Maulana Azad National Institute of Technology

(MANIT) Bhopal

Lab-Assignment

Principles of Programming Languages

(CSE-219)

Submitted by: Jishan Shaikh (Scholar no. 161112013)

Section: CSE-1 (2016-20)

Submitted to: Department of CSE,

MANIT Bhopal

Submission date: November 7, 2017

Index of programs

1. Write a program using Class and Object in C++.

Answer: Class: A Class is building block of object-oriented programming in C++. It is a user defined data-type, where user can customize its own data-type in a packet. Class holds its own data-members and data-functions which can be accessed and used by instances of that class. A Class is similar to a 'Structure' in C programming language. A Class is a blueprint of an object.

Data-members & functions of a class: There are three in-fields of a class named: public, protected and private. By default, the data-types are considered as private unless until explicitly mentioned as its type. Rules for single data-member/function are also applicable for all data-members/functions inside a class i.e. For a name of variables and identifiers.
All data-members whether public, private or protected of parent class can be accessed by **child** class inherited from it. Private data-members of parent class can not be accessed by its child or any its inherited class. Protected data can be accessed by any of its inherited class which was inherited publicly or protectively.

Class declaration in C++: Here is a snippet of class declaration in C++, where we declare some data-fields such as name of student as a string of characters, scholar number as long long type of integer, mobile-number as private and its GPA as double data-type inside protected field of student using a 'Class'.

```
class class_name{
    public:
        char[50] name;
        long long int scholar_number;
    private:
        long long int mobile_number;
    protected:
        double GPA;
};
```

Object: An object is an instance of a class. When a class is declared no memory is allocated, but when an instance of a class i.e. An object is created, memory equal to the data-elements of the class is allocated.

Program:

```
#include <iostream>
using namespace std;
class student{
    public:
```

```
        char[50] name;
        long int scholarnumber;
        float GPA;
};
int main(){
   student S1;
   student S2
   S1.name=”Jishan”;
   S1.scholarnumber=161112013;
   S1.GPA=8.5;
   S2.name=”Johny”;
   S2.scholarnumber=161112000;
   S2.GPA=8.0;
   cout << ”Name: “ + S1.name << endl;
   cout << ”Scholar number: + S1.scholarnumber << endl;
   cout << ”GPA: ” + S1.GPA << endl;
   cout << ”Name: “ + S2.name << endl;
   cout << ”Scholar number: + S2.scholarnumber << endl;
   cout << ”GPA: ” + S2.GPA << endl;
}
```

2. What is inheritance in C++? Explain the types of inheritance in C++ with example programs.

Answer: Inheritance: Its a matter of fact that code-sharing is sometimes very useful in case of large code-base or files of a big software. C++ also allows code-sharing through the concept of inheritance. Inheritance is one of the key features of object-oriented programming (OOP).

Inheritance is a common feature in our real world also. For example, organizational hierarchy in a large organization comes from this concept. Other examples in non-technical and technical ground is given in figure 1, 2 and 3.

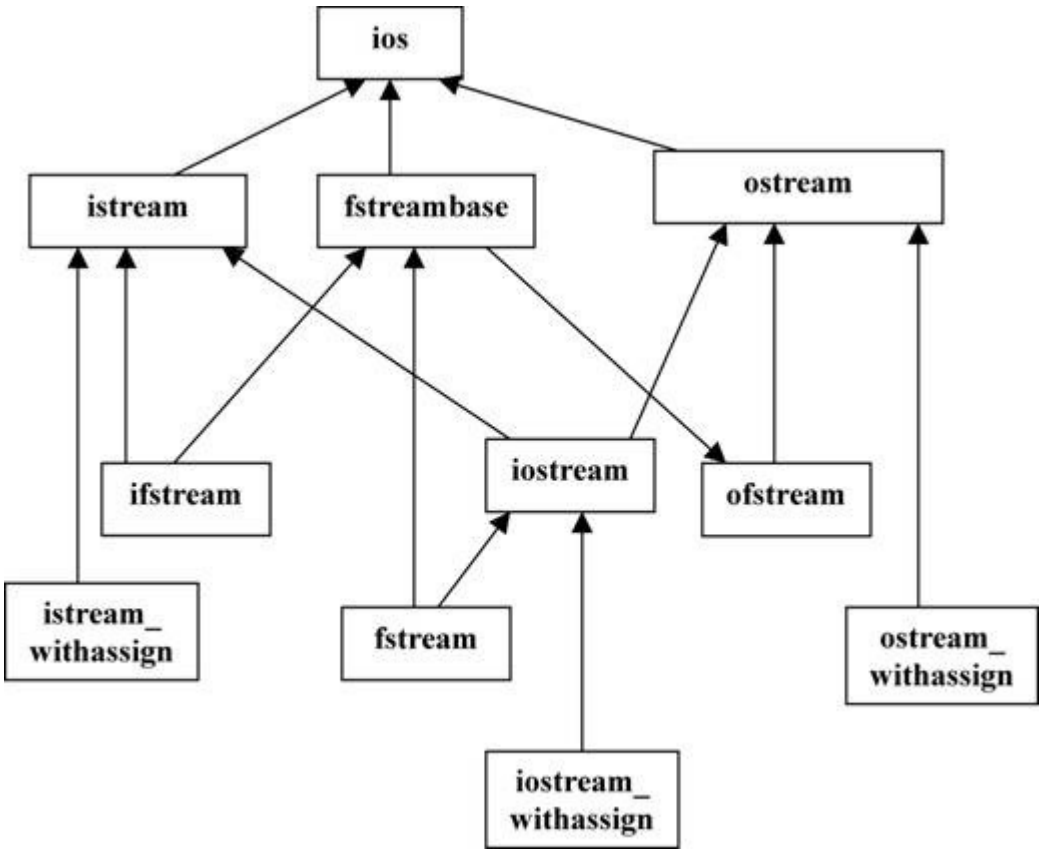

Figure 1. Header inheritance in C++ internal structure.

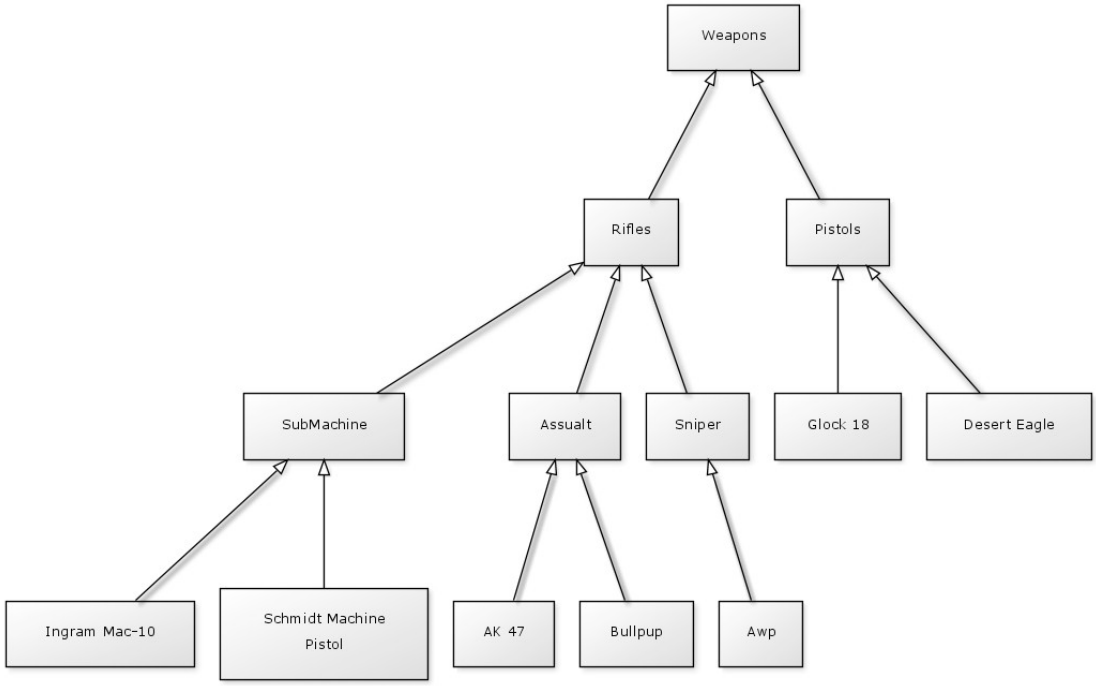

Figure 2. A Real world example of inheritance of weapons.

