Chapter 11: Friends, Overloading, and const

Instructor: Mark Edmonds

edmonds_mark@smc.edu

`friend` functions

- `friend` functions are NOT member methods but have access to **`private`** members of that class
- `friend` functions must be named inside the class definition
- `friend` functions are always public regardless of where they are placed in the class definition
- We wary of using `friend` functions; they defeat the purpose of encapsulation, but can be necessary in some circumstances

Number example

- Let's look at a Number class that uses a friend function to add two Numbers together
- Notice we didn't have to define the `add` function in the scope of `Number` (i.e. we didn't have to write `Number Number::add(Number left, Number right)`)
  - This is a bit unusual, but it's because `friend` functions aren't actually a member of the class

Example NumberDriver.cpp

```
1  // NumberDriver.cpp : Defines the entry point for the console
       application.
2  //
3
4  #include <iostream>
5  #include <cstdlib>
6
7  #include "Number.h"
8
9  using namespace std;
10
11 Number add( Number left, Number right );
12
13 int main(int argc, char* argv[])
14 {
15     cout << "----120----" << endl;
16     Number n = Number( 120 );
```

```
17      n.printRomanNumeral();
18
19      cout << "----99----" << endl;
20      n.setValue( 99 );
21      n.printRomanNumeral();
22
23      Number four = Number( 4 );
24      Number five = Number( 5 );
25      Number nine = add( four, five );
26
27      cout << "----9----" << endl;
28      nine.printRomanNumeral();
29
30      return( 0 );
31  }
```

Example Number.h

```
1   #ifndef NUMBER_H
2   #define NUMBER_H
3
4   #include <iostream>
5   #include <cstdlib>
6
7   using namespace std;
8
9
10  class Number {
11  public:
12      Number();
13      Number( int initValue );
14
15      void setValue( int v );
16      int  getValue();
17      void printRomanNumeral();
18
19      friend Number add( Number left, Number right );
20  private:
21      int value;
22  };
23
24  #endif
```

Example Number.cpp

```cpp
#include <iostream>
#include <cstdlib>

#include "Number.h"

using namespace std;

// note the friend function is not a member function
Number add( Number left, Number right ) {
    Number temp = Number( left.value + right.value );
    return( temp );
}

Number::Number() {
    value = 0;
}

Number::Number( int initValue ) {
    value = initValue;
}

void Number::setValue( int v ) {
    value = v;
}

int  Number::getValue() {
    return( value );
}

void Number::printRomanNumeral() {
    // uses cout
    int remainder = value;
    int thousands = remainder / 1000;
    remainder -= thousands * 1000;
    int fivehundreds = remainder / 500;
    remainder -= fivehundreds * 500;
    int hundreds = remainder / 100;
    remainder -= hundreds * 100;
```

```
39      int fiftys = remainder / 50;
40      remainder -= fiftys * 50;
41      int tens = remainder / 10;
42      remainder -= tens * 10;
43      int fives = remainder / 5;
44      remainder -= fives * 5;
45      int ones = remainder;
46
47      int i;
48
49      for (i = 1; i <= thousands; ++i)
50          cout << "M";
51      if (fivehundreds == 1 && hundreds == 4) {
52          cout << "CM";
53          fivehundreds = 0;
54          hundreds = 0;
55      }
56      for (i = 1; i <= fivehundreds; ++i)
57          cout << "D";
58      if (hundreds == 4) {
59          cout << "CD";
60          hundreds = 0;
61      }
62      for (i = 1; i <= hundreds; ++i)
63          cout << "C";
64      if (fiftys == 1 && tens == 4) {
65          cout << "XC";
66          fiftys = 0;
67          tens = 0;
68      }
69      for (i = 1; i <= fiftys; ++i)
70          cout << "L";
71      if (tens == 4) {
72          cout << "XL";
73          tens = 0;
74      }
75      for (i = 1; i <= tens; ++i)
76          cout << "X";
77      if (fives == 1 && ones == 4) {
78          cout << "IX";
79          ones = 0;
80          fives = 0;
81      }
```

```
82      for (i = 1; i <= fives; ++i)
83          cout << "V";
84      if (ones == 4) {
85          cout << "IV";
86          ones = 0;
87      }
88      for (i = 1; i <= ones; ++i)
89          cout << "I";
90      cout << endl;
91  }
```

