Lecture 8

C-Strings, C++ Strings
And Multidimensional Arrays

"Absolute C++"
Chapter 9, 5.4

An Introduction to C-strings

- A C string is not a class, it is simply an array of characters.
- Any given C string of length `n` will correspond to a character array of *at least* `n+1` characters. The extra character is a NULL byte which terminates the string.
- Most of us old time C programmers (well, OK, I'm not really *that* old) tended to stick with C strings instead of "converting".
- Or we wrote our own String classes which suited our needs better than the C++ string class.
- Either way, it's good to know about C strings.
- Some parts of the language still depend on them:
  - ifstream::open
  - iostream::getline

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:

```
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...

Demonstration #1

C Strings

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…

```
int main()
{
  char cStr[80];  // Allocate a C string
  cout << "Enter a string> ";
  cin.getline(cStr);
  // etc., etc..
}
```

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:

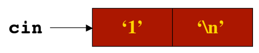
```
int main()
{
  int k;
  char cStr[80];  // Allocate a C string
  cout << "Enter a number> ";   cin >> k;
  cout << "Enter a string> ";   cin.getline(cStr,79);
  cout << "Number is: " << k << ", string is: " << cStr
       << endl;
}
```

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:

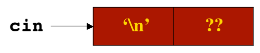


More Dangers (cont)

- 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 Dangers (cont)

- Now, when the next line of code is executed...

```
cout << "Enter a string> "; cin >> cStr;
```

- cin sees the newline as the first character in the input stream buffer and assumes we haven’t “seen” it before.
- It processes the newline viewing it as the terminating newline for our string read.
- This leaves us with an empty string in our string variable.
- So what can we do?
  - Try to extract the newline before actually calling “cin >> cStr”
  - Find a different way to read in data
    - Fall back to char I/O
    - Read in ints, etc., as strings, then convert

Demonstration #2

Reading in Data

More info on C strings

- How do we assign initial values to C strings?

```
char str1[50] = "This is a test";
char str2[] = "We're already 1/3 way through the semester";
char *str3 = "This is another way";
```

- 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.

More info on C strings

- Be careful of this

```
char str1[80],str2[80];
cin.getline(str1,79);
cin.getline(str2,79);
if (str1 == str2) {
  // Do something useful, presumably…
}
// Rest of code here
```

- 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?

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

- A multi-dimensional array (of ints) is declared as follows:

```
int mdarray[7][8];    // 7 rows, 8 colums
```

- This creates a multi-dimensional array of 7 rows and 8 columns.
- You can have as many “dimensions” as system resources allow.
- The following is also legal:

```
char foo[5][6][7][8][9];
```

- What does it represent?
- I have no idea.
- But it’s legal!

Multidimensional Arrays (cont)

- To access an element, use both indices:

```
mdarray[2][2] = 56;
cout << "mdarray[2][2] = " << mdarray[2][2] << endl;
```

- You can also initialize multi-dimensional arrays when you declare them.
- Both of the following initializations are legal:

```
// The following declaration is more "human friendly"
int mda[3][4] = { {1,2,3,4},{4,2,5,4},{5,5,5,5} };
// But the compiler can figure out the more complex version
int mda[3][4] = { 1,2,3,4,4,2,5,4,5,5,5,5 };
```

Back to C strings

- Even more danger to be wary of…

```
char *str1 = "This is a test";  // Allocates 15 bytes
char *str2 = "Hello world";     // Allocates 12 bytes
strcpy(str2,str1);              // Copies str1 to str2?
```

- 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?
- Yup! 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?
  - 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 str = "Hello World";
  cout << "str value is: " << str << 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 str = "Hello World";
  cout << "3rd character of str is: " << str[3] << endl;
  cout << "4th character of str is: " << str.at(4) << endl;
  cout << "str contains " << str.length() << " characters.";
  string str2 = str1 + ", how are you?";
  cout << "str2 is: " << str2 << endl;
}
```

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)   (str[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:

```
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