**ECE 1111: Engineering Computation I**

**Laboratory No. 6: Basic C Syntax, Atomic Types and Command Line Programming**

**Goal:** Demonstrate that you can create a simple C program, save its output to a file, and postprocess that file using Unix commands. We also want you to feel comfortable using Google search to find answers to your questions. Some of these exercises in this lab are not extensively explained. You will need to use Google search to learn more.

**Deliverables:**

1. *Source Code:* Do all your work in this directory:

/data/courses/ece\_1111/current/labs/lab\_06/<lastname\_firstname>

You must use a make file and comment your code, following the guidelines and templates provided in class. Create directories p01 through p04 for the corresponding parts below.

1. *Check Off:* You will arrange a meeting with your TA during the lab session. The TA will ask you to make a small modification to your print statement, compile and run your program, and manipulate the output file using command line tools.

**Description:**

1. (p01) Simple Print Statements and Redirecting Standard Output (stdout)

We have previously seen how a simple C program to print a message to the terminal can be implemented using the following commands:

#include <stdio.h>

int main(int argc, char\*\* argv) {

fprintf(stdout, "hello world\n");

}

Modify this code to declare a floating-point variable named “sum”, set its value to argv[1], and print the value and the program name to stdout. Your output should follow this format:

program name: <insert argv[0] here> sum = <insert argv[1] here>

The program name resides in the variable argv[0], and a text string corresponding to the value of sum is held in argv[1]. Later in the course we will explain what these are. For now, review the class lectures to see how you can operate on these quantities.

Your program name should be p01.exe. You should also have a makefile named Makefile in your p01 directory.

Create a directory called “*tmp*” using the *mkdir* command. Store the output of your program in this directory by redirecting a program to a file:

p01.exe 27.272727 > ./tmp/p01\_1.txt

As we discussed in class, the “>” sign instructs the program to direct output to the corresponding file. The “~” character denotes your home directory. Re-run this command and send the output to your home directory:

p01.exe 27.272727 > ~/p01\_2.txt

p01.exe 27.272727 > $HOME/p01\_3.txt

You now have 3 versions of the same file in the directory tree that starts with your home directory. Let’s demonstrate how we can use find to locate and manipulate these files.

Run find to locate these files:

cd

find . -name "\*.txt"

The first command returns you to your home directory. The second command searches all the files in the directory tree that starts with your home directory. How many files are returned?

1. (p02) Postprocessing with Command Line Tools

Filter this output using grep:

find . -name "\*.txt" | grep prog

How many files are returned? Why?

find . -name "\*.txt" | grep prog\_2 | wc

What did the *wc* command do?

Execute the following command and explain what it does:

find . -name "\*.txt" -exec ls -l {} \;

Is there another way to implement the functionality provided by this command?

Using what you have learned, in the following directory tree:

/data/courses/ece\_1111/resources/data

count the number of times the symbol “bckg” occurs in any file ending in “\*.csv” and containing the characters “a”, “b” and “c” in its filename. Produce a sorted list of each file that contains at least one of these symbols. A file should only appear once in this list.

1. (p03) Variables and Precision

In this lab, you will need to use a loop. Though we haven't covered this yet, you can easily read about these and follow the templates below. There are two types of loops you might need:

for loop:

for (long i = 0; i < 128; i++) {

fprintf(stdout, "the value of i is: %d\n", i);

}

while loop:

bool status = true;

long i = 0;

while (status) {

fprintf(stdout, "the value of i is: %d\n", i);

i++;

}

The first loop, which is a for loop, simply iterates from 0 to 127. The second loop runs as long as the Boolean variable, status, is true. Ideally, you would set this variable to false inside the while statement when some condition, such as end of file, failed.

Write a program that prints out an ASCII table similar to this one: [*http://www.asciitable.com/*](http://www.asciitable.com/). Your output should look as similar to this as possible:

----------------------------

| Dec | Hex | Oct | Char |

| 000 | 000 | 000 | (null) |

...

| 065 | 041 | 101 | A |

| 066 | 042 | 102 | B |

| 127 | 07F | 177 | DEL |

----------------------------

Note that "..." is not part of the requirement. It simply indicates I am skipping entries in the table. Your program should print all 256 values, if they are printable.

**You can skip the unprintable character such as “NUL” and “ACK”. Just print “N/A” for these unprintable entries. However, you must understand what these unprintable characters represent and why they exist.**

1. (p04) Implementing the Command “cat”

Write a program that functions similar to the Unix command "cat":

cat filename.txt

Your program should process only one file, but I should be able to specify that file from the command line (e.g., "filename.txt" can be any valid filename). Your program should print the contents of the file to stdout line by line as cat does.

Note that you can easily find code for this on the Internet or in several textbooks. Read the file character by character for now. Later we will relax that requirement.

**Summary:**

One of the attractive features of the C programming language is that you can easily manipulate low-level software such as bit and byte manipulations. Virtually everything in the Unix operating system is written in C. This was one of the major innovations of Unix – the operating system was written in what was considered a high-level language rather than assembly code.

Simple programs such as “cat” are very useful. Learning how to loop over a file until you reach the end of file is a very common way that we write flexible code. Since you don't know how many lines are in a file, you have to loop until there is no more data to read.

Most of the programs we write will process command line arguments. This is another way we make our programs flexible. You should be able to process any file the user specifies from the command line, and you should produce informative error messages when the file doesn't exist.

In this lab, you have learned some basic C syntax, such as how to print text messages and how to create a variable. You have reinforced your ability to create, compile and run programs.

Combining simple Unix commands into more powerful commands using pipes is one way we can create complex programs very quickly. The command line is an important part of the tools you use to be productive on a computer. In this lab, you have learned how to redirect the output of your program to a file, and how to postprocess those files using command line tools.