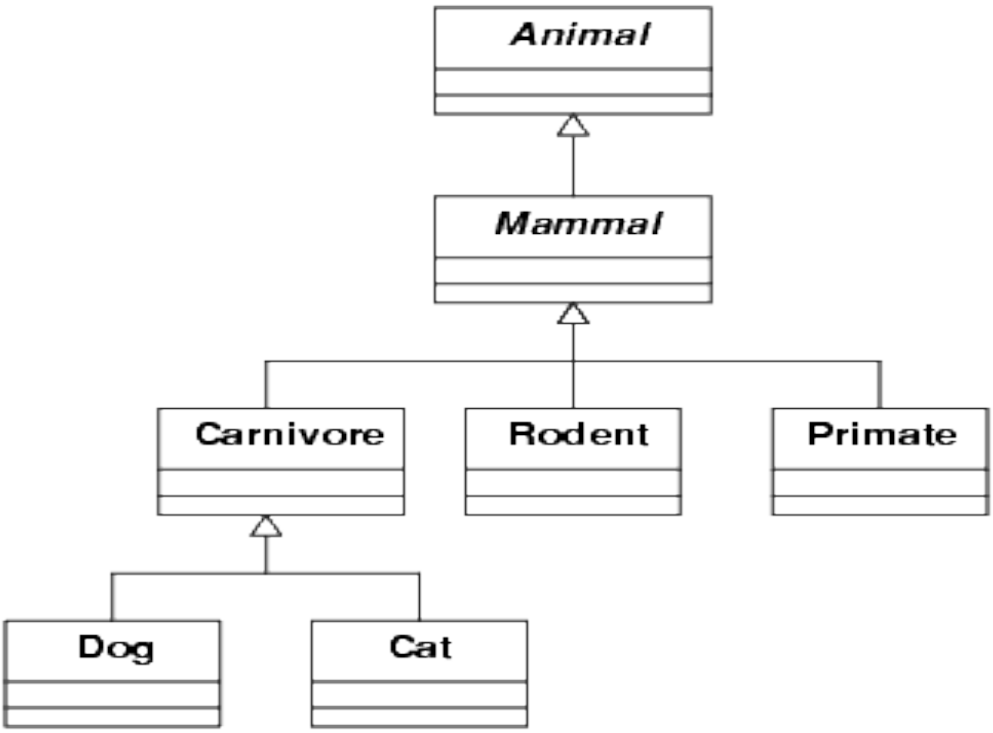

Figure 3. Another Real world example of inheritance of animals.

Program to demonstrate simple inheritance in C++:

```
#include <iostream>
using namespace std;
class vehicle{
    public:
        int price;
        float performance_index;
};
class bike: public vehicle{
    public:
        double weight_in_kg;
        char[] name;
};
int main(){
    bike Bike1;
    Bike1.price=25000;
    Bike1.performance_index=77.5;
    Bike1.name=”Pulsar_Old”;
    Bike1.weight_in_kg=225;
         cout<<”The  price-performance  ratio  of”  +
```

```
    Bike1.name + ”is” +
    (Bike1.price)/(Bike1.performance_index)<<endl;
20.     return 0;
21. }
```

Types of inheritance: There are major three types of inheritance in C++:

1. **Single inheritance:** Single inheritance enables a derived class to inherit properties and behavior from a single parent class. It allows a derived class to inherit the properties and behavior of a base class. This makes the code much more elegant and less repetitive. Single inheritance is safer than multiple inheritance if it is approached in the right way. It also enables a derived class to call the parent class implementation for a specific method if this method is overridden in the derived class or the parent class constructor. In other words, In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.

Program in C++ for Single inheritance:

```
#include <iostream>
using namespace std;
class Person{
     public:
        string profession;
        int age;
        Person(): profession("unemployed"), age(16){}
        void display(){
             cout << "My profession is: " <<
profession << endl;
             cout << "My age is: " << age << endl;
             walk();
             talk();
        }
        void walk(){ cout << "I can walk." << endl;}
        void talk(){ cout << "I can talk." << endl;}
};
class MathsTeacher : public Person{
    public:
       void teachMaths(){ cout << "I can teach Maths."
<< endl;}
};
class Footballer : public Person{
    public:
       void playFootball() { cout << "I can play
Football." << endl;}
};
int main(){
     MathsTeacher teacher;
     teacher.profession = "Teacher";
     teacher.age = 23;
     teacher.display();
     teacher.teachMaths();
     Footballer footballer;
     footballer.profession = "Footballer";
     footballer.age = 19;
     footballer.display();
     footballer.playFootball();
     return 0;
}
```