const revisited

- Remember our qualifier **const**. It makes the type unmodifiable, meaning the value is not allowed to be changed.
  - If we wrote `const int a = 5;` it means the integer `a` is initialized to 5 and cannot be changed to a value other than 5
- For a quick example:

```
1  const int DAYS_IN_WEEK = 7;
2
3  for (int i = 0; i < DAYS_IN_WEEK; i++) {
4      read_textbook_chapter();
5      study();
6  }
```

- **const** can also be applied to function parameters (as was briefly mentioned with the notes on pass-by-reference)
  - **const** is unnecessary with call-by-value parameters
  - **const** can be applied to call-by-reference parameters
    - We should prefer to use call-by-reference when we are passing objects (from either structs or classes)
    - If no changes are made to the object, we can pass by constant reference (e.g. `const MyClass& my_object`)
- **const** can also be applied to member functions. This means the function is not allowed to modify the object; i.e. the function is not allowed to modify an member variables.

`const` Number example

- Notice our `printRomanNumeral()` method is a **`const`** method (not allowed to modify member variables of the object)
- Notice we passed `left` and `right` by constant reference to the `add()` `friend` function

Example ConstDriver.cpp

```
1  // ConstDriver.cpp : Defines the entry point for the console
       application.
2  //
3
4  #include <iostream>
5  #include <cstdlib>
6
7  #include "Number.h"
8
9  using namespace std;
10
11 int main(int argc, char* argv[])
12 {
13     Number four = Number( 4 );
14     Number five = Number( 5 );
15     Number nine = add( four, five );
16
17     cout << "----9----" << endl;
18     nine.printRomanNumeral();
19
20     return( 0 );
21 }
```

Example Number.h

```
1  #ifndef NUMBER_H
2  #define NUMBER_H
3
4  #include <iostream>
5  #include <cstdlib>
6
7  using namespace std;
8
```

```
9
10 class Number {
11 public:
12     Number();
13     Number( int initValue );
14
15     void setValue( int v );
16     int  getValue() const;
17     void printRomanNumeral() const;
18
19     friend Number add( const Number& left, const Number& right );
20 private:
21     int value;
22 };
23
24 #endif
```

Example Number.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3
4  #include "Number.h"
5
6  using namespace std;
7
8  // note the friend function is not a member function
9  // note the use of the const modifier
10 Number add( const Number& left, const Number& right ) {
11     Number temp = Number( left.value + right.value );
12     return( temp );
13 }
14
15 Number::Number() {
16     value = 0;
17 }
18
19 Number::Number( int initValue ) {
20     value = initValue;
21 }
22
23 void Number::setValue( int v ) {
```

```
24      value = v;
25  }
26
27  // note the use of the const modifier
28  int  Number::getValue() const {
29      return( value );
30  }
31
32  // note the use of the const modifier
33  void Number::printRomanNumeral() const {
34      // uses cout
35      int remainder = value;
36      int thousands = remainder / 1000;
37      remainder -= thousands * 1000;
38      int fivehundreds = remainder / 500;
39      remainder -= fivehundreds * 500;
40      int hundreds = remainder / 100;
41      remainder -= hundreds * 100;
42      int fiftys = remainder / 50;
43      remainder -= fiftys * 50;
44      int tens = remainder / 10;
45      remainder -= tens * 10;
46      int fives = remainder / 5;
47      remainder -= fives * 5;
48      int ones = remainder;
49
50      int i;
51
52      for (i = 1; i <= thousands; ++i)
53          cout << "M";
54      if (fivehundreds == 1 && hundreds == 4) {
55          cout << "CM";
56          fivehundreds = 0;
57          hundreds = 0;
58      }
59      for (i = 1; i <= fivehundreds; ++i)
60          cout << "D";
61      if (hundreds == 4) {
62          cout << "CD";
63          hundreds = 0;
64      }
65      for (i = 1; i <= hundreds; ++i)
66          cout << "C";
```

```
67      if (fiftys == 1 && tens == 4) {
68          cout << "XC";
69          fiftys = 0;
70          tens = 0;
71      }
72      for (i = 1; i <= fiftys; ++i)
73          cout << "L";
74      if (tens == 4) {
75          cout << "XL";
76          tens = 0;
77      }
78      for (i = 1; i <= tens; ++i)
79          cout << "X";
80      if (fives == 1 && ones == 4) {
81          cout << "IX";
82          ones = 0;
83          fives = 0;
84      }
85      for (i = 1; i <= fives; ++i)
86          cout << "V";
87      if (ones == 4) {
88          cout << "IV";
89          ones = 0;
90      }
91      for (i = 1; i <= ones; ++i)
92          cout << "I";
93      cout << endl;
94  }
```

Overloading operators

```
1  Number four = Number(4);
2  Number five = Number(5);
3
4  // Wouldn't it be wonderful if we could do the following
5  Number nine = four + five;
```

- Adding two numbers may make intuitive sense, but `Number` is a class. What does it mean to add two classes together?
  - For our `Number` class, you may be able to imagine what this would mean

  - But what about for a car? What about for a bank account?
- Most of the operators we have used thus far can be *overloaded*
  - `+`, `-`, `==`, `/`, `*`, `++`, `--`, `+=`, `-=`, `*=`, `/=` can all be overloaded
  - We cannot overload the `::` and `.` operators
- These operators are all just functions, but we have to list their arguments differently than what we are used to
- Operators are defined as `friend` functions, typically with **`const`** argument parameters

```
1 friend Number operator +(const Number& left, const Number& right);
2
3 friend bool operator ==(const Number& left, const Number& right);
```

Number overloaded operator example

- This example shows how to overload a number of operators

Example OperatorDriver.cpp

```
1  // OperatorDriver.cpp : Defines the entry point for the console
       application.
2  //
3
4  #include <iostream>
5  #include <cstdlib>
6
7  #include "Number.h"
8
9  int main(int argc, char* argv[])
10 {
11     using namespace std;
12     using namespace cs52;
13
14     Number four = Number( 4 );
15     Number five = Number( 5 );
16
17     cout << "----9----" << endl;
18     Number nine = four + five;
19     nine.printRomanNumeral();
20
21     cout << "----1----" << endl;
22     Number one = five - four;
```

```
23      one.printRomanNumeral();
24
25      if (four == five) {
26          cout << "four equals five" << endl;
27      }
28      else {
29          cout << "four does not equal five" << endl;
30      }
31
32      return( 0 );
33  }
```

Example Number.h

```
1   #ifndef NUMBER_H
2   #define NUMBER_H
3
4   #include <iostream>
5   #include <cstdlib>
6
7   namespace cs52 {
8
9   class Number {
10  public:
11      Number();
12      Number( int initValue );
13
14      void setValue( int v );
15      int  getValue() const;
16      void printRomanNumeral() const;
17
18      friend Number operator + ( const Number& left, const Number& right
            );
19      friend Number operator - ( const Number& left, const Number& right
            );
20      friend bool   operator ==( const Number& left, const Number& right
            );
21      friend Number operator / ( const Number& left, const Number& right
            );
22      friend Number operator * ( const Number& left, const Number& right
            );
23
```

