Case Study: Creating a String Class

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**

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

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 &);
  bool operator<(const char *);
bool operator<(const char *);  
bool operator>=(MyString &);  
bool operator>=(const char*);  
bool operator<=(MyString &);  
bool operator<=(const char *);  

// Friends  
friend ostream &operator<<(ostream &, const MyString &);  
friend istream &operator>>(istream &, MyString &);  
};  

#endif

Contents of MyString.cpp

// Implementation file for the MyString class  
#include <cstring> // For string library functions  
#include "MyString.h"  
using namespace std;  

const MyString MyString::operator=(MyString &right)  
{  
    if (len != 0)  
        delete [] str;  
    str = new char[length() + 1];  
    strcpy(str, right.getValue());  
    len = right.length();  
    return *this;  
}  

const char *MyString::operator=(const char *right)  
{  
    if (len != 0)  
        delete [] str;  
    len = strlen(right);  
    str = new char[len + 1];  
    strcpy(str, right);  
    return str;  
}
Creating a String Class

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    len = strlen(str);
77    return str;
78 }
79
80 //******************************************************************************
81 // Overloaded == operator. * 
82 // Called when the operand on the right is a MyString * 
83 // object. Returns 1 if right.str is the same as str. * 
84 //******************************************************************************
85
86 int MyString::operator==(MyString &right)
87 {
88    return !strcmp(str, right.getValue());
89 }
Creating a String Class

```c++
int MyString::operator==(const char *right)
{
    return !strcmp(str, right);
}

int MyString::operator!=(MyString &right)
{
    return strcmp(str, right.getValue());
}

int MyString::operator!=(const char *right)
{
    return strcmp(str, right);
}

bool MyString::operator>(MyString &right)
{
    bool status;
    if (strcmp(str, right.getValue()) > 0)
        status = true;
    else
        status = false;
    return status;
}
```
Creating a String Class

bool MyString::operator>(const char *right) {
    bool status;
    if (strcmp(str, right) > 0)
        status = true;
    else
        status = false;
    return status;
}

//**************************************************
// Overloaded < operator.                             *
// Called when the operand on the right is a MyString *
// Returns true if str is less than right.str.        *
//*********************************************************

bool MyString::operator<(MyString &right) {
    bool status;
    if (strcmp(str, right.getValue()) < 0)
        status = true;
    else
        status = false;
    return status;
}

//********************************************************
// Overloaded < operator.                              *
// Called when the operand on the right is a string.    *
// Returns true if str is less than right.              *
//********************************************************

bool MyString::operator<(const char *right) {
    bool status;
    if (strcmp(str, right) < 0)
        status = true;
    else
        status = false;
    return status;
}

//*********************************************************
// Overloaded >= operator.                              *
// Called when the operand on the right is a MyString    *
// Returns true if str is greater than or equal to right. *
//*********************************************************

bool MyString::operator>=(MyString &right) {
    bool status;
    if (strcmp(str, right.getValue()) >= 0)
        status = true;
    else
        status = false;
    return status;
}

//**********************************************************
// Overloaded > operator.                               *
// Called when the operand on the right is a MyString     *
// Returns true if str is greater than right.str.         *
//**********************************************************

bool MyString::operator>=(const char *right) {
    bool status;
    if (strcmp(str, right) >= 0)
        status = true;
    else
        status = false;
    return status;
}
if (strcmp(str, right.getValue()) >= 0)
    status = true;
else
    status = false;
return status;

//*********************************************************
// Overloaded >= operator.                                *
// Called when the operand on the right is a string.      *
// Returns true if str is greater than or equal to right. *
//*********************************************************

bool MyString::operator>=(const char *right)
{
    bool status;
    if (strcmp(str, right) >= 0)
        status = true;
    else
        status = false;
    return status;
}

//**********************************************************
// Overloaded <= operator.                                 *
// Called when the operand on the right is a MyString      *
// object. Returns true if right.str is less than or equal *
// to right.str.                                           *
//**********************************************************

bool MyString::operator<=(MyString &right)
{
    bool status;
    if (strcmp(str, right.getValue()) <= 0)
        status = true;
    else
        status = false;
    return status;
}

//****************************************************
// Overloaded <= operator.                             *
// Called when the operand on the right is a string.   *
// Returns true if str is less than or equal to right. *
//****************************************************

bool MyString::operator<=(const char *right)
{
    bool status;
    if (strcmp(str, right) <= 0)
        status = true;
    else
        status = false;
Creating a String Class

```cpp
return status;
}

//*************************************************
// Overloaded stream insertion operator (<<).    *
//*************************************************

ostream &operator<<(ostream &strm, const MyString &obj)
{
    strm << obj.str;
    return strm;
}

//*************************************************
// Overloaded stream extraction operator (>>).   *
//*************************************************

istream &operator>>(istream &strm, MyString &obj)
{
    strm.getline(obj.str, obj.len);
    strm.ignore();
    return strm;
}
```

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:

```cpp
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:

```cpp
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:
Creating a String Class

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)
Creating a String Class

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

Program 14-18  (continued)

    // Display the MyString object values.
    cout << "name1: " << name1.getValue() << endl;
    cout << "name2: " << name2.getValue() << endl;
    cout << "name3: " << name3.getValue() << endl;
    cout << "string1: " << string1.getValue() << endl;
    cout << "string2: " << string2.getValue() << endl;

    // Test the overloaded relational operators.
    if (name1 == name2)
        cout << "name1 is equal to name2.\n";
    else
        cout << "name1 is not equal to name2.\n";
    if (name3 == "joe")
        cout << "name3 is equal to joe.\n";
    else
        cout << "name3 is not equal to joe.\n";
    if (string1 > string2)
        cout << "string1 is greater than string2.\n";
    else
        cout << "string1 is not greater than string2.\n";
    if (string1 < string2)
        cout << "string1 is less than string2.\n";
    else
        cout << "string1 is not less than string2.\n";
    if (string1 >= string2)
        cout << "string1 is greater than or equal to string2.\n";
    else
        cout << "string1 is not greater than or equal to string2.\n";
    if (string1 >= "ABC")
        cout << "string1 is greater than or equal to ABC.\n";
    else
        cout << "string1 is not greater than or equal to ABC.\n";
    if (string1 <= string2)
        cout << "string1 is less than or equal to string2.\n";
    else
        cout << "string1 is not less than or equal to string2.\n";
    if (string2 <= "DEF")
        cout << "string2 is less than or equal to DEF.\n";
    else
        cout << "string2 is not less than or equal to DEF.\n";
    return 0;
}
**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.