More Class Concepts

The this Pointer

this is a special pointer used inside member functions to point to the object itself. *this (this dereferenced) represents the "*current*" object. **this** is the address of the object. **this** is most commonly used to return by reference.

Example 5-1 - The **this** Pointer

```
1  // File: ex5-1.cpp - the this pointer
2  #include <iostream>
3  using namespace std;
4
5  class Thing
6  {
7    private:
8      int x;
9      double y;
10    public:
11     Thing(int arg1 = 0, double arg2 = 0.0);          // constructor
12     void copyThing(Thing&);
13     void printThing(void) { cout << x << ' ' << y << endl; }
14 };
15
16 Thing::Thing(int arg1, double arg2) : x(arg1), y(arg2)
17 { }
18
19 void Thing::copyThing(Thing& z)
20 {
21   if (this == &z)
22   {
23    cout << "Don't copy me to myself\n";
24    return;
25   }
26   x = z.x;
27   y = z.y;
28 }
29
30 int main(void)
31 {
32   Thing a(5,3.14);
33   Thing b(1);
34   Thing c;
35   a.printThing();
36   b.printThing();
37   c.printThing();
38   c.copyThing(a); // copy Thing-a to c
39   c.printThing();
40   b.copyThing(b); // copy Thing-b to b
41   b.printThing();
42 }
```

Chaining Functions

Functions may be "chained" by returning a reference to the class type.

Example 5-2 - Chaining Functions

```
1  // File: ex5-2.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Circle
7  {
8  private:
9      double radius;
10  public:
11      Circle (double r) : radius(r) {}          // constructor
12      Circle& area() ;
13      Circle& circumference() ;
14  };
15
16  Circle& Circle::area()
17  {
18      cout<<"The area of the Circle is "
19          << 3.14 * radius * radius << endl;
20      return (*this);
21  }
22  Circle& Circle::circumference()
23  {
24      cout<<"The circumference of the Circle is "
25          << 2. * 3.14 * radius << endl;
26      return (*this);
27  }
28
29  int main()
30  {
31      Circle c1(5);
32      c1.area().circumference();
33
34      Circle c2(4.45);
35      c2.circumference().area();
36  }
```

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

The area of the circle is 78.5
The circumference of the circle is 31.4
The circumference of the circle is 27.946
The area of the circle is 62.1799
```

- ✔ Why are the area() and circumference() functions not defined as const?

Static Data Members

A static data member is a data member that is shared by all instances of the class. There is only one occurrence of the static data member regardless of how many class objects exist. Static data members are still access controlled (private vs. public). Private static data members may not be accessed by non-member functions. **Non-const** static data members must be defined (initialized) outside of the class definition and outside of member function definitions.

Example 5-3 - Static Data Member

```
1  // File: ex5-3.cpp - static data member
2
3  #include <iostream>
4  using namespace std;
5
6  // function prototype
7  void funk();
8
9  class Circle
10  {
11  private:
12    double radius;
13    static unsigned numCircles;
14  public:
15    Circle (double r = 1.0) : radius(r) { numCircles++;}
16    Circle (const Circle& C) : radius(C.radius) { numCircles++;}
17    ~Circle () {numCircles--;}
18    void printCircleCount();
19  };
20
21  unsigned Circle::numCircles = 0;     // static member definition
22
23  void Circle::printCircleCount()
24  {
25    cout << "Number of Circles = " << numCircles << endl;
26  }
27
28
29  int main()
30  {
31    Circle c1(5.);
32    c1.printCircleCount();
33    Circle c2;
34    c2.printCircleCount();
35    c1.printCircleCount();
36    {
37          Circle c3(1.5);
38          c3.printCircleCount();
39    }
40    c1.printCircleCount();
41    Circle c4(c1);
42    c1.printCircleCount();
43
44    funk();
```

```
45    c1.printCircleCount();
46  }
47
48  void funk()
49  {
50    Circle tempLocal;
51    tempLocal.printCircleCount();
52  }
```

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

Number of Circles = 1
Number of Circles = 2
Number of Circles = 2
Number of Circles = 3
Number of Circles = 2
Number of Circles = 3
Number of Circles = 4
Number of Circles = 3
```

Static Member Functions

A static member function is a member function that cannot access the this pointer for an object. This means that the static member function cannot access the non-static data members of a class. It is used to access the static data members of a class. Static member functions are called using the class name and scope resolution operator, providing it has public access, as illustrated in the example below. Static member functions may not be const member functions.