```
24 private:
25     int value;
26 };
27
28 }
29
30 #endif
```

Example Number.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3
4  #include "Number.h"
5
6  namespace cs52 {
7
8  // note the friend operator functions are not member functions
9  // note the use of the const modifier
10 Number operator+ ( const Number& left, const Number& right ) {
11     Number temp = Number( left.value + right.value );
12     return( temp );
13 }
14
15 Number operator- ( const Number& left, const Number& right ) {
16     Number temp = Number( left.value - right.value );
17     return( temp );
18 }
19
20 Number operator/ ( const Number& left, const Number& right ) {
21     Number temp = Number( left.value / right.value );
22     return( temp );
23 }
24
25 Number operator* ( const Number& left, const Number& right ) {
26     Number temp = Number( left.value * right.value );
27     return( temp );
28 }
29
30 bool operator== ( const Number& left, const Number& right ) {
31     return( left.value == right.value );
32 }
```

```
33
34 Number::Number() {
35     value = 0;
36 }
37
38 Number::Number( int initValue ) {
39     value = initValue;
40 }
41
42 void Number::setValue( int v ) {
43     value = v;
44 }
45
46 // note the use of the const modifier
47 int  Number::getValue() const {
48     return( value );
49 }
50
51 // note the use of the const modifier
52 void Number::printRomanNumeral() const {
53     // uses cout
54     int remainder = value;
55     int thousands = remainder / 1000;
56     remainder -= thousands * 1000;
57     int fivehundreds = remainder / 500;
58     remainder -= fivehundreds * 500;
59     int hundreds = remainder / 100;
60     remainder -= hundreds * 100;
61     int fiftys = remainder / 50;
62     remainder -= fiftys * 50;
63     int tens = remainder / 10;
64     remainder -= tens * 10;
65     int fives = remainder / 5;
66     remainder -= fives * 5;
67     int ones = remainder;
68
69     int i;
70
71     for (i = 1; i <= thousands; ++i)
72         cout << "M";
73     if (fivehundreds == 1 && hundreds == 4) {
74         cout << "CM";
75         fivehundreds = 0;
```

```
76          hundreds = 0;
77      }
78      for (i = 1; i <= fivehundreds; ++i)
79          cout << "D";
80      if (hundreds == 4) {
81          cout << "CD";
82          hundreds = 0;
83      }
84      for (i = 1; i <= hundreds; ++i)
85          cout << "C";
86      if (fiftys == 1 && tens == 4) {
87          cout << "XC";
88          fiftys = 0;
89          tens = 0;
90      }
91      for (i = 1; i <= fiftys; ++i)
92          cout << "L";
93      if (tens == 4) {
94          cout << "XL";
95          tens = 0;
96      }
97      for (i = 1; i <= tens; ++i)
98          cout << "X";
99      if (fives == 1 && ones == 4) {
100         cout << "IX";
101         ones = 0;
102         fives = 0;
103     }
104     for (i = 1; i <= fives; ++i)
105         cout << "V";
106     if (ones == 4) {
107         cout << "IV";
108         ones = 0;
109     }
110     for (i = 1; i <= ones; ++i)
111         cout << "I";
112     cout << endl;
113 }
114
115 }
```

Overloading << and >>

- `cout` and `cin` use these operators to print information to the screen and extract information from user input
- For our number class, it may look like this:

```
friend ostream& operator <<(ostream& outs, const Number& n);

friend istream& operator >>(istream& ins, Number& n);
```

Example InsertExtract.cpp

```
// InsertExtract.cpp : Defines the entry point for the console
   application.
//

#include <iostream>
#include <cstdlib>

#include "Number.h"

int main(int argc, char* argv[])
{
    using namespace std;
    using namespace cs52;

    Number four = Number( 4 );
    Number five = Number( 5 );
    Number n;

    cout << "---9---" << endl;
    cout << four + five;

    cout << "Enter an integer:";
    cin  >> n;
    cout << "-------" << endl;
    cout << n;

    return( 0 );
}
```

