# Department of Electrical Engineering

**CS212**

**Object Oriented Programming**



# Lab 7: Friend Functions, Friend Classes, Static Variables and Inheritance

**Class**: BEE - 12C

**Date**: November 8th, 2021

**Time**: Monday (1400 – 1700)

**Name**: Muhammad Umer

**CMS ID**: 345834

**Learning Objectives**

Understanding and applying following concepts about inheritance in C++

* Inheritance syntax in C++
* Protected Access specifier
* Constructors and Destructors in base and derived class and their order of execution
* Effect of Base Access in Inheritance (public, protected and private)

**Tools**

* Microsoft Visual Studio 2013

**Inheritance**

Inheritance is a process of creating new classes (derived class) from existing (base) class. We do it for one of the following reasons:

* Reusability
* Code/Software Reliability
* Improved Conceptualization and overall design of program

**Example:**

Suppose we have a class “Cuboid” having three properties: *length*, *width* and *height*.

#include <iostream>

using namespace std;

class Cuboid {

 private:

  double length;

 protected:

  double width;

 public:

  double height;

  Cuboid() : length(2), width(3), height(4) {

    cout << "Base Class: Default Constructor " << endl;

  }

  Cuboid(double a, double b, double c) : length(a), width(b), height(c) {

    cout << "Base Class: Parameterized Constructor " << endl;

  }

  void setData(double a, double b, double c) {

    length = a;

    width = b;

    height = c;

  }

  double getLength() { return length; }

  void displayData() {

    cout << "Length = " << length << endl;

    cout << "Width = " << width << endl;

    cout << "Height = " << height << endl;

  }

  ~Cuboid() { cout << "Base Class: Destructor\n" << endl; }

};

class myCuboid : public Cuboid {

 public:

  myCuboid() { cout << "Derived Class: Default Constructor\n" << endl; }

  void calculateVolume() {

    cout << "Volume of Cuboid: " << getLength() \* width \* height << endl;

  }

  ~myCuboid() { cout << "Derived Class: Destructor\n" << endl; }

};

int main() {

  myCuboid object;

  object.calculateVolume();

  object.displayData();

  return 0;

}

**Lab Tasks**

**Task 1:**

1. Understand and execute the given example.
2. Verify the *three cases* of *Base Class Access*

**After creating an Accessor *getLength()* function in the body of the base class, we can verify various cases for running the code.**

1. **Public:** The code runs perfectly as the ***void displayData()*** function we utilize in main is made a public member of the derived class.

**Terminal Output**

**PS D:\NUST\Semester 3\Object Oriented Programming\Labs> lab7.exe**

Base Class: Default Constructor

Derived Class: Default Constructor

Volume of Cuboid: 24

Length = 2

Width = 3

Height = 4

Derived Class: Destructor

Base Class: Destructor

1. **Protected:** The code will return an error as the public member ***void displayData()*** function we utilize in main is made a protected member of the derived class; one that is inaccessible in the ***int*** ***main()*** function.

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:

In function 'int main()':

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:47:21: error: 'void Cuboid::displayData()' is inaccessible within this context

   47 |   object.displayData();

      |   ~~~~~~~~~~~~~~~~~~^~

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:25:8: note: declared here

   25 |   void displayData() {

      |        ^~~~~~~~~~~

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:47:21: error: 'Cuboid' is not an accessible base of 'myCuboid'

   47 |   object.displayData();

      |   ~~~~~~~~~~~~~~~~~~^~

Build finished with error(s).

1. **Private:** The code will yet again return an error as the public member ***void displayData()*** function we utilize in main is made a private member of the derived class; one that is inaccessible in the ***int*** ***main()*** function.

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:

In function 'int main()':

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:47:21: error: 'void Cuboid::displayData()' is inaccessible within this context

   47 |   object.displayData();

      |   ~~~~~~~~~~~~~~~~~~^~

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:25:8: note: declared here

   25 |   void displayData() {

      |        ^~~~~~~~~~~

D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7\lab7.cpp:47:21: error: 'Cuboid' is not an accessible base of 'myCuboid'

   47 |   object.displayData();

      |   ~~~~~~~~~~~~~~~~~~^~

Build finished with error(s).

1. Try to access the three data members of base class in derived class (in some function) for each of three cases, and if access is not possible (in any case), try to find a solution. Note: do not change the access of data members: length, width and height in base class.

