FAST

National University of Computer and Emerging Sciences Peshawar

OOP Lab # 11

C++ (Inheritance)

Computer Instructor: Muhammad Abdullah Orakzai

DEPARTMENT OF COMPUTER SCIENCE

Programming

الذی علم بالقلم۔ علم الانسان ما لم یعلم۔

Contents

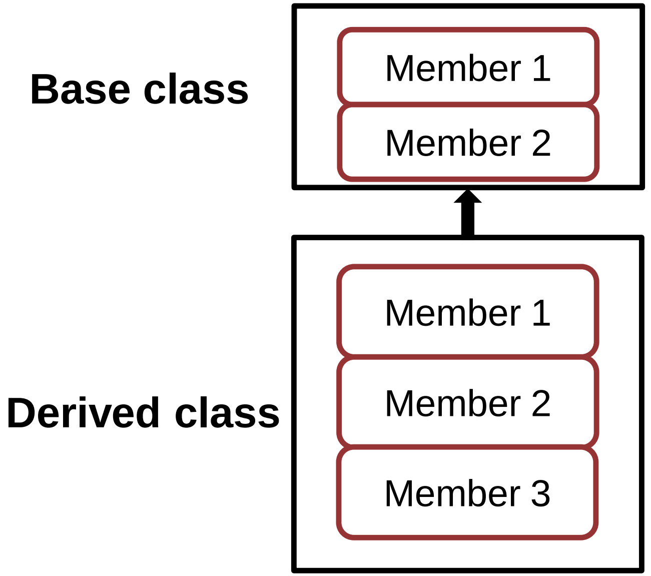
1) C++ Inheritance
2) C++ Protected Access Specifier
3) C++ Defining Derived Classes
4) C++ Public Inheritance
5) C++ Private Inheritance
6) C++ Protected Inheritance

Inheritance

- Inheritance is the second most important feature of Object Oriented Programming.
- In inheritance the code of existing class is used for making new class.
- This saves time for writing and debugging the entire code for a new class.
- To inherit means to receive. In inheritance a new class is written such that it can access or use the members of an existing class. The new class that can access the members of an existing class is called ***derived class*** or ***child class***. The existing class is called the ***base class*** or ***parent class***.
- The derived class can use the data members and member functions of the base class. It can have its own data members and member functions. Thus a derived can even be larger than a base class.

Inheritance...

- The figure shows the relationship between a derived class and the base class.
- The arrow is drawn from derived class to the base class.
- The direction of arrow indicates that the derived class can access members of the base class but the base class cannot access members of its derived class.
- The figure shows that the derived class has only one member of its own i.e. member 3.



Inheritance…

- The two members i.e. member 1 and member 2 shown in figure are the members of the base class. The derived class can also access these two members of the base class.
- Thus, whereas an object of the **base class** can access only two members, an object of the **derived class** can access three members.
- The inheritance relationship enables a derived class to inherit features from its base class. Furthermore, the derived class can add new features of its own. Therefore, rather than create completely new classes from scratch, you can take advantage of inheritance and reduce software complexity.

Inheritance…

- A new class can be derived from one or more existing classes. Based upon the member of the base classes from which a class is derived, the inheritance is divided into two categories.

- Single Inheritance
- Multiple Inheritance

Inheritance...

Single Inheritance

In single inheritance, the new class is derived from only one base class.

Multiple Inheritance

In multiple inheritance, the new class is derived from more than one base classes.

Protected Access Specifiers

- The public members of a class are accessible by all functions in the program and the private members of a class are accessible only by member functions and friend functions of that class. Similarly, the protected members of a class are accessible by the member functions and the friend functions of that class.
- The **protected members** of a base class are , however, **accessible** by members of its derived classes but the **private members** of the base class **are not accessible directly** by members of its derived classes. This is the main difference between the **protected** and the **private** access specifiers.
- The protected members of a base class fall between private and public member. These members are public for the derived class but for the rest of the program, these are treated as private.

