Laboratory: Processing the Command Line in C

Summary:

Prerequisites: Familiarity with basic C (arrays, main, compiling, etc.).

- Preparation
- Exercises
  - Exercise 1: Counting Arguments
  - Exercise 2: Argument 0
  - Exercise 3: Using The Command Name
  - Exercise 4: Argument 1
  - Exercise 5: Squaring the Argument
  - Exercise 6: Printing All Arguments
  - Exercise 7: Adding Up the Command Line
- For Those With Extra Time
  - Extra 1: q2l, Revisited
  - Extra 2: q2l Re-revisited
  - Extra 3: Computing from the Command Line

Preparation

- a. Log in to your MathLAN workstation. (Of course, you’ve probably already done that if you’re reading this laboratory.)
- b. Open a terminal window into which you can type commands.
- c. Create a directory for this lab, such as ~/CSC161/CommandLineLab/.
- d. In that directory, create the standard Makefile.

Exercises

Exercise 1: Counting Arguments

- a. Write a program that prints out the number of arguments it receives. That is, you just need to print argc and then exit. Call your program argc.
- b. Determine what happens for each of the following calls.
  - ./argc
  - ./argc 5
  - ./argc a b c
Exercise 2: Argument 0

As you may have just noticed, programs always seem to have at least one argument. Let’s figure out what that argument is.

a. Write a program, arg0 that prints out the value of argv[0] (a string).

b. Determine what output you get for each of the following.

- ./arg0
- ./arg0 5
- ./arg0 a b c
- ./arg0 "a b c"

c. Copy arg0 to argzero using the cp command in the shell. What do you expect to have happen when you repeat the previous commands, substituting argzero for arg0?

d. Check your answer experimentally.

e. Make a soft link from arg0 to argh0.

   ln -s arg0 argh0

Then try running argh0 with a few different arguments.

Exercise 3: Using The Command Name

Consider the following program.

```c
#include <stdio.h>
#include <string.h>

/**
 * hi.c - A not-so-simple "Hello world!" program.
 *
 #include <stdio.h>
 #include <string.h>

 /**
 * hi.c - A not-so-simple "Hello world!" program.
 */
 int main (int argc, char *argv[])
 {
 if (strcmp (argv[0], "./surprise") == 0)
  {
   printf ("*** SURPRISE ***\n");
  } // if
 else
  {
```
printf ("Hi, my name is %s.\n", argv[0]);
}
return 0;
} // main

a. Save the file as hi.c and compile it so that the executable is named hi.

b. What do you expect to have happen when you run it with the following command? (Note the strcmp returns 0 if given two identical strings; you can think of the 0 as representing “no difference”.)

./hi

c. Check your answer experimentally.

d. Link hi to hello using the following command.

ln -s hi hello

e. What do you expect to have happen when you run the following command?

./hello

f. Check your answer experimentally.

g. Link hi to surprise using the following command.

ln -s hi surprise

h. What do you expect to have happen when you run the following command?

./surprise

i. Check your answer experimentally.

Exercise 4: Argument 1

a. Write a program, arg1 that prints out the element with index 1 from the argv array. (That element is a string.)

b. What do you expect that program to print out for each of the following commands?

- ./arg1 hello
- ./arg1 a b c
- ./arg1 "a b c"
- ./arg1 ‘ls’
- ./arg1

c. Check your answer experimentally.
**Exercise 5: Squaring the Argument**

a. Write a program, `square` that takes one number on the command line and prints out the square of that number.

You may find it useful to use the `atoi` procedure which you can access when you include `stdlib.h`.

b. Determine what happens when you provide `square` with no arguments.

c. Determine what happens when you provide `square` with something other than a number as the first argument.

**Exercise 6: Printing All Arguments**

a. Write a program, `args`, that prints out all of the arguments, with each argument preceded by the argument number.

b. Predict the output for each of the following commands.

- `./args`
- `./args a b c`
- `./args "a b c"`
- `./args 'ls'`
- `./args a*`

c. Check your predictions experimentally.

**Exercise 7: Adding Up the Command Line**

Write a program, `sum`, that assumes all of its command-line arguments are integers and computes their sum.

Your program should report an error message if it receives no command-line arguments.

**For Those With Extra Time**

**Extra 1: q2l, Revisited**

Rewrite `q2l` from the the I/O lab so that the program checks for a value on the command line.

- If there’s a value on the command line, it should assume that it’s a number of quarts and convert it to a corresponding number of liters.
- If there’s no value on the command line, it should prompt the user for a number of quarts and convert to the corresponding number of liters.
Extra 2: q2l Re-revisited

Add error checking (including error checking for the command line) to the revised q2l.

Extra 3: Computing from the Command Line

Write a program, compute, that takes as its first argument an operation (initially, * or +) and as its remaining arguments a series of numbers, and that computes the specified operation on the numbers.

% ./compute * 3 4 5
60
% ./compute + 3 4 5
12
% ./compute * 5
5
% ./compute + 5
5

Copyright © 2010 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.