Inheritance

Inheritance is a relationship between two classes such that one class takes on (inherits) the properties and behaviors (types, data members and member functions) of another class. The *derived* class inherits from a *base* class. This process facilitates code reuse and is a formal method of expressing natural relationships between types.

A *derived* class may be the *base* for another class. Several classes may inherit from one class. A *derived* class may inherit from several classes. Just like people! This is called *multiple inheritance.*

The following example illustrates some of the basic inheritance concepts.

Example 7-1 - First inheritance example

```
1  // File: ex7-1.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Base
7  {
8  protected:
9      int b;
10  public:
11      Base(int n);
12      void print() const
13      {
14          cout << "Base data is " << b << endl;
15      }
16  };
17
18  Base::Base(int n) : b(n)
19  {
20      cout << "created Base object: " << this << endl;
21  }
22
23  class Derived : public Base
24  {
25  private:
26      int d;
27  public:
28      Derived(int x,int y);
29      void print() const;
30      void printBase() const
31      {
32          cout << this << "'s Base is " << b << endl;
```

```
33      }
34  };
35
36  Derived::Derived(int x, int y) : Base(x), d(y)
37  {
38      cout << "created Derived object: " << this << endl;
39  }
40
41  void Derived::print(void) const
42  {
43      cout << "Derived data is " << d << endl;
44      Base::print();
45  }
46
47  int main()
48  {
49      Base b1(5);
50
51      // print base object
52      b1.print();
53      cout << endl;
54      Derived d1(3,4);
55
56      // print derived object
57      d1.print();
58      cout << endl;
59
60      d1.printBase();
61
62      // call base class print() from derived class object
63      d1.Base::print();
64      cout << endl;
65
66      cout << "how big is an int? " << sizeof(int) << endl;
67      cout << "how big is a Base? " << sizeof b1 << endl;
68      cout << "how big is a Derived? " << sizeof d1 << endl;
69  }
```

******** Output ********

```
created Base object: 0x69fefc
Base data is 5

created Base object: 0x69fef4
created Derived object: 0x69fef4
Derived data is 4
Base data is 3

0x69fef4's Base is 3
Base data is 3
```

```
how big is an int? 4
how big is a Base? 4
how big is a Derived? 8
```

Inheritance Notes

- The base class data members are usually protected. Thus, they may be accessible in the derived class.
- Public inheritance is the most common type of inheritance. In public inheritance, the protected base members are accessible and are also protected in the derived class. Also, the public base members remain public in the derived class. In any type of inheritance, private base members are not accessible in any "place" except in base class member functions. Access in derived classes to the base members by inheritance type is summarized in the following table:

Access to base class members in a derived class Base

Base Class Members	**Public Inheritance**	**Private Inheritance**	**Protected Inheritance**
Private	not accessible	not accessible	not accessible
Protected	protected	private	protected
Public	public	private	protected

- The derived class constructor automatically makes a call to the base class constructor. You can cause a certain base class constructor to be called by using constructor initialization list syntax. If you don't, then the default base class constructor is called (and it had better be there).

- The base class constructor executes before the derived class constructor, and the derived destructor will execute before the base destructor.

- The derived class will use the accessible member functions of the base class unless it has a function of the same *signature*.

- The following members are not inherited by the derived class:
  constructors
  destructors
  friend functions

- Static data members may be inherited and hence, are shared among the base and derived class objects, providing they have public or protected access. Further, static member functions may also be inherited.

- Derived classes are also called subclasses, base classes are also called superclasses.

Inheritance Examples

The following example illustrates a typical inheritance situation. Suppose you have a number class in which addition with a plus sign is defined. This class works well, but you would also like to be able to use it for subtraction. To do so, define your "own" class and inherit the number class. Add a subtraction function to your class.

Example 7-2 - Adding functionality to a class using inheritance

```
1  // File: ex7-2.cpp - Adding functionality to a class using
   inheritance
2
3  #include <iostream>
4  using namespace std;
5
6  class Number
7  {
8  protected:
9      int x;
10  public:
11      Number() {}
12      Number(int n) : x(n) { }
13      Number(const Number& n) : x(n.x) { }
14      int get_x() const
15      {
16          return x;
17      }
18      Number& operator=(const Number& z)
19      {
20          x = z.x;
21          return *this;
22      }
23      Number  operator+(const Number& y) const
24      {
25          return x + y.x;
26      }
27  };
28
29  ostream& operator<<(ostream& out, const Number& obj)
30  {
31      out << obj.get_x();
32      return out;
33  }
34
35  class MyNumber : public Number
36  {
37  public:
38      MyNumber() {}
39      MyNumber(int n) : Number(n) { }
40      MyNumber(const Number& n) : Number(n) {}
```

```
41      MyNumber(const MyNumber& m) : Number(m) {}
42      MyNumber  operator-(const MyNumber& y) const
43      {
44          return x - y.x;
45      }
46  };
47
48  int main(void)
49  {
50      Number n1(4), n2(5);
51      Number n3;
52      cout << "n1=" << n1 << endl;
53      cout << "n2=" << n2 << endl;
54      n3 = n1 + n2;
55      cout << "n3=" << n3 << endl;
56      cout << endl;
57
58      MyNumber mn1(7), mn2(4);
59      MyNumber mn3;
60      cout << "mn1=" << mn1 << endl;
61      cout << "mn2=" << mn2 << endl;
62
63      mn3 = mn1 + mn2;
64      cout << "mn3=" << mn3 << endl;
65
66      mn3 = mn1 - mn2;
67      cout << "mn3=" << mn3 << endl;
68
69      MyNumber mn4(n1);
70      cout << "mn4=" << mn4 << endl;
71
72      MyNumber mn5(mn1);
73      cout << "mn5=" << mn5 << endl;
74  }
```

```
******  Output  ******

n1=4
n2=5
n3=9

mn1=7
mn2=4
mn3=11
mn3=3
mn4=4
mn5=7
```

- What is the purpose of the default constructors in both classes?

- How can the MyNumber copy constructor pass a MyNumber& to the Number copy constructor?

- What is returned from the operator+ and operator- functions?