Protected Access Specifiers...

```
// C++ program to demonstrate protected access modifier
#include <iostream>
using namespace std;

// base class
class Parent
{
    // protected data members
    protected:
    int id_protected;
}; //parent class ends
```

Protected Access Specifiers...

```
// sub class or derived class from public base class
class Child : public Parent
{
    public:
    void setId(int id)
    {
        // Child class is able to access the inherited
        // protected data members of base class
        id_protected = id;
    }
    void displayId()
    {
        cout << "id_protected is: " << id_protected << endl;
    }
};  // child class ends
```

Protected Access Specifiers...

```
// main function
int main() {

    Child obj1;
    // member function of the derived class can
    // access the protected data members of the base class
    obj1.setId(81);
    obj1.displayId();
    return 0;
}  // end of main() function
```

Defining Derived Classes

The syntax for defining a derived class is slightly different from the syntax of the base class definition.

The declaration of a derived class also includes the name of the base class from which it derived.

The general syntax for defining a derived class is:

```
class  sub_class_name  :  specifier base_class_name
{
  members to derived class
} ;
```

Defining Derived Classes…

class sub_class_name represents name of the derived class

: (colon) sets relation between the classes

specifier represents the access specifiers. It may be public, private or protected.

base_class_name represents the name of the base class.

Defining Derived Classes…

For Example, a class “student” is defined as

```
class student
{
        private:
        char name[15], address[15];
        public:
        void input(void);
        void show(void);
}
```

Defining Derived Classes…

The class ***student*** has two data members and two member functions.

Suppose the marks obtained by a student in three different subjects and the total marks of these are to be included as new data members in the above class. This is done by adding new members in the class. There are two ways in which these new members can be added to the class. These are:

- Add new members in the original class

Or

- Define a new class that has the new members and that also uses members of the existing "**student**" class. Using the members of an existing is the principle of inheritance. The new class is the derived class. The existing class serves as the base for the derived class.

Defining Derived Classes…

Deriving a new class from an existing class reduces the size of the program. It also eliminates duplication of code within the program.

For example, let the name of the new class be ***marks***. This class uses the members of the existing ***student*** class.

Defining Derived Classes…

```
class marks  : public students
{
        private:
        int s1,s2,s3,total;
        public:
        void inputmarks(void);
        void show_detail(void);
} ;
```

Defining Derived Classes…

The class ***marks*** is derived from the class ***student***. The class ***marks*** is the derived class. The class ***student*** is the base class.

The derived class ***marks*** has four data members of integer type and two member functions . It also uses the code of the base class ***student***.

The derived class cannot access directly the private data members of the base class **student** by using the dot operator. These members are only accessible to the derived through the interface function within the base class.

The program given below explains the above example.

Defining Derived Classes...

```
#include<iostream>
using namespace std;
class student
{
    private:
    char name[15], address[15];
    public:
    void input(void)
    {
        cout<<"Enter your name: ";
        cin>>name;
        cout<<"Enter address:";
        cin>>address;
    }
```

Defining Derived Classes...

```
void show(void)
    {
        cout<<"Name is: "<<name<<endl;
        cout<<"Address is :"<<address<<endl;
    }
}; // end of base class
```

Defining Derived Classes...

```
//derived class
class marks : public student
{
    private:
    int s1, s2,s3,s4, total;
    public:
    void inputmarks(void)
    {
        cout<<"Enter marks of sub1: ";cin>>s1;
        cout<<"Enter marks of sub2: ";cin>>s2;
        cout<<"Enter marks of sub3: ";cin>>s3;
        total= s1+s2+s3;
    }
    void show_detail(void);
};  // end of derived class
```

Defining Derived Classes...

```
int main()
{
    marks mmm;
    mmm.input();
    mmm.inputmarks();
    mmm.show_detail();

} // end of main() function
```

Defining Derived Classes...

```
void marks :: show_detail()
{
    show();
    cout<<"Marks of 1st subject: "<<s1<<endl;

    cout<<"Marks of 2nd subject: "<<s2<<endl;

    cout<<"Marks of 3rd subject: "<<s3<<endl;

    cout<<"Total Marks          : "<<total<<endl;

}
```

Output:

```
Enter your name: Sana
Enter address: Kohat
Enter marks of sub1: 99
Enter marks of sub2: 77
Enter marks of sub3: 66

Name is: Sana
Address is : Kohat
Marks of 1st subject: 99
Marks of 2nd subject: 77
Marks of 3rd subject: 66
Total Marks         : 242
```

Defining Derived Classes...

The class “***marks”*** is defined as derived class. The keyword “***public***” and the name of the base class “***student***” followed by colon (:) are written while defining the derived class. This shows that objects of the derived class are able to access public members of the base class. It is called ***Public Inheritance***.

