CC-211L

Object Oriented Programming

Laboratory 05

Friend Classes Functions and static Members

Version: 1.0.0

Release Date: 03-01-2023

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Friend functions
- Friend Classes
- Use of "this" pointer
- static Data Members
- static Member Function

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Lab Instructor	Dr.Prof. Sanam	
Teacher Assistants	Syed M.Jafar Ali Naqvi	bitf21m032@pucit.edu.pk
	M. Usman Munir	bitf19m020@pucit.edu.pk
	Aqdas Shabir	bitf21m022@pucit.edu.pk
	Ibad Nazir	bitf21m024@pucit.edu.pk
	Umer Farooq	bitf21m010@pucit.edu.pk

Background and Overview:

Friend Function:

A friend function can be granted special access to private and protected members of a class in C++. They are the non-member functions that can access and manipulate the private and protected members of the class for they are declared as friends.

Friend Class:

A **friend class** can access private and protected members of other classes in which it is declared as a friend. It is sometimes useful to allow a particular class to access private and protected members of other classes. For example, a LinkedList class may be allowed to access private members of Node.

'this' Pointer:

The "this" pointer is a pointer accessible only within the non-static member functions of a class. It points to the object for which the member function is called. Static member functions don't have a "this" pointer.

static Keyword:

When static keyword is used, variable or data members or functions cannot be modified again. It is allocated for the lifetime of program. Static functions can be called directly by using class name.

Static variables are initialized only once. Compilers persist the variable till the end of the program. Static variable can be defined inside or outside the function. They are local to the block. The default value of static variable is zero. The static variables are alive till the execution of the program.

Activities:

Pre-Lab Activities:

Friend Functions:

A friend function in C++ is a function that is declared outside a class but is capable of accessing the private and protected members of the class. There could be situations in programming where we want two classes to share their members. These members may be data members, class functions or function templates. In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data of members of the class.

Generally, non-member functions cannot access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes.

Example:

The picture code below shows the declaration, definition and working of friend function

```cpp
#include<iostream>
using namespace std;
class Distance{
private:
    int meter;
    // Friend Function
    // This allows the addFive which is a friend funtion to \n
    // Access the private and protected members of the class Distance
    friend int addFive(Distance D); // this is how a friend function is declared
public:
    Distance() : meter(0) {}
};
// friend function definition
int addFive(Distance d) {
    //accessing private members
    cout << "Distance Before: " << d.meter << endl;
    d.meter += 10;
    return d.meter;
}
int main() {
    Distance D;
    cout << "Distance After: " << addFive(D);
    return 0;
}
```

Fig. 01 (Working of a Friend Function)

Output:

Fig. 02 (Working of a Friend Function)

Example (Add Members of Two Different Classes using Friend Function):

```cpp
#include <iostream>
using namespace std;
// Forward declaration - We gurantee compiler that ClassB is implement after class A \n
// This is necessary otherwise the friend function declaration below won't work \n
class ClassB;
class ClassA{
private:
    int numA;
    // friend function declaration
    friend int add(ClassA n1, ClassB n2);
public:
    // constructor to initialize numA to 20
    ClassA() : numA(20) {}
};
class ClassB {
private: int numB;
       // friend function declaration
       friend int add(ClassA num1, ClassB num2);
public:
    // constructor to initialize numB to 30
    ClassB() : numB(30) {}
};
// access members of both classes
int add(ClassA objectA, ClassB objectB) {
    return (objectA.numA + objectB.numB);
}
int main()
{
    ClassA objectA;
    ClassB objectB;
    cout << "Sum: " << add(objectA, objectB);
    return 0;
}
```

Fig. 03 (Working of a Friend Function)

Output:

Fig. 04 (Working of a Friend Function)

Explanation:

In this program, **Class A** and **Class B** have declared add( ) as a friend function. Thus, this function can access private data of both classes. One thing to notice here is the friend function inside **Class A** is using the **Class B**. However, we haven 't defined **Class B** at this point. For this to work, we need a forward declaration of **Class B** in our program (This point is also discussed in comments).

Task 01: Complex Numbers **[Estimated time 20 minutes / 10 marks]**

Write a C++ to add two complex numbers using friend function. Your program should have a class named as **Complex,** your class should also have a function to print the result.

Sample Output:

Fig. 05 (Pre-Lab Task)

6 + 12i is the result of addition of (1+4i and 5+8i)

Task 02: Prime Sum **[Estimated time 30 minutes / 20 marks]**

Write a program in C++ to Check Whether a Number can be Express as Sum of Two Prime Numbers using the friend function.

Pictorial Representation:

Sample Output:

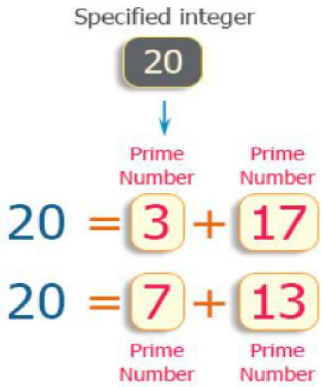

Fig. 06 (Pre-Lab Task)

Sample Output:

Fig. 07 (Pre-Lab Task)

Use your knowledge to understand where to use friend function in this program.

