Computer Science 352 Summer 2023
Programming Assignment 3
Due 6/23/2023 by 7pm

This PA for CS 352 has multiple parts. The first part will require you to write some bash commands to get you some more practice with using the bash shell. The second part will require you to write a program that is a simplified version of the **cut** unix command. You should ensure that you are following all of the rules from the style guide, which can be found on the class website. For the project, you should end up with the following directory structure:

```
pa3
├── shell
│   └── commands.txt
└── scut
    ├── makefile
    └── scut.c
```

The makefile should have two rules. One named **`scut`** to build the **`scut`** executable, and another named **`clean`** to remove the executable.

Bash Commands

You should write a bash command for each of the descriptions shown in this section, and put each in the file **commands.txt**. You should format it as follows:

```
(1)
first_command
(2)
second_command
```

You may only use the following commands:

`cd find ls mkdir touch cat grep cut uniq wc history ncal xargs cal find sort head tail tr rev`

You may (and should) also use pipes, redirection, and semicolons. If you are unsure how any of these work, use the man pages. Each command should be a one-liner.

1. A bash command to find every file containing the token **`printf`** that ends with either **`.c`** or **`.h`**. This command should search through all files in the current working directory and the entire subdirectory tree.

```
└── test
    ├── model
    │   ├── board.c
    │   └── board.h
    ├── view
    │   ├── menu.cpp
    │   └── menu.hpp
    ├── controller
    │   └── inputs.cpp
    └── main.c
```

   If the CWD is test, and if only the files **`menu.cpp`**, **`board.c`**, and **`main.c`** contain **`printf`**, the command should print:

```
$ your_command
./main.c
./model/board.c
```

   The resulting files should be sorted with the **`sort`** command.

2. A bash command to search through the past 50 commands that a user has run via bash, and search for only ones containing “cd” or “ls”. All other commands should be ignored. For example, say that the user has run a variety of different kinds of commands in the past 50 commands, and only 3 contain those keywords. Those would be:

```
$ ls -lta
$ cd /usr/lib
$ grep "cdna"
```

   In that case, if we ran the command, we should get:

```
$ your_command
cd /usr/lib
grep "cdna"
ls -lta
```

   The commands should be sorted via the sort command.

SCut (Simple Cut)

You should write a C program named **`scut.c`**. This program will be a simplified version of the **cut** command that can be found on many unix systems, including lectura. You should have already used this command by now. I recommend spending a bit of time playing around with the command and reading the man page (**`$ man cut`**) for the regular command before starting this program. The **`cut`** command can be used for extracting particular characters / columns of data from the standard input and print them to standard output. Your program will do the same, but it will have less configuration options compared to the regular command.

This program should expect to have *exactly* two command-line arguments. The first is a flag that specifies what mode to run the program in. You must support three options.

- **`-l`** stands for “letter” and specifies that the columns for the cut selections are on a per-character basis
- **`-w`** stands for “whitespace” and specifies that the cut selection columns will be separated by any whitespace (spaces or tabs)
- **`-c`** stands for “comma” and specifies that the cut selection columns will be separated by commas, such as from a CSV file

The second command-line argument will be the selection sequence. Basically, this will be a combination of digits, separated by commas and/or dashes. These sequences will specify which columns to print out in the final result from the cut command. A few examples:

- **`1,3,5-7`** : This sequence specifies that we want to extract columns 1, 3, 5, 6, and 7.
- **`1-5,7,22,25-29`** : This sequence specifies that we want to extract columns 1, 2, 3, 4, 5, 7, 22, 25, 26, 27, 28, and 29.

The program will get its input from standard in. The input may have multiple lines, so the program should continue reading input until it receives an **EOF** indicating that the program may stop processing input lines and may complete and then exit / return. For the sake of simplicity for reading in the lines, you may assume that every line will be less than 100 characters long. This simplifies the program in two ways. (1) Is that when you read in the lines you may use a char * buffer of length 102 (to account for possible newline / null character). (2) is that when you are processing the second argument, you may assume that a column number will not be wider than two digits (two chars in a char array).

