COMS 1003: Introduction to Computer Programming in C

Arrays, String, and Command Line Arguments

September 22nd 2005
Outline

• Relate to Readings
• Arrays
  − Data Collections for multiple objects of the same type
• Strings
  − sequences of characters
  − arrays of characters
  − string literals and string constants
  − string operations (string.h)
• Command line argument processing
Readings

• PCP: Chapters
  – 2 & 3
  – 5

• TCPL: Chapters
  – 2
  – 1.2
Simple Data Collections

• Store a student grade:
  – int locasto_grade = 0;

• What if we have 70 students?
  – int student0_grade = 0;
  – int student1_grade = 0;
  – int student2_grade = 0;
  – int student...
  – int student69_grade = 0;
  – whew!
Arrays

- Arrays are typed collections of the same data type
- Arrays allow for grouping of data under one common name
- Array elements are accessed by giving an offset, or index into the array.
- Offsets (or indexes) are always integer values
  - it doesn't make sense to say “give me the value at element number 5.34"
Array Organization

- Arrays have a name
  - e.g., student_grades
- Arrays have a length
  - not the number of elements in the array, but rather how much memory the array takes up
  - in C, this length is not stored with the array. You, the programmer, must keep track of it
- Array elements must be consistently typed
  - all elements must be of the same type
Accessing Array Elements

• Each element is at a unique position in the array
  - position is indicated by the subscript or index value
  - the value of the subscript or index is NOT the value of the element @ that index or position

```c
int student_array[70];
student_array[45] = 100;
student_array[2] = 50;
student_array[34] = 34;
student_array[0] = 100;
```
Declaring an Array

- Very similar to declaring a single variable of that type
  
  ```
  //declare an integer variable
  int myinteger;
  ```

- Just add brackets and a size
  
  ```
  //declare an int array
  int part_numbers[400];
  ```

- **type**
- **name**
- **size (number of elements)**
Initializing an Array

• There are several ways to initialize the data in an array. The most straightforward are to have an explicit loop or series of statements that initialize each element. This is tedious.

• The second way is to provide a comma separated list of elements in braces, like so:

```c
int temperatures[] = {89,54,100,0,34,-40};
//there is no need to specify a size, b/c
//compiler does it for you.
```
Accessing Array Elements II

- Array elements can be accessed by writing the name of the array and the position of the index of the item in the array.
- This expression is equivalent to the value of the array at that index

```c
int data[2];
data[0] = 400;
data[1] = 3;
printf("value at data[1] is %d\n", data[1]);
```
float sample_values[6] = {0.0};
sample_values[0] = -1.43;
sample_values[5] = 23.47;

//programming error!
sample_values[6] = 7.1;
Array Notes

• The previous slide shows us:
  – arrays start indexing from 0, not 1
  – thus, the array length has a maximum index of (length-1)

• Also, note that C doesn't do bounds checking, so you can inadvertently read or write past the end (or beginning) of an array and the language won't stop you. However, this is (99.99% of the time) an error or not what you wanted to do.
Advanced Array Topics

• There are other ways to access array elements
  - we'll cover one when we cover pointers
  - we'll cover the other when we talk about repetition control structures

• Arrays can be nested; that is, you can have an array of arrays, or an array of array of arrays, or ...
  - the basic idea is that you just add more []'s per dimension
  - a two-dimensional array is an array of arrays or a table
Multi-dimensional Arrays

//declare a two dimensional array of integers
int num_students = 70;
int num_grades = 7;
int class_grades[num_students][num_grades];
//also legal:
int class_grades[70][7];

//access an element by providing subscripts
class_grades[45][6] = 3;

//print the 6th student's 5th grade
printf("%d\n", class_grades[5][4]);
Strings
What are Strings?