In-Lab Activities:

Understanding the Friend Classes:

A friend class can access private and protected members of other classes in which it is declared as a friend. It is sometimes useful to allow a particular class to access private and protected members of other classes.

Example Code:

Earlier we discussed how a friend function of a class can access its members. If we make the whole class (ClassB) a friend of other class (ClassA). Now any function of ClassB will have access to all the members of ClassA. Refer to the picture code below for better understanding:

```cpp
#include <iostream>
using namespace std;
// Forward declarationclass
class ClassB;
class ClassA
{
private:
    int numA;
    // friend function declaration
    friend class ClassB;
public:
    // constructor to initialize numA to 12
    ClassA() : numA(12) {}
};
class ClassB {
private:
    int numB;
public:
    // constructor to initialize numB to 1
    ClassB() : numB(1) {}
    // Member function to add numA from Class A and numB from classB
    int add()
    {
        ClassA objectA;
        return objectA.numA + numB;
    }
};
int main()
{
    ClassB objectB;
    cout << "Sum: " << objectB.add();
    return 0;
}
```

Fig. 08 (Friend Class)

Output:

Fig. 09 (Friend Class)

Since **ClassB** is a friend class, we can create objects of **ClassA** inside of **ClassB**.

'this' pointer:

Every object in C++ has access to its own address through an important pointer called 'this' pointer. The 'this' pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a 'this' pointer, because friends are not members of a class. Only member functions have a 'this' pointer.

Example:

```
1  #include<iostream>
2  using namespace std;
3  /* local variable is same as a member's name */
4  class Test
5  {
6  private:
7  int x;
8  public:
9      void setX(int x) { x = x; }
10     void getX() {
11         cout << "Value of X is: " << x << endl;
12     }
13 };
14 int main()
15 {
16     Test obj;
17     obj.setX(10);
18     obj.getX();
19     return 0;
20 }
```

Fig. 10 ('this Pointer')

Output:

Output will not be 20, rather program will display some garbage value. Our program failed to invoke the member.

```
Microsoft Visual Studio Debug Console
Value of X is: -858993460
```

Fig. 11 ('this' Pointer)

Solution:

See line# 9 change it to **this->x=x;**.

```cpp
#include<iostream>
using namespace std;
/* local variable is same as a member's name */
class Test
{
private:
int x;
public:
    void setX(int x) { this->x = x; }
    void getX() {
        cout << "Value of X is: " << x << endl;
    }
};
int main()
{
    Test obj;
    obj.setX(10);
    obj.getX();
    return 0;
}
```

Fig. 12 ('this Pointer')

'this' pointer can also be used to return reference to the calling object:

Example Code:

Kindly observe the code and comments carefully for the better understanding:

```cpp
#include<iostream>
using namespace std;
/* local variable is same as a member's name */
class Test
{
private:
int x;
public:
    Test& setX(int x) {
        this->x = x;
        return *this;//Return test object
    }
    void getX() {
        cout << "Value of X is: " << x << endl;
    }
};
int main()
{
    Test obj;
    obj.setX(10).getX(); //Return value
    return 0;
}
```

Fig. 13 ('this Pointer')

Output:

```
Microsoft Visual Studio Debug Console
Value of X is: 10
```

Fig. 14 ('this' Pointer)

static data members:

- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is initialized before any object of this class is being created, even before main starts.
- It is visible only within the class, but its lifetime is the entire program

Example:

```
#include <iostream>
using namespace std;
class Student
{
private:
int rollNo;
char name[10];
int marks;
public:
static int objectCount; // this is how we declare the static data member using static keyword
Student() {
    objectCount++; // everytime the constructor gets called the object counter will increment by 1
}
void getdata() {
    cout << "Enter roll number: ";
    cin >> rollNo;
    cout << "Enter name: ";
    cin >> name;
    cout << "Enter marks: ";
    cin >> marks; }
void putdata() {
    cout << "Roll Number = " << rollNo << endl;
    cout << "Name = " << name << endl;
    cout << "Marks = " << marks << endl; cout << endl;
}
};
int Student::objectCount = 0;
int main(void) {
    Student s1; s1.getdata(); s1.putdata();
    Student s2; s2.getdata(); s2.putdata();
    cout << "Total objects created = " << Student::objectCount << endl; return 0;
}
```

Fig. 15 (static Data member)

Output:

```
Microsoft Visual Studio Debug Console
Enter roll number: 22
Enter name: Zain
Enter marks: 88
Roll Number = 22
Name = Zain
Marks = 88

Enter roll number: 21
Enter name: Saad
Enter marks: 88
Roll Number = 21
Name = Saad
Marks = 88

Total objects created = 2
```

Fig. 16 (static Data member)

static Member function:

Static member functions in C++ are the functions that can access only the static data members. These static data members share a single copy of themselves with the different objects of the same class. A function can be made static by using the keyword static before the function name while defining a class.

