An Introduction to C-strings

- Even though C strings are based on the concept of a simple array of characters, cin and cout try to deal with them as best as they can.
- Consider the following code:

```cpp
int main()
{
    char cStr[80]; // Allocate a C string
    cout << "Enter a string> ";
    cin >> cStr;
    cout << "You entered… " << cStr;
}
```

- Let’s make sure this works the way we think it does...

Reading in Strings...

- Why did input stop in the middle of our sentence?
- Because cin is designed to stop reading when whitespace is encountered.
- So how do we read in strings with spaces?
- Read in a character at a time, or rely on getline()
- Only catch is that getline() relies on a C style string.
- You need to allocate space for one before calling...

```cpp
int main()
{
    char cStr[80]; // Allocate a C string
    cout << "Enter a string> ";
    cin.getline(cStr,79);
    cout << "Number is: " << k << ", string is: " << cStr << endl;
}
```

More Dangers...

- Be careful when mixing the reading of strings with the reading of integers.
- You might get some unexpected behavior!
- Consider the following code:

```cpp
int main()
{
    int k;
    char cStr[80]; // Allocate a C string
    cout << "Enter a number> "; cin >> k;
    cout << "Enter a string> "; cin.getline(cStr,79);
    cout << "Number is: " << k << ", string is: " << cStr << endl;
}
```
More Dangers (cont)

- The output of this simple program might look like this:
  Enter a number> 1
  Enter a string>
  Number is 1, string is:

  - So what is going on?
  - The input stream (that which cin reads from) is thought of as an array of characters.
  - So when we enter "1" above, we put the following two characters into the input stream buffer:


More Dangers (cont)

- When the following line of code is executed...
  cout << "Enter a number> "; cin >> k;

  - cin manages to grab the "1" out of the input stream buffer but leaves the newline there. This leaves us with something like this:


More info on C strings

- Be careful of this
  char str1[50] = "This is a test";
  char str2[80] = "We’re already 1/3 way through the semester";
  char *str3 = "This in another way";

  - The above comparison will always be false
  - str1 and str2 are pointers (remember, arrays are pointers)
  - The comparison compares the pointer values, not the values of what each points at.
  - How do you compare C strings?

More info on C strings

- How do we assign initial values to C strings?

  - The first method allows you to provide an initial value to a C string defined to hold 49 characters (+1 for the NULL byte)
  - The second method allows you to provide an initial value AND allow the compiler to figure out the size (length of initial value + 1 for NULL byte)
  - The third method is an alternative syntax for the second method.
Standard C Library Routines
- Here are a few standard C library string routines:
  ```
  strcpy(char *s1, char *s2) -- Copy s2 to s1
  strcat(char *s1, char *s2) -- Append s2 to s1
  strlen(char *s1) -- Return length of s1
  strcmp(char *s1, char *s2) -- Compare s2 and s1
  ```
- Savitch, page 357, has a full reference for these functions.
- To answer the question posed on the last slide, you would compare the strings like this:
  ```
  char str1[80], str2[80];
  cin.getline(str1, 79);
  cin.getline(str2, 79);
  if (!strcmp(str1, str2))
    // Strings are equal.
  ```

Multidimensional Arrays (cont)
- To access an element, use both indices:
  ```
  mdarray[2][2] = 5;
  cout << mdarray[2][2] << endl;
  ```
- You can also initialize multi-dimensional arrays when you declare them.
- Both of the following initializations are legal:
  ```
  int mdarray[3][4] = {{1,2,3,4},{4,2,5,4},{5,5,5,5}};
  ```
- The following declaration is more "human friendly"
  ```
  int md[3][4] = {{1,2,3,4},{4,2,5,4},{5,5,5,5}};
  ```
  - But the compiler can figure out the more complex version

Back to C strings
- Even more danger to be wary of...
  ```
  char str1 = "This is a test";
  cout << str2 = "Hello world";
  ```
- Note the use of strcpy to copy str2 to str1.
- Problem is, str1 is bigger than str2.
- So what does C++ do? Does it only copy as many characters as will fit into str2?
- Naaah, it copies all of them and writes beyond the boundary of str2.
- Doesn’t that cause problems?
- Yip! In this case you’d be overwriting stack memory which is likely to cause problems immediately.

The C++ String Class
- So what can we do?
  ```
  string s = "Hello World";
  ```
  - Be very, very careful
  - Be prepared for lots of debugging
  - OR, use the C++ string class
  - We’ve used the string class in lecture before, but haven’t gone over it in any detail.
  - It has many options built in to the class instead of needing to rely on library functions like C strings do.
  - Let’s review some usage of the C++ string class:
  ```
  void main()
  {
    string s = "Hello World";
    cout << "3rd character of str is: " << s[3] << endl;
    cout << "4th character of str is: " << s[4] << endl;
    cout << "str contains " << s.length() << " characters.";
    string s2 = s + " how are you?";
    cout << "str2 is: " << s2 << endl;
  }
  ```

The C++ String Class (cont)
- Note how we can use the string class as if it were a built in type:
  ```
  Assign values directly to a string variable (overloaded operators)
  "output" string values directly to streams (cout)
  etc.
  ```
- We can also take advantage of some of the many member functions present in the string class:
  ```
  void main()
  {
    string s = "Hello World";
    cout << "3rd character of str is: " << s[3] << endl;
    cout << "4th character of str is: " << s[4] << endl;
    cout << "str contains " << s.length() << " characters.";
  }
  ```
The C++ String class--some member funcs
- Here are some common member functions you might use:

- `Str.Substr(pos, length)` -- returns the substring starting at position that is length long
- `Str.C_str()` -- return a C-style string (read only)
- `Str.at(i)` -- read/write access to the character at position i.
- `Str1 += str2` -- Concatenate str2 onto str1
- `Str.length()` -- Return the length of str
- `Str.find(str1)` -- Find the index of the first occurrence of str1 in str
- `Str.find(str1, pos)` -- Find the index of the first occurrence of str1 in str, starting at position pos.

The C++ String class--comparison operators
- Unlike C strings, you can compare C++ strings directly using the standard comparison operators:

```cpp
void main()
{
    string str1;
    string str2;

    cout << "Enter two strings… " << endl;
    ReadData(str1); // This function was defined in DEMO 2
    ReadData(str2);

    if (str1 == str2)
        cout << "The strings are equal!" << endl;
    else
        cout << "The strings are not equal" << endl;
}
```

Demonstration #3
Comparing C++ Strings

Lecture 8
Final Thoughts