* One possible solution is to make an Accessor function in the body of the base class to return the value of the private data member ***length***.

**Accessor**

double getLength() {

return length;

}

**Task 2**:

A software based startup company has two software development teams. Each team has four employees. After completion of each successful project delivered to the client, a bonus is given to the team that has completed the project. Each year the CEO of the company announces performance based increments (based on the annual profit of the company) which is added in the basic salary of each employee.

Create a set of classes to implement the above scenario by using Inheritance, friend functions wherever applicable.

You have to keep track of the number of projects completed by the company and the overall profit after deducting the salaries of staff and miscellaneous expenditures.

In the driver class, create an object of Company and add employees. The programming interface should have provisions to include/organize employees in development teams, assign project to a development team and print salaries after performance based increments.

**Code**

#include <iostream>

using namespace std;

int random(int min, int max);

class TeamProto;

class Employees {

public:

    double employeeSalary = 25000; // $ p/h

};

class Company {

    friend class TeamProto;

    int totalProjects;

    double netProfit;

public:

    Company(double projectCount1, double projectCount2, double sumSalaries1,

        double sumSalaries2)

    {

        totalProjects = projectCount1 + projectCount2;

        netProfit = (random(75000, 90000) \* totalProjects) - sumSalaries1 - sumSalaries2;

    }

    void displayInfo()

    {

        cout << "\t\tCompany Data" << endl;

        cout << "Total Projects Completed: " << totalProjects << endl;

        cout << "Net Profit: " << netProfit << endl;

    }

};

class TeamProto : public Employees {

    double employee\_1, employee\_2, employee\_3, employee\_4;

    double performanceIncrement = 0.1;

public:

    int projectCount;

    TeamProto(int a) { projectCount = a; }

    void setSalaries(int x)

    {

        employee\_1 = random(33050, 42000) + (employeeSalary \* performanceIncrement \* x);

        employee\_2 = random(33050, 42000) + (employeeSalary \* performanceIncrement \* x);

        employee\_3 = random(33050, 42000) + (employeeSalary \* performanceIncrement \* x);

        employee\_4 = random(33050, 42000) + (employeeSalary \* performanceIncrement \* x);

    }

    double sumSalaries = employee\_1 + employee\_2 + employee\_3 + employee\_4;

    friend void employeeData(TeamProto);

    friend void teamData(TeamProto);

};

int random(int min, int max) // range : [min, max]

{

    static bool first = true;

    if (first) {

        srand(time(NULL)); // seeding for the first time only!

        first = false;

    }

    return min + rand() % ((max + 1) - min);

}

void employeeData(TeamProto team)

{

    cout << "Employee 1 Salary: " << team.employee\_1 << endl;

    cout << "Employee 2 Salary: " << team.employee\_2 << endl;

    cout << "Employee 3 Salary: " << team.employee\_3 << endl;

    cout << "Employee 4 Salary: " << team.employee\_4 << endl;

}

void teamData(TeamProto team)

{

    cout << "\nCollective Team Salary: "

         << team.employee\_1 + team.employee\_2 + team.employee\_3 + team.employee\_4

         << endl;

    cout << "Project Count: " << team.projectCount << endl;

}

int main()

{

    TeamProto team1(3), team2(6);

    team1.setSalaries(team1.projectCount);

    team2.setSalaries(team2.projectCount);

    Company Z(team1.projectCount, team2.projectCount, team1.sumSalaries,

        team2.sumSalaries);

    Z.displayInfo();

    cout << "\tData of Team 1";

    teamData(team1);

    employeeData(team1);

    cout << "\n\tData of Team 2";

    teamData(team2);

    employeeData(team2);

    return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Terminal Output**

PS D:\NUST\Semester 3\Object Oriented Programming\Labs\Lab 7> lab7.exe

Company Data

Total Projects Completed: 9

Net Profit: 703089

Data of Team 1

Collective Team Salary: 178236

Project Count: 3

Employee 1 Salary: 47529

Employee 2 Salary: 44174

Employee 3 Salary: 45663

Employee 4 Salary: 40870

Data of Team 2

Collective Team Salary: 205795

Project Count: 6

Employee 1 Salary: 53092

Employee 2 Salary: 53214

Employee 3 Salary: 50127

Employee 4 Salary: 49362

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**