STL String

aka typedef basic_string<char> string

What we will cover

- The STL String class
- String I/O
- Concatenation
- Comparisons
- String Length and Accessing Individual Elements
- Searching and Substrings
- Splicing and Erasure
- Retrieving a C-style string

The STL String Class

- Strings are objects that represent sequences of characters
  - Until now we've been doing this with C-style char arrays
- Part of the std namespace
- Found inside the <string> header
- typedef basic_string<char> string;
  - Designed specifically to operate with single-byte chars

STL String advantages

- No fixed size
  - STL strings will grow and shrink to fit the data that is contained within it.
- Helper functions
  - STL strings have many features to help concatenating, comparing, and converting that improve on the simple “str” functions that work with C-style strings.

STL String disadvantages

- No fixed size
  - Because strings have to change in size, they are slower to concatenate and append than pure C-style strings.

Declaring a string

- Include the <string> header

```
#include <string>

using namespace std;

void main() {
    string my_string;
}
```

OR

```
#include <string>

void main() {
    std::string my_string;
}
```

String I/O

- Supported by cin

```
#include <string>
#include <iostream>

void main() {
    std::string my_string;
    std::cin >> my_string;
}
```

- Passing an input stream to getline will read an entire line at a time

```
#include <string>
#include <iostream>

void main() {
    std::string my_string;
    std::getline(std::cin, my_string, '\n');
}
```

Concatenation

- Use the = operator assignment
- Use the + operator for concatenation

```
#include <string>
#include <iostream>

void main() {
    std::string my_string1 = "a string";
    std::string my_string2 = " is this";
    std::string my_string3 = my_string1 + my_string2;

    std::cout << my_string3 << std::endl;   // Will output "a string is this"
}
```

Comparisons

- C-style string comparisons require special functions
- For C++ strings, all typical relational operators work as expected
- Can compare 2 C++ strings, or a C++ string and a c string or static string

```
#include <string>
#include <iostream>

void main() {
    std::string passwd;
    std::getline(std::cin, passwd, '\n');
    if(passwd == "xyzzy") {
        std::cout<<"Access allowed";
    }
}
```

String Length and Accessing Individual Elements

- To find the string's length use the length() or size() functions (both return the exact same value)

```
#include <string>
#include <iostream>

void main() {
    std::string my_string1 = "ten chars.";
    int len = my_string1.length(); // or .size();
}
```

- Strings are not guaranteed to be null terminated
  - Its possible to store bytes with a value of 0 inside a C++ string

String Length and Accessing Individual Elements

- Can access individual characters, just like an array

```
#include <string>
#include <iostream>

void main() {
    std::string my_string = "meow";
    for(int i = 0; i < my_string.length(); i++){
        std::cout << my_string[i];
    }
}
```

- Iterators can also be used

Searching and Substrings

- int find(string pattern, int possiton)
  - Search for *pattern* from the given *position*
- rfind
  - Searching begins at the end of the string
- Special value string::npos indicates no match found
- string substr(int position, int length)
  - Create a new string consisting only of the slice beginning at *position* and of *length* characters

Searching and Substrings

```
void main() {
   std::string input;
   int cat_appearances = 0;

   std::getline(std::cin, input, '\n');
   for(int i = input.find("cat", 0); i != std::string::npos; i = input.find("cat", i)) {
      cat_appearances++;
      i++;  // Move past the last discovered instance to avoid finding same string
   }
   std::cout << cat_appearances;
}
```

```
void main() {
   std::string my_string = "abcdefghijklmnop";
   std::string first_ten_of_alphabet = my_string.substr(0, 10);
   std::cout << "The first ten letters of the alphabet are " << first_ten_of_alphabet;
}
```

Splicing and Erasure

- Remove part or all of a string, or insert new text
- Remove part of a string

```
std::string my_removal = "remove aaa";
my_removal.erase(7, 3); // erases aaa
```

- Delete a whole string

```
std::string str = "whiskers";
str.erase(0, str.length());
```

- Splice (insert) one string into another

```
std::string my_string = "ade";
my_string.insert(1, "bc");
std::cout << my_string << std::endl;// my_string is now "abcde"
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Retrieving a c-style string

- You can retrieve a char* from a C++ string
- Useful when working with C library functions
- The c_str() member function returns the char*
  - Returns a constant pointer
  - Do not need to free/delete it

```
void main() {
   std::string my_string = "meow";
   std::cout << strlen(my_string.c_str());
}
```

Conclusion

- STL strings are a simple alternative to c-style strings, and are easy to use
- Provide many convenient functions
- Relational operators work as expected
- Can be used as input to C library functions requiring a char* via the c_str() function

References

- std::string - C++ standard library - Cprogramming.com. 2015. *std::string - C++ standard library - Cprogramming.com*. [ONLINE] Available at:http://www.cprogramming.com/tutorial/string.html. [Accessed 15 April 2015].
- string - C++ Reference. 2015. *string - C++ Reference*. [ONLINE] Available at: http://www.cplusplus.com/reference/string/string/. [Accessed 15 April 2015].

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX