Case Study: Creating a String Class

Earlier in this book you were introduced to the C++ standard library `string` class. The `string` class automatically handles many of the tedious tasks involved in using strings, such as dynamic memory allocation, and bounds checking. It also overloads operators, such as + and =, and offers many member functions that ease the job of working with strings. In this section, however, you will learn to write your own string handling class. In the process, you will see examples of the copy constructor and overloaded operators in full action.

The `MyString` Class

The `MyString` class defined in this section is an abstract data type for handling strings. It offers several advantages over standard C++ character array manipulation:

- Memory is dynamically allocated for any string stored in a `MyString` object. The programmer using this class doesn't need to be concerned with how large to make an array.
- Strings may be assigned to a `MyString` object with the = operator. The programmer using this class does not have to call the `strcpy` function.
- One string may be concatenated to another with the += operator. This eliminates the need for the `strcat` function.
- Strings may be tested with the relational operators. The programmer using this class doesn't have to call the `strcmp` function.

The following program listings show the class implementation.

Contents of `MyString.h`

```
1  // Specification file for the MyString class
2  #ifndef MYSTRING_H
3  #define MYSTRING_H
4  #include <iostream>
5  using namespace std;
6
```

```
class MyString;  // Forward declaration.
ostream &operator<<(ostream &, const MyString &);
istream &operator>>(istream &, MyString &);

// MyString class. An abstract data type for handling strings.

class MyString
{
private:
    char *str;
    int len;
public:
    // Default constructor
    MyString()
        { str = NULL; len = 0; }

    // Copy constructor
    MyString(MyString &right)
        { str = new char[right.length() + 1];
          strcpy(str, right.getValue());
          len = right.length(); }

    // The following constructor initializes the
    // MyString object with a C-string
    MyString(char *sptr)
        { len = strlen(sptr);
          str = new char[len + 1];
          strcpy(str, sptr); }

    // Destructor
    ~MyString()
        { if (len != 0) delete [] str; }

    // The length function returns the string length.
    int length() const
        { return len; }

    // The getValue function returns the string.
    const char *getValue() const
        { return str; };

    // Overloaded operators
    const MyString operator+=(MyString &);
    const char *operator+=(const char *);
    const MyString operator=(MyString &);
    const char *operator=(const char *);
    int operator==(MyString &);
    int operator==(const char *);
    int operator!=(MyString &);
    int operator!=(const char *);
    bool operator>(MyString &);
    bool operator>(const char *);
    bool operator<(MyString &);
```

```
60      bool operator<(const char *);
61      bool operator>=(MyString &);
62      bool operator>=(const char*);
63      bool operator<=(MyString &);
64      bool operator<=(const char *);
65
66      // Friends
67      friend ostream &operator<<(ostream &, const MyString &);
68      friend istream &operator>>(istream &, MyString &);
69  };
70
71  #endif
```

Contents of `MyString.cpp`

```
1   // Implementation file for the MyString class
2   #include <cstring> // For string library functions
3   #include "MyString.h"
4   using namespace std;
5
6   //*********************************************
7   // Overloaded = operator. Called when operand     *
8   // on the right is another MyString object.       *
9   // Returns the calling object.                    *
10  //*********************************************
11
12  const MyString MyString::operator=(MyString &right)
13  {
14      if (len != 0)
15          delete [] str;
16      str = new char[right.length() + 1];
17      strcpy(str, right.getValue());
18      len = right.length();
19      return *this;
20  }
21
22  //*********************************************
23  // Overloaded = operator. Called when operand     *
24  // on the right is a C-string.                    *
25  // Returns the str member of the calling object.  *
26  //*********************************************
27
28  const char *MyString::operator=(const char *right)
29  {
30      if (len != 0)
31          delete [] str;
32      len = strlen(right);
33      str = new char[len + 1];
34      strcpy(str, right);
35      return str;
36  }
37
```

```
38  //************************************************
39  // Overloaded += operator. Called when operand    *
40  // on the right is another MyString object.       *
41  // Concatenates the str member of right to the    *
42  // str member of the calling object.              *
43  // Returns the calling object.                    *
44  //************************************************
45
46  const MyString MyString::operator+=(MyString &right)
47  {
48      char *temp = str;
49
50      str = new char[strlen(str) + right.length() + 1];
51      strcpy(str, temp);
52      strcat(str, right.getValue());
53      if (len != 0)
54            delete [] temp;
55      len = strlen(str);
56      return *this;
57  }
58
59  //************************************************
60  // Overloaded += operator. Called when operand    *
61  // on the right is a string. Concatenates the     *
62  // str member of right to the str member of       *
63  // the calling object.                            *
64  // Returns the str member of the calling object.  *
65  //************************************************
66
67  const char *MyString::operator+=(const char *right)
68  {
69      char *temp = str;
70
71      str = new char[strlen(str) + strlen(right) + 1];
72      strcpy(str, temp);
73      strcat(str, right);
74      if (len != 0)
75            delete [] temp;
76      return str;
77  }
78
79  //*************************************************
80  // Overloaded == operator.                        *
81  // Called when the operand on the right is a MyString *
82  // object. Returns 1 if right.str is the same as str. *
83  //*************************************************
84
85  int MyString::operator==(MyString &right)
86  {
87      return !strcmp(str, right.getValue());
88  }
89
```

```
90  //*************************************************
91  // Overloaded == operator.                        *
92  // Called when the operand on the right is a string. *
93  // Returns 1 if right is the same as str.         *
94  //*************************************************
95
96  int MyString::operator==(const char *right)
97  {
98      return !strcmp(str, right);
99  }
100
101 //*********************************************************
102 // Overloaded != operator.                                 *
103 // Called when the operand on the right is a MyString      *
104 // object. Returns true if right.str is not equal to str. *
105 //*********************************************************
106
107 int MyString::operator!=(MyString &right)
108 {
109     return strcmp(str, right.getValue());
110 }
111
112 //*************************************************
113 // Overloaded != operator.                        *
114 // Called when the operand on the right is a string. *
115 // Returns true if right is not equal to str.     *
116 //*************************************************
117
118 int MyString::operator!=(const char *right)
119 {
120     return strcmp(str, right);
121 }
122
123 //*********************************************************
124 // Overloaded > operator.                                  *
125 // Called when the operand on the right is a MyString      *
126 // object. Returns true if str is greater than right.str. *
127 //*********************************************************
128
129 bool MyString::operator>(MyString &right)
130 {
131     bool status;
132
133     if (strcmp(str, right.getValue()) > 0)
134         status = true;
135     else
136         status = false;
137     return status;
138 }
139
140 //*************************************************
141 // Overloaded > operator.                         *
142 // Called when the operand on the right is a string. *
143 // Returns true if str is greater than right.     *
144 //*************************************************
145
```

```
146 bool MyString::operator>(const char *right)
147 {
148     bool status;
149
150     if (strcmp(str, right) > 0)
151         status = true;
152     else
153         status = false;
154     return status;
155 }
156
157 //*****************************************************
158 // Overloaded < operator.                             *
159 // Called when the operand on the right is a MyString  *
160 // object. Returns true if str is less than right.str. *
161 //*****************************************************
162
163 bool MyString::operator<(MyString &right)
164 {
165     bool status;
166
167     if (strcmp(str, right.getValue()) < 0)
168         status = true;
169     else
170         status = false;
171     return status;
172 }
173
174 //*****************************************************
175 // Overloaded < operator.                             *
176 // Called when the operand on the right is a string. *
177 // Returns true if str is less than right.            *
178 //*****************************************************
179
180 bool MyString::operator<(const char *right)
181 {
182     bool status;
183
184     if (strcmp(str, right) < 0)
185         status = true;
186     else
187         status = false;
188     return status;
189 }
190
191 //*****************************************************
192 // Overloaded >= operator.                            *
193 // Called when the operand on the right is a MyString *
194 // object. Returns true if str is greater than or     *
195 // equal to right.str                                 *
196 //*****************************************************
197
198 bool MyString::operator>=(MyString &right)
199 {
200     bool status;
201
```

```
202     if (strcmp(str, right.getValue()) >= 0)
203         status = true;
204     else
205         status = false;
206     return status;
207 }
208 
209 //*********************************************************
210 // Overloaded >= operator.                                *
211 // Called when the operand on the right is a string.      *
212 // Returns true if str is greater than or equal to right. *
213 //*********************************************************
214 
215 bool MyString::operator>=(const char *right)
216 {
217     bool status;
218 
219     if (strcmp(str, right) >= 0)
220         status = true;
221     else
222         status = false;
223     return status;
224 }
225 
226 //*********************************************************
227 // Overloaded <= operator.                                *
228 // Called when the operand on the right is a MyString      *
229 // object. Returns true if right.str is less than or equal *
230 // to right.str.                                          *
231 //*********************************************************
232 
233 bool MyString::operator<=(MyString &right)
234 {
235     bool status;
236 
237     if (strcmp(str, right.getValue()) <= 0)
238         status = true;
239     else
240         status = false;
241     return status;
242 }
243 
244 //*************************************************
245 // Overloaded <= operator.                        *
246 // Called when the operand on the right is a string.   *
247 // Returns true if str is less than or equal to right. *
248 //*************************************************
249 
250 bool MyString::operator<=(const char *right)
251 {
252     bool status;
253 
254     if (strcmp(str, right) <= 0)
255         status = true;
256     else
257         status = false;
```

```
258     return status;
259 }
260
261 //************************************************
262 // Overloaded stream insertion operator (<<).     *
263 //************************************************
264
265 ostream &operator<<(ostream &strm, const MyString &obj)
266 {
267     strm << obj.str;
268     return strm;
269 }
270
271 //************************************************
272 // Overloaded stream extraction operator (>>).    *
273 //************************************************
274
275 istream &operator>>(istream &strm, MyString &obj)
276 {
277     strm.getline(obj.str, obj.len);
278     strm.ignore();
279     return strm;
280 }
```

