Creating and initializing C++ strings

Creating and initializing strings is a straightforward proposition and fairly flexible. In the **SmallString.cpp** example below, the first **string**, **imBlank**, is declared but contains no initial value. Unlike a C **char** array, which would contain a random and meaningless bit pattern until initialization, **imBlank** does contain meaningful information. This **string** object is initialized to hold no characters and can properly report its zero length and absence of data elements using class member functions.

The next string, **heyMom**, is initialized by the literal argument Where are my socks? This form of initialization uses a quoted character array as a parameter to the **string** constructor. By contrast, **standardReply** is simply initialized with an assignment. The last string of the group, **useThisOneAgain**, is initialized using an existing C++ **string** object. Put another way, this example illustrates that **string** objects let you do the following:

Create an empty **string** and defer initializing it with character data.

Initialize a **string** by passing a literal, quoted character array as an argument to the constructor.

Initialize a **string** using the equal sign (**=**).

Use one **string** to initialize another.

```
//: C03:SmallString.cpp
#include <string>
using namespace std;

int main() {
string imBlank;
string heyMom("Where are my socks?");
string standardReply = "Beamed into deep "
"space on wide angle dispersion?";
string useThisOneAgain(standardReply);
} ///:~
```

These are the simplest forms of **string** initialization, but variations offer more flexibility and control. You can do the following:

Use a portion of either a C **char** array or a C++ **string**.

Combine different sources of initialization data using **operator+**.

Use the **string** object s **substr( )** member function to create a substring.

Here s a program that illustrates these features:

```
//: C03:SmallString2.cpp
#include <string>
#include <iostream>
using namespace std;

int main() {
string s1("What is the sound of one clam napping?");
string s2("Anything worth doing is worth overdoing.");
string s3("I saw Elvis in a UFO");
// Copy the first 8 chars:
string s4(s1, 0, 8);
cout << s4 << endl;
// Copy 6 chars from the middle of the source:
string s5(s2, 15, 6);
cout << s5 << endl;
// Copy from middle to end:
string s6(s3, 6, 15);
cout << s6 << endl;
// Copy many different things:
string quoteMe = s4 + "that" +
// substr() copies 10 chars at element 20
```

```
s1.substr(37, 1);
cout << quoteMe << endl;
} ///:~
```

The **string** member function **substr( )** takes a starting position as its first argument and the number of characters to select a second argument. Both arguments have default values. If you say **substr( )** with an empty argument list, you produce a copy entire **string**, so this is a convenient way to duplicate a **string**.

Here s the output from the program:

```
What is
doing
Elvis in a UFO
What is that one clam doing with Elvis in a UFO?
```

Notice the final line of the example. C++ allows **string** initialization techniques to be mixed in a single statement, a flexible a convenient feature. Also notice that the last initializer copies *just one character* from the source **string**.

Another slightly more subtle initialization technique involves the use of the **string** iterators **string::begin( )** and **string::end** technique treats a **string** like a *container* object (which you ve seen primarily in the form of **vector** so far you ll see many mo containers in Chapter 7), which uses *iterators* to indicate the start and end of a sequence of characters. In this way you can h **string** constructor two iterators, and it copies from one to the other into the new **string**:

```
//: C03:StringIterators.cpp
#include <string>
#include <iostream>
#include <cassert>
using namespace std;

int main() {
string source("xxx");
string s(source.begin(), source.end());
assert(s == source);
} ///:~
```

The iterators are not restricted to **begin( )** and **end( )**; you can increment, decrement, and add integer offsets to them, allov extract a subset of characters from the source **string**.

C++ strings may *not* be initialized with single characters or with ASCII or other integer values. You can initialize a string with of copies of a single character, however:

```
//: C03:UhOh.cpp
#include <string>
#include <cassert>
using namespace std;

int main() {
// Error: no single char inits
//! string nothingDoing1('a');
// Error: no integer inits
//! string nothingDoing2(0x37);
// The following is legal:
string okay(5, 'a');
assert(okay == string("aaaaa"));
} ///:~
```

The first argument indicates the number of copies of the second argument to place in the string. The second argument can on single **char**, not a **char** array.

Thinking in C++ Vol 2 - Practical Programming
Prev Home