CMSC 106
Lecture Set #9 – More About Arrays and Sorting

Set Started:
Monday, October 31, 2010

C-Strings

- Definition
  - An array of characters
  - Where the used portion is terminated by a null character
- <string.h>
  - Library that acts on C-strings
  - Most will crash if given something that does not fit the definition above
- Creating and Initializing a string

```
char name1[4] = {'J','a','n','\0'};
char name2[6] = "Plane";
```

- Characters, strings and numeric values are all different length of the string and the sizeof operator

$$"0" \neq '0' \neq 0$$

  - sizeof operator tells the size of the variable or type
  - strlen uses the definition of C-string to find number of used characters

Input and Output

- Output
  - %s in printf format string
  - puts() function takes a string as the only argument
- Input
  - dangerous to use %s in scanf or to use gets() function
  - `char *fgets(char *buffer, int bufferSize, FILE *stream);`
    - read a line into buffer (at most bufferSize-1 characters)
      - null byte added at end of buffer
    - reads from stream – for standard input just type **stdin** as the name of the stream
    - returns NULL on error or end of file
    - on success returns pointer to the space where you read into (here called the buffer)

Strings

- Zero or more characters followed by null char '\0'
  - also called NUL
  - not counted as part of string
  - string.h defines prototypes for string routines
- Some String Functions
  - size_t strlen(char const *str);
    - returns count of characters in str
    - up to but not including the null character
  - char *strncpy(char *dst, char const *src, size_t len);
    - copy src to dst (a better version of strcpy)
    - copy until '\0' in src or at most len characters
    - pad extra characters will '\0'
    - Safety tip: dst[len-1] = '\0'; to force termination of new string
  - char *strncat(char *dst, char const *src, size_t len);
    - append src onto the end of dst (a better version of strcat)
    - always appends NUL to end of dst string

More String Functions

- Comparison
  - int strncmp(char const *s1, char const *s2, size_t len);
    - returns 0 if string equal up to len
    - returns a negative value if s1 is less than s2
    - returns a positive value if s1 is greater than s2
- Searching
  - char *strchr(char const *str, int ch);
  - char *strrchr(char const *str, int ch);
    - finds the first occurance of ch in str
    - strrchr finds the last occurance
    - returns NULL if not found
  - char *strstr(char const *s1, char const *s2);
    - find the first occurrence of s2 in s1

Character Functions

Prototypes in ctype.h

- Classifying characters
  - parameter is int, but it's a character
  - int isspace(int ch);
    - returns true if ch ' ', '\n', '\t', form feed, or carriage return
  - int isdigit(int ch);
    - returns true if its 0 through 9
  - int islower(int ch); and isupper(int ch);
    - return true if it's a-z for islower and A-Z and isupper
  - int isalpha(int ch);
    - returns true if it's a-z or A-Z
  - int isalnum(int ch);
    - returns true if it's a-z or A-Z or 0-9
- Transformation
  - int toupper(int ch), int tolower(int ch)
    - converts to upper/lower case

typedef

- Allows you to define a new type
- Format:

typedef whatItIs whatYouWantToCallIt;

- For example:

typedef int Bool;

- Array example:

typedef char MyString[MAX];

Array Sorting

- To put the elements of an array in order according to some criteria
- Necessary characteristics:
  - Need to have a way to determine "greater" and "lesser"
    - Numeric (use <. >, <= or >=)
    - Strings (use strcmp)
  - Need to be able to change the order based on that criteria
  - Need to continue the process until all elements of the array are in order based on that criteria

Algorithm

- an algorithm is an effective method for solving a problem using a finite sequence of instructions
- Must include:
  - What needs to be done
  - These steps must then be presented in an order
- There are many algorithms available for sorting – we will just look at a few basic ones here

Three Sorting Algorithms

- Bubble Sort
  - Traverses the array "bubbling up" the highest value by comparing every successive pair and swapping those two if needed
- Insertion Sort
  - Inserts the first element into an empty array and assumes that one is in the correct place. Then inserts each additional element by sliding the others down as needed so that one value can be inserted into the correct place of the new array
- Selection Sort
  - Searches through the array to find the smallest and swaps it so that it is now in the correct place (the 0th element), then repeats using the remainder of the array to find the next smallest and swap it into the 1st place, etc until all are in the correct positions

Bubble Sort – Step by Step

make (size-1) passes over the array
  for each element in the array except the last one
  (this means indexes between 0 and (size -2))
    compare that element to the one immediately after it in the array
    (all will have one immediately after because the loop stopped at size-2)
      if these two items are in the wrong order, swap them

Bubble Sort Example Sorting (5 1 4 2 8)

- **First Pass:**
  (**5 1** 4 2 8) (**1 5** 4 2 8), Swap since 5 > 1
  (1 **5 4** 2 8) (1 **4 5** 2 8), Swap since 5 > 4
  (1 4 **5 2** 8) (1 4 **2 5** 8), Swap since 5 > 2
  (1 4 2 **5 8**) (1 4 2 **5 8**), No Swap on since 5 <= 8
  Second Pass:
  (**1 4** 2 5 8) (**1 4** 2 5 8) No Swap on since 1 <= 4
  (1 **4 2** 5 8) (1 **2 4** 5 8), Swap since 4 > 2
  (1 2 **4 5** 8) (1 2 **4 5** 8) No Swap on since 4 <= 5
  (1 2 4 **5 8**) (1 2 4 **5 8**) No Swap on since 5 <= 8
  Third Pass:
  (**1 2** 4 5 8) (**1 2** 4 5 8) No Swap on since 1 <= 2
  (1 **2 4** 5 8) (1 **2 4** 5 8) No Swap on since 2 <= 4
  (1 2 **4 5** 8) (1 2 **4 5** 8) No Swap on since 4 <= 5
  (1 2 4 **5 8**) (1 2 4 **5 8**) No Swap on since 5 <= 8

Insertion Sort – Step by Step (for non-descending order)

create an empty array where you can put the sorted elements
make 1 pass over the original array
  for each element in the original array
  (this means indexes between 0 and (size -1))
  insert that element into the correct position in the new array
    find the first element in the new array that is larger than the one you are inserting
    - if there are none larger just insert it at the end of the used portion in the new array
    - if there are one or more larger – they all slide down to make room then you put the insertion element into that position you found

Insertion Sort Example Inserting 5,7,0,3,4,2,6,1

- **5** *0 0 0 0 0 0 0* *(it is in the correct place)*
- 5 **7** *0 0 0 0 0 0* *(inserts after because greatest)*
- **0** 5 7 *0 0 0 0 0* *(everyone slides down)*
- 0 **3** 5 7 *0 0 0 0* *(5 and 7 slide down)*
- 0 3 **4** 5 7 *0 0 0* *(5 and 7 slide down)*
- 0 **2** 3 4 5 7 *0 0* *(3, 5 and 7 slide down)*
- 0 2 3 4 5 **6** 7 *0* *(7 slides down)*
- 0 **1** 2 3 4 5 6 7 *(2,3,4,5,6 and 7 slide down)*

Selection Sort – Step by Step (For non-descending order)

Make size - 1 passes over the array
(call these pass 0, 1, 2, … size-1)
  On pass n, consider the portion of the array that is between the n position and the end of the array (on this pass you will get the n^{th} element into his correct spot)
    1) Find the smallest element in that remaining portion of the array
    2) Swap that value into the n^{th} position

Selection Sort Example

64 25 12 22 11 (original unsorted list)

11 25 12 22 64 (find that 11 is the smallest and swap it with the 64 so the 11 can be in the 0th place)

11 **12** 25 22 64 (find that 12 is the smallest in the remainder of the array and swap it (with 25) into the 1st place)

11 12 **22** 25 64 (find that 22 is the smallest in the remainder of the array and swap it (with 25) into the 2nd place)

11 12 22 **25** 64 (find that 25 is the smallest in the remainder of the array and swap it (with 25) into the 3rd place)