Example 7-3 - Inherit the deck class

```
1  // File: ex7-3.cpp - Inherit the deck class
2
3  #include <iostream>
4  #include <cstdlib>
5  using namespace std;
6
7  class Card
8  {
9  private:
10      int value;
11      int suit;
12  public:
13      Card(int n = 0);
14      Card(int val, int s);
15      int get_value() const
16      {
17          return value;
18      }
19      int get_suit() const
20      {
21          return suit;
22      }
23  };
24
25  Card::Card(int n) : value(n % 13), suit(n / 13)
26  { }
27
28  Card::Card(int val, int s) :value (val), suit(s)
29  { }
30
31  ostream& operator<<(ostream& out, const Card& crd)
32  {
33      const string valueStr[13] =
34      {
35          "two","three","four","five","six","seven",
36          "eight","nine","ten","jack","queen","king","ace"
37      };
38      const string suitStr[4] =
   {"clubs","diamonds","hearts","spades"};
39      out << valueStr[crd.get_value()] << " of " <<
   suitStr[crd.get_suit()];
40      return out;
41  }
42
43  class Deck
44  {
45  protected:
46      const int DeckSize;
47      Card* ptrCard;
48
```

```
49  public:
50      Deck(int = 0);
51      ~Deck();
52      Card* get_ptrCard() const
53      {
54          return ptrCard;
55      }
56      int getDeckSize() const
57      {
58          return DeckSize;
59      }
60      void shuffle();
61  };
62
63  Deck::Deck(int n) :     DeckSize(n), ptrCard(new Card[n])
64  { }
65
66  Deck::~Deck()
67  {
68      delete [] ptrCard;
69      ptrCard = nullptr;
70  }
71
72  ostream& operator<<(ostream& out, const Deck& deck)
73  {
74      for (int i = 0; i < deck.getDeckSize(); i++)
75          out << deck.get_ptrCard()[i] << endl;
76      return out;
77  }
78
79  void Deck::shuffle()
80  {
81      cout << "I am shuffling the Deck\n";
82      Card temp;
83      for (int i = 0; i < DeckSize; i++)
84      {
85          int k = rand() % DeckSize;
86          temp = ptrCard[i];
87          ptrCard[i] = ptrCard[k];
88          ptrCard[k] = temp;
89      }
90  }
91
92
93  class PokerDeck : public Deck
94  {
95  public:
96      PokerDeck();
97  };
98
99  PokerDeck::PokerDeck() : Deck(52)
100  {
```

```
101      for (int i = 0; i < DeckSize; i++) ptrCard[i] = Card(i);
102  }
103
104  class PinocleDeck : public Deck
105  {
106  public:
107      PinocleDeck();
108  };
109
110  PinocleDeck::PinocleDeck() : Deck(48)
111  {
112      for (int i = 0; i < DeckSize; i++) ptrCard[i] =
   Card(i%6+7,i/2%4);
113  }
114
115  int main()
116  {
117      PokerDeck pokerD;
118      cout << "This is a poker deck\n" << pokerD << endl;
119      PinocleDeck pinocleD;
120      cout << "This is a pinocle deck\n"<< pinocleD << endl;
121      pokerD.shuffle();
122      pinocleD.shuffle();
123  }
```

```
******  Output  ******

two of clubs                            <= the poker deck starts here
three of clubs
four of clubs
five of clubs
six of clubs
seven of clubs
eight of clubs
 ..
 ..
 ..
queen of spades
king of spades
ace of spades

nine of clubs                           <= the pinocle deck starts here
ten of clubs
jack of diamonds
queen of diamonds
king of hearts
ace of hearts
nine of spades
ten of spades
jack of clubs
queen of clubs
king of diamonds
ace of diamonds
nine of hearts
ten of hearts
 ..
 ..
 ..
ace of diamonds
nine of hearts
ten of hearts
jack of spades
queen of spades
king of clubs
ace of clubs
nine of diamonds
ten of diamonds
jack of hearts
queen of hearts
king of spades
ace of spades
…
```

Example 7-4 - Account classes

```
1  // File: ex7-4.cpp - Derive Savings and Checking from Account
2
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  class Account
8  {
9  protected:
10      unsigned long long accountNum;
11      double balance;
12      double intRate;         // annual interest rate
13  public:
14      Account(unsigned long long num = 0,double bal = 0,double = 0);
15      void deposit(double amount);
16      void withdraw(double amount);
17      void month_end();
18      friend ostream& operator<<(ostream& out, const Account&
    account);
19  };
20
21  ostream& operator<<(ostream& out, const Account& account)
22  {
23      out << fixed << setprecision(2);
24      out << "Account: " << account.accountNum << "  balance = $" <<
    account.balance << endl;
25      return out;
26  }
27
28  Account::Account(unsigned long long acc_no,double init_bal,double
    i_rate)
29      : accountNum(acc_no), balance(init_bal), intRate(i_rate)
30  {
31      cout << "* New Account\t";
32      cout << *this << endl;
33  }
34
35  void Account::deposit(double amount)
36  {
37      cout << "Account: " << accountNum << "  deposit = $" << amount
    << endl;
38      balance += amount;
39  }
40
41  void Account::withdraw(double amount)
42  {
43      cout << "Account: " << accountNum << "  withdraw = $" << amount
    << endl;
44      balance -= amount;
```

```
45  }
46
47  void Account::month_end()
48  {
49      cout << "Account month-end processing: " << accountNum << endl;
50      balance *= (1.+intRate/12.);
51      cout << *this << endl;
52  }
53
54  class SavingsAccount : public Account
55  {
56  public:
57      SavingsAccount(long acc_no, double init_bal = 50., double
   i_rate = .02)
58          : Account(acc_no,init_bal,i_rate) { }
59  };
60
61
62  class CheckingAccount : public Account
63  {
64  private:
65      double min_balance;
66      double service_charge;
67  public:
68      CheckingAccount(unsigned long long, double, double =
   300.,double = 3.,double = .01);
69      void process_check(double amt)
70      {
71          withdraw(amt);
72      }
73      void month_end(void);
74  };
75
76  CheckingAccount::CheckingAccount(unsigned long long acc_no, double
   init_bal,
77                                   double min_bal, double
   service_chg, double i_rate)
78      : Account(acc_no,init_bal,i_rate),
79        min_balance(min_bal),
80        service_charge(service_chg)
81  { }
82
83  void CheckingAccount::month_end()
84  {
85      cout << "Checking Account month-end processing: " << accountNum
   << endl;
86      balance *= (1.+intRate/12.);
87      if (balance < min_balance) balance -= service_charge;
88      cout << *this << endl;
89  }
90
91  int main()
```

```
92  {
93       SavingsAccount Mysavings(1234560ULL,500.);
94       CheckingAccount Mychecking(1234561ULL,1000.);
95       Mysavings.deposit(100.);
96       cout << Mysavings << endl;
97       Mysavings.withdraw(200.);
98       cout << Mysavings << endl;
99       Mychecking.deposit(100.);
100       cout << Mysavings << endl;
101       Mychecking.process_check(200.);
102       cout << Mychecking << endl;
103       Mysavings.month_end();
104       Mychecking.month_end();
105  }
```

```
******  Output  ******

* New account  account: 1234560  balance = 500

* New account  account: 1234561  balance = 1000

account: 1234560  deposit = 100
account: 1234560  balance = 600

account: 1234560  withdraw = 200
account: 1234560  balance = 400

account: 1234561  deposit = 100
account: 1234561  balance = 1100

account: 1234561  withdraw = 200
account: 1234561  balance = 900

account month-end processing: 1234560
account: 1234560  balance = 401.666656

checking account month-end processing: 1234561
account: 1234561  balance = 903
```

Example 7-5 - Triangle classes
This example demonstates two levels of inheritance.

```
1  // File: ex7-5.cpp - Triangle classes
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  class Triangle
8  {
9  protected:
10      double a,b,c;
11  public:
12      Triangle(double s1,double s2,double s3) : a(s1), b(s2), c(s3)
   {}
13      double area()const;
14      double perimeter() const
15      {
16          return a + b + c;
17      }
18      friend ostream& operator<<(ostream&, const Triangle&);
19  };
20
21  double Triangle::area() const
22  {
23      double s = perimeter()/2.0;    //    s = semiperimater
24      return sqrt(s*(s-a)*(s-b)*(s-c));
25  }
26
27  ostream& operator<<(ostream& out, const Triangle& triangle)
28  {
29      out << &triangle << ": sides "
30          << triangle.a << ' ' << triangle.b << ' ' << triangle.c;
31      return out;
32  }
33
34  class Isosceles : public Triangle
35  {
36  public:
37      Isosceles(double base, double leg) : Triangle(base,leg,leg) {}
38  };
39
40  class Equilateral : public Isosceles
41  {
42  public:
43      Equilateral(double side) : Isosceles(side,side) {}
44  };
45
46
47  int main()
48  {
```

```
49      Triangle t1(3,4,5);
50      cout << t1 << endl;
51      cout << "perimeter=" << t1.perimeter() << "  area=" <<
   t1.area() << endl;
52
53      Isosceles t2(2,4);
54      cout << t2 << endl;
55      cout << "perimeter=" << t2.perimeter() << "  area=" <<
   t2.area() << endl;
56
57      Equilateral t3(5);
58      cout << t3 << endl;
59      cout << "perimeter=" << t3.perimeter() << "  area=" <<
   t3.area() << endl;
60
61  }
```

```
******  Output  ******

0x6afee8: sides 3 4 5
perimeter=12  area=6
0x6afed0: sides 2 4 4
perimeter=10  area=3.87298
0x6afeb8: sides 5 5 5
perimeter=15  area=10.8253
```

Private Inheritance

Private inheritance may be used to represent a "has-a" relationship between two classes. (Fortunately) this type of inheritance is not all that common. Private inheritance is more commonly replaced by containment, or a container relationship. Instead of a "has-a" relationship between classes, private inheritance is more commonly used to express an "in terms of" relationship. Here are some notes regarding private inheritance:

- Private inheritance is the default inheritance type, even though public inheritance is by far the more common type of inheritance. This is what you get if you leave off the "public" after the colon in the class definition.

