

References

Chapter 15 – Inheritance, Polymorphism, and Virtual Functions

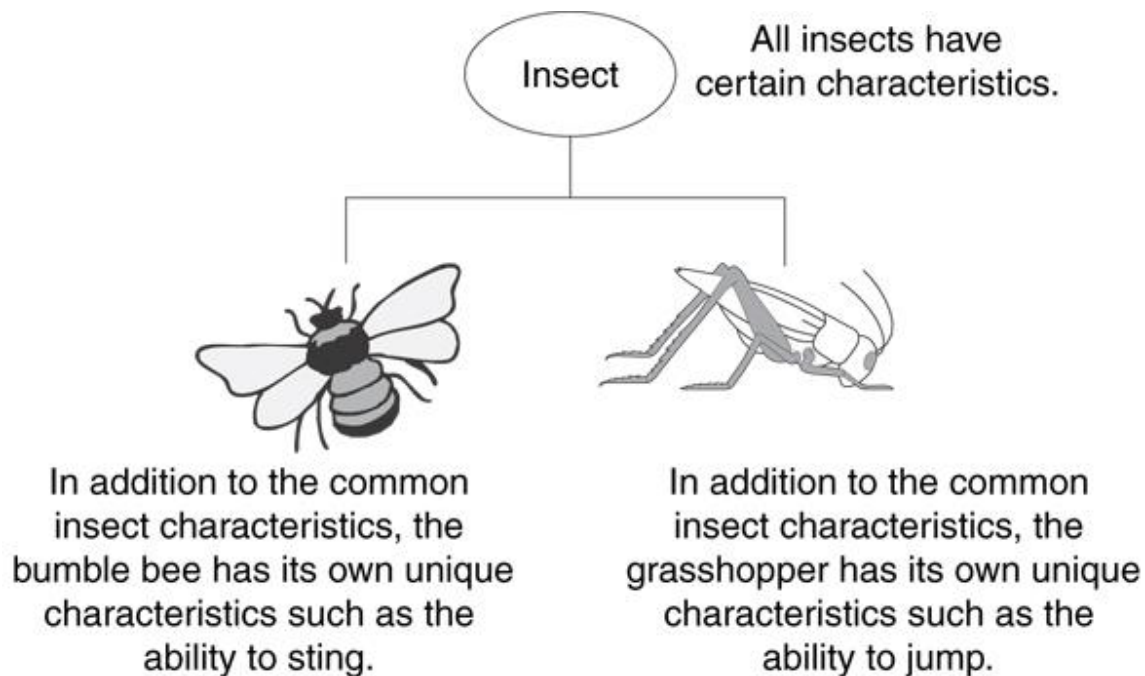
Starting Out With C++. From Control Structures through Objects (Eighth Edition)

Interview Question:

- What are the three access specifiers in C++?
- What are the three types of inheritances in C++?

What is Inheritance?

The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important features of Object Oriented Programming.

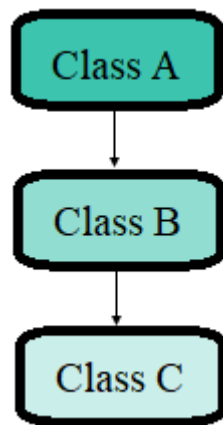


As shown in the example above, both bumble bee and grasshopper inherit from Insect. The class Insect is called Base Call, while Bumble Bee and Grasshopper are derived class.

Inheritance establishes an “is a” relationship between classes. A grasshopper is an insect. But an insect is not a grasshopper.

Sample 1

The following program consists of three classes, i.e. A, B and C. Class **B publicly inherits** from class A, and class C **publicly inherits** from class B.



```
#include <iostream>
using namespace std;
class A
{
protected:
    int A_Var = 0;
public:
    A(int val = 10)
    {
        A_Var = val;
        cout << "A constructor: A_Var = " << A_Var << endl;
    }
    ~A()
    {
        cout << "A destructor: A_Var = " << A_Var << endl;
    }
}
```

```

    }
    void getVal()
    {
        cout << "A_Var = " << A_Var << endl;
        return;
    }
};

class B: public A
{
protected:
    int B_Var = 0;
public:
    B(int val = 100)
    {
        B_Var = val;
        cout << "B constructor: B_Var = " << B_Var << endl;
    }
    ~B()
    {
        cout << "B destructor: B_Var = " << B_Var << endl;
    }
    void getVal()
    {
        cout << "A_Var = " << A_Var << endl;
        cout << "B_Var = " << B_Var << endl;
        return;
    }
};

class C: public B
{
protected:
    int C_Var = 0;
public:
    C(int val = 1000)

