

CC-211L

Object Oriented Programming

Laboratory 10

Inheritance - II

Version: 1.0.0

Release Date: 23-03-2023

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
 - Inheritance
 - Base Class
 - Derived Class
 - Constructor
 - Destructor
- Activities
 - Pre-Lab Activity
 - Constructor and Destructor in Derived Classes
 - Single Inheritance
 - Single Inheritance with Arguments
 - Multiple Inheritance
 - Multiple Inheritance with Arguments
 - Task 01
 - In-Lab Activity
 - public, protected, and private Inheritance
 - Public Inheritance
 - Protected Inheritance
 - Private Inheritance
 - Relationship among objects in an Inheritance Hierarchy
 - Invoking Base-Class Functions from Derived-Class Objects
 - Derived-Class Member-Function Calls via Base-Class Pointers
 - Task 01
 - Task 02
 - Task 03
 - Post-Lab Activity
 - Task 01
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

Learning Objectives:

- Constructors in Derived Classes
- Destructors in Derived Classes
- Public protected and private Inheritance
- Relationship among objects in Inheritance

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	Azka Saddiqa	azka.saddiqa@pucit.edu.pk
Teacher Assistants	Saad Rahman	bsef19m021@pucit.edu.pk
	Zain Ali Shan	bcsf19a022@pucit.edu.pk

Background and Overview:

Inheritance:

Inheritance is one of the core concepts of object-oriented programming (OOP) languages. It is a mechanism where you can derive a class from another class for a hierarchy of classes that share a set of attributes and methods.

Base Class:

A base class is an existing class from which the other classes are determined, and properties are inherited. It is also known as a superclass or parent class. In general, the class which acquires the base class can hold all its members and some further data as well.

Derived Class:

A derived class is a class that is constructed from a base class or an existing class. It tends to acquire all the methods and properties of a base class. It is also known as a subclass or child class.

Constructor:

A constructor in C++ is a special method that is automatically called when an object of a class is created.

Destructor:

A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete.

Activities:

Pre-Lab Activities:

Constructor and Destructor in Derived Class:

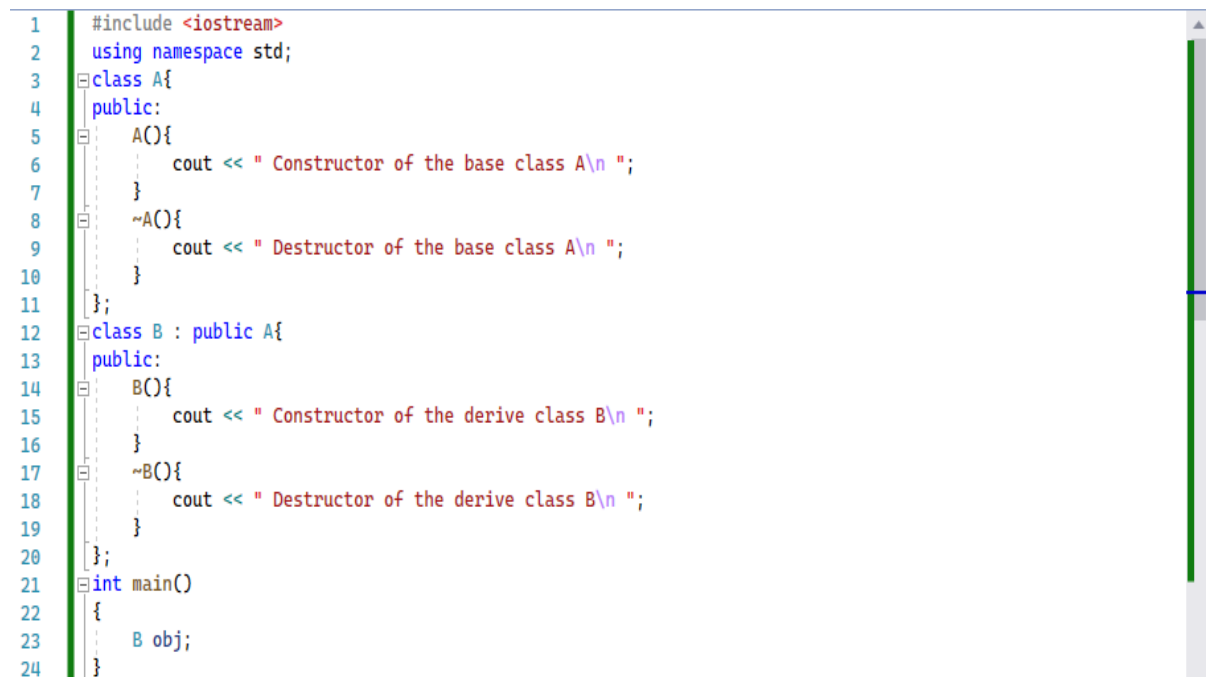
Single Inheritance:

When an object of the derived class is created, a part of the base class is also included with that object. The base class is initiated to include the base class part with the object of the derived class and then the derived class part is included. So, when the derived class object is created, the constructor of the base class is automatically executed first and then the constructor of the derived class is executed.

The base class constructors are not inheritance by derived class. however, a derived class constructor always calls the constructor for its base class first to initialize base class data members for the objects of the derived class. If the derived class constructor is omitted, the derived class default constructor calls the base class constructor.

Destructors are called in the reverse order of constructor calls. So, a derived class destructor is called its base class destructor.

Example:



```
1  #include <iostream>
2  using namespace std;
3  class A{
4  public:
5      A(){
6          cout << " Constructor of the base class A\n ";
7      }
8      ~A(){
9          cout << " Destructor of the base class A\n ";
10     }
11 };
12 class B : public A{
13 public:
14     B(){
15         cout << " Constructor of the derive class B\n ";
16     }
17     ~B(){
18         cout << " Destructor of the derive class B\n ";
19     }
20 };
21 int main()
22 {
23     B obj;
24 }
```

Fig. 01 (Constructor and Destructor in Single Inheritance)

Output:



```
Microsoft Visual Studio Debug Console
Constructor of the base class A
Constructor of the derive class B
Destructor of the derive class B
Destructor of the base class A
```

Fig. 02 (Constructor and Destructor in Single Inheritance)

Single Inheritance with arguments:

The derived class is responsible for calling the constructor of the base class. In case of constructor of the base class with arguments, the syntax to define constructor of the derived class is different. The base class constructor is connected with the derived class constructor by using the colon (:) in the header of the derived class constructor. The parameters of the base class constructor are also given as parameter list in the derived class constructor. In derived class constructor, the parameters of the base class come first than the derived class constructor.

Example:

```

1  #include <iostream>
2  using namespace std;
3  class A{
4  private:
5      int x, y;
6  public:
7      A(int m, int n){
8          x = m;
9          y = n;
10     }
11     void print(){
12         cout << " value of x = " << x << endl;
13         cout << " value of y = " << y << endl;
14     }
15 };
16 class B : public A{
17 private:
18     float i, j;
19 public:
20     B(int m, int n, float a, float b) : A(m, n){
21         i = a;
22         j = b;
23     }
24     void show(){
25         cout << " value of i = " << i << endl;
26         cout << " Value of j = " << j << endl;
27     }
28 };
29 int main(){
30     B obj(1, 2, 20.1, 10.3);
31     obj.print();
32     obj.show();
33 }

```

Fig. 03 (Constructor and Destructor in Single Inheritance with arguments)

Output:

```

Microsoft Visual Studio Debug Console
value of x = 1
value of y = 2
value of i = 20.1
value of j = 10.3

```

Fig. 04 (Constructor and Destructor in Single Inheritance with arguments)

Multiple Inheritance:

In multiple inheritance, the constructors of the base classes and constructors of the derived class are automatically executed when an object of the derived class is created. The constructors of the base classes are executed first and then the constructor of the derived class is executed. The constructors of

base classes are executed in the same order in which the base classes are specified in derived class (i.e., from left to right).

Similarly, the destructors are executed in reverse order, i.e., derived class constructor is executed first and then constructors of the base classes are executed (from right to left).