The derived class “**marks**” can access the “**input()**” and “**show()**” member functions of the base class. It cannot access other private members of the base class.

An object “***mmm***” of the class “marks” is created. The member function “***input()”*** of the class “***student***” is called through “***mmm***” object of the class “***marks***”. Similarly, the “***show()”*** function is also called in the “***show_detail()”*** member function of the class “***marks***” since the derived class marks has been declared as public of the “***student***” class. The objects of the “***marks***” class can access only the public members of the base class “**student**”.

Types of Inheritance w.r.t Access Control

There are three kinds of inheritance w.r.t access control

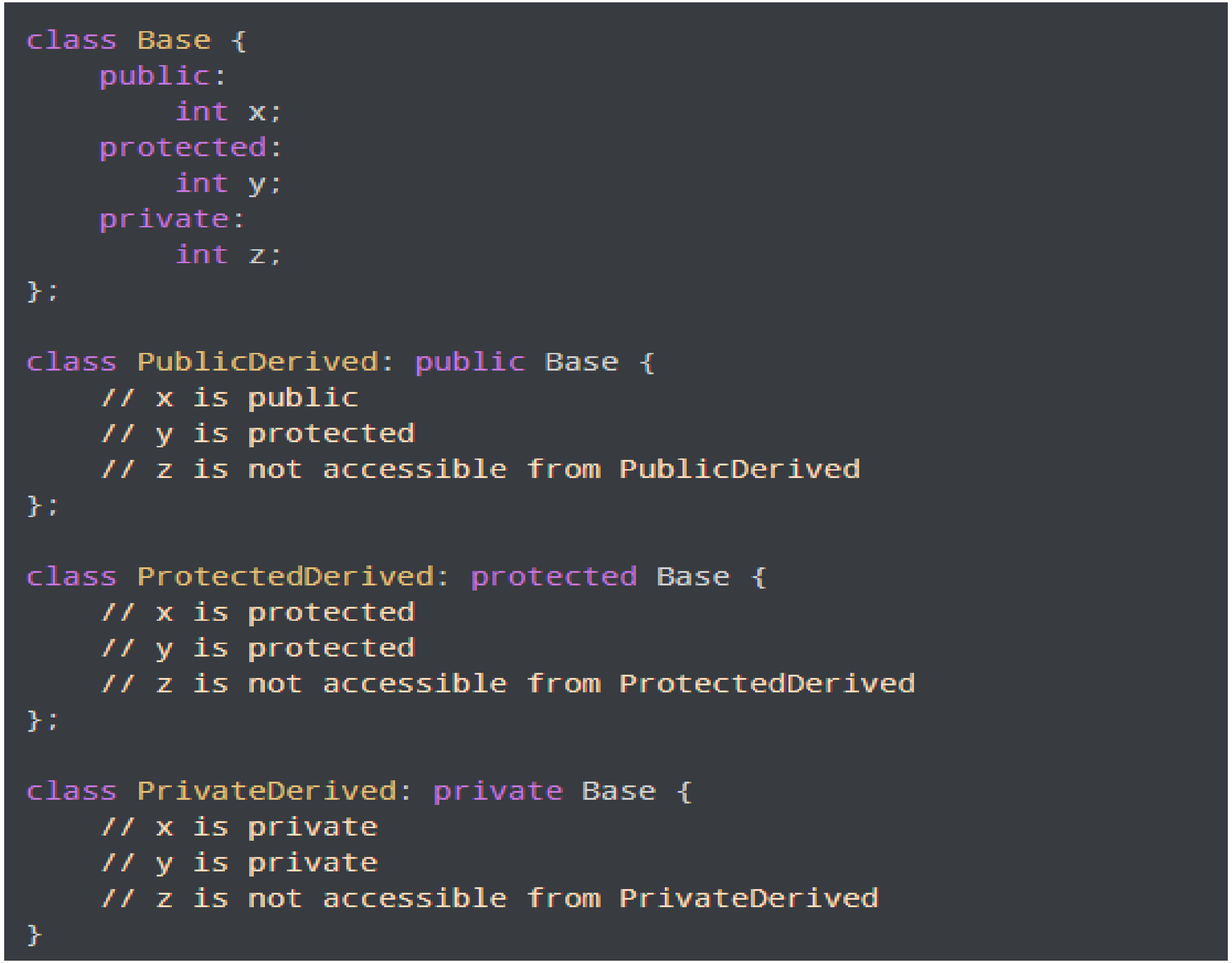
1) Public Inheritance
2) Private Inheritance
3) Protected Inheritance

1) Public Inheritance

- In public inheritance, the public members of the base class become the public members of the derived class.
- Thus the objects of the derived class can access public members (both data and functions) of the base class.
- Similarly, the protected data members of the base class also become the protected members of derived class.
- **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.
- If a derived class is declared in public mode, then the members of the base class are inherited by the derived class just as they are.