Example Number.h

```
 #ifndef NUMBER_H
#define NUMBER_H

#include <iostream>
#include <cstdlib>

namespace cs52 {

class Number {
public:
    Number();
    Number( int initValue );

    void setValue( int v );
    int  getValue() const;
    // instead of using cout, accept an ostream
    void printRomanNumeral( ostream& outs ) const;

    friend Number operator + ( const Number& left, const Number& right
        );
    friend Number operator - ( const Number& left, const Number& right
        );
    friend bool   operator ==( const Number& left, const Number& right
        );
    friend Number operator / ( const Number& left, const Number& right
        );
    friend Number operator * ( const Number& left, const Number& right
        );

    friend std::ostream& operator <<( std::ostream& outs, const Number&
         n );
    friend std::istream& operator >>( std::istream& ins, Number& n );

private:
    int value;
};

}

#endif
```

Example Number.cpp

```
#include <iostream>
#include <cstdlib>

#include "Number.h"
using namespace std;

namespace cs52 {

// note the friend operator functions are not member functions
// note the use of the const modifier
ostream& operator <<( ostream& outs,
                      const Number& n ) {
    n.printRomanNumeral( outs );
    return( outs );
}

istream& operator >>( istream& ins,
                      Number& n ) {
    int i = 0;
    ins >> i;
    n.setValue( i );
    return( ins );
}

Number operator+ ( const Number& left, const Number& right ) {
    Number temp = Number( left.value + right.value );
    return( temp );
}

Number operator- ( const Number& left, const Number& right ) {
    Number temp = Number( left.value - right.value );
    return( temp );
}

Number operator/ ( const Number& left, const Number& right ) {
    Number temp = Number( left.value / right.value );
    return( temp );
}

Number operator* ( const Number& left, const Number& right ) {
    Number temp = Number( left.value * right.value );
```

```
42      return( temp );
43  }
44
45  bool operator== ( const Number& left, const Number& right ) {
46      return( left.value == right.value );
47  }
48
49  Number::Number() {
50      value = 0;
51  }
52
53  Number::Number( int initValue ) {
54      value = initValue;
55  }
56
57  void Number::setValue( int v ) {
58      value = v;
59  }
60
61  // note the use of the const modifier
62  int  Number::getValue() const {
63      return( value );
64  }
65
66  // note the use of the const modifier
67  void Number::printRomanNumeral( ostream& outs ) const {
68      // uses cout
69      int remainder = value;
70      int thousands = remainder / 1000;
71      remainder -= thousands * 1000;
72      int fivehundreds = remainder / 500;
73      remainder -= fivehundreds * 500;
74      int hundreds = remainder / 100;
75      remainder -= hundreds * 100;
76      int fiftys = remainder / 50;
77      remainder -= fiftys * 50;
78      int tens = remainder / 10;
79      remainder -= tens * 10;
80      int fives = remainder / 5;
81      remainder -= fives * 5;
82      int ones = remainder;
83
84      int i;
```

```
85
86      for (i = 1; i <= thousands; ++i)
87          outs << "M";
88      if (fivehundreds == 1 && hundreds == 4) {
89          outs << "CM";
90          fivehundreds = 0;
91          hundreds = 0;
92      }
93      for (i = 1; i <= fivehundreds; ++i)
94          outs << "D";
95      if (hundreds == 4) {
96          outs << "CD";
97          hundreds = 0;
98      }
99      for (i = 1; i <= hundreds; ++i)
100         outs << "C";
101     if (fiftys == 1 && tens == 4) {
102         outs << "XC";
103         fiftys = 0;
104         tens = 0;
105     }
106     for (i = 1; i <= fiftys; ++i)
107         outs << "L";
108     if (tens == 4) {
109         outs << "XL";
110         tens = 0;
111     }
112     for (i = 1; i <= tens; ++i)
113         outs << "X";
114     if (fives == 1 && ones == 4) {
115         outs << "IX";
116         ones = 0;
117         fives = 0;
118     }
119     for (i = 1; i <= fives; ++i)
120         outs << "V";
121     if (ones == 4) {
122         outs << "IV";
123         ones = 0;
124     }
125     for (i = 1; i <= ones; ++i)
126         outs << "I";
127     outs << endl;
```

```
128 }
129
130
131 }
```