Example:

```

1  #include <iostream>
2  using namespace std;
3  class base1{
4  public:
5      base1(){
6          cout << "Constructor of class base1\n";
7      }
8      ~base1(){
9          cout << "Destructor of class base1\n";
10     }
11 };
12 class base2{
13 public:
14     base2(){
15         cout << "Constructor of class base2\n";
16     }
17     ~base2(){
18         cout << "Destructor of class base2\n";
19     }
20 };
21 class derive1 : public base1, public base2
22 {
23 public:
24     derive1(){
25         cout << "Constructor of class derive1\n";
26     }
27     ~derive1(){
28         cout << "Destructor of class derive1\n";
29     }
30 };
31 int main(){
32     derive1 x;
33     cout << " Destructors are: " << endl;
34 }

```

Fig. 05 (Constructor and Destructor in Multiple Inheritance)

Output:



```

Microsoft Visual Studio Debug Console
Constructor of class base1
Constructor of class base2
Constructor of class derive1
Destructors are:
Destructor of class derive1
Destructor of class base2
Destructor of class base1

```

Fig. 06 (Constructor and Destructor in Multiple Inheritance)

Multiple Inheritance with arguments:

The constructor of the derived class calls the constructors of the base classes in the same order in which they are specified in the header of the derived class. The syntax define constructors in multiple inheritance with arguments is also like the definition of single inheritance with arguments. The constructors of the base classes relate to the constructor of the derived class by using colon (:) and separated by commas.

Example:

```
1  #include <iostream>
2  using namespace std;
3  class base1{
4  public:
5      base1(int x, int y){
6          cout << "Sum of values passed to base1: " << x + y << endl;
7      }
8  };
9  class base2{
10 public:
11     base2(string str)
12     {
13         cout << "String passed to base2: " << str << endl;
14     }
15 };
16 class derive1 : public base1, public base2{
17 public:
18     derive1(int x, int y, string str, double d) : base1(x, y), base2(str){
19         cout << "Value passed to derive1: " << d;
20     }
21 };
22 int main(){
23     derive1 obj(2,3,"Hello World", 1.5);
24 }
```

Fig. 07 (Constructor and Destructor in Multiple Inheritance with arguments)

Output:

Microsoft Visual Studio Debug Console

```
Sum of values passed to base1: 5
String passed to base2: Hello World
Value passed to derive1: 1.5
```

Fig. 08 (Constructor and Destructor in Multiple Inheritance with arguments)

Task 01: Car Manufacturing**[Estimated time 25 minutes / 20 marks]**

You are working on a software project for a car manufacturing company. The project involves designing a class hierarchy to represent different types of cars. You are required to implement:

- A base class **'Car'** and two derived classes **'ElectricCar'** and **'GasolineCar'**
- The **'ElectricCar'** class will have a member variable **'batteryLife'** to represent the battery life of the car
- The **'GasolineCar'** class will have a member variable **'fuelTankCapacity'** to represent the fuel tank capacity of the car

Your task is to implement the constructors and destructors for each class in such a way that the member variables are initialized properly and all allocated resources are released when the objects are destroyed.

Specifically, your implementation should:

- Initialize the member variables of each class appropriately in the constructors
- Ensure that any resources allocated in the constructors are properly released in the destructors
- Demonstrate how the constructors and destructors work in the context of inheritance, for example, by creating objects of the derived classes and calling their constructors and destructors and showing how the base class constructor and destructor are also called

