C-Strings and the string Class

Chapter 12

C-String

- What is a string?

C-String

- What is a string?
  - A set, sequence, array, or string, of characters

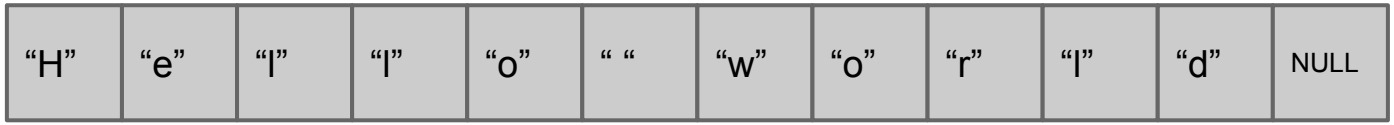
- Up until now, the string Class has been used for most character arrays

C-String

- A C-String is a sequence of characters in memory that are NULL terminated

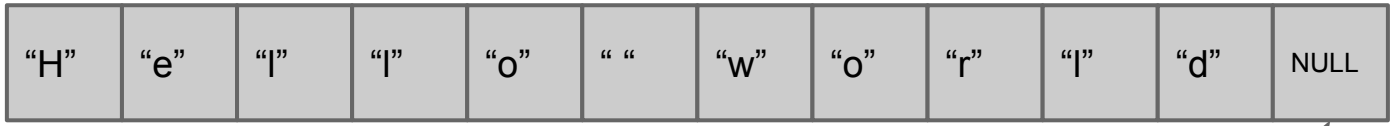
C-String

The string “Hello world” would look like this in memory



C-String

The string “Hello world” would look like this in memory



Notice the NULL terminator. It is very important

C-String

- In a C-String, why is the NULL terminator so important?

C-String

- In a C-String, why is the NULL terminator so important?
  - How would we know when or where the string ended?

C-String

- C-Strings come in different forms
  - String literals
  - Programmer defined arrays of characters
  - Pointers to character

C-String Literal

- C-Strings can come in literal form...literally

C-String Literal

- C-Strings can come in literal form...literally
  - A string literal is a string constant, ie “Bob”

C-String Literal

- C-Strings can come in literal form...literally
  - A string literal is a string constant, ie “Bob”

- When a compiler comes across a string literal, such as “Bob” it allocates enough space for the string plus a null terminator

C-String Literal

- C-Strings can come in literal form...literally
  - A string literal is a string constant, ie “Bob”

- When a compiler comes across a string literal, such as “Bob” it allocates enough space for the string plus a null terminator
  - Hence, “bob” would be 4 bytes long

C-String Literal

- String literals do not change and are hard coded into the program

C-String Literal

- String literals do not change and are hard coded into the program

- We can create strings on the fly

Programmer Defined Arrays of Characters

- A programmer defined array of characters allows a program to create its own C-Strings

Programmer Defined Arrays of Characters

- A programmer defined array of characters allows a program to create its own C-Strings
  - Don’t forget the NULL terminator!

Programmer Defined Arrays of Characters

- A programmer defined array of characters allows a program to create its own C-Strings
  - Don’t forget the NULL terminator!

```
char myString[6];
myString[0] = ‘H’;
myString[1] = ‘e’;
myString[2] = ‘l’;
myString[3] = ‘l’;
myString[4] = ‘o’;
myString[5] = NULL;
```

Programmer Defined Arrays of Characters

- An array of characters can even be initialized with a String literal

Programmer Defined Arrays of Characters

- An array of characters can even be initialized with a String literal

```
char myString[6] = “Hello”;
```

Programmer Defined Arrays of Characters

- An array of characters can even be initialized with a String literal

char myString[6] = “Hello”;

- When initializing with a String literal, the array size is optional. The compiler will choose the appropriate size for you.

Programmer Defined Arrays of Characters

- An array of characters can even be initialized with a String literal

char myString[6] = “Hello”;

- When initializing with a String literal, the array size is optional. The compiler will choose the appropriate size for you.

char myString[] = “Hello”;

Pointers To Characters

- A pointer to a character array can also be used

Pointers To Characters

- A pointer to a character array can also be used

```
char myString[] = “Hello”;
char* pPointer = myString;
```

Pointers To Characters

- A pointer to a character array can also be used

```
char myString[] = “Hello”;
char* pPointer = myString;
```

- An advantage to pointers, is that the pointer can be reassigned
- How would we pass a character array into a function?

Pointers To Characters

- A pointer to a character array can also be used

```
char myString[] = “Hello”;
char* pPointer = myString;
```

- An advantage to pointers, is that the pointer can be reassigned
- How would we pass a character array into a function?
- What about a dynamically allocated C-String?

Command Line Arguments

- Now that we know C-Strings, we can look at passing command line arguments into our program

Command Line Arguments

int main(int argc, char** pArgv)

Command Line Arguments

int main(int argc, char** pArgv)

- The argument ‘pArgv’ is an array of C-Strings
  - Each C-String represents a command line argument

- The argument ‘argc’ is the number of arguments in ‘pArgv’

Command Line Arguments

int main(int argc, char** pArgv)

- The argument ‘pArgv’ is an array of C-Strings
  - Each C-String represents a command line argument

- The argument ‘argc’ is the number of arguments in ‘pArgv’

- Note: pArgv[0] will be the name of your executable
  - If your command was “MyProgram.exe firstArg”
    - pArgv[0] = “MyProgram.exe”

C-String Helper Functions

- Functions
  - strlen
  - strcat
  - strcpy
  - strcmp
  - strstr

C-String Helper Functions

- What are some common tasks we would want to do with a C-String?

C-String Helper Functions

- What are some common tasks we would want to do with a C-String?
  - Convert a string into a value

String Conversions

- Functions
  - atoi
  - atol
  - atof
  - itoa

The C++ string Class

- There is an easier alternative to C-Strings…
  - The C++ string Class

The C++ string Class

- There is an easier alternative to C-Strings…
  - The C++ string Class

```
std::string myString;
```

The C++ string Class

- There is an easier alternative to C-Strings…
  - The C++ string Class

```
std::string myString;
```

http://www.cplusplus.com/reference/string/string/

The C++ string Class

- Constructors
  - string()
  - string(const char* s)
  - string(const string &s)
  - string(const char*, int n)
  - string(int n, char ch)
  - string(const string &s, int p, int n)

The C++ string Class

- Overloaded Operators
  - >>
  - <<
  - =
  - +=
  - +
  - [ ]
  - Relational Operators
    - <, >, <=, ...

The C++ string Class

- Member functions
  - append
  - at
  - c_str
  - compare
  - copy
  - find
  - empty
  - insert
  - length
  - replace
  - substr