`CSSE 132` – Introduction to Systems Programming
Rose-Hulman Institute of Technology

Exam 2 Practice - Coding Part

Name:______________________________ Section: 1 2 3

This part of the two-part exam is **open course material**. You may use any of the following resources:

- Your computer
- Your assignments and labs submitted in your individual GIT repository for this term
- The CSSE 132 course website and things directly linked from it

You are not allowed to use other Internet resources, instant messaging, your smartphone, or other communication means during this part of the exam. Use of other resources is considered academic dishonesty and will result in a penalty grade.

To start the coding part:

- ☐ First read and sign the honesty statement below.
- ☐ Pull your Git repository on your Linux. You will be using the files in the `exam2practice` directory. If your repository does not have this directory, ask your instructor for help.

I attest that all of my code for this exam is submitted to my Git repository and I have not received help on this exam from any source other than the acceptable sources listed above.

Your Signature:______________________________ Date:______________

Problem 1 (20 pts) Complete the functions in `part1.c`. Specifications can be found in the comments. Make and run `./test` to check your work.

Problem 2 (20 pts) Create a program called `part2`. This program will read a file and squeeze the consecutive spaces in the file. When run, it takes one input argument as the file name to perform squeezing.

HINT: The new line character `'\n'` *will break consecutive spaces. Namely, consecutive spaces cannot continue across multiple lines in a file.*

Your program must satisfy the following specifications:

- ☐ You must implement this in a new file called `part2.c` in your `exam2practice` directory.
- ☐ Do not include any of your code from `part1.c`. We provide a correct implementation of `part1.c` so you can use any functions from `part1.c`, even if you did not complete them. Add this exact line to the top of your `part2.c` file:
  `#include "support/part1.h"`
- ☐ The provided `Makefile` already has a rule that will make `part2` from `part2.c`. You can type `make part2` to compile it.
- ☐ When executed from the command line with the proper arguments, it will print the squeezed version of the file on the screen. For example (`aloha.txt` is in your repo):

```
pi@my-pi:~$ ./part2 aloha.txt
 Aloha is the Hawaiian word for love, affection, peace, compassion and mercy.
```

- ☐ If the wrong number of arguments are given, print the below error message:

```
pi@my-pi:~$ ./part2
Usage: ./part2 filename
```

- ☐ If the program fails to open the file specified, print below error message including that non-existent filename:

```
pi@my-pi:~$ ./part2 notthere.txt
Error: Could not open file notthere.txt
```

- ☐ You can assume the number of bytes in each line in the file is less than 512 bytes. *(Hint: when using* `fgets` *to iterate lines,* `fgets` *will return* `NULL` *when reaches to the end of file.)*
- ☐ To test your program, you can first manually run the program with certain files, e.g., `./part2 aloha.txt`. If the output looks good to you, you can use the provided test script to do a more thouroughly test by typing `./test_part2.sh`

(Continue on next page)

Problem 3 (10 pts) In this part, you need to create a program called `part3`. This program will interact with user input, i.e., it takes keyboard input and output result accordingly. In this problem, you need to write an **addition calculator**. To simplify the implementation, the calculator only sums two **one-digit** decimal numbers (i.e., 0 to 9). The workflow of the program can be described as below:

1) Once launched, the program will print a message as follows:
   `Type the first number to add:` (followed by a new line character `\n`)
2) Then user types the first one-digit number and hit the enter key
3) Once received the first number, the program will print another message as follows:
   `Type the second number to add:` (followed by a new line character `\n`)
4) Then user types the second one-digit number and hit the enter key
5) Finally, the program will print the result formatted as
   `The result is:  X` (followed by a new line character `\n`)
   Where `X` is the addition result. The program will **exit** after printing the result.

A complete example can be shown as follows

```
$ ./part3
Type the first number to add:
4
Type the second number to add:
9
The result is: 13
$
```

Note that `$` is the command prompt. Yours may appear differently. Your program output should match the format exactly (including the spaces and new lines) to pass the test script.

The detailed implementation requirements are listed blow:

- ☐ You need to create a file called `part3.c` in your `exam2practice` directory.
- ☐ The provided `Makefile` already has a rule that will make `part3` from `part3.c`. You can type `make part3` to compile it.
- ☐ HINT: The program will exit after printing out result. Therefore, *NO loop* is needed. You can always assume the program inputs are legit.
- ☐ To test your program, please type `./test_part3.sh` to run the test script. The script may freeze if your code has infinite loop(s) or never exits.

(Continue on next page)

IMPORTANT*:* When you are finished with this part of the exam:

- ☐ Add any files you created or modified to Git. (`git add part1.c part2.c part3.c`)
- ☐ Commit your solutions to Git. (`git commit -m "finish exam 2 practice"`)
- ☐ Push your repository to the Git server. (`git push`)
- ☐ Double check if the push is successful. Run `git log --stat -1` and make sure you see your latest commit with correct file modification information. Also, double check if the **first line** of the log message includes `origin/master` in the parentheses. If not, try to `git push` again.
- ☐ Ask your instructor to confirm your submission.