2. **Multiple inheritance:** Multiple inheritance a feature of some object-oriented programming languages in which a class or an object inherits characteristics and properties from more than one parent class or object. This is contrary to the single inheritance property, which allows an object or class to inherit from one specific object or class. Although there are certain benefits associated with multiple inheritance, it does increase ambiguity and complexity when not designed or implemented properly. In other words, multiple inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one sub-class is inherited from more than one base classes.

Program in C++ for Multiple inheritance:

```
#include <iostream>
using namespace std;
class Mammal{
  public:
    Mammal(){
        cout << "Mammals can give direct birth." << endl;
    }
};
class WingedAnimal{
  public:
    WingedAnimal(){
      cout << "Winged animal can flap." << endl;
    }
};
class Bat: public Mammal, public WingedAnimal{
};
int main(){
    Bat b1;
    return 0;
}
```

3. **Multi-level inheritance:** In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance. In other words, this is a type of inheritance where a derived class is created from another derived class.

Program in C++ for Multi-level inheritance:

```
#include <iostream>
using namespace std;
class A{
    public:
      void display(){
          cout << "Base class content.";
      }
};
class B : public A{
};
class C : public B{
};
int main(){
    C obj;
    obj.display();
    return 0;
}
```

The Accessibility of members in inherited subsequent classes is defined in table 1.

	Private data-members			**Protected data-members**			**Public data-members**		
Inherited type	Public	Protected	Private	Public	Protected	Private	Public	Protected	Private
Own class	YES	YES	YES	YES	YES	YES	YES	YES	YES
Derived class	NO	NO	NO	YES	YES	YES	YES	YES	YES
2nd derived class	NO	NO	NO	YES	YES	NO	YES	YES	NO

Table 1. Data-types and its accessing in subsequent inherited classes. (Up-to level 2)

3. Write C++ programs to demonstrate Operator overloading and function overloading in C++.

Answer: Operator overloading: The meaning of an operator is always same for variable of basic types like: int, float, double etc. For example: To add two integers, + operator is used.

However, for user-defined types (like: objects), you can redefine the way operator works. For example: If there are two objects of a class that contains string as its data members. You can redefine the meaning of + operator and use it to concatenate those strings.

This feature in C++ programming that allows programmer to redefine the meaning of an operator (when they operate on class objects) is known as **operator overloading.**

Program to demonstrate operator overloading:

```
#include <iostream>
using namespace std;
class Test{
   private:
      int count;
   public:
       Test(): count(5){}
       void operator ++(){
          count = count+1;
       }
       void Display(){
          cout<<"Count: "<< count;
       }
};
int main(){
    Test t;
        //This calls "function void operator ++()" function
    ++t;
    t.Display();
    return 0;
}
```

Function Overloading: In C++ programming, two functions can have same name if number and/or type of arguments passed are different.
These functions having different number or type (or both) of parameters are known as overloaded functions. For example:

```
int test(){}
int test(int a){}
```

3. float test(double a){}
4. int test(int a, double b){}

Here, all 4 functions are overloaded functions because argument(s) passed to these functions are different.
Notice that, the return type of all these 4 functions are not same. Overloaded functions may or may not have different return type but it should have different argument(s).

1. // Error code
2. int test(int a){}
3. double test(int b){}

The number and type of arguments passed to these two functions are same even though the return type is different. Hence, the compiler will throw error.

Program-1 to demonstrate Function overloading:

```
#include <iostream>
using namespace std;
void display(int);
void display(float);
void display(int, float);
int main(){
    int a = 5;
    float b = 5.5;
    display(a);
    display(b);
    display(a, b);
    return 0;
}
void display(int var) {
    cout << "Integer number: " << var << endl;
}
void display(float var) {
    cout << "Float number: " << var << endl;
}
void display(int var1, float var2) {
    cout << "Integer number: " << var1;
    cout << " and float number:" << var2;
}
```

