Iteration Lab

In this lab, you will be writing your own code to perform tasks that require the use of loops. You can use `for` loops, `while` loops, or both. Each activity is labeled as "level I", "level II", or "level III", based on the complexity of thought and coding they require. Everyone must complete at least one activity from each level.

Activity 1: (level 1) Finding the sum of the odd numbers, 1 through 2*n* + 1.
Write a Java program to find the sum of the odd numbers from 1 to 2*n* + 1, inclusive; *n* represents any positive integer that is entered by the user. For example, if the user enters 7, then 2*n* + 1 is 15. The program needs to sum up 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15. When your program is finished, it should output the number with a message similar to "The sum of odd numbers from 1 through 15 is 64."

Activity 2: (level 1) Checking whether a number is prime.
Write a Java program that checks whether the integer entered by a user is prime. When finished, the program should print out the answer with a simple message such as, "15 is not prime." or "31 is prime". Here are some helpful points:

- The number 1 is not considered prime.
- 2 is the lowest prime number.
- The definition of a prime number is that it has exactly two factors: 1 and itself. For example, 23 is prime, because it is divisible only by 1 and by 23.
- You will want to start by testing your code on small prime numbers. Once it is working, test it on some larger numbers as well. Here are some large prime numbers for testing: 499, 1217, 2731, 7879, 19319, and 79393.

Activity 3: (level 2) Checking whether a String is a palindrome.
Write a Java program that checks whether the String entered by a user is a palindrome. There is more than one approach that works. Here are some tips:

- Recall that you can use .nextLine() to accept a String as keyboard input (just as you would use .nextDouble() or .nextInt() to accept a number).
- You can use .charAt(i) to access the character at position i in a string. But remember that Java starts counting from 0, so that the .charAt(0) returns the first character in the String, .charAt(1) returns the second character, and so forth.
- Characters can be compared in Java using the == operator, just like numbers.
- If you want to compare two String, you'll have to use the .equals method. For example, to compare string1 with string2, you would use string1.equals(string2). Java will check character by character that the strings are equal.
- Case counts: 'A' and 'a' are not the same character, and "Apple" and "apple" not equal strings.

© 2023 Great Hearts America

Activity 4: (level 2) Calculate the *n*th Fibonacci number.
Write a program that computes the *n*th Fibonacci number in the sequence 1, 1, 2, 3, 5, 8, 13, 21, ...
The user should input *n*, a positive integer. The program then prints a message telling the *n*th Fibonacci number. Have the program check that the integer the user enters is positive. If the user enters a non-positive integer, the program should prompt them to try again. If you want an extra challenge, check also that the number is not too big for the computer to handle (hint: look back in your notes on data types to find the largest possible size of int variables, and notice that Fibonacci numbers get large quite quickly). You can design the program to deal with non-integers as well. In order to do this, you can accept input as a double, then after checking, you can convert it to an int. The following command converts double x to int i:

```
i = (int)x;
```

Activity 5: (level 3) Finding the greatest common divisor.
Write a Java program that computes the greatest common divisor, GCD (also called greatest common factor, GCF) of two user-supplied integers. Recall that the GCD is the largest number that will divide evenly into both numbers. For example, for the numbers 24 and 32, the GCD is 8. When complete, the program should print out a simple message stating, for example, “The GCD of 24 and 32 is 8.” Here are some tips:

- This problem can be solved by testing one number at a time, starting with 1, 2, *etc*. But that can take a very long time. See if you can find a way to cut down on the number of tests you need to do. For instance, what is the largest number you would ever benefit from trying?
- It might help with efficiency if you think about what the highest number is that you should try if: both numbers are even or both numbers are odd. Also, think about what the answer should be if the two numbers are equal.

BONUS: Research Euclid’s algorithm for finding GCD, and implement it in Java.

Activity 6: (level 3) Finding the least common multiple of two numbers.
Write a Java program that computes the least common multiple (LCM) of two user-supplied integers. The least common multiple is the smallest number into which both numbers will divide evenly. For example, the LCM of 24 and 36 is 72. Here are some points to consider:

- What if both numbers are zero?
- What if just one of the two numbers is zero?
- Incidentally, there is a simple way to calculate the LCM of two numbers, if you already have the GCD. If you want to research that, you may be able to make quick work of this activity by expanding your solution to Activity 5.

© 2023 Great Hearts America

Activity 7: (level 1) The Collatz Conjecture

The Collatz sequence, named after Lothar Collatz, is a list of numbers formed according to the following rules: Starting with any positive integer, *n*, the first term of the Collatz sequence is $C_1 = n$. Each successive term C_{k+1} is formed from the previous term C_k according to the function:

$$C_{k+1} = \begin{cases} \frac{1}{2}C_k & if\ n\ is\ even \\ 3C_k + 1 & if\ n\ is\ odd \end{cases}$$

For example, if n = 5, the sequence goes 5, 16, 8, 4, 2, 1. For n = 7, the sequence goes 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

The Collatz conjecture is that *This sequence will eventually reach the number 1, regardless of the starting positive integer.* This is a simple idea, but it remains unproven! Perhaps you will be the one to provide a proof someday?

Write a Java program that prints out the Collatz sequence for a user-supplied starting integer. The program should stop at that point and print out the number of steps taken to reach 1. If it does not eventually arrive at 1, either your program did something wrong, or you have found a counterexample to the conjecture and proven it false (but this is unlikely, as the conjecture has been checked for every number less than 2^{68}, or approximately 300 quintillion!).

© 2023 Great Hearts America