

Additional Class Features- Part 2

MyString Class Revisited

In this part we expand the MyString class; adding more advanced features.

Returning references

In C++, a function may return references.

Returning references can be very useful. A function that returns a reference can be used on the left side of an assignment statement. The effect of this is very powerful; we demonstrate this soon with the operator[] function.

We begin with a simple program having a function that returns a reference.

```
#include <iostream>
using namespace std;

int intVal;

// function retRef declaration
int& retRef();

void main()
{
    retRef() = 55;
    cout << intVal; // outputs 55
} // end main

function retRef definition
int& retRef()
{
    return intVal;
} // end retRef
```

We declared the function retRef() to return a reference to int. In the function definition, the statement

Week 11 - Part 2: MyString Class Revisited

2

```
return intVal; // returns a reference to intVal
```

will not return the value of the global variable `intVal`, but returns the variable `intVal` itself. In the main function, the statement

```
retRef() = 55; // gets translated as intVal = 55;
```

sets the value of `intVal` to 55, because `retRef()` returned a reference to `intVal`.

We must be careful when returning a reference. The object that we refer to must not go out of scope. For example, consider the following rewritten version of `retRef()`.

```
int& retRef()
{
    int x = 55; // x now is a local variable
    return x; // returns a reference to x
} // end retRef
```

In the above function, `x` is local to `retRef()` and will go out of scope when `retRef()` returns. Therefore, the reference returned by `retRef()` is useless.

Most recent C++ compilers will flag it as an error when we try to return a reference to a local variable.

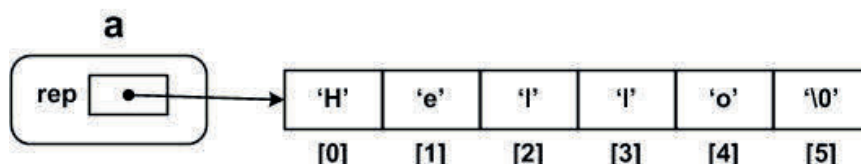
The subscript operator[]

An important use of returning a reference is found in overloading the subscript operator.

Consider the `MyString` class. Suppose that `a` is an object of `MyString` declared as follows.

```
MyString a("Hello");
```

We know that, the above declaration creates object `a` having the C-String `Hello` as shown below.



Week 11 - Part 2: MyString Class Revisited

3

Suppose that we want to replace the character 'H' in position 0 with 'M', so that the modified C-String in a is Mello.

We can accomplish this by adding a member function to the MyString class. The function declaration is as follows.

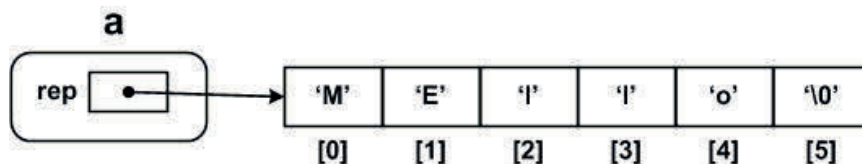
```
char& operator[](int);
```

By this, we are overloading the unary operator []. The name of the function is operator[]. It takes an int argument, which specifies the position of the element in the dynamic array pointed to by rep. Notice that the function returns a reference to a char.

If this operator[] function is defined for the class, we can use any MyString object just as a character array. For example, the statements

```
a[0] = 'M';  
a[1] = 'E';
```

modify the C-String in object a to Mello as illustrated below.



As we know, in C++, there is no bounds checking for arrays. It is therefore possible to overflow or underflow an array. For example, look at the following statements and the comments next to them.

```
a[-1] = 'X'; // underflow index -1 out of bounds  
a[6] = 'Y'; // overflow index 6 out of bounds
```

For our example object a, though index 5 is within the array bounds, we should not allow the replacement of the null character '\0' in position 5 with any other character.

The following definition of operator[] function implements the bounds-checking of the array as required.

```
char& MyString::operator[](int index)
{
    assert ( index >= 0 && index < strlen(rep) );

    return rep[index];
} // end function operator[]
```

The assert statement

In the above function we used the assert statement. The assert statement consists of the identifier assert followed by a logical expression in parantheses and ends with a semicolon. If the logical expression is false, then the program is terminated and an error message is issued. If the logical expression is true, nothing happens and the program proceeds to the next statement after the assert statement. Thus, in the above function, a reference to the char variable at position index is returned only if the index value is in the range from 0 to strlen(rep) - 1.

The assert statement is defined in the library <cassert>, so any program that uses an assert statement must contain

```
#include <cassert>
```

directive.

Comparing two MyString objects

Occasionally, we may need to compare two objects of MyString to find whether they hold the same C-String or not. Sometimes, we may want to know whether the CString in an object is less than the C-String of another object; in the lexical sense. To do so, we can overload the comparison operators ==, !=, <, >, <=, and >=.

We will implement these operator functions as nonmember (friend) functions of MyString class. Doing so will allow the left-hand side of the == operator to be a C-style string or a MyString object. If one of the operands of operator == is a C-style string, a temporary MyString object will be created by the MyString(char*) constructor. Thus, if lhs and rhs are MyString objects, then the following statements are legal.

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/807066132043006143>