Program-2 to demonstrate Function overloading in C++:

```
#include <iostream>
using namespace std;
int absolute(int);
float absolute(float);
int main(){
    int a=-5;
    float b=5.5;
     cout << "Absolute value of " << a << " = " << absolute(a) << endl;
     cout << "Absolute value of " << b << " = " << absolute(b);
    return 0;
}
int absolute(int var){
 if(var<0)
     var=-var;
    return var;
}
float absolute(float var){
    if(var<0.0)
        var=-var;
    return var;
}
```

4. Write C++ programs to demonstrate following features in C++:

a) **Exception Handling**

b) **Sorting using Standard Template Library (STL)**

c) **Searching using Standard Template Library (STL)**

Answer: Exception handling in C++:

Example: 1 The output of program explains flow of execution of try/catch blocks-

```
#include <iostream>
using namespace std;
int main(){
   int x=-1;
   // Some code
   cout << "Before try \n";
   try{
      cout << "Inside try \n";
      if(x<0){
            throw x;
            cout << "After throw (Never executed) \n";
         }
      }
      catch(int x){
         cout << "Exception Caught \n";
      }
      cout << "After catch (Will be executed) \n";
      return 0;
}
```

Output:

```
Before try
Inside try
Exception caught
After catch (Will be executed)
```

Example: 2 There is a special catch block called 'catch all' catch(…) that can be used to catch all types of exceptions. For example, in the following program, an int is thrown as an exception, but there is no catch block for int, so catch(…) block will be executed.

```
#include <iostream>
using namespace std;
int main(){
    try{
       throw 10;
    }
    catch(char *excp){
        cout << "Caught " << excp;
    }
       catch(...){
           cout << "Default Exception\n";
       }
       return 0;
  }
```

Output:

```
Default expression
```

Example: 3 Implicit type conversion doesn't happen for primitive types. For example, in the following program 'a' is not implicitly converted to int.

```
#include <iostream>
using namespace std;
int main(){
    try{
       throw 'a';
    }
    catch(int x){
        cout << "Caught " << x;
    }
       catch(...){
           cout << "Default Exception\n";
       }
    return 0;
}
```

Output:

```
Default Exception
```

Example: 4 Nesting of try-catch blocks-

```
#include <iostream>
 using namespace std;
int main(){
    try{
        try{
            throw 20;
        }
        catch(int n){
             cout << "Handle Partially ";
                Throw;
           }
       }
       catch(int n){
           cout << "Handle remaining ";
       }
       return 0;
  }
```

Output:

Handle Partially Handle remaining

Sorting using Standard Template Library:

```
#include <iostream>
#include <algorithm>
using namespace std;
void show(int a[]){
    for(int i=0; i<10; ++i)
        cout << '\t' << a[i];
}
int main(){
    int a[10]={1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
      cout << "\n The array before sorting is : ";
      show(a);
      sort(a, a+10);
      cout << "\n\n The array after sorting is : ";
      show(a);
      return 0;
  }
```

Output:

```
The array before sorting is : 1    5    8    9    6
   7    3    4    2    0
The array after sorting is : 0    1    2    3    4
   5    6    7    8    9
```

Searching using Standard Template Library:

```
#include <iostream>
#include <algorithm>
using namespace std;
void show(int a[], int arraysize){
    for(int i=0; i<arraysize;++i)
        cout << '\t' << a[i];
}
int main(){
    int a[]={1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
         int asize=sizeof(a)/sizeof(a[0]);
         cout << "\n The array is : ";
        show(a, asize);
        cout << "\n\nLet's say we want to search for 2
   in the array";
        cout << "\n So, we first sort the array";
        sort(a, a + 10);
        cout << "\n\n The array after sorting is : ";
        show(a, asize);
        cout << "\n\nNow, we do the binary search";
```

```
        if (binary_search(a, a + 10, 2))
               cout << "\nElement found in the array";
        else
                   cout << "\nElement not found in the
array";
          cout << "\n\nNow, say we want to search for
10";
        if (binary_search(a, a + 10, 10))
               cout << "\nElement found in the array";
        else
                   cout << "\nElement not found in the
array";
       return 0;
}
```

Output:

```
The array before sorting is : 1    5     8     9     6
   7     3     4     2     0
Let’s say we want to search for 2 in the array
So, we first sort the array
The array after sorting is : 0     1     2     3     4
   5     6     7     8     9
Now, we do the binary search
Element found in the array
Now, say we want to search for 10
Element not found in the array
```

End of the assignment :)