- Strings are arbitrarily long sequences of characters
- C keeps many things as simple as possible, so strings in C are not first class data objects
  - Rather, they are simply character arrays
  - This makes sense, but you have to keep some rules in mind when operating on strings
- Just remember that a string is always an array of characters (and treat it as such) and you'll be fine
Character Basics

- Characters in C are 8-bit (1 byte) values that can sometimes be treated like small integers.
- How many unique integer values can you specify with 8 bits?
- In a program, you may represent a character like:
  ```c
  char somechar = 'B';
  ```
  but numbers work equally well:
  ```c
  char anotherchar = 66;
  ```
Examples of Strings

• You've seen some strings before
  - String literals: a sequence of characters in quotation marks inside the text or body of a program
  - For example:
    ```
    printf("result is %d\n", result);
    
    the "result is %d\n" is a string literal.
    ```

• A character array is the other common way to refer to a string
  ```
  char studentname[30];
  ```
String Gotcha's

• In order to truly treat a character array as a string, you must make sure it is null-terminated
  – the last character in the array (or of the string if the string is smaller than the array length) must be a null character
  – the null character is written as '\0' (backslash zero)
  – recall the '\n' for newlines
  – the C compiler automatically null-terminates string literals
A simple example

//declare and initialize 2 char arrays
char myname[] = {'M','i','k','e','\0'};
char yourname[] = {'E','l','i','\0'};

//change some of the values in one string
myname[1] = 'a';
myname[2] = 't';
myname[3] = 't';

//note that chars are really very small ints
yourname[0] = 65;
//what is yourname?
Char Arrays as ... char arrays

- Every string is a character array
- Not every character array is a string
  - character arrays are just collections of characters
  - can hold any legal char value (1 byte or 8 bits of information)
  - interpretation depends on context
  - the data stored in a character array doesn't need to be treated like a string
  - nevertheless, you can still treat it like a string. C allows you to shoot yourself in the foot if you really want to
String Operations

- Many basic string operations are tedious to write
- So these operations are provided as functions in the standard library
- To use them, your program should
  #include <string.h>
- Operations include:
  - strnlen (return the length of the string)
  - strncmp (compare two strings lexicographically)
  - strncpy (copy one string to another)
String Properties

• The length of a string is the number of characters in the string, not counting the '\0' null terminator
• Strings are compared by comparing their basic elements: the characters that they contain
  – compared in lexicographic order
• Semantics are consistent when you deal with multi-dimensional char arrays:
  – char classnames[70][30]; is an array of character arrays (array of strings)
Command Line Arguments
echoargs Demo
Command Line Input

• One of the ways to supply input to your program is via the command line
• But this data comes from the execution environment...how do you refer to it in your code?
• Fortunately, the C language provides a place for this input: the arguments to the main() function
  - argc : an integer specifying how many arguments
  - argv : an array of strings holding the actual values
The signature of `main()`

/* main is a function returning an integer
 * and taking 2 parameters: an integer and
 * an array of strings
 */
int main(int argc, char* argv[])
{
    //argc is the argument count (the length of
    // the argv array)
    //argv is the collection of argument values
    //argv is an array of strings (themselves
    // arrays of characters)
    return 0;
}
The `argv[]` array

- In the signature of main, the declaration of `argv[]` may look strange. It's got a funny asterisk. Ignore that for right now
  - imagine that main is declared like so:
    ```c
    int main(int argc, char argv[][[]])
    {}  
    ```
- In fact, you can write the above and your program will still compile and work, but the compiler will warn you (see program)
Parsing the Command Line

- Command line arguments provide a convenient way of controlling startup options for your program.
- Now that your program can refer to these data objects (via argv[]), it must be able to compare them against known options and enable or disable certain actions based on the content.
Common Options

• Commonly implemented options include:
  – -v or –version (sometimes means 'verbose')
  – -h or –help (print out usage information)

• Your homeworks will need to implement at least these two options

• Other options will be problem-specific, but the general rules for examining them are the same
The `strncmp()` function

- How do you compare two strings?
- Could do it by examining the character arrays character by character
- The standard library provides a function for this common task called `strncmp()`
- It compares two strings up to a certain supplied length
- See the man page for more detail
calling

strncmp("hello", "hello", 5)
returns 0 because the strings are equal
(including the invisible \0 at the end)

calling

strncmp("yes", "nah", 3)
returns 1 because the strings are different
and "yes" is lexicographically greater than "nah"

calling

strncmp("nah", "yes", 3)
returns -1 b/c the strings are different and
"nah" is lexicographically less than "yes"
shoutarg Demo