CS 250 - Fall 2021 - Lab 11: Recursion Part A

Available of Part A: Nov 17, 2021.
Available of Part B: Nov 22, 2021.
Due date: **Nov 29, 2021, 11:59 PM**

Lab 11 will expand to two weeks. This document is the first part of this lab that guides you through programming some basic recursive problems.

Recursion is a powerful design technique. Recursion can be a difficult concept to master, and it is worth concentrating on in isolation before using it in large programs. Therefore, this week's lab is structured as several small problems that can be solved separately.

Objectives:

The objectives of this lab are:

- To motivate the use of recursion.
- Simple recursion
- Recursion with a return value
- Practice exploiting recursive methods to solving a set of practical problems: positive integer power of a number, digit sum of a number, conversion of a number from base 10 to other, computing the Fibonacci number

Prerequisite:

- Recursion tutorial on Javatpoint

Part A1: Computing the Power

In the first part, we will develop a recursive method to calculate a positive integer power n of a number x (x^n). This equation is defined as follows:

$$\begin{aligned} x^n &= 1 \qquad\quad if\ n = 0 \\ &= x * x^{n-1} \quad if\ n > 0 \end{aligned}$$

The above equation of power is very suitable to implement this function using the recursion approach.

1. Create a new project named Lab11A and create a class named PowerRecursive

2. Copy the following skeleton to replace the generated code in Eclipse

```
public class PowerRecursive {
        /**
         * This method calculates the power x^n
         * @param x: the base integer number
         * @param n: the positive integer number of the exponential
         * @return: the positive integer power n of x (x^n)
         */
        public static int powerFunc(int x, int n) {
                //TODO: add code below
        }

        public static void main(String[] args) {
                // TODO Auto-generated method stub

        }
}
```

3. **Complete the method powerFunc(int x, int n)**: You need to use a recursive approach to implement this method. Use the following steps:

- If n < 0, throw a new IllegalArgumentException with the message "n should be positive!"
- Implement the base case: if n = 0, return 1
- Implement the recursive case: if n > 0, return x * powerFunc(x, n - 1)

4. Implement the **main()** method to test the newly implemented **powerFunc** method. Implement it with the following steps:

- Create a **Scanner** instance to read input from the console
- Prompt the text "Enter the base (x): "
- int x = Use the above Scanner instance to read input of integer
- Prompt the text "Enter the exponential (n - positive) only: "
- int n = Use the above Scanner instance to read input of integer
- Calculate x^n using the powerFunc method and print the value to console.

Part A2: Count the number of digits of a positive integer

The second program that demonstrates the recursive algorithm is counting the number of digits of an integer number **n**. This problem can be solved recursively using the following equation:

- If n < 10, the number of digits is only one
- Otherwise, the number of digits is one (the unit digit) + the number of digits in the rest of the number n (the rest when taking off the unit digit)

An example to demonstrate this method: the number of digits in 1234 is 1 (the unit digit 4) + the number of digits in 123.

1. Create the second class CountDigits in Lab11A

2. Copy the following skeleton code to replace the generated code of CountDigits

```
import java.util.Scanner;
public class CountDigits {
        /**
         * This method count the number of digits of an integer num
         * @param n: the positive integer number
         * @return: the positive integer power n of x (x^n)
         */
        public static int countDigits(int n) {
                //TODO: add code below
        }

        public static void main(String[] args) {
                // TODO Auto-generated method stub
        }
}
```

3. **Complete the method countDigits(int n)**: You need to use a recursive approach to implement this method. Use the following steps:

- If n < 10, return 1
- Otherwise, return 1 + countDigits (n / 10)

4. Implement the **main()** method to test the newly implemented **countDigits** method. Implement it with the following steps:

- Create a **Scanner** instance to read input from the console
- Prompt the text “Please enter a positive integer: ”
- int num = Use the above Scanner instance to read input of integer
- If num <= 0, print the error message "Please run again and enter a positive integer!"
- Otherwise, count the digits of num using the countDigits method and print the value to console.