In-Lab Activities:**public, protected, and private Inheritance:**

In C++ inheritance, we can derive a child class from the base class in different access modes. E.g.,

```
class Base {
.... ..
};

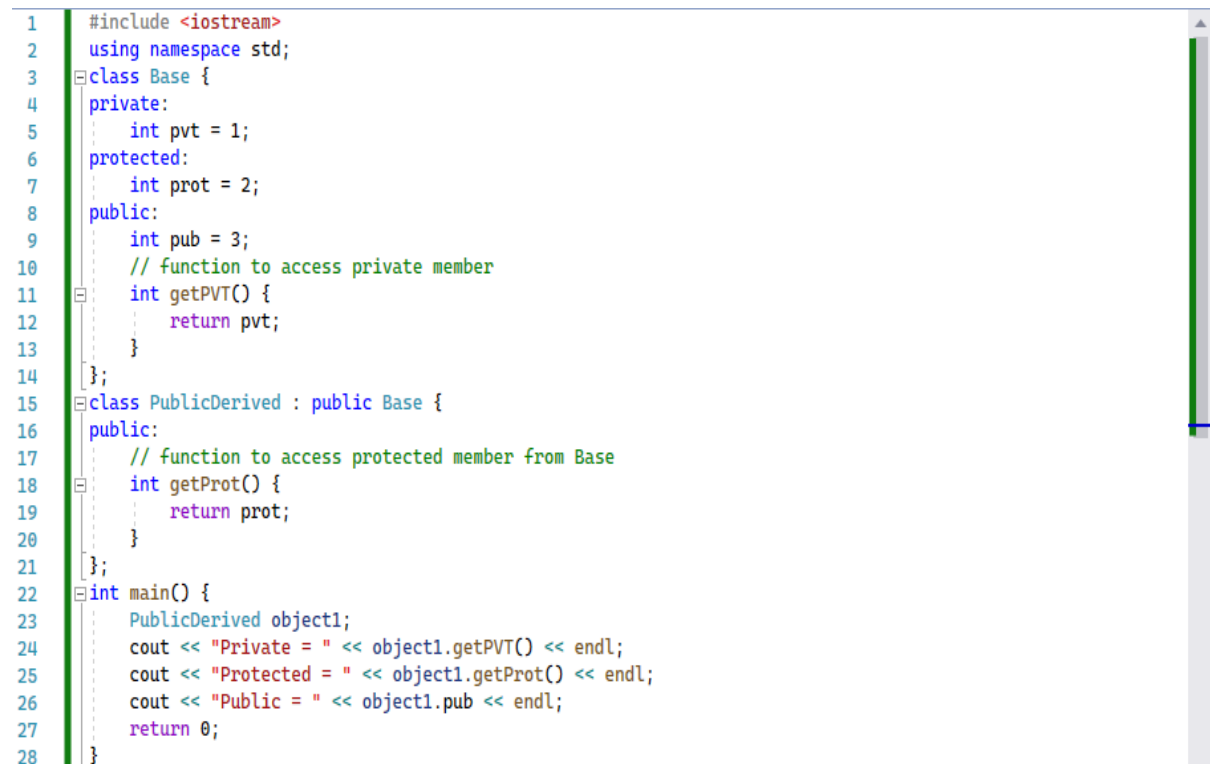
class Derived : public Base {
.... ..
};
```

Notice the keyword `public` in the code `'class Derived : public Base'`

This means that we have created a derived class from the base class in public mode. Alternatively, we can also derive classes in protected or private modes. These 3 keywords (public, protected, and private) are known as access specifiers in inheritance.

Public Inheritance:

Public inheritance makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.

Example:


```
1  #include <iostream>
2  using namespace std;
3  class Base {
4  private:
5      int pvt = 1;
6  protected:
7      int prot = 2;
8  public:
9      int pub = 3;
10     // function to access private member
11     int getPVT() {
12         return pvt;
13     }
14 };
15 class PublicDerived : public Base {
16 public:
17     // function to access protected member from Base
18     int getProt() {
19         return prot;
20     }
21 };
22 int main() {
23     PublicDerived object1;
24     cout << "Private = " << object1.getPVT() << endl;
25     cout << "Protected = " << object1.getProt() << endl;
26     cout << "Public = " << object1.pub << endl;
27     return 0;
28 }
```

Fig. 09 (public Inheritance)

Output:


```

Microsoft Visual Studio Debug Console
Private = 1
Protected = 2
Public = 3

```

Fig. 10 (public Inheritance)

