CS 240 Summer 2017

Lab 4: Parsing bits and strings, static libraries, and system() (200 pts)

Due: 07/12/2017 (Wed), 11:59 PM

Objective

The objective of this lab is to practice parsing input viewed as bits and strings. We will also employ system utilities commonly used with C programs such as archives for static linking and system() to execute legacy binaries from within an app.

Lab 4 Code Base

The C code base for lab4 is available as a tarball

/u/data/u3/park/pub/cs240/lab4.tar

on our lab machines.

Problems [200 pts]

Problem 1 (70 pts)

This is an extension of Problem 4, Lab 3. Code an additional function with function prototype

```
void calc_diff(int, float *, float *, float *);
```

that takes four arguments. The first argument specifies vector size, the second and third argument the two input vectors, and the fourth argument a pointer to a 1-D array, float u[MAXSIZE], which stores the result of subtracting the vector x[] from y[]. read_vectors() should be reused as is and u[] should be printed to stdout from main().

An important difference is the system aspect of the implementation where you are asked to create a library of object files using the archiving utility ar. Your main.c compiled separately into main.o using -c will then statically link with your library, call it libmymath.a

```
gcc -o mymain main.o -static -L <pathname-of-lib-directory> -l mymath
```

where <pathname-of-lib-directory> specifies the pathname of the directory where the archive libmymath.a is located. The archive should contain three object files: read_vectors.o, calc_dotmag.o, calc_diff.o. To create the initial archive run

```
ar rcs libmymath.a read_vectors.o
```

To check the content of the archive run

```
ar t libmymath.a
```

Add further object files to the archive (do it one by one) by running

```
ar rcs libmymath.a calc_dotmag.o
```

and similarly for calc_diff.o. The option r inserts an object file into the archive, replacing a previous version if one exists. The modifier c creates an archive if it doesn't exist. The s modifier maintains an object file symbol index. After performing updates manually, modify the Makefile submitted in lab3 so that archive entries are automatically updated whenever object files are updated. Recall that actions, such as running gcc, after specifying a dependency (which are updated based on time stamps as noted in class), are added by inserting a tab followed by an action (in our case, ar with arguments). Test that the Makefile works correctly by selectively deleting object files and/or writing to .c files. Deposit the source code, object code, object code archive, and Makefile in v20/.

Problem 2 (50 pts)

Write a program, main() in main.c, that reads an integer of type unsigned int from stdin and inspects its bits by using the shift operator >> discussed in class. Inspect the four least significant bits of the int variable by performing an & with a suitable mask that blocks out/ignores all other bits. Output the 4-bit value using printf() in decimal and hexadecimal formats. Inspect the remaining 28 bits by right shifting the unsigned int variable using the shift operator >> four bits at a time in a loop, masking as before, and printing their 4-bit values in decimal and hexadecimal formats. Verify that the integer input 10 is represented as 000...01010. What is the largest value that you can store in unsigned int? Use your code to test that it stores and outputs the bits of the largest value correctly. Submit the code in v21/.

Problem 3 (80 pts)

Part A: Parsing

As in Problem 2 of lab3, read a string from stdin using getchar() terminated by '\n'. The input should be a URL of protocol type HTTP, for example,

```
http://www.cs.purdue.edu/homes/park/cs240/index.html
```

Note that upper and lower case are both allowed and treated as equivalent for the protocol (HTTP) and domain name (www.cs.purdue.edu) components which include subdomains and subsubdomains. For example,

hTTp://www.CS.Purdue.EDU/homes/park/cs240/index.html

is syntactically valid. IP addresses can be used in place of domain name and port numbers are allowed. That is,

http://128.10.19.20/homes/park/cs240/lab4/lab4.html

and

HTTP://WWW.cs.Purdue.edu:443/homes/park/cs240/index.html

are both valid.

Write your own string processing code such that the domain name or IP address (in dotted decimal format) is extracted from the URL. In the above examples, WWW.cs.Purdue.edu or 128.10.19.20 are extracted from the URL input. For our purposes, assume that domains must be of type .gov and .org. If the URL is ill-formatted, then indicate so by printing an error message. For example,

htp://www.CS.purdue.EDU/homes/park/cs240/index.html

is ill-formatted ("htp") and so is

http://www.CS.purdue.EDU/homes/park/cs240/index.html

since .edu is not allowed. In the case of a domain name, convert all upper case letters in the protocol and domain name (if any) to lower case. Output to stdout the converted domain name using printf(). In the case of an IP address, output its 32 bit values (use a slight modification of the bit inspection code in Problem 2).

Part B: Finding IP address or domain name using legacy app

In the case of domain name input, after extracting, converting, and printing the domain name (a string), find its IP address using an existing (i.e., legacy) app. In our case, the executable binary *host*. If you execute

% host www.in.gov

in your shell you will find that host outputs to stdout the IP address 208.40.244.65. In the case of IP address input, the reverse holds. That is, if you run

% host 208.40.244.65

the app will perform a reverse look-up and output www.in.gov. There are other ways of converting IP address and domain names but we will use a method that makes use of an existing app. To run an existing binary from within a C program, we will use the function system() which takes a string that is the binary to be executed as input. For example, if your C code include the following statement at the end of main()

system("host www.in.gov");

then the same output to stdout would be produced. Since the argument of host is not fixed but extracted from the user supplied input URL at run-time, you must create a string that concatenates "host" with the domain name or IP address string extracted. The two strings must be separated by a space and terminate with '\0'. Include the header file stdlib.h. Implement, compile, and test your code with five URLs and verify that it works as specified in the problem. Write your own string processing code, i.e., do not use string processing library functions. Deposit your code as main.c in v22/. Put each function you write in its own .c file and use Makefile to manage compilation.

Bonus Problem (20 pts)

Look into how domain names can be translated to IP addresses (in dotted decimal form) without using system(). Modify the code of Problem so that it does not use system(). Submit the code in v23/.

Turn-in Instructions

Electronic turn-in instructions:

i) For code submissions, follow the directions specified in the problems.

ii) Use turnin to manage lab assignment submissions. Rename lab4 to mylab4. Go to the parent directory of your mylab4 directory and type the command

turnin -v -c cs240 -p lab4 mylab4

Please note the assignment submission policy specified in the course home page.

Back to the CS 240 web page