Part A3: Computing the Fibonacci numbers

Fibonacci sequence is a well-known mathematical sequence in which the Fib(n) - the nth term of the sequence can be defined as follows:

$$Fib(0) = 0$$
$$Fib(1) = 1$$
$$Fib(n) = Fib(n-1) + Fib(n-2)\ if\ n > 1$$

The mathematical expression of the Fibonacci sequence makes it naturally fit for recursive implementation.

The following steps guide you through implementing the code for this part:

1. Create a new class Fibonacci in Lab11A

2. Copy the following code to replace the generated code of that class:

```
public class Fibonacci {
        /**
         * This method calculates the Fibonacci of nth term recursively
         * @param n:
         * @return: the Fibonacci value of the nth term
         */
        public static int fibRecursive(int n){
                //TODO: add code below
        }

        public static void main(String[] args) {
                // TODO Auto-generated method stub
        }
}
```

3. **Complete the method fibRecursive(int n)**: You need to use a recursive approach to implement this method. Use the following steps:

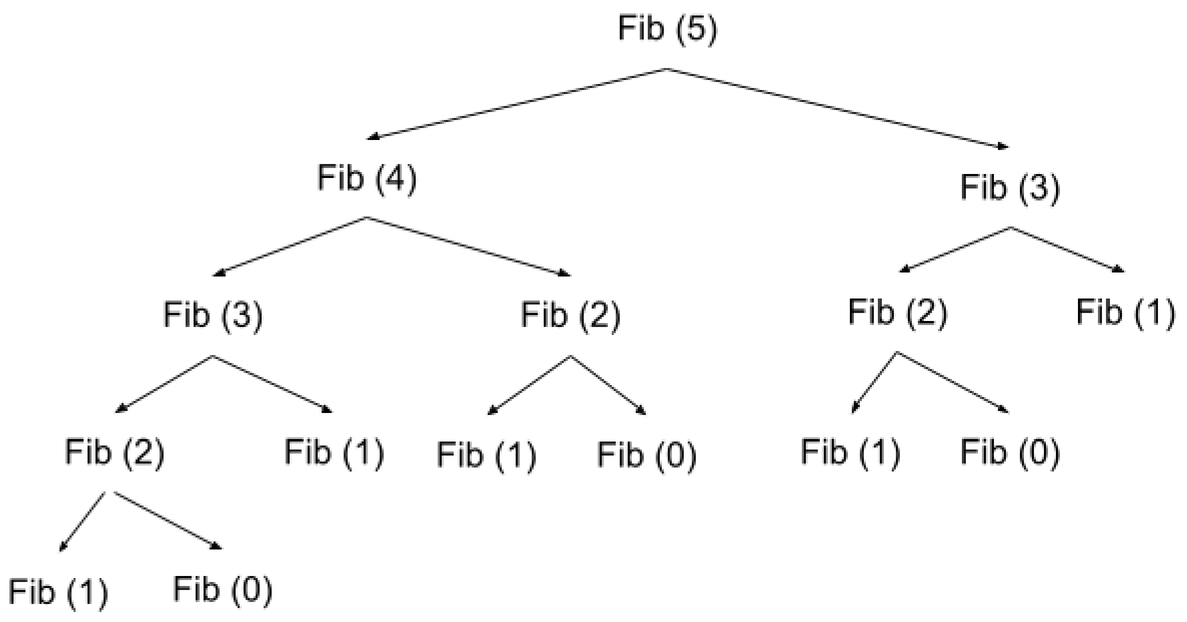
- If n = 0 or n = 1, return n
- Otherwise, return fibRecursive(n-1) + fibRecursive(n-2)

4. **Complete the main() method:** This method will use the Scanner to ask the user to input the number **n** and use the method **fibRecursive** to calculate the **nth** term of the Fibonacci sequence. Use the following steps:

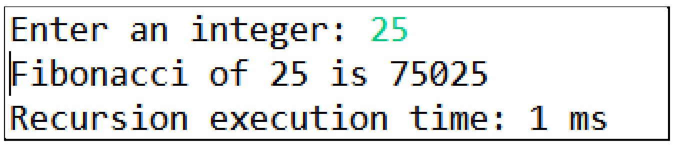
- Create a **Scanner** instance to read input from the console
- Prompt the text “Please enter a positive integer: ”
- int n = Use the above Scanner instance to read input of integer
- If n < 0, print the error message "Please run again and enter a positive integer!"