The Copy Constructor

Because the `MyString` class has a pointer as a member and dynamically allocates memory to store its string value, a copy constructor is provided. This function will cause the object to properly set up its data when initialized with another `MyString` object.

The Overloaded = Operators

The `MyString` class has two overloaded = operators. The first is for assigning one `MyString` object to another. This operator function is called when the operand on the right of the = sign is a `MyString` object, as shown in the following code segment:

```
MyString first("Hello"), second;
second = first;
```

The second version of `MyString`'s = operator is for assigning a traditional C-string to a `MyString` object. This operator function is called when the operand on the right of = is a string literal or any pointer to a C-string (such as the name of a char array). This is shown in the following program segment:

```
MyString name;
char who[] = "Jimmy";
name = who;
```

The Overloaded += Operators

The += operator is designed to concatenate the string on its right to the `MyString` object on its left. Like the = operators, `MyString` has two versions of +=. The first version is designed to work when the right operand is another `MyString` object, as shown in the following program segment:

```
MyString first("Hello "), second("world");
first += second;
```

The second version of the += operator will be called when the right operand is a literal string or any pointer to a character. It is shown here:

```
MyString first("Hello ");
first += "World";
```

The Overloaded == Operators

The `MyString` object has overloaded versions of the == operator for performing equality tests. Like the other operators, the first version is designed to work with another `MyString` object and the second is designed to work with a traditional C-string.

The == operator functions return an integer that can be treated as a Boolean value. Both functions use `strcmp` to compare the operands, and then returns the negative of `strcmp`'s return value. (Recall that `strcmp` uses inverted logic: It returns 0 when its arguments are equal, and returns a nonzero value when they are not equal.) So, these operator functions return `true` if the string contained in the right operand matches the `str` member of the calling object. If the strings of the two operands do not match, the functions return `false`. These operator functions allow the programmer using this class to construct relational expressions such as those shown in the following program segments:

```
MyString name1("John"), name2("John");
if (name1 == name2)
    cout << "The names are the same.\n";
else
    cout << "The names are different.\n";
```

```
MyString name1("John");
if (name1 == "Jon")
    cout << "The names are the same.\n";
else
    cout << "The names are different.\n";
```

The Overloaded > and < Operators

The `MyString` object has two overloaded versions of the > operator for performing greater-than tests, and the < operator for performing less-than tests. The first version of each is designed to work with another `MyString` object and the second is designed to work with a traditional C-string. (The functions use the library function `strcmp` to determine if a greater-than or less-than relationship exists.)

The > functions return a `true` if the `str` member of the calling object is greater than the string contained in the right operand. Otherwise, the functions return `false`. The < functions return a `true` if the `str` member of the calling object is less than the string contained in the right operand. Otherwise, they return `false`.

These operator functions allow the programmer using this class to construct relational expressions such as those shown in the following program segments:

```
MyString name1("John"), name2("Jon");
if (name1 > name2)
```

```
    cout << "John is greater than Jon.\n";
else
    cout << "John is not greater than Jon.\n";
MyString name1("John");
if (name1 < "Jon")
    cout << "John is less than Jon.\n";
else
    cout << "John is not greater than Jon.\n";
```

The Overloaded >= and <= Operators

The `MyString` object has two overloaded versions of the >= operator for performing greater-than or equal-to tests, and the <= operator for performing less-than or equal-to tests. The first version of each is designed to work with another `MyString` object and the second is designed to work with a traditional C-string. (The functions use the library function `strcmp` to determine if a greater-than or less-than relationship exists.)

The >= functions return a `true` if the `str` member of the calling object is greater than or equal to the string contained in the right operand. Otherwise, the functions return `false`. The <= functions return `true` if the `str` member of the calling object is less than or equal to the string contained in the right operand. Otherwise, they return `false`.