- Private inheritance is used to indicate that one class "contains" another class, but the containment is limited to exactly one instance of the base class.

- The (privately) derived class inherits the base class public and protected members, but does not "pass them on". That is, the derived class must provide it's own public interface to any base class members desired.

- Private inheritance is used when you want to make use of the base class, but you wish to hide the base class public interface or you wish to provide your own public interface.

Example 7-6 – Private inheritance

The following example demonstrates private inheritance. The objective is to create a name class that is defined in terms of the "standard" string class. To keep the class simple, the name class has a simple user interface, thus hiding the complexity of the string class.

```
1  // Example 7-6 - private inheritance

2
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  class name : private string
8  {
```

```
9  public:
10   name(const char *);
11   void print() const;
12   string first_last() const;
13   string initials() const;
14   void change_last(const string& new_last);
15 };
16
17 name::name(const char* n) : string(n) {}
18
19 void name::print() const {
20   cout << c_str() << ".\n";
21 }
22
23 string name::first_last() const {
24   size_t comma_pos = find(',');
25   size_t second_space = find_last_of(' ');
26   return substr(comma_pos+2,second_space-comma_pos-2) +
27          ' ' + substr(0,comma_pos);
28 }
29
30 string name::initials() const {
31   string inits;
32   inits = data()[find(',')+2];
33   return inits + data()[length()-1] + *data();
34 }
35
36 void name::change_last(const string& new_last) {
37   replace(0,find(','),new_last);
38 }
39
40 int main() {
41   name joe("Bentley, Joseph E");
42   joe.print();
43   cout << joe.first_last() << endl;
44   cout << joe.initials() << endl;
45   joe.change_last("Smith");
46   joe.print();
47   return 0;
48 }
```

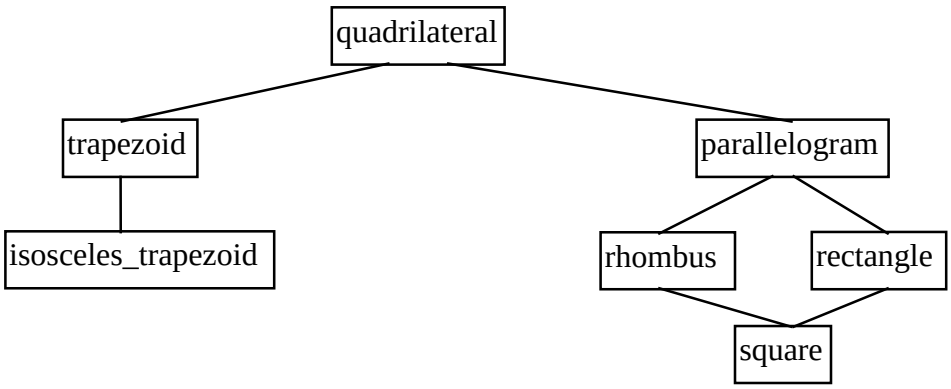
```
******  Output  ******

Bentley, Joseph E.
Joseph Bentley
JEB
Smith, Joseph E.
```

Multiple Inheritance

Example 7-7 - First Multiple Inheritance Example

```
File: ex7-7.cpp - multiple inheritance

#include <iostream>
using namespace std;

class one {
  protected:
    int a,b;
  public:
     one(int z,int y) { a = z; b = y; }
     void show(void) const { cout << a << ' ' << b << endl; }
};

class two {
  protected:
    int c,d;
  public:
    two(int z,int y) { c = z; d = y; }
    void show(void) const { cout << c << ' ' << d << endl; }
};

class three : public one, public two
{
  private:
    int e;
  public:
    three(int,int,int,int,int);
    void show(void) const
    { cout <<a<< ' ' <<b<< ' ' <<c<< ' ' <<d<< ' ' <<e<< endl;}
};

three::three(int a1, int a2, int a3, int a4, int a5)
: one(a1,a2),two(a3,a4)
{
  e = a5;
}

int main(void)
{
  one abc(5,7);
  abc.show();                                  // prints 5 7
  two def(8,9);
  def.show();                                  // prints 8 9
  three ghi(2,4,6,8,10);
  ghi.show();                                  // prints 2 4 6 8 10
  return 0;
}
```