Lets walk through an example of how the program should behave, which will hopefully give you a better idea of how it should be functioning.

Assume that you have written the program correctly, and you compile it using the following command:

```
$ gcc -Wall -Werror -std=c11 -o scut scut.c
```

Let’s also assume that you have a file in this same directory named **`stuff.csv`** and it has the following contents:

```
alice,30,532,AZ,S
bob,25,3411,CA,Z
jonas,40,8192,AZ,T
greg,50,400,UT,C
```

Lets try testing out our scut program. If we select columns 1 and 3 with the -l option, we should get:

```
$ cat stuff.csv | scut -l 1,3
a i
b b
j n
g e
```

If we instead use the -c option for comma separation, we would instead get

```
$ cat stuff.csv | scut -c 1,3
alice 532
bob 3411
jonas 8192
greg 400
```

We could also do a test with **printf** and the -w option:

```
$ printf "the big lazy fox will\ngo to sit in the cave\nover by the small mountain" | scut -w 2,4,5
big fox will
to in the
by small mountain
```

In these first few examples, I did not use the “dash” to specify a range of columns. Supporting both commas and ranges will be trickier than just supporting columns. You may want to focus on getting multi-column support only with commas working first. In fact, there will be some test cases on gradescope that do not use dashes, so even if you do not get dash support working, you can still pass at least some of the tests. After you get that fully working and tested, you can move on to supporting column ranges. After you have added support for this, you should be able to replicate the following two examples:

```
$ cat stuff.csv | scut -c 1,3-5
alice 532 AZ S
bob 3411 CA Z
jonas 8192 AZ T
greg 400 UT C

$ cat stuff.csv | scut -l 1-2,7-15
a l 3 0 , 5 3 2 , A Z
b o , 3 4 1 1 , C A ,
j o 4 0 , 8 1 9 2 , A
g r 0 , 4 0 0 , U T ,
```

You may also assume the column-specification string will always specify columns in-order. For example, **“1,7,2,3-4”** would be invalid and you do not have to support this. If you wanted columns 1, 2, 3, 4, and 7, you could instead do **“1,2,3,4,7”** or **“1-4,7”**.

A few things that you can or cannot assume for this program:

- Your program may assume that no single line of input will exceed 128 characters
    - Your program should not assume anything further about the input lines
    - There could be empty lines, lines with special characters, lines with or without whitespace, etc.
- Your program should not assume any min or max number of input lines
    - Could have 0 input lines, could have 1,000, could have 100,000, etc.
- Your program should check to ensure that there are exactly two command-line arguments. If not, it should print **`"expected 2 command line arguments.\n"`** to standard error and then exit with a non-zero exit code.
- Your program should check to ensure that the first command-line argument is one of the three shown in this spec. If it is not, then it should print **`"Invalid delimiter type.\n"`** to standard error and then exit with a non-zero exit code.
- The selection argument (second argument) must begin with a digit, end with a digit, and in total contain only digits, commas, and dashes. If any of these are not true, it should print **`"Invalid selection.\n"`** to standard error and then exit with a non-zero exit code.

C Standard Library Restriction

You may only use two functions from **`string.h`**: **`strlen`** and **`strcmp`**. Other than these two, you may not use any other functions from **`string.h`**. You may use **`stdlib.h`**, **`stdio.h`**, and **`stdbool.h`**. You may not use any other standard library functionality.

Submitting Your PA

After you have completed PA, double check that the file structure and file / directory names match what is shown in the project overview on the first page. Also, before you submit, you should ensure that your code compiles and runs correctly on lectura and that the code follows the rules from the style guide. Remove all files from the pa3 directory and subdirectories other than the ones shown on the first page of this spec. This includes test files, executable files, and other file types.

Once you are ready to submit, zip up the **`pa3`** directory by running:

```
$ zip -r pa3.zip ./pa3
```

Then, turn this file to the PA 3 dropbox on gradescope. There will be some test cases that will not be visible until after the grades get published. You should ensure you pass the visible ones, and test your program thoroughly.