Example 5-4 - Static Member Function

```
1  // File: ex5-4.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Circle
7  {
8  private:
9      double radius;
10      static unsigned numCircles;
11   public:
12      Circle(double r = 1.0);
13      ~Circle();
14      static void print_numCircles();
15      static void resetNumCircles();
16  };
17
18  unsigned Circle::numCircles = 0 ;
19
20  Circle::Circle(double r) : radius(r)
21  {
```

```
22      numCircles++;
23  }
24
25  Circle::~Circle()
26  {
27      numCircles--;
28  }
29
30  void Circle::print_numCircles()
31  {
32      cout << "number of Circles = " << numCircles << endl;
33  }
34
35  void Circle::resetNumCircles()
36  {
37      numCircles = 0;
38  }
39
40  int main()
41  {
42      Circle c1(5.);
43      Circle::print_numCircles();
44      Circle c2(4.);
45      Circle::print_numCircles();
46      Circle::resetNumCircles();
47      Circle::resetNumCircles();
48      Circle::print_numCircles();
49      Circle c3(1.);
50      Circle::print_numCircles();
51  }
```

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

number of Circles = 1
number of Circles = 2
number of Circles = 0
number of Circles = 1
```

Friend Functions

A friend function is a non-member function that has access to the private parts of a class. Friendship can be granted in three ways:

- to an independent (non-class member) function
- to a class member function of another class
- to another class (to all functions in that class)

A friend function is always a non-class member. A function outside of a class cannot "seek friendship". Friendship is only granted by a class to another function (or class). A friend has access to all private members.

It is common practice for friend functions to have arguments which include references to the (friendship-granting) class.

Example 5-5 - An independent friend

```
1  // File: ex5-5.cpp - a friend to the Circle class
2
3  #include <iostream>
4  using namespace std;
5
6  class Circle
7  {
8    private:
9      double radius;
10    public:
11     Circle (double r = 1.0) : radius(r) { }
12    friend void print(const Circle&);
13  };
14
15
16  int main(void)
17  {
18    Circle c1(5.);
19    print(c1);
20    Circle c2;
21    print(c2);
22  }
23
24  void print(const Circle& c)
25  {
26    cout << "This Circle has radius " << c.radius<< endl;
27  }
```

Friendly advice

- Friend functions are not affected by their location in a class definition or any access specifiers.

- Granting friendship to another function or class is not reciprocal. If class xyz declares that class abc is a friend, then class xyz is not necessarily a friend to class abc.

- Friendship is not transitive. If class xyz grants friendship to class abc, and class abc grants friendship to class def, then the friendship from xyz is not automatically granted to def.

- Friendship is not inherited. The friend of a base class is not a friend to a class derived from the base. Further, if a base class, B, is a friend to another class, C, classes derived from B are not friends of C. (My friends are not necessarily my children's friends, and my children's friends are not my friends.)

- Class member functions operate on the object that invokes the function. Friend functions operate on objects that are passed as arguments.

Granting friendship to another class

- If class dog grants friendship to class cat, then any function of the cat class can access any member of the dog class.

- The word class is optional in the grant of friendship to another class.

Granting friendship to a function of another class

- To grant friendship to a member of another class, you must indicate the class name and function name using the scope resolution operator.

- If you want the dog class to grant friendship to the meow function of the cat class, you must:

  1) forward declare the dog class.

  2) define the cat class, declaring the meow function, but not defining it.

  3) define the dog class, identifying the friend function, cat::meow().

  4) define the cat member functions.

Example 5-6 - A friend to the card and deck classes

```
1  // File: ex5-6.cpp - a friend to the card and deck classes
2
3  #include <iostream>
4  #include <cstdlib> // needed for rand() function
5  #include <string>
6  using namespace std;
7
8  const string value_name[13] =
9  "two","three","four","five","six","seven","eight","nine","ten","jack","queen
   ","king","ace"};
10
11  const string suit_name[4] =
12  {"clubs","diamonds","hearts","spades"};
13
14  const int HandSize = 5;
15  const int DeckSize = 52;
16
17  class Deck;  // forward declare the Deck class
18
19  class Hand
20  {
21  private:
22      int card[HandSize];
23  public:
24      Hand() { }
25      void dealMe(Deck&);
26      void print(const Deck&) const;
27  };
28
29  class Card
30  {
31  private:
32      int value;
33      int suit;
34  public:
35      Card(int arg = 0) : value(arg%13), suit(arg%4) { }
36      int get_value() const
37      {
38          return value;
39      }
40      int get_suit() const
41      {
42          return suit;
43      }
44      void print(void) const;
45      friend void Hand::print(const Deck&) const;
46  };
47
48  void Card::print() const
49  {
50      cout << value_name[value] << " of " << suit_name[suit] << endl;
51  }
52
53
54  class Deck
```

