Lab 2: Pointers and Input

Due date: Friday 19 January before 6:30pm. Submissions made after the deadline will not be accepted.

Introduction

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

To start, login to MarkUs and navigate to the `lab2` assignment. Like the previous lab, this triggers the starter code for this lab to be committed to your repository. 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.

If the integer is zero, the program prints the full string to standard output followed by a single newline `\n` character. If the integer is between 1 and 9 inclusive the program prints only the corresponding digit from the string to stdout (again followed by a newline). In both of these cases the program returns 0.

If the integer is less than 0 or greater than 9, the program prints the message "ERROR" (followed by a newline) 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.

You may assume that the user correctly enters a 10-character string and an integer. You must not print anything else to stdout. *Do not prompt the user to enter the values.*

Hint: If you skipped the video titled, Input, Output and Compiling Video 3: Reading Input (scanf), now would be a good time to go watch it.

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 with condition `while (scanf(...) != EOF)` rather than ignoring the return value from `scanf`.

After each integer the output produced is as before:

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

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

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

Do *not* print any extra newline characters or any prompts for the user.

How do you end standard input?

One way to run your program is to redirect the input to come from a file. Using that approach, 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 testingit.

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`.