Computer Science 381
Programming Unix in C
The College of Saint Rose
Fall 2013

Lab 3: Input/Output in C

Due: 11:59 PM, Thursday, September 19, 2013

In this week's lab, you will work through some examples focused on how C handles input and output, including to and from files.

Before you start, create a directory on a Unix or Mac system to contain your work for this lab.

You may do your work on any C-capable Unix-like system (such as a lab Mac or mogul).

Readings from K&R

Skim through Chapters 2 and 3 of K&R. Most of what is there is identical to or at least very similar to what you likely know well from other programming languages you know. Much of what we will look at this week is in Chapter 7, though that chapter assumes you have seen some of the material in Chapters 4-6. Even so, it is useful to be able to use higher-level input and output right away, so we will consider much of the content of that chapter this week as well.

Command-line Parameters

You have likely seen Java applications that take command-line parameters (the `String args[]` parameter to the `main` method of a class). A C program that wishes to make use of command-line parameters must declare two parameters to the `main` function, traditionally named `argc` and `argv`.

The parameter `argc` to the `main` function is a count of how many command-line strings are included in `argv`, which is an array of strings.

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

Note: `argv[0]` is not the first parameter, it is the program name itself, and this array entry for the program name is included in the value of `argc`.

Even when we enter numbers for command-line parameters, the operating system will provide them to your program as strings. So we need to be able to convert strings to a numeric equivalent.

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

This is done with the overly complicated `strtol` function, which we use, then check error conditions. There's a lot here we have not yet seen.

- The man page for `strtol` tells us we need to include two additional header files, `stdlib.h` and `limits.h`.
- It also tells us about the parameters to `strtol`, which are the string which we would like to convert to a number, a pointer into the string at the point beyond which we matched a number (which we don't care about, so we pass in `NULL`), and the base to use for the conversion. We also see that the number is the return value.
- Error checking for `strtol` is messy – we need to check the variable `errno`, defined in `errno.h`, to see if an error condition was encountered. If so, `errno` will be a non-zero value and we print an error message and exit.
- We use `fprintf` instead of `printf` when printing the error message. This is because we want to give this output special significance. Rather than sending it to the *standard output*, which is what `printf` would do, we send it to *standard error*, by using `fprintf` and specifying `stderr` as the first parameter. Java supports the same idea: use `System.err` instead of `System.out`.
- Other than that, it works just like `printf`. We give it a format string. In this case, it includes one specifier, a `%s`, which means to expect an additional parameter which is a character string (well, really a pointer to a `null`-terminated array of `char`). Here, the string is `argv[0]`, the first command-line parameter, which is always the name of the program. This labels the error message with the program name.
- Once we have detected the error, we don't want to continue, so we call the `exit` function with an error code of 1 to terminate execution. We could also use the call `return 1;`.
- Note that the error check here has two `%s`'s, so we have two additional parameters to `fprintf`, both pointers to strings.

Practice Program:
Write a C program that takes an arbitrary number of command-line parameters, each of which should represent an integer value. Print out the sum of the values provided. Call your C program `argadder.c` (5 points)

Formatted Keyboard Input

We have seen how to use the `getchar` function to get input from the keyboard or redirected from a file, one character at a time. But often, we'd like to read input as words or numbers.

C's standard mechanism for this is the `scanf` function.

See Example:
`/home/cs381/examples/scanf-example`

- `scanf` is a very strange thing. It will make a bit more sense once you have more experience with the `printf` function, but for now we can summarize what we see there as "read in an integer value (represented by the `%d` in the *format string*), and put it into the place pointed at by the address of `x`, then return the number of values that matched the input with the correct format." Similarly for the `double` value using a `%lf` in the format string.

- The `scanf` call forces us to think a bit about *pointers*, which are the key to understanding so much of how C works. `scanf`'s parameters after the format string are always a list of pointers to a place in memory where there is room to put the values being read in. In this case, we want the `int` value to end up in the local variable `x`, so we have to take the address of the variable with the & operator. Don't worry, it will make better sense when you see more examples.
- The `scanf` to read in a string is a special case and does not require the & operator. This is because the name of a C string already is a pointer to the first element in the array. Again, much more on this when we study C pointers in more detail.
- Next, we check to make sure that the input to `scanf` did, in fact, represent an `int` value. If not, we print an error message and exit. Otherwise, we continue.

Practice Program:
Write a program `inputadder.c` that takes its input from the keyboard rather than from the command line. Your program should read in integer values one by one, accumulating a sum as you go, until you encounter an invalid (non-integer) or the end of the input (someone types `Ctrl-d`). At that point, print out the sum and exit. (5 points)

File I/O

Read Section 7.5 of K&R to learn about reading from and writing to files in C.

Practice Program:
Write a program `everynth.c` that takes 3 command-line parameters: the names of two files and a number we'll call n. The program should take read a series of integers from the first file, and write every n^{th} number to the second file. (10 points)

Note: for this program, you can decide when to stop processing input numbers when an `fscanf` call returns a value other than 1, indicating that it could not find another number.

Programming Assignment

Note: the programming assignment portion will be due as part of the next lab submission, but you are strongly encouraged to get as much of it done this week as possible.

There is collection of graph data files that represent various highway systems at `http://courses.teresco.org/chm/graphs.html`. Each of the ".gra" files is in the format described under "Graph Data" at `http://courses.teresco.org/chm/`.

Write a program `extremes.c` that reads the "waypoints" portion of a graph file whose file name is specified as a command-line parameter and find the easternmost, westernmost, northernmost, and southernmost waypoints in the file. For each, print out the waypoint label and its latitude and longitude.

Notes:

- You can safely ignore the "road segments" portion of the input file for this assignment.
- You can safely assume that no waypoint label is longer than 256 characters.
- By including the `\n` in your `fscanf` format string, you can automatically move the input to the next line.
- You can copy one C string to another using the `strcpy` function, defined in `string.h`. If you have strings declared as

  ```
  char str1[256];
  char str2[256];
  ```

  the following would copy the contents of `str1` to `str2`:

  ```
  strcpy(str2, str1);
  ```

- Valid values for latitudes range from -90 (southernmost) to 90 (northernmost) and for longitudes range from -180 (westernmost) to 180 (easternmost).
- If you were doing this in an object-oriented language like Java or C++, you might create an object to represent each waypoint and provide methods to compare them, print them, *etc.*. But here, you can just use three variables, a string (array of `char`) and two doubles to represent the current waypoint you are considering, and extra sets of strings and two doubles to keep track of each "most extreme" in some direction waypoint you've encountered so far.

This program is worth 30 points, broken down as shown at the end of this document.

Executables for reference solutions to all programs are available on mogul in `/home/cs381/labs/io`.

Submission

Please submit all required files as email attachments to *terescoj@strose.edu* by 11:59 PM, Thursday, September 19, 2013. 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.

Grading

Grading Breakdown	
Practice program `argadder.c` correctness	5 points
Practice program `inputadder.c` correctness	5 points
Practice program `everynth.c` correctness	10 points
`extremes.c` correctness	20 points
`extremes.c` design	3 point
`extremes.c` documentation	3 points
`extremes.c` style	3 point
`extremes.c` efficiency	1 point