```
55  {
56      friend class Hand;
57  private:
58      Card d[DeckSize];
59      int next_Card;
60  public:
61      Deck();
62      void shuffle();
63      void deal(int=HandSize);
64      void print() const;
65  };
66
67  Deck::Deck()
68  {
69      for (int i = 0; i < DeckSize; i++) d[i] = Card(i);
70      next_Card = 0;
71  }
72
73  void Deck::shuffle()
74  {
75      int i, k;
76      Card temp;
77      cout << "I am shuffling the Deck\n";
78      for (i = 0; i < DeckSize; i++)
79      {
80          k = rand() % DeckSize;
81          temp = d[i];
82          d[i] = d[k];
83          d[k] = temp;
84      }
85  }
86
87  void Deck::print() const
88  {
89      for (int i = 0; i < DeckSize; i++) d[i].print();
90  }
91
92  void Hand::dealMe(Deck& deck)
93  {
94      for (int i = 0; i < HandSize; i++) card[i] = deck.next_Card++;
95  }
96
97  void Hand::print(const Deck& deck) const
98  {
99      cout << "here is your hand:\n";
100      for (int i = 0; i < HandSize; i++) deck.d[card[i]].print();
101  }
102
103  int main (void)
104  {
105      Deck poker;
106      poker.shuffle();
107      poker.print();
108      Hand Joe;
109      Hand Mary;
110      Joe.dealMe(poker);
```

```
111      Mary.dealMe(poker);
112      cout << "\nOk, Joe ";
113      Joe.print(poker);
114      cout << "\nOk, Mary ";
115      Mary.print(poker);
116  }
```

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

```
I am shuffling the deck
ten of hearts
ace of diamonds
queen of hearts
three of hearts
four of diamonds
eight of diamonds
eight of spades
eight of hearts
seven of spades
six of hearts
     .
     .
     .

Ok, Joe here is your hand:
ten of hearts
ace of diamonds
queen of hearts
three of hearts
four of diamonds

Ok, Mary here is your hand:
eight of diamonds
eight of spades
eight of hearts
seven of spades
six of hearts
```

- ✔ Does Hand::print() have to be declared as a friend of the Card class?
- ✔ How can you change the code to eliminate all friend functions?

Example 5-7 - More friendly poker

```
1  // File: ex5-7.cpp
2
3  #include <iostream>
4  #include <cstdlib>            // needed for rand() function
5  #include <string>
6  #include <cassert>
7  #include <ctime>
8  using namespace std;
9
10  const string value_name[13] = {"two","three","four","five","six",
11
   "seven","eight","nine","ten","jack","queen","king","ace"
12                                 };
13  const string suit_name[4] = {"clubs","diamonds","hearts","spades"};
14
15  const int HandSize = 5;
16  const int DeckSize = 52;
17
18  class Deck;
19
20  class Hand
21  {
22      friend void threeOrFourOfAKind(const Hand&);
23
24  public:
25      Hand(const string&, Deck&);
26      void print() const;
27      string getName() const
28      {
29          return name;
30      }
31      const Deck& getDeck() const
32      {
33          return deck;
34      }
35  private:
36      string name;
37      int Card_no[HandSize];
38      Deck& deck;
39      void dealMe(Deck&);
40  };
41
42
43  Hand::Hand(const string& n, Deck& d) : name(n), deck(d)
44  {
45      dealMe(deck);
46  }
47
48  class Card
49  {
50  private:
51      int value;
52      int suit;
53  public:
54      Card (int x = 0) : value(x%13), suit(x%4) { }
```

