UNIVERSITY OF ENGINEERING AND TECHNOLOGY

(NAROWAL CAMPUS)

Object-Oriented Programming Lab Manual

Created by: Muhammad Abdullah

Registration Number: 2022-CS-525

Topics: Object Oriented Programming in C++,

Unions & User-Defined Header file

Lab Manual

(Object-Oriented Programming Lab)

- ## Object Oriented Programming:

OOPS is abbreviated as Object Oriented Programming system in which programs are considered as a collection of objects. Each object is nothing but an instance of a class. A class is simply a representation of a type of object. It is the blueprint/plan/template that describes the details of an object. An object is an instance of a class. It has its own state, behavior, and identity. OOP follows a Bottom-up programming approach. The main aim of OOP is to bind data and the functions that operate on them together so that no other part of the code can access this data except this function. There are four main pillars of Object Oriented Programming which are as follows:

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

Encapsulation

Encapsulation is an attribute of an object, and it contains all data which is hidden. That hidden data can be restricted to the members of that class. Levels are Public, Protected, and Private. The main advantage of encapsulation is that data is hidden and protected from random access by outside non-member methods of a class. In encapsulation, data (variables) are declared private, and methods (functions) are declared public. Access specifiers allow us to restrict the scope or visibility of a package, class, constructor, methods, variables, or other data members. The three types of most common access specifiers are Public, Private, and Protected. Public Modifiers mean that class, variable, or method is accessible throughout from within or outside the class, within or outside the package, etc. It provides the highest level of accessibility. A private Modifier means that class, variable or method is not accessible from within or outside the class, within or outside the package, etc. A private field or method can't be inherited to a subclass. This provides the lowest level of accessibility. Protected Modifiers mean that a class, variable, or method is accessible from classes in the same package, sub-classes in the same package, and subclasses in other packages but not accessible from classes in other packages.

Abstraction

Abstraction means displaying only essential information and hiding the details. Abstraction allows the hiding of unnecessary data from the user. This reduces program complexity efforts. It displays only the necessary information to the user and hides all the internal background details. If we talk about data abstraction in a programming language, the code implementation is hidden from the user and only the necessary functionality is shown or provided to the user. We can achieve data abstraction by using an Abstract class and interface.

Inheritance:

Inheritance is a concept where one class shares the structure and behavior defined in another class. If Inheritance is applied to one class is called Single Inheritance, and if it depends on multiple classes, then it is called multiple Inheritance. In other words, it is a mechanism of acquiring properties or behaviors of an existing class to a new class. The Base Class, also known as the Parent Class is a class, from which other classes are derived. The Derived Class, also known as Child Class, is a class that is created from an existing class.

Polymorphism:

Polymorphism is nothing but assigning behavior or value in a subclass to something that was already declared in the main class. Simply, polymorphism takes more than one form. Polymorphism is the ability of an object to take on many forms. We can define polymorphism as the ability of a message to be displayed in more than one form. Polymorphism is mainly divided into two types Compile time Polymorphism (CTP) which is also called static polymorphism or early binding and Runtime Polymorphism (RTP) which is also called dynamic polymorphism or late binding. Compile time polymorphism is achieved by function overloading or operator overloading. Runtime polymorphism refers to the process when a call to an overridden process is resolved at the run time. This type of polymorphism is achieved by Function Overriding.

Constructor and Destructor Methods:

Constructor is a special type of member function that is used to initialize an object. It is similar to functions but its name should be the same as its class name and must have no explicit return type. It is called when an object of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory. We use the constructor to assign values to the class variables at the time of object creation. Constructors have the following types:

1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor
4. Static Constructor
5. Private Constructor

A constructor with 0 parameters is known as a default constructor. If a constructor is declared private, we cannot create an object of the class. A copy constructor is a constructor which uses the existing object to create a new object. It copies variables from another object of the same class to create a new object. A static constructor is automatically called when the first instance is generated, or any static member is referenced. The static constructor is explicitly declared by using a static keyword.

A destructor is a type of member function that is used to destroy an object. It is called automatically when the object goes out of scope or is explicitly destroyed by a call to delete. It destroys the objects when they are no longer in use. A destructor has the same name as the class, preceded by a tilde (~).

- ## Separate Interface and Implementation:

  In C++, the term "separate interface and implementation" generally refers to a design technique where the definition of a class's interface, which includes the public member functions and variables, is declared in a header file, while the implementation of those functions and variables is defined in a separate source file.

  This technique provides several benefits, such as Encapsulation: By hiding the implementation details of a class, you can prevent clients from accessing or modifying its internal state directly, which can help maintain the integrity and correctness of your program. Modularity: By separating the interface and implementation, you can modify the implementation of a class without affecting the code that uses it, as long as you don't change the interface. Reusability: By defining a clear interface, you can reuse the same class in different contexts and applications, without having to rewrite the entire implementation every time.