```

```

{
    C_Var = val;
    cout << "C constructor: C_Var = " << C_Var << endl;
}
~C()
{
    cout << "C destructor: C_Var = " << C_Var << endl;
}
void getVal()
{
    cout << "A_Var = " << A_Var << endl;
    cout << "B_Var = " << B_Var << endl;
    cout << "C_Var = " << C_Var << endl;
    return;
}

};
int main()
{

    cout << endl << "\\1\\" << endl;
    C c1;

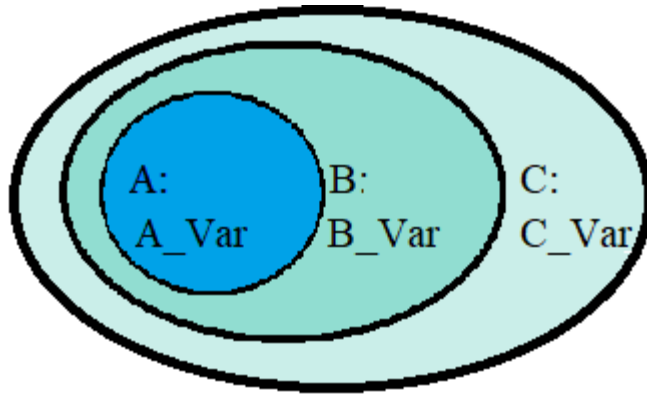
    cout << endl << "\\2\\" << endl;
    c1.getVal();

    cout << endl << "\\3\\" << endl;

    return 0;
}

```

Each one of the three classes owns a **protected** data member. That is, class A owns A_Val, B owns B_Val, and C owns C_Val. But when a C object is constructed, it actually owns all these three data members. The figure below shows how they are related.



This example demonstrates public inheritances among three classes and each of the classes owns a protected data member. In the following sections we will explain the three types access specifiers and the three types of inheritances.

The tree types of access specifiers

There are 3 access specifiers for a class/struct/Union in C++. These access specifiers define how the members of the class can be accessed. Of course, any member of a class is accessible within that class (Inside any member function of that same class). Moving ahead to type of access specifiers, they are:

Public - The members declared as Public are accessible from outside the Class through an object of the class.
Protected - The members declared as Protected are NOT accessible from outside the class BUT only in a class derived from it.
Private - These members are only accessible from within the class. No outside Access is allowed. Not even the derived class of the base class.

Let us summarize the different access types according to which functions can access them in the following way:

Access	Public	Protected	Private
Members of the same class	yes	yes	yes
Members of derived class	yes	yes	no
Non- members	yes	no	no

Note: Non-members represent any classes or functions from outside the class, such as main, another class or any global functions.

The tree types of Inhetances

There are 3 types of Inheritances:

Inheritance Type	The X of base class become Y of the derived class	
Public Inheritance	Public -> public Protected -> protected	All Public members of the Base Class become Public Members of the derived class & All Protected members of the Base Class become Protected Members of the Derived Class.
Protected Inheritance	Public -> protected Protected -> protected	All Public members of the Base Class become Protected Members of the derived class & All Protected members of the Base Class become Protected Members of the Derived Class.
Private Inheritance	Public -> private Protected -> private	All Public members of the Base Class become Private Members of the Derived class & All Protected members of the Base Class become Private Members of the Derived Class.

Exercise 1A:

class Grade
private members: char letter; float score; void calcGrade(); public members: void setScore(float); float getScore(); char getLetter();


When Test class inherits
from Grade class using
public class access, it
looks like this:



class Test : public Grade
private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int); ???
private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int); ???

Answer

class Grade
<pre>private members: char letter; float score; void calcGrade(); public members: void setScore(float); float getScore(); char getLetter();</pre>

When Test class inherits
from Grade class using
public class access, it
looks like this: 

class Test : public Grade
<pre>private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int);</pre>

<pre>private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int); void setScore(float); float getScore(); float getLetter();</pre>

Exercise 1B:

If the Inheritance is **protected**, what would be the result?

class Grade
private members: char letter; float score; void calcGrade(); public members: void setScore(float); float getScore(); char getLetter();

class Test : protected Grade
private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int);


When Test class inherits
from Grade class using
protected class access, it
looks like this: →

private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int); ???

Answer

class Grade
<pre>private members: char letter; float score; void calcGrade(); public members: void setScore(float); float getScore(); char getLetter();</pre>

class Test : protected Grade
<pre>private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int);</pre>

When Test class inherits from Grade class using protected class access, it looks like this: 

<pre>private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int); protected members: void setScore(float); float getScore(); float getLetter();</pre>

Exercise 1C:

If the Inheritance is **private**, what would be the result?

class Grade
private members: char letter; float score; void calcGrade(); public members: void setScore(float); float getScore(); char getLetter();


class Test : private Grade
private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int);

When Test class inherits
from Grade class using
private class access, it
looks like this:

private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int); ???

Answer

class Grade
<pre>private members: char letter; float score; void calcGrade(); public members: void setScore(float); float getScore(); char getLetter();</pre>

When Test class inherits
from Grade class using
private class access, it
looks like this: 

class Test : private Grade
<pre>private members: int numQuestions; float pointsEach; int numMissed; public members: Test(int, int);</pre>

<pre>private members: int numQuestions; float pointsEach; int numMissed; void setScore(float); float getScore(); float getLetter(); public members: Test(int, int);</pre>



Story:

John is dating a girl and he wants to use his father's Porsche. So he talks to his father.

"Dad, may I borrow your Porsche for tonight?"

"What is that for?"

"Dad, you know that I am dating Lisa, and I need to pick her up tonight."

"Son. That is not the right way to date a girl."

"Dad, I just learned C++. I know that the Porsche is your **protected property**. Based on the C++ Inheritance rule, I inherit from you and thus can use your protected property."

“Hmph... Son I happen to know what C++ too. Well, now I am going set my Porsche as private, not proteted.”

Discussions:

In a public Inheritance the private data members of the base class become the private data members of the derived class. Is this statement true or false?

Exercise 2:

Use the sample code in Sample 1 (or Sample_Inhertiance1.txt) to do the following exercises.

Step 1:

Predict the output and run the program to check if your prediction is correct. Explain why the three constructors and the three destructors display the messages in such a sequence.

Step 2:

The object is a C object. Explain why both B_Var and A_Var have valid default values, i.e. 100 and 10, respectively.

Step 3:

Comment out the constructor of C. Explain why even in the case a C object can still be constructed.

Step 4:

Comment out `getVal()` in C. Explain why even in the case that C does not have a `getVal()`, the `c1.getVal()` still works.

Step 5:

Change the three access specifiers of the three variables, i.e. `A_Val`, `B_Val` and `C_Val`, to private. Can the program compile? Why?

Discussions:

Exercise 3:

Step 1:

Predict the output and run the program to check if your prediction is correct.

Step 2:

In Part 4 the called `getVar()` function is of Class A while the `getVar()` in Part 3 is of Class B. Can you explain the reason why?

Step 3:

Comment out the `getVar()` function in Class A and compile again. Can it compile?

Step 4: Change

```
int getVar()
```

in Class A to

```
virtual int getVar()
```

Compile and run the program. What will you get?

```
#include <iostream>
using namespace std;
class A
{
protected:
    int A_Var;
public:
    A(int val = 10)
    {
        A_Var = val;
        cout << "A constructor: A_Var = " << A_Var << endl;
    }
    int getVar()
    {
        cout << "A::getVar = ";
        return A_Var;
    }
};

class B: public A // Change public to protected or private and see what can
happen
{
private:
    int B_Var;
public:
    B(int val = 100)
    {
        B_Var = val;
```



```

        cout << "B constructor: B_Var = " << B_Var << endl;
    }
    int getVar()
    {
        cout << "B::getVar = ";
        return B_Var;
    }
};

int main()
{

    cout << endl << "\\1\\" << endl;
    B bObj;

    cout << endl << "\\2\\" << endl;
    cout << bObj.getVar() << endl;

    cout << endl << "\\3\\" << endl;
    B *bPtr = nullptr;
    bPtr = &bObj;
    cout << bPtr->getVar() << endl;

    cout << endl << "\\4\\" << endl;
    A *aPtr = nullptr;
    aPtr = &bObj;
    cout << aPtr->getVar() << endl;

    return 0;
}

```

Discussions:

- If Class B inherits from Class A, then we can assign the address of a B object to an A pointer. (But assign an A object to a B pointer will have to do a casting first.) An analogy of this is that the son can live in the father's house

forever. However, if the father wants to move in the son's house, it won't be so easy.

- Which function to call is determined by the pointer type, not the object. That is why in Part 4 above, the `getVar()` is a member of Class A, not B.
- In Step 4 above, after making the `getVar()` a virtual function, the program compile and run. But the `getVar()` function is now the member of Class B. How this actually works will be explained in the article **Polymorphism**.