These operator functions allow the programmer using this class to construct relational expressions such as those shown in the following program segments:

```
MyString name1("John"), name2("Jon");
if (name1 >= name2)
    cout << "John is greater than or equal to Jon.\n";
else
    cout << "John is not greater than or equal to Jon.\n";
MyString name1("John");
if (name1 <= "Jon")
    cout << "John is less than or equal to Jon.\n";
else
    cout << "John is not less than or equal to Jon.\n";
```

Program 14-17 shows how `MyString`'s += operator performs string concatenation. Additionally, the program's source code demonstrates how `MyString` allows the programmer to treat strings much like any other built-in data type.

Program 14-17

```
1  // This program demonstrates the MyString class.
2  #include <iostream>
3  #include "MyString.h"
4
5  int main()
6  {
7      // Define and initialize several MyString objects.
8      MyString object1("This"), object2("is");
9      MyString object3("a test.");
10     MyString object4 = object1;
11     MyString object5("is only a test.");
```

(program continues)

Program 14-17 *(continued)*

```
12      // Define a C-string.
13      char string1[] = "a test.";
14
15      // Display the MyString objects.
16      cout << "object1: " << object1 << endl;
17      cout << "object2: " << object2 << endl;
18      cout << "object3: " << object3 << endl;
19      cout << "object4: " << object4 << endl;
20      cout << "object5: " << object5 << endl;
21
22      // Display the C-string.
23      cout << "string1: " << string1 << endl;
24
25      // Test the overloaded += operator.
26      object1 += " ";
27      object1 += object2;
28      object1 += " ";
29      object1 += object3;
30      object1 += " ";
31      object1 += object4;
32      object1 += " ";
33      object1 += object5;
34      cout << "object1: " << object1 << endl;
35
36      return 0;
37  }
```

Program Output

```
object1: This
object2: is
object3: a test.
object4: This
object5: is only a test.
string1: a test.
object1: This is a test. This is only a test.
```

Program 14-18 shows how `MyString`'s relational operators can be used to compare strings with the same ease that numeric data types are compared.

Program 14-18

```
1   // This program demonstrates the MyString class.
2   #include <iostream>
3   #include "MyString.h"
4   using namespace std;
5
6   int main()
7   {
8       // Define several MyString objects.
9       MyString name1("Billy"), name2("Sue");
10      MyString name3("joe");
11      MyString string1("ABC"), string2("DEF");
12
```

(program continues)

Program 14-18 *(continued)*

```
13      // Display the MyString object values.
14      cout << "name1: " << name1.getValue() << endl;
15      cout << "name2: " << name2.getValue() << endl;
16      cout << "name3: " << name3.getValue() << endl;
17      cout << "string1: " << string1.getValue() << endl;
18      cout << "string2: " << string2.getValue() << endl;
19
20      // Test the overloaded relational operators.
21      if (name1 == name2)
22          cout << "name1 is equal to name2.\n";
23      else
24          cout << "name1 is not equal to name2.\n";
25
26      if (name3 == "joe")
27          cout << "name3 is equal to joe.\n";
28      else
29          cout << "name3 is not equal to joe.\n";
30
31      if (string1 > string2)
32          cout << "string1 is greater than string2.\n";
33      else
34          cout << "string1 is not greater than string2.\n";
35
36      if (string1 < string2)
37          cout << "string1 is less than string2.\n";
38      else
39          cout << "string1 is not less than string2.\n";
40
41      if (string1 >= string2)
42          cout << "string1 is greater than or equal to string2.\n";
43      else
44          cout << "string1 is not greater than or equal to string2.\n";
45
46      if (string1 >= "ABC")
47          cout << "string1 is greater than or equal to ABC.\n";
48      else
49          cout << "string1 is not greater than or equal to ABC.\n";
50
51      if (string1 <= string2)
52          cout << "string1 is less than or equal to string2.\n";
53      else
54          cout << "string1 is not less than or equal to string2.\n";
55
56      if (string2 <= "DEF")
57          cout << "string2 is less than or equal to DEF.\n";
58      else
59          cout << "string2 is not less than or equal to DEF.\n";
60
61      return 0;
62  }
```

Program 14-18 *(continued)*

Program Output

```
name1: Billy
name2: Sue
name3: joe
string1: ABC
string2: DEF
name1 is not equal to name2.
name3 is equal to joe.
string1 is not greater than string2.
string1 is less than string2.
string1 is not greater than or equal to string2.
string1 is greater than or equal to ABC.
string1 is less than or equal to string2.
string2 is less than or equal to DEF.
```