16.216: ECE Application Programming

Fall 2012

Lecture 21 & 22: Key Questions
October 26, 2012

1. (Review) Explain the relationship between pointers and arrays, as well as the use of pointer arithmetic.

2. Explain how 2-D arrays are passed to functions.

3. **Example:** Say we have a program that stores student exam scores in a 2-D array:
   - Each row represents an individual student
   - Each column represents one of the 3 exams

Write functions to:

- Calculate the exam average for each student and store it in a 1-D array that is accessible in the main program
  - Assume all exams have equal weight
- Calculate the average for each exam and store it in a 1-D array that is accessible in the main program
- Each function takes the same arguments:
  - The 2-D array
  - The # of students in the class
  - The 1-D array that will be used to hold the averages

3 (cont.) Extra space to write functions

4. Describe how character arrays can be used to represent strings in C.

5. Describe the C functions used for:
a. Copying one string to another

b. Comparing two strings

c. Checking the number of characters in a string

d. Concatenating two strings together

6. **Example:** What does the following program print?

```
int main() {
      char s1[15];
      int n1;
      char s2[10] = ".216";
      int n;

      strncpy(s1, "16", 15);
      n1 = strlen(s1);
      printf("s1 = %s\n", s1);
      printf("Length of s1 = %d\n\n", n1);

      printf("%c\n\n", s1[1]);

      strncat(s1,s2,10);
      n1 = strlen(s1);
      printf("s1 = %s\n", s1);
      printf("Length of s1 = %d\n\n", n1);

      // Assume user inputs: ABC ABD
      printf("Enter two strings:");
      scanf("%s%s", s1, s2);
      n = strncmp(s1, s2, 15);
      if (n > 0)
            printf("%s > %s\n", s1, s2);
      else if (n < 0)
            printf("%s < %s\n", s1, s2);
      else
            printf("%s == %s\n", s1, s2);
      return 0;
}
```

7. **Example**: Write a function for each of the following:

a. `int readStrings(char *s);`

Repeatedly read strings from standard input until the input string matches `s`. Return the number of strings read.

b. `void copyNull(char *s1, char *s2, int n);`

Copy the first *n* characters of `s2` into `s1`, and make sure that the new version of `s1` terminates with a null character.

c. `int fillString(char *s);`

Repeatedly read strings from standard input and concatenate them to `s` until there is no room in the string. Return the final length of the string.

For example, if `s` is a 6-character array already holding “abcd”:

- User enters “e”—string is full; return 5
- User enters “ef”—there’s not enough room; return 4