- Otherwise, calculate the nth term of the Fibonacci sequence using the fibRecursive method and print the value to console.
- Before calling fibRecursive, you can call System.nanoTime() to get the start time.
- After calling fibRecursive, you can call System.nanoTime() to get the end time.
- Calculate the execution time of fibRecursive(n) in milliseconds by subtracting the end time with start time and then divide the subtraction result with 1000000

5. Run this Java class, input a small number less than 35, and notice the results. If you try a bigger number, it takes a long time to compute and return the value. This issue is because the recursive version of Fibonacci makes a lot of repetitive recursion calls. Check the following diagram to understand the method call diagram to calculate Fib(5)



You can see in the above diagram that Fib(3), Fib(2), and Fib(1) are repeatedly called multiple times. This approach is a very inefficient method to calculate the Fibonacci sequence. The bigger the number, the more repetition of those Fibonacci terms calculations in the middle of the execution. Below is the sample output of this part:



```
Enter an integer: 25
Fibonacci of 25 is 75025
Recursion execution time: 1 ms
```

Lab Assignment Part A

The recursive version of the Fibonacci calculation at Part A3 is inefficient because of the repetitions of the calculations of the intermediate Fibonacci terms. To avoid the recomputing of Fibonacci terms to calculate the nth term of the Fibonacci sequence more efficiently, we calculate these sequences from the beginning of the sequence, instead of from the end (n), saving them in the array to avoid the recomputation. By saving the Fibonacci sequence in the array, we do not need to recalculate a specific Fibonacci if we can look it up. This assignment asks you to implement the iterative version of the Fibonacci as follows:

1. Add a new method in the class Fibonacci:

```
/**
 * This method calculates the Fibonacci of nth term iteratively using an array to
 *    store the intermediate values to avoid the recomputation
 * @param n: the input number
 * @return: the Fibonacci value of the nth term
 */
public static int fibIterative(int n) {
     //TODO: add code below
}
```

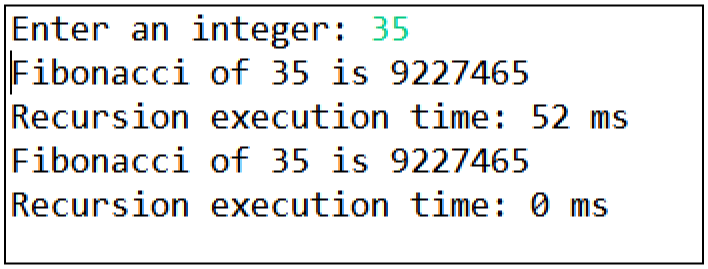
2. Implement the iterative version as follows:

- Create an array of integers the size of n + 1
- Initialize the first two elements of the array to 0 and 1, corresponding to the first two elements of the Fibonacci sequence.
- Then use a for loop from 2 to n and computing each element of the array as the sum of the two previous elements: **array[i] = arr[i-1] + arr[i-2]**
- Return the value of the array at the index n: **array[n]**

3. Modify the code of the **main**() method and execute it as follows:

- Add the code to call the new method **fibIterative(n)** to calculate the nth term of the Fibonacci sequence after calling (do not remove the code to call fibRecursive() on the input number
- Compare the running time of two versions of the Fibonacci sequence calculations.

Below is the sample output of this assignment:



```
Enter an integer: 35
Fibonacci of 35 is 9227465
Recursion execution time: 52 ms
Fibonacci of 35 is 9227465
Recursion execution time: 0 ms
```

Lab 11A Result Submission:

You need to submit the results of the following sections to the D2L assignment item of Lab11:

- Complete the required Lab Assignment Part A on week 13

- Complete the required Lab Assignment Part B on week 14

- Compress the whole Eclipse source code of your Lab11A and Lab11B project in zip format using the Export function