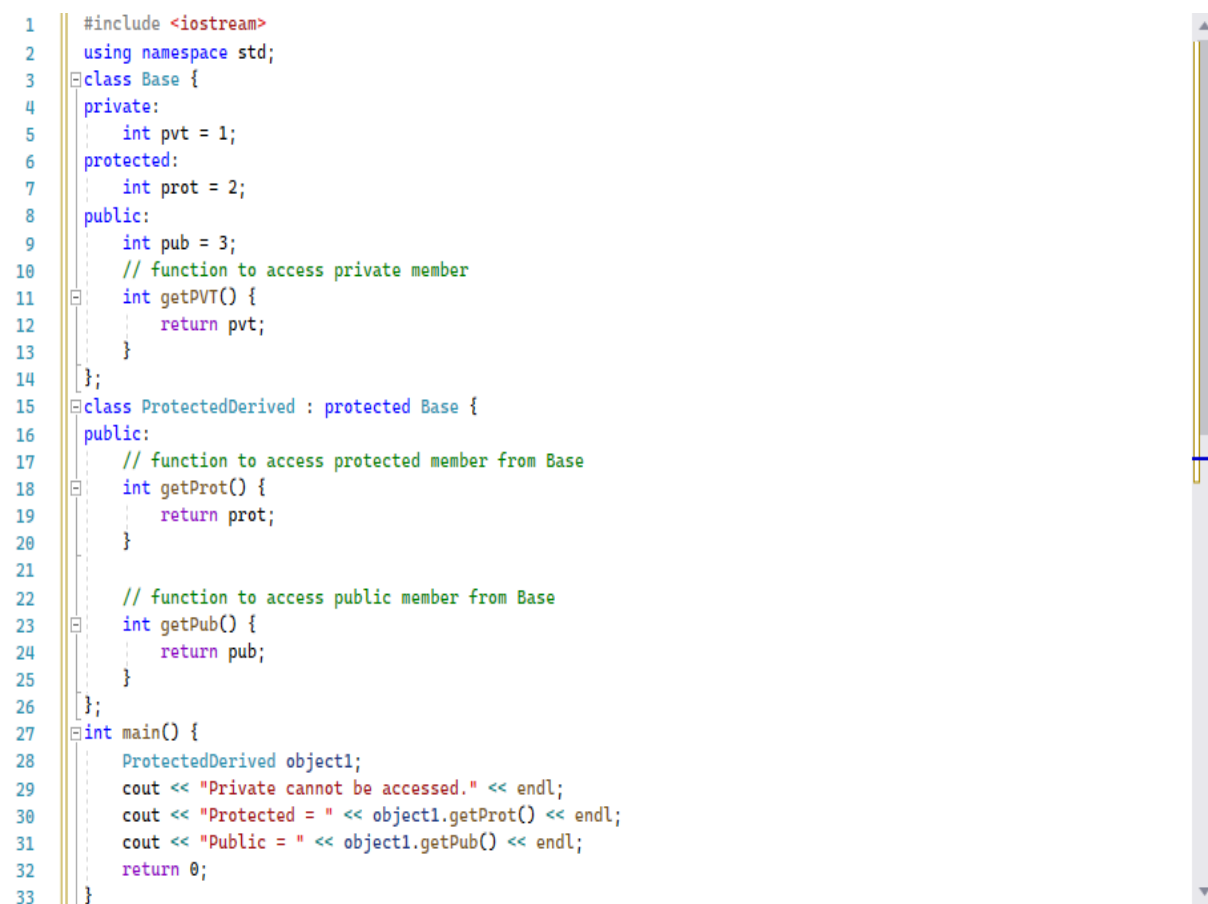
Since private and protected members are not accessible from main(), we need to create public functions getPVT() and getProt() to access them.

Accessibility:

Accessibility	Private Members	Protected Members	Public Members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes	Yes

Protected Inheritance:

Protected inheritance makes the public and protected members of the base class protected in the derived class.

Example:


```

1  #include <iostream>
2  using namespace std;
3  class Base {
4  private:
5      int pvt = 1;
6  protected:
7      int prot = 2;
8  public:
9      int pub = 3;
10     // function to access private member
11     int getPVT() {
12         return pvt;
13     }
14 };
15 class ProtectedDerived : protected Base {
16 public:
17     // function to access protected member from Base
18     int getProt() {
19         return prot;
20     }
21
22     // function to access public member from Base
23     int getPub() {
24         return pub;
25     }
26 };
27 int main() {
28     ProtectedDerived object1;
29     cout << "Private cannot be accessed." << endl;
30     cout << "Protected = " << object1.getProt() << endl;
31     cout << "Public = " << object1.getPub() << endl;
32     return 0;
33 }

