CSE2421 Lab 1 (3 points)

Getting to know Linux, your choice of editor, Makefile, and GDB

Please read 2_Unix_Linux_Debugging.pdf before you start

Step 1: Log into stdlinux.cse.ohio-state.edu

1.1 If you plan to use CLI, you can use ssh command on your local Linux/Mac machine; you can install putty if your local machine uses Windows. If you plan to use GUI, you need to install FastX and follow its instructions.

1.2 Your username and password should be the same as your OSU account. If you are CSE student, you should already have access to stdlinux. If you do not have access, please email help@cse.ohio-state.edu

Step 2: get familiar with Linux commands

2.1 Try the Linux commands listed in the slides. If you don't know how to use one command, use "man <command>" to find the manual, or simply google it

2.2 Finish the following task: under your home directory ("cd ~" will take you to home directory), create a directory called "cse2421"; then create a directory "lab1" under cse2421

Step 3: choose and get familiar with your editor

3.1 Since this step highly depends on the editor you choose, we will not provide detailed instructions here. Eventually you should put a text file with any content you like under the lab1 directory you just created in Step 2.

3.2 Go to the lab1 directory and use the "cat" command to show the content of the file. Check whether it is as expected.

Step 4: edit some c and h files

You should create the following three files with your editor: lab1.c, lab1_func.c, and local_file.h. Note do not copy past code from this document: it is likely to cause many problems.

lab1.c

```
#include <stdio.h> /* this tells the preprocessor to copy IO library prototypes
                              and other information from the file /usr/include/stdio.h */
#include "local_file.h" /* this include statement references a file in the current directory */

int main(void)
{
        unsigned int maxEntries;
        int getchar_return_value; /* note manual page says getchar() returns an integer value */

        printf("This program reads in a number, then a series of keyboard characters. The number\n");
        printf("indicates how many characters follows. The number can be no higher than %d.\n", MAX_NUM);
        printf("Then the specified number of characters follow. These characters can be any\n");
        printf("key on a regular keyboard.\n");

        /* Read the first number entered to know how many entries will follow */
        /* We are going to assume that the number will be 1 or 2 digits in length */
        /* and it will be followed by a newline,'\n', character */
        printf("Please enter the number of entries, followed by the enter/return key: ");
        getchar_return_value = getchar(); /* read the first ASCII character of our max number */
        /* if it's not a newline, convert to non-ASCII value, we can assume it is a number */
        if (getchar_return_value != '\n'){
                maxEntries = getchar_return_value - '0';
                getchar_return_value = getchar(); /* potentially read the second character of our number */
                /* if it's not a newline, convert to non-ASCII value, we can assume it is a number */
                if (getchar_return_value != '\n') {
                        maxEntries = maxEntries*10 + getchar_return_value - '0';
                        getchar(); /* trash the '\n' */
                }
        }
        else maxEntries=MAX_NUM; /* if first ASCII character on the line is a '\n', then assume max */

        if (maxEntries > MAX_NUM) {
                printf("Specified number of characters is invalid. It must be between 1 and %d.\n", MAX_NUM);
                return(0);
        }

  #ifdef DEBUG
        printf("entering function\n");
  #endif
        print_chars(maxEntries);
  #ifdef DEBUG
        printf("returned from function\n");
  #endif
        return(0);
}
```

```
lab1_func.c
#include <stdio.h>      /* this tells the preprocessor to copy IO library prototypes
                        and other information from the file /usr/include/stdio.h */
#include "local_file.h" /* this is an include file located in the current directory */
int getchar_return_value;
void print_chars(unsigned int maxEntries){
        int i;
        printf("enter the %u characters: ", maxEntries);
        for(i = 0; i < maxEntries; i++){
                getchar_return_value = getchar();
                if(i==0) printf("The keyboard values are: \n");
                if (getchar_return_value != EOF){ /* we got a valid value */
                        putchar(getchar_return_value); /* print it back out */
                        putchar('\n');
                }
                else{
                        printf("fewer than %u characters entered, number of characters set to %d\n", maxEntries, i);
                        maxEntries = i;
                        break;
                }
        }
        return;
}
```

```
local_file.h
#ifndef LOCAL_FILE_H
#define LOCAL_FILE_H
#define MAX_NUM 25
void print_chars(unsigned int max);
#endif
```

Some notes:

1. Quotes are double quotes, not two single quotes
2. Indentation: you could use tab or spaces; typically 4 spaces for indentation; tab is typically 8 spaces in length, but you can reconfigure it to 4 spaces

Step 5: compile your program

5.1 use the following command “gcc -ansi -pedantic ~~-std=c89~~ –g -o lab1 lab1.c lab1_func.c”; you can google the meaning of each option. If the compilation succeeds, run lab1 to see how it goes
5.2 Write your own Makefile. Again, you are encouraged to write a Makefile with multiple rules, but one rule is fine. The only requirement is that when the grader types “make”, it should generate an executable file called “lab1”

Step 6: try GDB

6.1 Try to use GDB to do the following: 1) set breakpoints when your program calls “getchar” (there are four locations; you need to set breakpoints for three of them; ignore the one without a return value); 2) after your program reaches the breakpoint, use the “next” command to execute the getchar function (breakpoints stop before the corresponding line is executed); 3) use the “print” command to print the values of the return value of getchar (check the ASCII code to see if the values are as expected); 4) let the program continue; 5) repeat steps 2-4 until the program finishes (note: don’t input too many characters)
6.2 Capture a screenshot of all steps above; if it is long, you can take multiple screenshots

Step 7: zip your files and submit on Carmen

7.1 Use the “zip” or “gzip” command to put all files in one .zip file: it should include the .c files, .h files, Makefile, and the screenshots. It does not need to include the executable file. Note: if you use CLI to connect to stdlinux, you can do the screenshots on your local machine; then you can use scp to copy them to stdlinux to do the zip or you can use scp to copy files from stdlinux to your local machine and do the zip on your local machine
7.2 Submit on Carmen: if you use GUI access to stdlinux, you can start a browser there and upload the zip file directly; if you use CLI access to stdlinux, you can use the “scp” command to copy the zip file to your local machine and upload it to Carmen on your local machine
7.3 Verify your submission: download it, unzip it on stdlinux, make, and run it

Grading criteria:

1. We will unzip your zip file, and run “make”. If it reports any errors, you will lose all points
2. We will then run your program. If it reports any errors, you will get a penalty based on the errors
3. Your source code should be properly indented (length should be 4 spaces; you can use either tab or space). Otherwise, you will lose .5 points
4. We will check your screenshots.