We will make some changes in the above code and make ourselves a static member function **getCount** that will display the **objectCount** (a static data member)

Example:

```
#include <iostream>
using namespace std;
class Student
{
private:
int rollNo; char name[10]; int marks;
public:
static int objectCount; // this is how we declare the static data member using static keyword
Student() {
    objectCount++; // everytime the constructor gets called the object counter will increment by 1
}
void getdata() {
    cout << "Enter roll number: ";
    cin >> rollNo;
    cout << "Enter name: ";
    cin >> name;
    cout << "Enter marks: ";
    cin >> marks; }
void putdata() {
    cout << "Roll Number = " << rollNo << endl;
    cout << "Name = " << name << endl;
    cout << "Marks = " << marks << endl; cout << endl;
}
static void getCount() {
    cout << "Object Count = " << objectCount << endl;
}
};
int Student::objectCount = 0;
int main(void) {
    Student s1; s1.getdata(); s1.putdata();
    Student s2; s2.getdata(); s2.putdata();
    Student::getCount();  return 0;
}
```

Fig. 17 (static Member function)

Task 01: Integer Array [Estimated time 20 minutes / 25 marks]

Write a program with a class integer that contains an array of integers. Initialize the integer array in the constructor of the class. Then create friend functions to the class:

- Find the largest integer in the array.
- Find the smallest integer in the array.
- Find the repeated elements in array.
- Sort the elements of array in ascending order.
- Create a destructor that sets all of the elements in the array to 0.

Task 02: Friend Class [Estimated time 20 minutes / 20 marks]

Develop a C++ program to find the area of a rectangle by converting the member of a class square (data member: side) which is a friend class of rectangle (data member: height, width). Declare Rectangle as a friend of Square so that Rectangle member functions could have access to the private member of square.

Task 03: Count Customers [Estimated time 20 minutes / 20 marks]

Create a class Customer.

Properties:

- customerID
- Customer Name
- Phone
- gender

It should have a static variables count and id. Its initial value will be 5.It should be decremented each time an object of this customer class is created. Create a function getCount() function to get the number of objects of customers created of customer class. If count reaches zero then throw exception in the constructor of the class and do not create object. Customer will be assigned from customerID from static variable id inside the constructor. Then id will be incremented to be assigned to the next object of the class. In this way, customerID of all customers will be unique. Create a display function for Customer class as well.

Task 04: Aggregated Family [Estimated time 40 minutes / 35 marks]

Class Date:

Properties:

Day, Month, Year

Method:

- DisplayDate function: display Data in this format: DD-MM-Year.

Class Member:

Properties:

- memberId, Name, gender, DateOfBirth (type Date)
- Id (static variable to automatically assign value to each family in constructor)

Method:

- Display member
- GetAge: It will return the age of the family member

Class Family:

Properties

- Family ID
- Members array of type Member (it will be a D-array, initially assigned nullptr in constructor)
- number of members (variable to calculate number of members in family)
- Id (static variable to automatically assign value to each family in constructor)

Methods:

- Display whole Family
- Add member to family
- Remove a member from a family
- Update data of a member of family.

Main function:

In main function, provide menu based functionality to access all functions of Family class.

Post-Lab Activities:

Task 01: BankAccount [Estimated time 40 minutes / 30 marks]

Village Community Banks (VICOBA) is a microfinance scheme to improve the economic status of people living in rural areas in Tanzania. In this scheme, members take out a loan and are required to pay back the money after a fixed number of years, and for each year, a fixed interest rate must be paid.

Create a VicobaAccount class that will help VICOBA track their customers' loans. Provide data members to represent the

- amount borrowed
- the annual interest rate
- the loan duration in years

The annual interest rate should be declared **static**.

Also provide a static member function that will be used to alter the annual interest rate, another function that will report the amount of money a customer has to repay based on the annual interest rate and the loan duration (assume simple interest is used) and a constructor that accepts the loan amount and duration. Finally, add get and set functions for all the data members.

Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .cpp file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** **[30 marks]**
  - Task 01: Complex Numbers [10 marks]
  - Task 02: Prime Sum [20 marks]
- **Division of In-Lab marks:** **[70 marks]**
  - Task 01: Integer Array [30 marks]
  - Task 02: Object Invocation [20 marks]
  - Task 03: Friend Class [20 marks]
- **Division of Post-Lab marks:** **[30 marks]**
  - Task 01: VicobaAccount [30 marks]

References and Additional Material:

- **Friend Function and Friend Classes**
  https://www.programiz.com/cpp-programming/friend-function-class
- **'this' Pointer**
  https://www.geeksforgeeks.org/this-pointer-in-c/
- **static Member function**
  https://www.geeksforgeeks.org/static-member-function-in-cpp/
- **static Data member**
  https://www.geeksforgeeks.org/static-data-members-c/

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:40: In-Lab Task
- Slot – 03 – 00:40 – 01:20: In-Lab Task
- Slot – 04 – 01:20 – 02:20: In-Lab Task
- Slot – 05 – 02:20 – 02:45: Evaluation of Lab Tasks
- Slot – 06 – 02:45 – 03:00: Discussion on Post-Lab Task