Computer Science 381
Programming Unix in C
The College of Saint Rose
Winter Immersion 2014

Lab 5: More Pointers and Arrays

Recommended Due Date: Wednesday, January 8, 2014

In this lab, you will gain more experience using arrays and pointers in C.

Reference solutions to all programs are available on mogul in `/home/cs381/labs/pointers2`.

Arrays, Arrays, Arrays

Have a look at this example, which demonstrates some of the ways we can declare, construct, initialize, and otherwise use arrays in C.

See Example:
`/home/cs381/examples/arrays`

The comments in this program describe its usage of the most important C features. Pay special attention to the usage of `malloc` to allocate chunks of memory and `free` to return them to the system when finished. It also shows the use of `realloc` to change the size of an allocated chunk of memory.

This demonstrates one of the key differences between C and Java: we have to tell C when we are finished with our allocated memory. Java uses *garbage collection* to reclaim memory no longer in use automatically. We will consider the merits of both approaches later in the semester; for now we simply need to remember that any memory we allocate in C must be released when we are done with it. Advice: when you add a `malloc()`, immediately add the corresponding `free()` in an appropriate place.

? Lab Question 1:

Draw a series of memory diagrams showing the state of the stack and heap variables in existence and their values (indicate unintialized values with a "?") at the execution points labeled with comments as "EXECUTION POINT"s A, B, C, and D. (8 points)

? Lab Question 2:

Explain the purpose of each component of the `malloc` statement on line 58 of `arrays.c`:

```
b = (double *)malloc(10*sizeof(double));
```

(2 points)

Lab Question 3:

Explain how the "`while (*f)`" loop that starts on line 112 of `arrays.c` works. What would happen if the first element of the array was a 0? What would happen if there was no 0 value in the array at all? (4 points)

Practice Program:

Extend your input adder program to remember all the values read in by storing them in a dynamically-allocated array. Name your program `inputsortadder.c`. At the end, the program should print the addition problem and its solution, first with the numbers in the same order as entered, then sorted in increasing order (you might use the `isort` function from last time). For example, given the input values 9, 4, and 5, the program would print

```
9+4+5=18
4+5+9=18
```

Your task here is to be able to create the array and continue to make it larger as needed. Start with a reasonably-sized array (perhaps 10 slots). If the user inputs 10 or fewer values, you are all set. But if an 11th value is input, you will need to resize the array to make space. Use the same strategy as Java's `Vector` or `ArrayList` class – double the size of the array each time it fills. (10 points)

Strings

We have seen that strings in C are simply represented as `NULL`-terminated arrays of `char`. The following example demonstrates more about this and gives examples of some of C's string processing functions.

See Example:
`/home/cs381/examples/strings`

Also read the man page for `string(3)`. It lists the string processing functions available in the standard C library.

Practice Program:

K&R Exercise 5-4. Call your program `strend.c` and include a `main` function that tests your program by passing the first two command-line parameters to `main` as the parameters `s` and `t` to your function. (6 points)

Pointers to Arrays and Multidimensional Arrays

Read Sections 5.6-5.10 of K&R.

Submission

Please submit all required files as email attachments to *terescoj@strose.edu*. You are recommended

to do so by Wednesday, January 8, 2014. Be sure to check that you have used the correct file names and that your submission matches all of the submission guidelines listed on the course home page.