CSE 1321L: Programming and Problem Solving

Lab 6

Repetition Structures

What students will learn:

- The purpose of repetition structures
- Three kinds of loops that are found in most languages
- Knowing when to use a specific kind of loop
- How to apply repetition to solve problems

Overview: If there's one thing computers are good at, it's repeating something over and over. The concept of repetition (which some call "iteration" and others "looping") is not terribly difficult since we humans repeat things in our daily lives. Any decent programming language is going to support iteration and usually allows for three different kinds of "looping templates". These templates are exactly what this lab is going to cover.

The three kinds of loops we'll cover are the for, while and do-while loop. You want to memorize the templates for these. Before that, it's important to know when to use them. Here's an overall guideline to help you out:

1. Use a for loop when you want to repeat something a certain number of times. For example, if you want to repeat something 100 times, and a for loop is a good candidate for that. Or, if you wanted to count from 50 to 3000 in increments of 10, you could do that too.
2. Use a while loop is useful when you don't know how many times something will repeat; the loop could "go on forever". As an example, if you ask a user to enter a number between 1-10 and they consistently enter 45, this could go on forever. Eventually (and hopefully), the user would enter a valid number.
3. Use a do-while loop when the loop must execute at least one time. The loops above can execute 0 times, but not this one! The reason is because, for all loops, there is a test to see if the loop should continue repeating. With a do-while loop, that test is at the bottom.

The best way to show these loops is through examples. The code below sums the numbers from 1 to 100. You should see that all loops have a 1) test to continue, and 2) make progress towards completing.

```
int sum = 0;
for (int i = 1; i <= 100; i++)
{ sum +=
  i;
}
```

```
int sum = 0; int
i = 1; while(i
<= 100) { sum +=
i; i++;
}
```

```
int sum =
0; int i =
1; do { sum
+= i; i++;
} while(i <= 100);
```

- The for loop starts a counter (called 'i') at 1. So long as i is <= 100, the loop continues. We then execute sum += i. Finally, i++ occurs and the test to continue is evaluated again.
- The while and do-while loops are nearly identical. However, notice that the do-while has the test to continue at the bottom – and also has a semicolon at the end. Be careful!

Lab6A: Largest of 10 For this lab, please use a **for loop**. They may seem a little more intimidating at first, but they are actually quite simple to set up.

The goal of this exercise is for you to create a program that will ask the user to input 10 **positive integer** numbers, **one at a time**. While it does this the program should also keep track of the largest number it has seen so far. After it has run 10 times, it should display the largest number you inputted.

Remember, the class name should be Lab6A.
The user input is indicated in **bold.**

Sample output:

```
Please enter 10 numbers and this program will display the largest.

Please enter number 1: 50
Please enter number 2: 51
Please enter number 3: 10
Please enter number 4: 1
Please enter number 5: 99
Please enter number 6: 1000
Please enter number 7: 1010
Please enter number 8: 42
Please enter number 9: 89
Please enter number 10: 1000

The largest number was 1010
```

Intentionally Left Empty, please proceed to Next Page

Lab6B: Pick a number between 1 and 1000 For this lab, make sure to please use a **while loop**. Many programming languages have a library for a **R**andom **N**umber **G**enerator (RNG). Whenever this RNG is used it will output one "random" number back to the user. Applications of such a generator can including something like the lottery.

Please keep in mind that the RNG will generate a random floating-point value between 0.0 and 1.0 and to adjust the range of the random numbers you need to either multiply or add to the result.
For example, if want the range to be from 0.0 to 50.0 we would *multiply* the result by 50. If wanted to move the range to 2.0 to 52.0 we would *add* 2 to the result. Re-read this passage and try to think about this a bit more deeply, it will click and make sense.
In this lab exercise, please write a program that asks the user to pick an **integer number** between 1 and 1000; **please use a while loop to verify that what was entered by the user is between the range specified earlier**. If the input is within range, please have an RNG (please ask your lab instructor for details on how to create one some details shown below) that should keep randomly generating numbers until it generates one matching the number entered by the user.

Also, please make sure that the program keeps track of how many guesses it takes. Have the program displays each guess and after than display the total guess count. Please refer to the sample output below.

Disclaimer: When using the RNG you are going to have to store the generated number as a double or a float. Please make sure to round up the generated number to the nearest ones digit in order to avoid an infinite loop.

Remember, the class name should be Lab6B.
The user input is indicated in **bold.**

Java

```
import java.util.Random;
public class generateRandom{
    public static void main(String args[])
    {
      Random rand = new Random();
      // Generate random integers in range 0 to 9
      int rand_int1 = rand.nextInt(10);} }
```

C#

```
public class generateRandom{
    public static void main(String args[])
    {
        Random rnd = new Random();
      // Generate random integers in range 0 to 9
        int example   = rnd.Next(10); } }
```

C++

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

int main() {
      srand((unsigned) time(0));
      int randomNumber;
      // Generate random integers in range 0 to 9
     randomNumber = (rand() % 10);
     cout << randomNumber << endl; }
```

Sample output:

```
Enter a number between 1 and 1000: 42
My guess was 56
My guess was 198
My guess was 239
My guess was 2
My guess was 5
...
My guess was 920
My guess was 42

I guessed the number was 42 and it only took me 231 guesses
```

Intentionally Left Empty, please proceed to Next Page

Lab6C: Cha-Ching For this lab, use a **do-while loop**.
A **sentinel loop** is a loop (a special while loop or a do-while loop) that continues to process data until it reaches a specific value(s) that signals that it should stop looping; this special value(s) is usually indicated as the condition of the while or do-while loop. A good example of a sentinel loop is the while loop that you had to write to verify user input in Lab6B, the special values were anything in the range of 1 to 1000. Another very common application for this is allowing a user to rerun a program.
Please write a very simple program that mimics a bank account. The program should start the user out with $1000. The program should print out a welcome menu **once** with the options present for the user. The program should allow the user to make a deposit, withdrawal, and see their current balance. Every time the user deposits or withdraws, the program should show the user their new balance; it should also ask the user if they want to return to the main menu. ***As long as*** **the user types 'Y' or 'y'** the program should keep running.

Remember, the class name should be Lab6C.
The user input is indicated in **bold.**

Sample output:

```
Welcome!
You have $1000 in your account.

Menu
0 – Make a deposit
1 – Make a withdrawal
2 – Display account value

Please make a selection: 1
How much would you like to withdraw? : 50
Your current balance is $950
Would you like to return to the main menu (Y/N)? : Y

Menu
0 – Make a deposit
1 – Make a withdrawal
2 – Display account value

Please make a selection: 0
How much would you like to deposit? : 100
Your current balance is $1050
Would you like to return to the main menu (Y/N)? : Y

Menu
0 – Make a deposit
1 – Make a withdrawal
2 – Display account value

Please make a selection: 500
Invalid entry, please try again.
Would you like to return to the main menu (Y/N)? : Y
```

```
Menu
0 - Make a deposit
1 - Make a withdrawal
2 - Display account value

Please make a selection: 2
Your current balance is $1050
Would you like to return to the main menu (Y/N)? : N

Thank you for banking with us!
```

Instructions:

- Programs must be working correctly.
- **Programs must be saved in files with the correct file name.**
- If working in Java or C#, class names must be correct.
- Programs must be working and checked by the end of the designated lab session.
- Programs (**only .java, .cs or .cpp files**) must be uploaded to Gradescope by due date.