1) Public Inheritance

- **Note:** private members of the base class are inaccessible to the derived class.



```
class Base {
    public:
        int x;
    protected:
        int y;
    private:
        int z;
};

class PublicDerived: public Base {
    // x is public
    // y is protected
    // z is not accessible from PublicDerived
};

class ProtectedDerived: protected Base {
    // x is protected
    // y is protected
    // z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
    // x is private
    // y is private
    // z is not accessible from PrivateDerived
}
```

Accessibility in public Inheritance

Accessibility	private members	protected members	public members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes	Yes

1) Public Inheritance…

The general syntax for deriving a public class from base class is:

```
class  sub_class_name  :  public base_class_name
{
   ----------
   ----------
} ;
```

Where

public	specifies the public inheritance
sub_class_name	represents the name of the derived class.
base_class_name	represents name of the base class

1) Public Inheritance...

```
#include<iostream>
using namespace std;
class A
{
    private:
    int a1, a2;
    protected:
    int pa1, pa2;
    public:
    void ppp(void)
    {
        cout<<"Value of pa1 of class A: "<<pa1<<endl;
        cout<<"Value of pa2 of class A: "<<pa2<<endl;
    }
}; // end of base class A
```

1) Public Inheritance...

```
//derived class
class B : public A
{
    public:
    void get(void)
    {
        cout<<"Enter value of pa1: "; cin>>pa1;
        cout<<"Enter value of pa2: "; cin>>pa2;
    }
};  // end of derived class B
```

1) Public Inheritance...

```
int main()
{
    B obj;
    obj.get();
    obj.ppp();
} // end of main() function
```

Output:

```
Enter value of pa1: 23
Enter value of pa2: 33

Value of pa1 of class A: 23
Value of pa2 of class A: 33
```

In the above program, the class B is publicly derived from class A.

The objects of the class B:

- Cannot access the private data members **a1** and **a2** of base class A.
- Can access the public member function **ppp()** of base class A.
- Can access the protected data members **pa1** and **pa2** of base class A.

1) Public Inheritance…

In this diagram, shape is the base class. The class rectangle is derived from shape. Every rectangle is a shape. Suppose class B is derived from class A. Then, B cannot directly access the private members of A. That is, the private members of A are hidden in B.

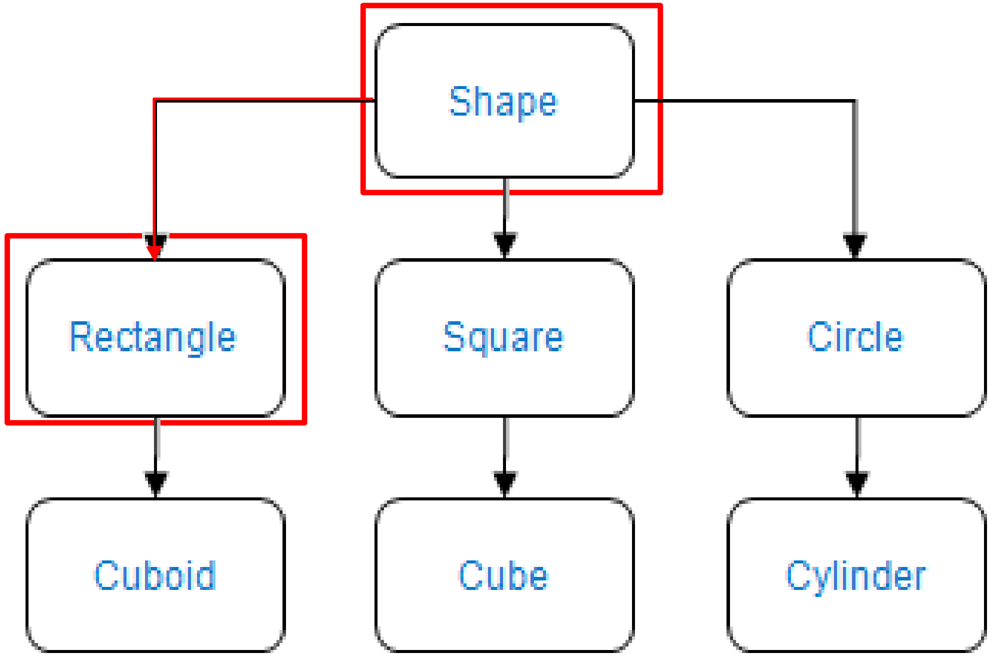
If ***memberAccessSpecifier*** is public—that is, the inheritance is public—then:

a) The public members of A are public members of B. They can be directly accessed in class B.

b) The protected members of A are protected members of B. They can be directly accessed by the member functions (and friend functions) of B.

c) The private members of A are hidden in B. They cannot be directly accessed in B. They can be accessed by the member functions (and friend functions) of B through the public or protected members of A.

1) Public Inheritance…



1) Public Inheritance...

```
#include <iostream>
using namespace std;
class Shape  // Base class
{
public:
    void setWidth(int w)
    {
        width = w;
    }
    void setHeight(int h)
    {
        height = h;
    }
protected:
    int width;
    int height;
};
```

1) Public Inheritance...

```
// Derived class
class Rectangle: public Shape
{
public:
    int getArea()
    {
        return (width * height);
    }
};
```

1) Public Inheritance...

```
int main(void)
{
Rectangle Rect;
Rect.setWidth(5);
Rect.setHeight(7);

// Print the area of the object.
cout << "Total area: " << Rect.getArea() << endl;
return 0;

}

/*
Total area: 35
*/
```

Test public inheritance

```
// tests publicly
#include <iostream>
#include <conio.h>
using namespace std;
class A //base class
{
private:
    int privdataA;
protected:
    int protdataA;
public:
    int pubdataA;
};
```

Test public inheritance...

```
class B : public A //publicly-derived class
{
public:
    void funct()
    {
        int a;
        a = privdataA; //error: not accessible
        a = protdataA; //OK
        a = pubdataA; //OK
    }
};
```

Test public inheritance...

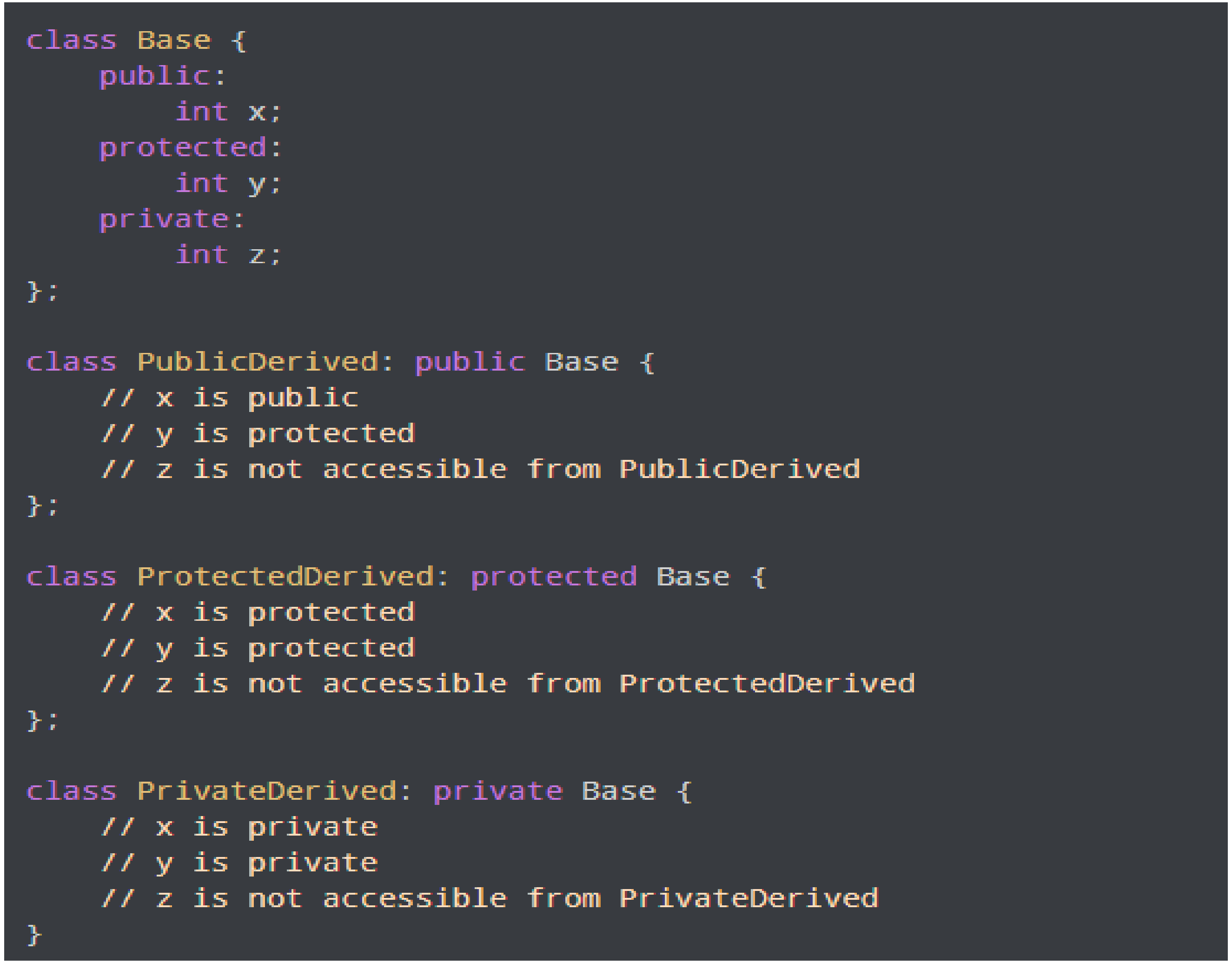
```
void main()
{
    int a;
    B objB;
    a = objB.privdataA; //error: not accessible

    a = objB.protdataA; //error: not accessible

    a = objB.pubdataA; //OK (A public to B)
}
```

