Lab 2

Due Sep 22, 2023 by 6:30p.m. **Points** 1

Lab 2: Pointers and Input

Introduction

The purpose of this lab is to practice writing C programs involving pointers and the `scanf` function.

To start, log into MarkUs and click on the lab. You can use the same process as Lab 1 to access the starter files: click the button at the bottom to add the files to your repository, then do a `git pull` to download the files to your computer.

We've described each problem briefly below, but for more detail on the first two problems, read through the starter code. There is no starter code for the final two problems.

1. invest.c

Your task is to implement a function `invest` that takes an amount of money and multiplies it by a given rate. It's your job to figure out the exact type of this function's arguments, given the sample usage in the `main` function in the starter code.

2. score_card.c

Your task is to implement a function `sum_card`, which takes an array of pointers to integers, and returns the sum of the integers being pointed to.

3. phone.c

Your task is to write a small C program called `phone.c` that uses `scanf` to read two values from standard input. The first is a 10 character string and the second is an integer. The program takes no command-line arguments. (You will write this program from scratch, so remember to add, commit, and push.)

If the integer is -1, the program prints the full string to stdout. If the integer i is between 0 and 9, the program prints only the corresponding character (i.e., at index i) from the string to stdout. In both of these cases the program returns 0.

If the integer is less than -1 or greater than 9, the program prints the message "ERROR" to stdout and returns 1.

We haven't learned about strings yet, but you will see that to hold a string of 10 characters, you actually need to allocate space for 11 characters. The extra space is for a special character that indicates the end of the string. Use this line

```
char phone[11];
```

to declare the variable to hold the string. Other than this line, there is no starter code for this program.

Note: the program must not print any prompts.

Here is some sample runs of the program ($ is the shell prompt). The line after `./phone` is input typed by the user:

```
$ ./phone
4161234567 3
1
$ ./phone
4161234567 -1
4161234567
$ ./phone
4161234567 9
7
$ ./phone
4161234567 10
ERROR
```

4. phone_loop.c

Your task is to write a C program called `phone_loop.c`. This program will again read from standard input using `scanf` and take no command-line arguments. Similar to `phone.c`, this program reads a 10-character string as the first input value but then it repeatedly reads integers until standard input is closed. (Hint: Use a while loop that checks the return value of `scanf` to accomplish this. Check the man page for `scanf` to determine the return value when the input stream is closed.)

After each integer the output produced is as before:

- if the integer is -1, the full string is printed
- if the integer is between 0 and 9, the individual character at that position is printed
- if the integer is less than -1 or greater than 9, the message "ERROR" is printed (to stdout)

In each case the printing is followed by a newline character.

When the program finishes running, `main` returns with a 0 return code if there were no errors and with a 1 return code otherwise.

Note: the program must not print anything other than then the values specified.

Here is an example of running the program. The parts typed by the user are shown in bold. The program prints that parts that are not bolded.

```
$ ./phone_loop
4147891234
3
7
6
1
-1
4147891234
0
4
11
ERROR
```

How do you end standard input?

One way to run your program is to redirect the input to come from a file. Then it is clear when the file ends. But how do you close standard input, when it is coming from the keyboard? You manually indicate the end of standard input from a keyboard by pressing Ctrl-D (on Unix) or Ctrl-Z (on Windows) and enter.

Submission

Use git to submit your final `invest.c`, `score_card.c`, `phone.c` and `phone_loop.c` -- make sure they're inside your `lab2` folder and named exactly as described in this handout, as that's where our test scripts will be looking for them. Do *NOT* add or commit executables to your repository. We will build executables by compiling your code as part of testing it.

IMPORTANT: make sure to review how to use git to submit your work to the MarkUs server; in particular, you need to run `git push`, not just `git commit` and `git add`.

Running a Simple Test on MarkUs

For each of the four programs, we have provided a simple test that will confirm that you have submitted a file that compiles without warnings and runs one easy test case. As a last step run these tests using the Run Tests button in the Automated Testing tab. If your submission does not pass these automated tests, it will get 0, but if it passes these tests, it it not necessarily fully correct. You still need to test that your code works correctly in all cases.