```

Fig. 11 (protected Inheritance)

Output:


```

Microsoft Visual Studio Debug Console
Private cannot be accessed.
Protected = 2
Public = 3

```

Fig. 12 (protected Inheritance)

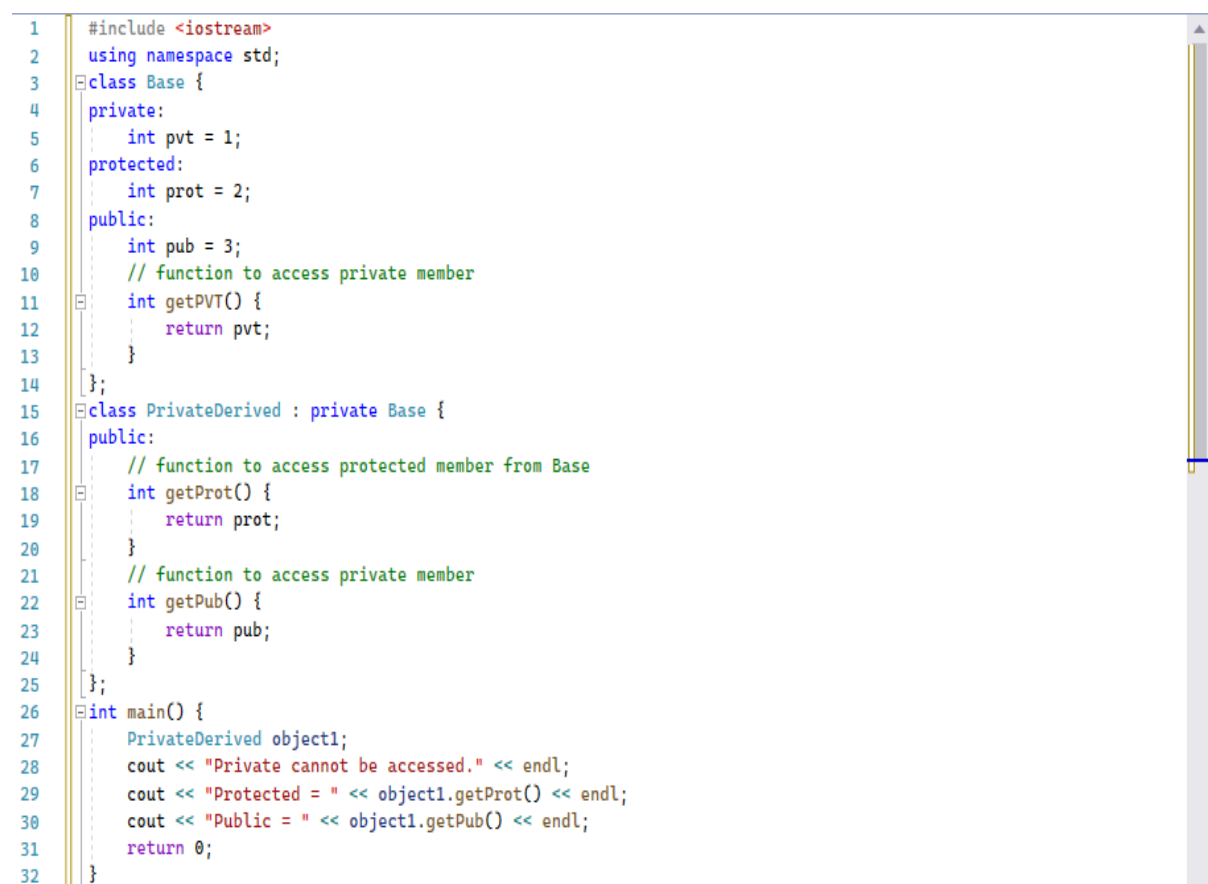
As we know, protected members cannot be directly accessed from outside the class. As a result, we cannot use getPVT() from ProtectedDerived. That is why we need to create the getPub() function in ProtectedDerived in order to access the pub variable.