2) Private Inheritance

- In Private Inheritance, the objects of the derived class cannot access the public members of the base class.
- Its objects can only access the protected data members of the base class.
- In this case, all the members of the base class become private members in the derived class.
- **private inheritance** makes the **public** and **protected** members of the base class **private** in the derived class.
- The private members of the base class are always private in the derived class.



```cpp
class Base {
    public:
        int x;
    protected:
        int y;
    private:
        int z;
};

class PublicDerived: public Base {
    // x is public
    // y is protected
    // z is not accessible from PublicDerived
};

class ProtectedDerived: protected Base {
    // x is protected
    // y is protected
    // z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
    // x is private
    // y is private
    // z is not accessible from PrivateDerived
}
```

Accessibility in private Inheritance

Accessibility	private members	protected members	public members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes (inherited as private variables)	Yes (inherited as private variables)

2) Private Inheritance…

The general syntax for deriving a private class from base class is:

```
class  sub_class_name  :  private base_class_name
{
   ----------
   ----------
} ;
```

Where

private	specifies the private inheritance
sub_class_name	represents the name of the derived class.
base_class_name	represents name of the base class

2) Private Inheritance...

```
#include<iostream>
using namespace std;
class A
{
    private:
    int a1, a2;
    protected:
    int pa1, pa2;
    public:
    void ppp(void)
    {
        cout<<"Value of pa1 of class A: "<<pa1<<endl;
        cout<<"Value of pa2 of class A: "<<pa2<<endl;
    }
}; // end of base class A
```

2) Private Inheritance…

```
//derived class
class B : private A   //privately-derived class
{
    public:
    void get(void)
    {
        cout<<"Enter value of pa1: "; cin>>pa1;
        cout<<"Enter value of pa2: "; cin>>pa2;
        cout<<"Value of pa1 of class A: "<<pa1<<endl;
        cout<<"Value of pa2 of class A: "<<pa2<<endl;
    }
};  // end of derived class B
```

2) Private Inheritance…

```
int main()
{
    B obj;
    obj.get();
    //obj.ppp();
} // end of main() function
```

Output:

```
Enter value of pa1: 12
Enter value of pa2: 33

Value of pa1 of class A: 12
Value of pa2 of class A: 33
```

In the above program, the class B is derived as private from the base class A.
The objects of the class B:

- Cannot access the private data members **a1** and **a2** of base class A.
- Cannot access the public member function **ppp()** of base class A.
- Can only access the protected data members **pa1** and **pa2** of base class A.