The next example illustrates a more complicated inheritance situation. It models the relationship between types of quadrilaterals. This relationship is shown in the following figure:



Note that the parallelogram class will be derived from the quadrilateral class, both the rhombus and rectangle classes will be derived from the parallelogram class. And the square is derived from both the rhombus and the rectangle classes. It's the square class that makes this multiple inheritance.

Example 7-8 - Quadrilaterals

```
1  // File: ex7-8.cpp
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  class quadrilateral
8  {
9    protected:
10      double a,b,c,d;
11    public:
12      quadrilateral(double s1,double s2,double s3,double s4)
13      : a(s1), b(s2), c(s3), d(s4) {}
14      quadrilateral() {}
15      void show() const
16      {
17         cout << "quadrilateral: " << this << "  sides "
18              << a << ' ' << b << ' ' << c << ' ' << d << endl;
19      }
20  };
```

```
21  class trapezoid : public quadrilateral
22  {
23    public:
24      trapezoid(double base1,double base2,double leg1,double leg2)
25      : quadrilateral(base1,leg1,base2,leg2) {}
26  };
27
28  class isosceles_trapezoid : public trapezoid
29  {
30    public:
31      isosceles_trapezoid(double base1,double base2,double leg)
32      : trapezoid(base1,leg,base2,leg) {}
33  };
34
35  class parallelogram : public quadrilateral
36  {
37    protected:
38      int angle;
39    public:
40      parallelogram(double s1,double s2, int ang)
41      : quadrilateral(s1,s2,s1,s2) { angle = ang; }
42      parallelogram() { }
43      void show_angles(void) const
44      {
45        cout << "angles = " << angle << ' ' << (180-angle) << endl;
46      }
47  };
48
49  class rectangle : virtual public parallelogram
50  {
51    public:
52      rectangle(double base, double height)
53      : parallelogram(base,height,90) {}
54      rectangle() {}
55  };
56
57  class rhombus: virtual public parallelogram
58  {
59    public:
60      rhombus(double side,int ang) : parallelogram(side,side,ang){}
61      rhombus() {}
62  };
63
64  class square : public rhombus,public rectangle
65  {
66    public:
67      square(double side) : parallelogram(side,side,90) {}
68  };
```

```
69  int main(void)
70  {
71    quadrilateral q1(1,2,3,4);
72    q1.show();
73
74    trapezoid q2(22,13,8,15);
75    q2.show();
76
77    isosceles_trapezoid q3(18,8,13);
78    q3.show();
79
80    parallelogram q4(4,3,45);
81    q4.show();
82    q4.show_angles();
83
84    rectangle q5(4,3);
85    q5.show();
86    q5.show_angles();
87
88    rhombus q6(5,45);
89    q6.show();
90    q6.show_angles();
91    cout << endl;
92
93    square q7(5);
94    q7.show();
95    q7.show_angles();
96
97    return 0;
98  }
```

```
******  Output  ******

quadrilateral: 0x3dc9ffd6  sides 1 2 3 4
quadrilateral: 0x3dc9ffb6  sides 22 8 13 15
quadrilateral: 0x3dc9ff96  sides 18 8 13 13
quadrilateral: 0x3dc9ff74  sides 4 3 4 3
angles = 45 135
quadrilateral: 0x3dc9ff52  sides 4 3 4 3
angles = 90 90
quadrilateral: 0x3dc9ff2e  sides 5 5 5 5
angles = 45 135

quadrilateral: 0x3dc9ff0a  sides 5 5 5 5
angles = 90 90
```

Note: The rectangle and rhombus classes both inherit the parallelogram class. Their inheritance is designated virtual, so that if a class is derived from both of the them, the parallelogram data will not be repeated in the class.