```
55      int get_value(void) const
56      {
57          return value;
58      }
59      int get_suit() const
60      {
61          return suit;
62      }
63      void print(void) const;
64  };
65
66  void Card::print() const
67  {
68      cout << (value_name[value]) << " of "  << (suit_name[suit]) << endl;
69  }
70
71
72  class Deck
73  {
74      friend void threeOrFourOfAKind(const Hand&);
75      friend class Hand;
76  public:
77      Deck();
78      void print(void) const;
79
80  private:
81      Card d[DeckSize];
82      int nextCard;
83      void shuffle(void);
84  };
85
86
87  Deck::Deck() : nextCard(0)
88  {
89      for (int i = 0; i < DeckSize; i++) d[i] = Card(i);
90      nextCard = 0;
91      shuffle();
92  }
93
94  void Deck::shuffle(void)
95  {
96      int k;
97      Card temp;
98      cout << "I am shuffling the Deck\n";
99      for (int i = 0; i < DeckSize; i++)
100      {
101          k = rand() % DeckSize;
102          temp = d[i];
103          d[i] = d[k];
104          d[k] = temp;
105      }
106  }
107
108  void Deck::print(void) const
109  {
110      for (int i = 0; i < DeckSize; i++) d[i].print();
```

```
111  }
112
113  void Hand::dealMe(Deck& deck)
114  {
115      assert(deck.nextCard < DeckSize-4);
116      for (int i = 0; i < HandSize; i++) Card_no[i] = deck.nextCard++;
117  }
118
119  void Hand::print() const
120  {
121      cout << "Ok " << name << ", here is your hand:" << endl;
122      for (int i = 0; i < HandSize; i++) deck.d[Card_no[i]].print();
123      threeOrFourOfAKind(*this);
124      cout << endl;
125
126  }
127
128  int main (void)
129  {
130      srand(time(0));
131      Deck poker;
132
133      Hand curly("Curly",poker);
134      Hand larry("Larry",poker);
135      Hand moe("Moe",poker);
136
137      curly.print();
138      larry.print();
139      moe.print();
140  }
141
142  void threeOrFourOfAKind(const Hand& who)
143  {
144      int temp;
145      int Card_count;
146      for (int i = 0; i < 3; i++)
147      {
148          Card_count = 1;
149          temp = (who.getDeck().d[who.Card_no[i]]).get_value();
150          for (int j = i + 1; j < HandSize; j++)
151              if (temp == who.getDeck().d[who.Card_no[j]].get_value())
   Card_count++;
152          if (Card_count > 2)
153          {
154              cout << "Hey, you have " << Card_count << ' ' <<
   value_name[temp] << "s.\n";
155          }
156      }
157  }
158
```

```
****** Sample Run ******

I am shuffling the Deck
Ok Curly, here is your hand:
jack of diamonds
six of hearts
eight of diamonds
nine of spades
six of clubs

Ok Larry, here is your hand:
ten of hearts
six of spades
seven of hearts
five of spades
eight of spades

Ok Moe, here is your hand:
three of hearts
ten of clubs
three of spades
queen of clubs
three of clubs
Hey, you have 3 threes.
```

Mutual Friendship

What if you want a function of one class to be a friend of a second class, and a function of the second class to be a friend of the first class? How do you do it?

Make the bark() function of the dog class a friend of the cat class and the meow() function a friend of the dog class.

Example 5-8 - Mutual friends

```
1  // File: ex5-8.cpp
2  // File: ex5-8.cpp
3
4  #include <iostream>
5  #include <string>
6  using namespace std;
7
8  class Cat;          // forward declaration
9  class Dog
10  {
11      string name;
12  public:
13      void bark(const Cat&) const;
14      Dog(const string& n) : name(n) { }
15      friend class Cat;
16  };
17  class Cat
18  {
19      string name;
20  public:
21      void meow(const Dog&) const;
22      Cat(const string& n) : name(n) { }
23      friend void Dog::bark(const Cat&) const;
24  };
25
26
27  void Dog::bark(const Cat& c) const
28  {
29      for (size_t i = 0; i < name.size(); i++) cout << " woof ";
30      cout << endl;
31  }
32  void Cat::meow(const Dog& d) const
33  {
34      for (size_t i = 0; i < name.size(); i++) cout << " meow ";
35      cout << endl;
36  }
```

```
37  int main()
38  {
39      Dog bart("Bart");
40      Cat socks("Socks");
41      bart.bark(socks);
42      socks.meow(bart);
43  }
```

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

 woof  woof  woof  woof  woof
 meow  meow  meow  meow
```