- ## Unions in C++:

  A union is a type of structure that can be used where the amount of memory used is a key factor. Similarly to the structure, the union can contain different types of data types. Each time a new variable is initialized from the union it overwrites the previous in C language but in C++ we also don't need this keyword and use that memory location. This is most useful when the type of data being passed through functions is unknown, using a union that contains all possible data types can remedy this problem. It is declared by using the keyword "union".

- ## Programs:

  Task 1: Write a program that contains Union and some operations

  Task 2: Create a class that contains some interface and implementation member functions

  Task 3: Create a class that contains Constructor and Destructor

  Task 4: Create a program of Default Constructor

  Task 5: Write a program using a separate interface and implementation technique

Task 1 Program:

```cpp
#include <iostream>
using namespace std;
union casualityMeter{
    int initialCasualties;
    int update;
    int finalCasualties;
};
int main(){
    casualityMeter earthQuick;
    earthQuick.initialCasualties = 10;
    cout<<"Number of Casualties at initial is: "<<earthQuick.initialCasualties<<endl;
    earthQuick.update = 15;
    cout<<"Number of Casualties at 1300 Hours is: "<<earthQuick.update<<endl;
    earthQuick.update = 20;
    cout<<"Number of Casualties at 1600 Hours is: "<<earthQuick.update<<endl;
    earthQuick.update = 25;
    cout<<"Number of Casualties at 1900 Hours is: "<<earthQuick.update<<endl;
    earthQuick.finalCasualties = earthQuick.update;
    cout<<"Number of Casualties at Final is: "<<earthQuick.finalCasualties<<endl;
    return 0;
}

```

Task 1 Output:

```
Number of Casualties at initial is: 10
Number of Casualties at 1300 Hours is: 15
Number of Casualties at 1600 Hours is: 20
Number of Casualties at 1900 Hours is: 25
Number of Casualties at Final is: 25

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 18928) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Task 2 Program:

```cpp
#include <iostream>
#include <string>
 using namespace std;
class player
 {
 private:
    string name, country;
    int age, t20, odi, rank;      // t20 = Runs in T20, odi = Runs in ODI
    float height, weight;

 public:
    void getInfo();
    void printInfo();
 };
int main()
 {
    player babarAzam;
    babarAzam.getInfo();
    babarAzam.printInfo();
    return 0;
 }
```

```
22  void player :: getInfo()
23  {
24      cout << " --: Player Information Entry :-- " << endl;
25      cout << "Enter Player Name: ";
26      getline(cin, name);
27      cout << "Enter Player Country: ";
28      getline(cin, country);
29      cout<<"Enter Player Rank: ";
30      cin>>rank;
31      cout << "Enter Player Age: ";
32      cin >> age;
33      cout << "Enter Player Height: ";
34      cin >> height;
35      cout << "Enter Player Weight: ";
36      cin >> weight;
37      cout << "Enter Player T20 Runs: ";
38      cin >> t20;
39      cout << "Enter Player ODI Runs: ";
40      cin >> odi;
41  }
42  void player :: printInfo(){
43      cout<<"Player Name: "<<name<<endl;
44      cout<<"Player Country: "<<country<<endl;
45      cout<<"Player Rank: "<<rank<<endl;
46      cout<<"Player Age: "<<age<<endl;
47      cout<<"Player Height: "<<height<<endl;
48      cout<<"Player Weight: "<<weight<<endl;
49      cout<<"Player T20 Runs: "<<t20<<endl;
50      cout<<"Player ODI Runs: "<<odi<<endl;
51  }
```

Task 2 Output:

```
 --: Player Information Entry :--
Enter Player Name: Babar Azam
Enter Player Country: Pakistan
Enter Player Rank: 1
Enter Player Age: 28
Enter Player Height: 1.8
Enter Player Weight: 00
Enter Player T20 Runs: 3485
Enter Player ODI Runs: 5089
Player Name: Babar Azam
Player Country: Pakistan
Player Rank: 1
Player Age: 28
Player Height: 1.8
Player Weight: 0
Player T20 Runs: 3485
Player ODI Runs: 5089

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 18304) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Task 3 Program:

```cpp
#include <iostream>
using namespace std;
class Data{
    private:
        int *intptr = NULL, length = 0;
    public:
        Data(int n){
            length = n;
            intptr = new int[length];
        }
        void getData();
        void search(int x);
        void printData();
        ~Data(){
            delete[] intptr;
            intptr = NULL;
            cout<<endl<<"CLEARED";
        }
};
int main(){
    int a;
    cout<<"Enter Length/Quantity of data: ";
    cin>>a;
    Data d(a);
    d.getData();
    cout<<"Enter a value to search in given data: ";
    cin>>a;
    d.search(a);
    cout<<" --: Data :-- "<<endl;
    d.printData();
    return 0;
}
void Data :: getData(){
    for(int i =0; i<length;i++){
        cout<<"Enter "<<i+1<<" Value: ";
        cin>>*(intptr+i);
    }
}
void Data :: search(int x){
    for(int i = 0 ; i<length;i++){
        if(x == *(intptr + i)){
            cout<<"Yes it is in our record"<<endl;
            return;
        }
    }
    cout<<"No it is not in out record"<<endl;
    return;
}
void Data :: printData(){
    for(int i =0; i<length;i++){
        cout<<*(intptr + i)<<" ";
    }
}
```

