LOOPS

Phillip's Time: 10 mins

The following sets of programs will test you on your ability to use loops in order to solve problems through repetition

1 SumEvens and SumOdds (24 Lines)

One of the basic functions of loops is to prompt the user for repeated input until the user enters a special terminating character. Once the terminating character is inputted, the program should exit the loop. This character is completely arbirtary and is decided by the programmer (or in this case, the annoying person writing your program)
SumEvens should start by prompting the user to type in an integer number. If the number -1 is entered, the loop should terminate and output the summation of ONLY the even numbers inputted.

Example Output:

> ./sumEvens
> Please enter a number (-1 to end program): 22
> Please enter a number (-1 to end program): 13
> Please enter a number (-1 to end program): 2
> Please enter a number (-1 to end program): 10
> Please enter a number (-1 to end program): -3
> Please enter a number (-1 to end program): -1
> Your total sum is: 34

```
>       ./sumEvens
>       Please enter a number (-1 to end program): 1
>       Please enter a number (-1 to end program): 1
>       Please enter a number (-1 to end program): -1
>       Your total sum is: 0

>       ./sumEvens
>       Please enter a number (-1 to end program): -1
>       Your total sum is: 0
```

Notice how there are no eamples of sumOdds. The reason for that is that if you can successfully implement sumEvens, then you can implement sumOdds by changing a single line your code.

Tips:

1. Don't try to keep track of all the numbers at once, you will hurt your head if you try this. Instead only work with the single input given to you per iteration of the loop
2. Remember what it means for a number to be even.
3. when using scanf, remember that it doesnt ever scan the newline, it'd be wise to account for this....read more here.

2 Basic Calculator (60 Lines)

Now that we know how to prompt a user for an input, lets expand it a bit into a full blown menu system.

Calculator should start by displaying an integer, followed by the operation corresponding to that integer (in this case 0. = End Program, 1. = Addition, 2. = Subtraction, 3. = Multiplcation, and 4. = Division). The user then should be prompted to input a choice. After the choice has been made, the user should now be prompted to input two numbers and carry out the chosen operation on those two numbers.

If an invalid choice is made, the program should display an error message (Error: Invalid choice) then exit the program with error code 1.

Example Output:

```
>       ./calculator
>       0. End Program
        1. Addition
        2. Subtraaction
        3. Multiplication
        4. Division
        Please enter a choice: 1
>       Please enter the first number followed by the second number: 1 4
>       The result is: 5

>       0. End Program
        1. Addition
        2. Subtraaction
        3. Multiplication
        4. Division
        Please enter a choice: 2
>       Please enter the first number followed by the second number: 2 5
>       The result is: -3

>       0. End Program
```

1. Addition
2. Subtraaction
3. Multiplication
4. Division

Please enter a choice: 3

> Please enter the first number followed by the second number: 5 5

> The result is: 25

> 0. End Program
1. Addition
2. Subtraaction
3. Multiplication
4. Division

Please enter a choice: 4

> Please enter the first number followed by the second number: 10 5

> The result is: 2

> 0. End Program
1. Addition
2. Subtraaction
3. Multiplication
4. Division

Please enter a choice: 0

>

> ./calculator

> 0. End Program

1. Addition

2. Subtraaction

3. Multiplication

4. Division

Please enter a choice: -1

> Error: Invalid Input

Tips:

1. To read one number after another with scanf, simply write scanf("%d %d", &num1, &num2), this tells scanf to look for an integer, followed by a space, followed by another integer and saves the two integers into num1 and num2. You will find this basic pattern matching scanning to be quite useful in the future....
2. Once again...be careful about that newline you input when you hit 'enter' on your keyboard.....

3 Red Light, Green Light! (39 Lines)

Another key use of loops is to process a set of commands.

When I was a kid, me and the neighborhood kids would play this stupid game called Red Light Green Light. The idea is that one kid would stand on the opposite side of a field and just yell the commands " Red Light" or "Green Light". If "Red Light" was yelled, we would all have to stop, else if "Green Light" was yelled, we would all run towards the person. If one of us got to the person at the end of the field, you won! Very engaging no?

