 }

 void Grade:: setGrades (int a, int b,int c)
 {
  g1=a; g2=b; g3=c;
 }
 void Grade:: printGrades()
 {
  cout << "\n\n--------------------------------";
  cout << "\nHere are the Grades you entered: " <<g1<<setw(4)<<g2<<setw(4)<<g3;
}
void Grade::printLetterGrade()

{
  float av;
  av = (g1+g2+g3)/3.0;
  cout << "\nYour Average and Letter Grade -> "<<av;
  if (av >= 90) cout <<"  A\n";
            else if (av >=80) cout << " B\n";
                else if (av >=70) cout <<" C\n";
                          else if (av >= 60) cout << " D\n";
                            else cout << " F\n";

}
```

Now that we covered briefly structs, classes and arrays, note that you can have an array as a member of a struct or a class. You can also have an array of instances of struct or an array of objects. For example, I gave you a program in Lab 10 using struct to keep track of inventory (parts). I added an array to hold 40 parts in the following example.

```
int main()
{
InvItem part, copyPart, manyParts[40];
getItem(part);
copyPart = part;
showItem(copyPart);
cin.ignore();getchar();
return 0;
}
```

If you wish to add items to the 5th array element, you could do it this way:
`manyParts[5]=part;`

or this way:
`manyParts[5].partNum = 2352; manyParts[5].description = "Pipe 8mm"` and so on.

Or you could read from the keyboard as follows:

```
cout<<"Enter the part number: "; cin>> manyParts[5].partNum;
cout<<"Enter the description: "; cin.get();
getline(cin, manyParts[5].description);
cout<<"Enter the quantity on hand: "; cin>> manyParts[5].onHand;
cout<<"Enter the unit price: "; cin>> manyParts[5].price;
```

Assignment:
Write a program to calculate area and perimeter of a rectangle using a Rectangle class.