Accessibility:

Accessibility	Private Members	Protected Members	Public Members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes	Yes (Inherited as protected variable)

Private Inheritance:

Private inheritance makes the public and protected members of the base class private in the derived class.

Example:


```

1  #include <iostream>
2  using namespace std;
3  class Base {
4  private:
5      int pvt = 1;
6  protected:
7      int prot = 2;
8  public:
9      int pub = 3;
10     // function to access private member
11     int getPVT() {
12         return pvt;
13     }
14 };
15 class PrivateDerived : private Base {
16 public:
17     // function to access protected member from Base
18     int getProt() {
19         return prot;
20     }
21     // function to access private member
22     int getPub() {
23         return pub;
24     }
25 };
26 int main() {
27     PrivateDerived object1;
28     cout << "Private cannot be accessed." << endl;
29     cout << "Protected = " << object1.getProt() << endl;
30     cout << "Public = " << object1.getPub() << endl;
31     return 0;
32 }

```

Fig. 13 (private Inheritance)

Output:

```

Microsoft Visual Studio Debug Console
Private cannot be accessed.
Protected = 2
Public = 3

```

Fig. 14 (private Inheritance)

As we know, private members cannot be directly accessed from outside the class. As a result, we cannot use getPVT() from PrivateDerived. That is why we need to create the getPub() function in PrivateDerived in order to access the pub variable.

Accessibility:

Accessibility	Private Members	Protected Members	Public Members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes (Inherited as private variable)	Yes (Inherited as private variable)

Relationship among objects in an Inheritance Hierarchy:**Invoking Base-Class Functions from Derived-Class Objects:**

A derived class can invoke a function of its base class by using the scope resolution operator (::) and the name of the function. This is useful when the derived class wants to call a function that is implemented in the base class, or to access a member variable that is defined in the base class.

Example:

```

1  #include <iostream>
2  using namespace std;
3  // Base class
4  class Animal {
5  protected:
6      void eat() {
7          cout << "The animal is eating." << endl;
8      }
9  };
10 // Derived class
11 class Dog : public Animal {
12 public:
13     void bark() {
14         cout << "The dog is barking." << endl;
15     }
16     void dogEat() {
17         Animal::eat(); // call the eat() function of the base class
18     }
19 };
20 int main() {
21     Dog myDog;
22     myDog.bark(); // call the bark() function of the derived class
23     myDog.dogEat(); // call the eat() function of the base class
24     return 0;
25 }

```

Fig. 15 (Relationship among objects)

Output:

```
Microsoft Visual Studio Debug Console
The dog is barking.
The animal is eating.
```

Fig. 16 (Relationship among objects)

Derived-Class Member-Function Calls via Base-Class Pointers:

Off a base-class pointer, the compiler allows us to invoke only base-class member functions. Thus, if a base-class pointer is aimed at a derived-class object, and an attempt is made to access a derived-class-only member function, a compilation error will occur.

Example:

```
1  #include <iostream>
2  using namespace std;
3  // Base class
4  class Shape {
5  public:
6      void draw() {
7          cout << "Drawing a shape." << endl;
8      }
9  };
10 // Derived class
11 class Rectangle : public Shape {
12 public:
13     void drawR(){
14         cout << "Drawing a rectangle." << endl;
15     }
16 };
17 // Main function
18 int main() {
19     Shape* myShape = new Rectangle();
20     myShape->drawR();
21     delete myShape;
22     return 0;
23 }
```

Fig. 17 (Relationship among objects)

Error:

	Code	Description
abc	E0135	class "Shape" has no member "drawR"

Fig. 18 (Relationship among objects)

Task 01: Animals**[Estimated time 30 minutes / 20 marks]**

- Create a base class called '**Animal**' with a private data member
 - name
- Add a constructor to the Animal class that initializes the name data member
- Create two derived classes from the Animal class called '**Mammal**' and '**Bird**'
- The Mammal class should have a private data member
 - numLegs
 - a constructor that initializes the name and numLegs data members
- The Bird class should have a private data member
 - wingSpan
 - a constructor that initializes the name and wingSpan data members
- Create two additional derived classes from the Mammal class called '**Dog**' and '**Cat**'
- The Dog class should have a private data member
 - breed
 - a constructor that initializes the name, numLegs, and breed data members
- The Cat class should have a private data member
 - color
 - a constructor that initializes the name, numLegs, and color data members
- Finally, create a new main() function that creates objects of all four classes (Mammal, Bird, Dog, and Cat) and test the constructors and destructors
- Make sure to test how the constructors and destructors of each class are called when creating and deleting objects

