CS 110 SI Bonus Session

Array-based Data Structures
Session Outline

1. Strings vs Char Arrays
2. The cctype functions
3. Examples
Strings vs Char Arrays

• All strings are char arrays
• But not all Char Arrays are strings.

• eg. char myLetters[20] = “Hello”;

• Creates an array:

```
['H']['e']['l']['l']['o']['\0']
```
This is known as a C-string

- C-strings were around in c, the precursor to c++.
- They have none of the functions which c++ strings have, such as length, substr, find, at.
- However, we have the ability to implement them ourselves.
The Char Data Type

- Recall that the char data type is, at its core, a very small unsigned integer
- The integer refers to a value in the ascii table
  - 'A' = 65, 'B' = 66 ... 'Z' = 90
  - 'a' = 97, 'b' = 98... 'z' = 122
  - '0' = 48, '1' = 49 ... '9' = 57
  - ' ' = 32
- Can you write a simple function which returns the ASCII code for a char character?
Char comparisons

• Based on the ASCII values, we can make some sweeping statements about char comparisons
  – When the case is the same, a letter which comes earlier in the alphabet is less than a letter which comes later in the alphabet
  – When the case is different, the upper case letter is less than the lower case letter
  – Numeric characters are less than alphabetical characters
Examples

• Can you write a function which compares two characters irrespective of case, and returns which one comes first?
The `<cctype>` header

- Determining a characters relationship to other characters requires an intimate knowledge of the ASCII table
- C (and c++) make this a bit easier using the cctype header.
Boolean Functions in cctype

- The following functions return booleans based on what a character is, and take a single character as an argument
  - isalpha
  - isupper
  - islower
  - isdigit
  - ispunct
  - + more
Cctype Char functions

- The following functions return a char, and take a char
  - tolower – always returns the lowercase equivalent
  - toupper – always returns the uppercase equivalent
Without using the cctype header, can you code your own toupper function?

- Algorithm: Check if the input is lowercase
  - If true, return the uppercase variant
  - If false, return the original character entered.
Let's leave our char arrays for a bit and look at the familiar c++ string class.

The designers implemented comparisons.

There are 4 cases for evaluating equality.

Case 1: The strings are of equal length and contain the exact same character.
   - The strings are ruled equal.
<string> compare rules

- Case 2: The strings are of the same length, but have different characters
  - Starting at the beginning of the string, check for equality until a char causes a mismatch, then return the char equality tests for that character

- Case 3: The strings are of different lengths, but have all the same characters for the first part
  - Eg. ['a'][b][c][d][e][f][g] and ['a'][b][c][d][e]
  - The string with more letters is greater
<string> comparator rules

• Case 4: The strings have a different number of characters, and have an inequality within the overlapping subset
  – Test equality until a character fails to be equal, then return the comparison for that character.

• Could you implement something similar for your char array class?
Examples

- Given the char array (char [20] rawData):
  - ['p']['H']['i']['L']['i']['P'][' ']['j'][' ']['f']['R']['y'][' '][' ']...

- Populate the following three arrays:
  - char FirstName[20] <= ['P']['h']['i']['l']['i']['p'][' ']...
  - MiddleInitial <= 'J';
  - char LastName[20] <= ['F']['r']['y'][' ']...
  - Capitalize the first letter and make the rest lower
Example:

• A text file named gibberish.txt contains less than 100 characters
• Read the file into an array
• Search through the file to see if the word “hello” exists in the file
• Do case insensitive matches, eg.
  – “HELLO” and “hello” and “HeLIO” should all match
• Every instance of the string that matches.
Variable Scope

- Global Variables – live as long as the program lives
- Local Variables – live only in the function they are created, from when the function is called to when it returns
- Static variables – live in the function, and persist between function calls