```
1  // Copying an object from a base class pointer.
2  // George F. Riley, Georgia Tech, Fall 2009
3  // ECE3090
4
5  // Sometimes, we need to make a copy of an object
6  // when we have a pointer or a reference to a base class
7  // of the object.  It's not obvious how to do this correctly
8  // as the simple solutions are not sufficient.
9  //
10 // This example demonstrates the use of an object
11 // cloning method that solves this problem nicely.
12
13 #include <iostream>
14
15 using namespace std;
16
17 // Define a virtual base class for clonable objects.
18 class ClonableObject
19 {
20 public:
21   // Define a pure virtual function called "Clone" method that returns
22   // a copy of itself
23   virtual ClonableObject* Clone() const = 0;
24 };
25
26 // Define a base class derived from Clone
27 class Base : public ClonableObject
28 {
29 public:
30  // All subclasses of Base must define PrintMe (for debugging)
31   virtual void PrintMe() const = 0;
32 };
33
34
35 // Make two distinct subclasses of Base, called A and B
36 class A : public Base
37 {
38 public:
39   A(int);
40   A(const A&);                      // Copy constructor
41   void            PrintMe() const;  // Print the object for debugging
42   ClonableObject* Clone() const;    // Make a clone of the object
43 public:
44   int a;
45 };
46
47 class B : public Base
48 {
49 public:
50   B(int);
51   B(const B&);                      // Copy constructor
52   void            PrintMe() const;  // Print the object for debugging
53   ClonableObject* Clone() const;    // Make a clone of the object
54 public:
55   int b;
56 };
```

Program objectcloning.cc

```
57
58  // Implementation of A
59  A::A(int a1)
60      : a(a1)
61  {
62  }
63
64  A::A(const A& old)
65      : a(old.a)
66  { // Copy constructor
67  }
68
69  void A::PrintMe() const
70  {
71    cout << "Hello from A, a is " << a << endl;
72  }
73
74  ClonableObject* A::Clone() const
75  {  // Return a copy of this A
76    return new A(*this);
77  }
78
79  // Implementation of B
80  B::B(int b1)
81      : b(b1)
82  {
83  }
84
85  B::B(const B& old)
86      : b(old.b)
87  { // Copy constructor
88  }
89
90  void B::PrintMe() const
91  {
92    cout << "Hello from B, b is " << b << endl;
93  }
94
95  ClonableObject* B::Clone() const
96  {  // Return a copy of this B
97    return new B(*this);
98  }
99
100 // Subroutine "Sub" has a reference to the Base class as a parameter
101 void Sub(const Base& base)
102 {
103   // For some reason, Sub  needs a copy of the object passed as a parameter
104   // However, class "Base" is a virtual base class (due to the PrintMe method)
105   // and no  objects of class Base can be created.  Further, the copy
106   // must be either and A or a B, depending on the type of the object
107   // passed by the caller.  The "Clone" function solves the problem nicely.
108   Base* c = (Base*)base.Clone();
109   c->PrintMe(); // Print the  new object for debugging
110 }
111
112
```

Program objectcloning.cc (continued)

```
113  int main()
114  {
115    // Create an A and B object
116    A a(1);
117    B b(2);
118    // Call the Sub method with a and b.  Both are valid  arguments to
119    // Sub since they are subclasses of Base.
120    Sub(a);
121    Sub(b);
122  }
123
124  // Output from this program is:
125  // Hello from A, a is 1
126  // Hello from B, b is 2
```

Program objectcloning.cc (continued)