Task 02: Student Data**[Estimated time 30 minutes / 20 marks]**

- Define a class '**Student**' as a base class with data members
 - admissionNo,
 - Name
 - age and
 - address as protected type and
 - getStudent() function for setting values for data members
- These members and functions are common to both derived classes
- Make two derived classes of Student class as '**Undergraduate**' and '**Graduate**' each containing data member
 - degree_program in which student is enrolled and
 - member functions getdegree() to set degree_program
 - how() to display whole information
- Output should be as follows:



```

Microsoft Visual Studio Debug Console

----Graduate Student----
Admission No: 1
Name: Ahmad
Age: 22
Degree: CS

----Undergraduate Student----
Admission No: 2
Name: Ali
Age: 18
Degree: SE

```

Fig. 19 (In-Lab Task)

Task 03: Employee**[Estimated time 30 minutes / 20 marks]**

- Create a class of '**Employee**' having data member
 - employee ID
 - name, and
 - designation
- This class should have at least two member functions. One function takes input from users and update the data members. The second function should display the value of data members
- Design another class '**Salary**' that has five data members
 - basic pay
 - human resource allowance
 - dearness allowance
 - profitability fund, and
 - net pay.
- This class should have following functions:
 - getEmployeeDetails() that take input from users in all the data members and calculate net pay using formula:

Net pay = basic pay + human resource allowance + dearness allowance - profitability fund

This function, at the start, should invoke the function of employee class that takes input from the user.

- Display() that first invokes display function of employee class and then display the data members of Salary class (including net pay)
- Design another class named '**BankCredit**' that has two data members:
 - bank name and
 - account number.
- It should have two member functions as:
 - getBankDetails() that first invokes the getter method of Salary class and then takes input from user for bank name and account number
 - Print() that first invokes display method of Salary class and then display the data members of BankCredit class
- In main(), declare array of objects of BankCredit class with size 5. It should then take input from the user about the number of employees. It should then invoke getBankDetails() and print() function for each employee

Post-Lab Activities:**Task 01: Restaurant Ordering System****[Estimated time 60 minutes / 40 marks]**

Design a class hierarchy for a restaurant ordering system. The system should have the following requirements:

- The restaurant offers different types of food items, such as appetizers, entrees, and desserts.
- Each food item should have a name, description, and price.
- The restaurant also offers different types of beverages, such as soft drinks, juices, and alcoholic drinks.
- Each beverage should have a name, description, and price.
- The restaurant offers different types of discounts, such as percentage-based discounts and dollar-based discounts.
- The restaurant offers different types of payment methods, such as credit cards, debit cards, and cash.
- The ordering system should be able to calculate the total cost of an order, including any discounts and taxes.

Your task is to design a class hierarchy that models these requirements, using appropriate inheritance. You should also write code to demonstrate how the classes work together in the context of the ordering system.

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:** [20 marks]
 - Task 01: Car Manufacturing [20 marks]
- **Division of In-Lab marks:** [60 marks]
 - Task 01: Animals [20 marks]
 - Task 02: Student Data [20 marks]
 - Task 03: Employee [20 marks]
- **Division of Post-Lab marks:** [40 marks]
 - Task 01: Restaurant Ordering System [40 marks]

References and Additional Material:

- **Inheritance**
<https://www.programiz.com/cpp-programming/inheritance>
- **Base and Derived Classes**
<https://learncplusplus.org/learn-c-inheritance-base-classes-and-derived-classes/>
- **Constructor and Destructor in Derived Classes**
<https://www.geeksforgeeks.org/order-constructor-destructor-call-c/>
- **public, protected, and private Inheritance**
<https://www.programiz.com/cpp-programming/public-protected-private-inheritance>

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