2) Private Inheritance…

If access specifier is private—that is, the inheritance is **private**—then:

a) The public members of A are private members of B. They can be accessed by the member functions (and friend functions) of B.

b) The protected members of A are private members of B. They can be accessed by the member functions (and friend functions) of B.

c) The private members of A are hidden in B. They cannot be directly accessed in B. They can be accessed by the member functions (and friend functions) of B through the public or protected members of A.

2) Private Inheritance…

```
// tests publicly- and privately-derived classes

#include <iostream>
#include <conio.h>
using namespace std;
class A //base class
{
private:
    int privdataA;
protected:
    int protdataA;
public:
    int pubdataA;
};
```

2) Private Inheritance...

```
class B : public A //publicly-derived class
{
public:
    void funct()
        {
            int a;
            a = privdataA; //error: not access
ible
            a = protdataA; //OK
            a = pubdataA; //OK
        }
};
```

2) Private Inheritance…

```
class C : private A //privately-derived class
{
public:
    void funct()
        {
            int a;
            a = privdataA; //error: not accessible
            a = protdataA; //OK
            a = pubdataA; //OK
        }
};
```

2) Private Inheritance...

```
void main()
{
    int a;
    B objB;
    a = objB.privdataA; //error: not accessible
    a = objB.protdataA; //error: not accessible
    a = objB.pubdataA; //OK (A public to B)

    C objC;

    a = objC.privdataA; //error: not accessible
    a = objC.protdataA; //error: not accessible
    a = objC.pubdataA; //error: not accessible (A private to C)
}
```