So naturally, we're going to code it. RedLightGreenLight should begin by scanning a set of

characters inputted by the user. The characters must only contain 'G' or 'R' for Green Light and Red Light respectively. Then for every character scanned, the program should print either "Running!!!" or "FREEZE!" depending on whether 'G' or 'R' was read. If the very last character read was 'G', then the program should also print "You Win!" (As if the last thing said was G eventually you will get to the other side of the field provided your not completely out of shape.), else if the very last thing read was 'R' then the program should print "You didn't win yet!"

If something other than 'G' or 'R' was scanned in, the program should complain and throw an error message (Error: Command not G or R), then return error code 1

Example Output:

> ./RLGL

> Enter the set of commands: GGGRG

> Running!!!

Running!!!

Running!!!

FREEZE!

Running!!!

You Win!

> ./RLGL

> Enter the set of commands: RRGR

> FREEZE!

FREEZE!

Running!!!

FREEZE!

You didn't win yet!

> ./RLGL

> Enter the set of commands: RRPR

> FREEZE!

FREEZE!

Error: Command not G or R

TIPS:

1. In order to scan a single character at a time from a set of characters, simply put your scanf inside a loop
2. An important question you should ask yourself is when should my program stop looping?
3. remember that scanf actually does return an integer value. It returns the number of successful arguements read. I.E scanf("%d", num) will return 1 if the user inputs an integer value.
4. What will scanf return if there is no more input left to scan....?
5. Don't let the number of lines fool you. This program is a little tricky for beginners

4 Tutor Panic! (21 Lines)

As a tutor, sometimes I get swamped by too many people and its tiring!!! So I (ill-advisingly) decided to make a program to help schedule tutees better.

The input for Tutor Panic will be a text file (student.txt), with each row containing 3 integers. The first integer is the students SID number, the second integer is the time the student wishes to start tutoring, and the third integer is the time the student wishes to end tutoring. students.txt is garenteed to be sorted by begining time. The numbers are also garenteed to be separated by one space.

TutorPanic should start by taking in student.txt and for each student, figuring out whether or not I can accomodate the student's schedule. I can accomodate them if the students begin time isn't within the timeframe of a previous student. If I can accomodate the student, the program should print out "Student {SID} can be accomodated from {Begin_Time} to {End_Time}. Furthermore, the CSIF opens at 8AM, so if a student request for me to tutor them at 3AM, I'm not going to be able to do that (assume we are also using a 24 hour clock). Students are also accomodated on a first come first serve basis, meaning if two students have the same time frame, I will accomodate the student that arrives first in the list.

Example Output:

```
> cat students.txt
> 999818578 13 14
  111222333 15 16
  223334444 17 23
  442345343 23 24

> ./TutorPanic < students.txt
> Student 999818578 can be accomodated from 13 to 14
  Student 111222333 can be accomodated from 15 to 16
  Student 223334444 can be accomodated from 17 to 23
  Student 442345343 can be accomodated from 23 to 24
```

```
>       cat students.txt
>       999818578 13 15
        111222333 14 16
        223334444 17 23
        442345343 23 24

>       ./TutorPanic <      students.txt
>       Student 999818578 can be accomodated from 13 to 15
        Student 223334444 can be accomodated from 17 to 23
        Student 442345343 can be accomodated from 23 to 24

>       cat students.txt
>       999818578 5 6
        111222333 12 16
        223334444 12 16
        442345343 23 24

>       ./TutorPanic <      students.txt
        Student 111222333 can be accomodated from 12 to 16
        Student 442345343 can be accomodated from 23 to 24

>       cat students.txt
```

```
>       999818578 5 6
        111222333 7 10
        223334444 9 15
        442345343 14 22

>       ./TutorPanic <      students.txt
```

Tips:

1. Since the input is garenteed to be sorted by begining time, all you really need to keep track of to see if there is a conflict is the information of the person directly before you..
2. This program uses a lot of elements from the previous ones! Take a look at those hints as they might be useful!