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

Homework 6

Name (Print):_____ RHIT Username:_____

Put your answers in the spaces reserved below the questions. **Do not** write your answers on **separate sheets of paper**.

1. (6 points) Consider the code below:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct db_entry {
5    char* name;
6    char* value;
7  };
8
9  void dbe_print(struct db_entry* entry)
10 {
11   fprintf(stdout, "%s =>%s\n", entry->name, entry->value);
12 }
13
14 int main(int argc, char** argv)
15 {
16   struct db_entry* e;
17   e->name = "hocus pocus";
18   e->value = "focus";
19   dbe_print(e);
20   return 0;
21 }
```

This code compiles, but crashes when it is run. Suspecting a problem with `dbe_print`, you comment out the line of code that prints the entry (11), however the program still crashes! You suspect the crash is caused by a memory allocation problem.

(a) What is the reason for the crash?

(b) Without changing existing code, you can solve this problem by inserting additional code to correct the crash and resolve the memory issue.

Indicate what code you would insert and the line numbers immediately before where your code will be inserted. (For example, "insert this code after line 19" or "insert this code just above line 9".)

2. (6 points) Complete the following code to 1) use buffered I/O to open `filename`, and 2) get the file size so that it can be printed out, and then 3) close the file.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv) {
5      FILE* f;
6      int size = 0;
7      char* filename = argv[1];
8
9      // open the file
10
11     f = ____________(______________________);
12     // get the file size and assign to size
13
14     _______(___________________);
15
16     size = ____________________;
17     // close the file
18
19     _______(___________________);
20
21     printf("File size is %d\n", size);
22     return 0;
23 }
```

3. (60 points) On your Linux, pull changes to your Git repository. In the `homework6` directory:

   (a) (34 points) Complete the functions in the `problems.c` file. Test it using the provided tests. Again, type `make` to compile and `./test` to run test cases.

   (b) (12 points) Create a program called `platin`. When the program runs, it should read lines of keyboard input using `fgets` and *pigify* each line, printing out the result. When an empty line is typed (i.e., enter is hit at the prompt), the program should exit.

   Tips: Be sure to remove the newline character from each line by converting it to a null character. You can safely assume there is only one word per line and the word can't be longer than 100 characters.

   In this example, '+' and '-' are used to indicate input and output. They are **NOT** part of what is printed. A + means it was typed, and a - means the program printed it. The last line of input is just the enter key:

```
  usr@linux: ~$ ./platin
+ awesome
- awesomeyay
+ sauce
- aucesay
+ foo
- oofay
+
  usr@linux: ~$
```

   Your program must satisfy the following specifications:

   - ☐ You must implement this in a new file called `platin.c` in your `homework6` directory.
   - ☐ To use your functions from `problems.c`, you can put the following line at the top of your file.

     ```
     #include "problems.c"
     ```

   - ☐ Remember that the input arguments of the `main` function are defined as follows

     ```
     int main(int argc, char** argv){
       ...
     }
     ```

   - ☐ The provided `Makefile` already has a rule that will make `platin` from `platin.c`. You can type `make platin` to compile it.
   - ☐ To test the `platin` program, you can first manually run it with `./platin` and type some words. If the output looks good to you, you can run the provided script `./test_platin.sh` for a thorough test.
     Warning: The test script will freeze, if your program `platin` doesn't quit on an empty line. In this case, press `ctrl + c` to terminate the program.

(c) (14 points) Create a program called `sameLine`. This program will take two input arguments as filenames of two files. When the program runs, it will read the two files and compare their contents line by line. Then it will print out the lines which 1) have the exactly same content in both files and 2) also appear at the same line number in both files. In other words, the program won't pint a line if the line content is the same but appears in different locations in the two files.

Your program must satisfy the following specifications:

- ☐ You must create a new file called `sameLine.c` in your `homework6` directory.
- ☐ The provided `Makefile` already has a rule that will make `sameLine` from `sameLine.c`. You can type `make sameLine` to compile it.
- ☐ You can assume the number of bytes in each line is less than 512 bytes, and every line ends with `'\n'`, including the last line.
- ☐ If the wrong number of arguments are given, print the error message below:

```
usr@linux:~$ ./sameLine
Usage: ./sameLine file1 file2
```

- ☐ If the program fails to open one or both files, print the error message below which includes the name of the failed file (assume only one file can fail)

```
usr@linux:~$ ./sameLine notthere1.txt notthere2.txt
Error: Could not open file notthere1.txt
usr@linux:~$ ./sameLine file.txt notthere2.txt
Error: Could not open file notthere2.txt
```

- ☐ The program will print out the common lines in the following format:

```
<line number>: <line content>
```

Pointy brackets are used to specify the format. Don't print them out. Once again, it only prints out the lines which have the same line content and line numbers in both files.

- ☐ The line numbers starts from 1 for each file.
- ☐ Hints:
  - When using `fgets` to iterate through the lines, `fgets` will return `NULL` when it reaches the end of file.
  - You can use the `strncmp` function to compare two strings (i.e., lines).
  - You can use a count variable to keep tracking the line number.
- ☐ To test your program, you can first manually run `./sameLine file1.txt file2.txt`. If the output looks good to you, you can use the provided test script to do a thorough test by typing `./test_sameLine.sh`.

When finished, submit your `problems.c`, `platin.c`, and `sameLine.c` to **Gradescope**.