Lab 5

Due Dates:
Submit this lab and the extended learning (optional) to TEACH by Monday at 11:59pm. You must get checked off by a TA by the beginning of lab 6.

Lab Overview and Objectives:
Computer programs that do only one thing are of very limited use. The ability for programs to inspect information and make decisions based on the information is called control flow. The basic IF statement has already been covered. This lab explores other control flow statements.

Prelab:
1. Write a couple lines of code to show how you could call these functions. If the function returns a value be sure to save it in an appropriately typed variable.
   A. void printLongest ( char one[ ], char two[ ], char three[ ] )
   B. float calcAverage ( int num1, int num2, int num3 )
2. Write a while loop that prints: 1 2 3 4 5 6

Task 1:
1. For the first time in cs/ece 151 we are putting code in separate functions and not completely in the main function. Look below for a model of how that can be done. We write the function above main so that it has been defined above the place it is used. It is similar to declaring a variable above the point where you start using the variable.

```
#include <stdio.h>

int pleaseAdd( int a, int b ) {   // define a function before it is called
    int c = a + b;    // a and b are the function parameters
    return c;
}

int main( ) {
    int x, y;    // declare variables before you use them
    scanf("%i %i", &x, &y);   // use the variables
    int z = pleaseAdd( x, y ); // call the function
    printf("The sum is %i\n", z);
    return 0;
}
```

2. For this lab, follow the model above and write three functions. The first counts from 0 to the parameter and prints the numbers out in a single line. The second function counts down to 0 from the parameter. The last function should count between the two parameters. If the first number is smaller than the second number, the function should count up. If the first number is
larger, the function should count down.

3. Once all three functions are written, write a main function that prompts the user for two positive numbers then uses the first for the count up and count down functions and both numbers for the count between function. After doing this once, the program should ask the user if they would like to quit or re-run the program.

Example output:

```
flip wallacch $ Lab5
Please enter 2 ints.
3 9
0 1 2 3
3 2 1 0
3 4 5 6 7 8 9
Do you want to run this again with 2 different ints? (Enter 1 for yes, 2 for no) 1
Please enter 2 ints.
10 6
0 1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1 0
10 9 8 7 6
Do you want to run this again with 2 different ints? (Enter 1 for yes, 2 for no) 2
```

**Task 2:**

You may have noticed that even when you only want to read one character from the keyboard, nothing happens until you press the "Enter" key. The computer will not allow you to immediately read the character after it was typed. This is because input into the computer program is "buffered": the computer remembers all the characters typed until it encounters the newline character ('\n'). At that point in time, all characters that were typed are made available to your program--this includes the '\n' too. You may then go through each of those characters one at a time by calling getchar over and over. In case you haven’t used getchar before, it is called like this:

```c
char myChar = getchar();
```

Often, a program will only be interested in the first character that was typed, so all the remaining characters on the line are extra and should be ignored. A common task is to call getchar to read in the first character from the buffer, remember that character, and then keep calling getchar over and over until the program encounters the newline character. None of these "extra" characters are remembered. This is called "flushing" the buffer.

Name your program for this task: flush.c

Write a function called getchar_and_flush that gets the first character AND flushes any remaining characters out of the input buffer before returning that character. Your function should take no arguments and return a single character (char). Use a while loop which is perfect when you do not know how many times the loop will execute. We do not know how many characters will appear before the newline character ('\n').

In the **main** function of your program, create a loop that will do the following:

1. Ask the user to type a character.
2. Print out the character the user typed as well as the ASCII number for the character.
   (You get the ASCII value simply by printing the char with %i instead of %c.)
3. Repeat forever until the user enters a special character like '~', 'Q', or some other character of your choosing.

Watch out: if you call getchar one too many times, your program will stop again waiting for input. The trick is to call getchar until you read the newline and then STOP.

Example output:
flip wallacch $ flush
I will run this program until you type a "*" character as the first character on the line!
Enter a char please. (* to quit)
I will enter lots of characters.
You entered I which has the ASCII value of 73.
Enter a char please. (* to quit)
more and more chars. Everything after the first char is removed.
You entered m which has the ASCII value of 109.
Enter a char please. (* to quit)
OK
You entered 0 which has the ASCII value of 79.
Enter a char please. (* to quit)
* because I want to quit
You entered * which has the ASCII value of 42.
BYE!
flip wallacch $

Get checked off by a TA or NO points. Submit your code for this lab by next Monday at 11:59pm.

Study Questions (written answers due at the beginning of Lab6):
1. Do you completely understand what the return statement does? It changes the flow of execution. It causes the program to “return” to the point where the function was called. It also can bring a value with it.
   return; // simply “returns” to where the function was called
   return x; // “returns” AND it brings the value of x

   Consider this function.
   int whatDoIReturn( int x ) {
       if(x<=1) {
           return -2;
       }
       int i;
       for(i=1; i<100; i++) {
           if(i%2 == 0 && i%5 == 0 && i%x == 0) {
               return i;
           }
       }
       return -1;
   }

   Describe and give examples of what this function can return. Your explanation should explain when each return statement is used.
2. Write a couple lines of code that calls the above function and prints out what it returns.

**Lab 5 Summary:**

<table>
<thead>
<tr>
<th>TASK</th>
<th>Completed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prelab</td>
<td>4pts.</td>
</tr>
<tr>
<td>Task 1</td>
<td>5pts.</td>
</tr>
<tr>
<td>Task 2</td>
<td>5pts.</td>
</tr>
<tr>
<td>Study Questions</td>
<td>4pts.</td>
</tr>
</tbody>
</table>

**Extended Learning:**

Write a program that walks an ‘*’ to the right and back to the left exactly like this demonstration. The first thing your program should do is ask the user how far to walk. You will write functions and not put the movement code in main.

Your main function must look like this:

```c
int main() {
    int x;
    printf("Enter how far to walk. ");
    scanf("%i", &x);
    // add some checking to see if x is <=1 or >10

    goThereAndBack(x);
    goStraightThereAndBack(x); // * stays on the same line
}
```

Hints:
- Pause 1 second: sleep(1);
- Move backward 1 space: printf("%c", 'b');
- Flush the output buffer and print my printf statement now: fflush(stdout);

Demonstrate your program to a TA at the beginning of Lab 6.
Submit your code to Lab5Extra by next Monday at 11:59pm.