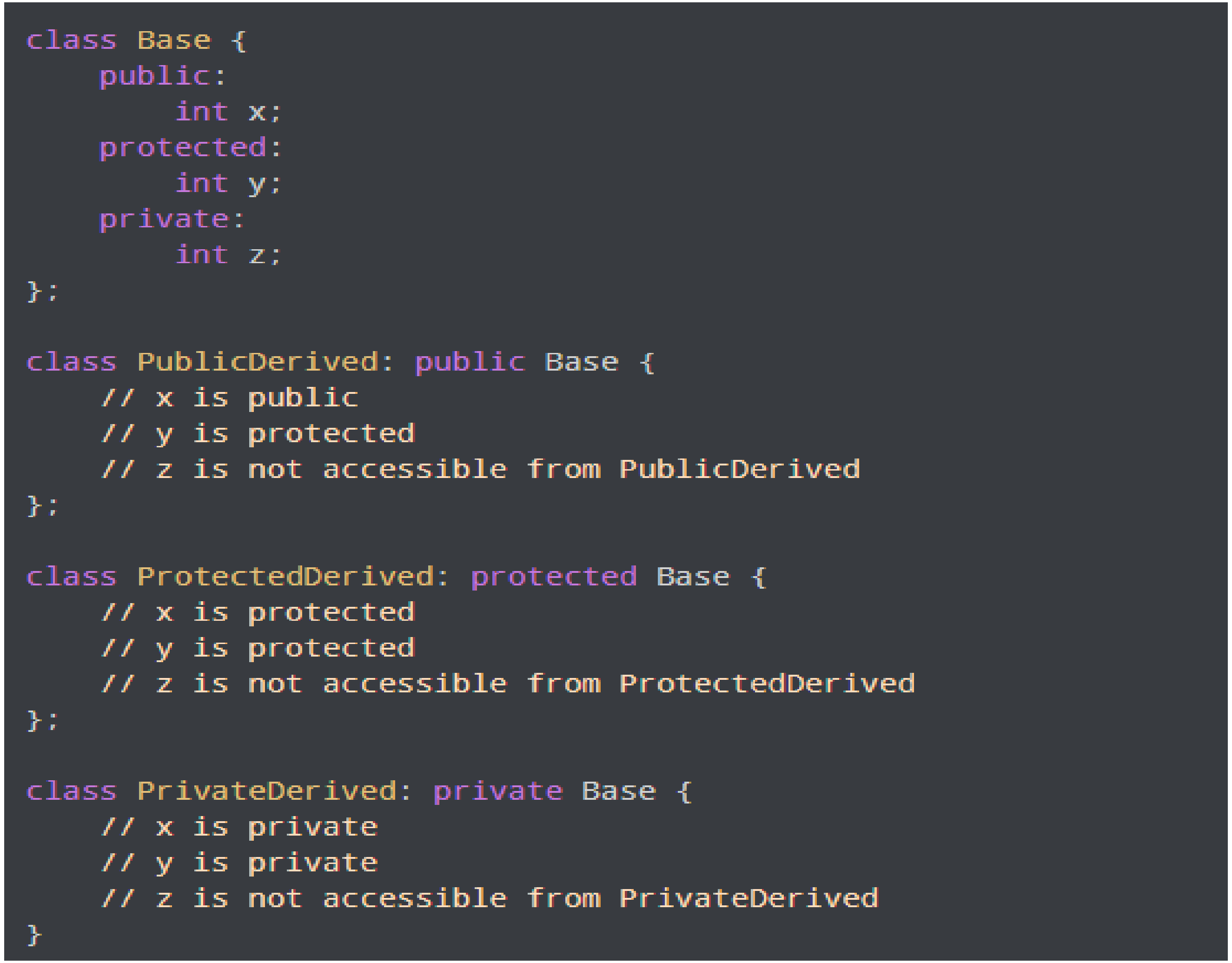
2) Private Inheritance…

Note:

If you don’t supply any access specifier when creating a class, private is assumed.

3) Protected Inheritance

- The object of the class that is derived as protected can only access the protected member of the base class.
- The public members of the base class become protected members in the derived class.
- **protected inheritance** makes the **public** and **protected** members of the base class **protected** in the derived class.
- The general syntax for deriving a protected class from base class is:



```
class Base {
    public:
        int x;
    protected:
        int y;
    private:
        int z;
};

class PublicDerived: public Base {
    // x is public
    // y is protected
    // z is not accessible from PublicDerived
};

class ProtectedDerived: protected Base {
    // x is protected
    // y is protected
    // z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
    // x is private
    // y is private
    // z is not accessible from PrivateDerived
}
```

Accessibility in protected Inheritance

Accessibility	private members	protected members	public members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes	Yes (inherited as protected variables)

3) Protected Inheritance…

```
class  sub_class_name  :  protected base_class_name
{
   ----------
   ----------
} ;
```

Where

protected	specifies the protected inheritance
sub_class_name	represents the name of the derived class.
base_class_name	represents name of the base class

3) Protected Inheritance...

```
#include<iostream>
using namespace std;
class A
{
    private:
    int a1, a2;
    protected:
    int pa1, pa2;
    public:
    void ppp(void)
    {
        cout<<"Value of pa1 of class A: "<<pa1<<endl;
        cout<<"Value of pa2 of class A: "<<pa2<<endl;
    }
}; // end of base class A
```

3) Protected Inheritance…

```
//derived class
class B : protected A   // protectedly-derived class
{
    public:
    void get(void)
    {
        cout<<"Enter value of pa1: "; cin>>pa1;
        cout<<"Enter value of pa2: "; cin>>pa2;
        cout<<"Value of pa1 of class A: "<<pa1<<endl;
        cout<<"Value of pa2 of class A: "<<pa2<<endl;
    }
};  // end of derived class B
```

3) Protected Inheritance…

```
int main()
{
    B obj;
    obj.get();
    //obj.ppp();
} // end of main() function
```

Output:

```
Enter value of pa1: 12
Enter value of pa2: 33

Value of pa1 of class A: 12
Value of pa2 of class A: 33
```

In the above program, the class B is derived as protected from the base class A.

The object of class B:

- Can only access the protected data members pa1 and pa2 of the base class A.

3) Protected Inheritance…

If Access Specifier is protected—that is, the inheritance is protected—then:

a) The public members of A are protected members of B. They can be accessed by the member functions (and friend functions) of B.

b) The protected members of A are protected members of B. They can be accessed by the member functions (and friend functions) of B.

c) The private members of A are hidden in B. They cannot be directly accessed in B. They can be accessed by the member functions (and friend functions) of B through the public or protected members of A.

More about Public, Protected and Private Inheritance in C++ Programming

https://www.programiz.com/cpp-programming/public-protected-private-inheritance

References

- https://beginnersbook.com/2017/08/cpp-data-types/
- http://www.cplusplus.com/doc/tutorial/basic_io/
- https://www.w3schools.com/cpp/default.asp
- https://www.javatpoint.com/cpp-tutorial
- https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp
- https://www.programiz.com/
- https://ecomputernotes.com/cpp/

THANK YOU