Task 3 Output:

```
Enter Length/Quantity of data: 5
Enter 1 Value: 901
Enter 2 Value: 947
Enter 3 Value: 130
Enter 4 Value: 425
Enter 5 Value: 463
Enter a value to search in given data: 1075
No it is not in out record
 --: Data :--
901 947 130 425 463
CLEARED
D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 19144) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Task 4 Program:

```cpp
#include <iostream>
using namespace std;
class solider{
    private:
        string name, fname, battalion, rank, course, idNumber, ww; //fname = Father Name , ww = War Wounded
        int age;
        float height, weight;
    public:
        solider(){
            name = "Muhammad Abdullah";
            fname = "Zahid Mehmood";
            rank = "Lieutenant Colonel";
            battalion = "4th Commando Battalion ";
            course = "17th PMA L/C";
            idNumber = "PA-1904";
            age = 30;
            height = 5.9;
            weight = 75;
            ww = "Yes";
        }
        void printData();
};
int main(){
    solider Abdullah;
    Abdullah.printData();
    return 0;
}
```

```cpp
void solider :: printData(){
    cout<<" --: Solider Information :--"<<endl;
    cout<<"Name: "<<name<<endl;
    cout<<"Father Name: "<<fname<<endl;
    cout<<"Age: "<<age<<endl;
    cout<<"Rank: "<<rank<<endl;
    cout<<"Army Number: "<<idNumber<<endl;
    cout<<"Battalion: "<<battalion<<endl;
    cout<<"Course: "<<course<<endl;
    cout<<"Height: "<<height<<endl;
    cout<<"Weight: "<<weight<<endl;
    cout<<"War Wounded: "<<ww<<endl;
}
```

Task 4 Output:

```
 --: Solider Information :--
Name: Muhammad Abdullah
Father Name: Zahid Mehmood
Age: 30
Rank: Lieutenant Colonel
Army Number: PA-1904
Battalion: 4th Commando Battalion
Course: 17th PMA L/C
Height: 5.9
Weight: 75
War Wounded: Yes

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 17720) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Task 5 Program:

```cpp
#include <iostream>
#include "Task5File.h"
using namespace std;
int main()
{
    player viratKholi;
    viratKholi.getInfo();
    viratKholi.printInfo();
    return 0;
}
```

Task 5 Header File:

```cpp
#ifndef Task5file
#define Task5file

#include <iostream>
#include <string>
using namespace std;
class player
{
private:
    string name, country;
    int age, t20, odi, rank; // t20 = Runs in T20, odi = Runs in ODI
    float height, weight;

public:
    void getInfo()
    {
        cout << " --: Player Information Entry :-- " << endl;
        cout << "Enter Player Name: ";
        getline(cin, name);
        cout << "Enter Player Country: ";
        getline(cin, country);
        cout << "Enter Player Rank: ";
        cin >> rank;
        cout << "Enter Player Age: ";
        cin >> age;
        cout << "Enter Player Height: ";
        cin >> height;
        cout << "Enter Player Weight: ";
        cin >> weight;
        cout << "Enter Player T20 Runs: ";
        cin >> t20;
        cout << "Enter Player ODI Runs: ";
        cin >> odi;
    }
```

```
35      void printInfo()
36      {
37          cout << "Player Name: " << name << endl;
38          cout << "Player Country: " << country << endl;
39          cout << "Player Rank: " << rank << endl;
40          cout << "Player Age: " << age << endl;
41          cout << "Player Height: " << height << endl;
42          cout << "Player Weight: " << weight << endl;
43          cout << "Player T20 Runs: " << t20 << endl;
44          cout << "Player ODI Runs: " << odi << endl;
45      }
46  };
47
48  #endif
```

Task 5 Output:

```
 --: Player Information Entry :--
Enter Player Name: Virat Kholi
Enter Player Country: India
Enter Player Rank: 2
Enter Player Age: 34
Enter Player Height: 1.75
Enter Player Weight: 00
Enter Player T20 Runs: 4139
Enter Player ODI Runs: 12898
Player Name: Virat Kholi
Player Country: India
Player Rank: 2
Player Age: 34
Player Height: 1.75
Player Weight: 0
Player T20 Runs: 4139
Player ODI Runs: 12898

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 4348) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```
