CS 250 - Spring 2021 - Lab 11: Recursion Part B

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 second part of this lab that guides you through programming some intermediate recursive problems related to string and array.

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 the second part 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: counting a number in an array, place hyphen character between characters, remove duplicate characters, and subset-sum

Prerequisite:

- Recursion tutorial on Javatpoint

Part B1: Counting a number in an array

The first problem is counting the number of occurrences of a number in an array of integers recursively. To solve this problem, we can use the idea of recursive linear search in the lecture. However, we will keep counting until the last element in the array.

1. Create a new project named Lab11B and create a class named CountNumInArray

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

```
public class CountNumInArray {
	/**
	 * This method counts the number of occurrences of num in the given array
	 * @param arr: the input array
	 * @param num: the number to be counted in the array
	 * @param index: the current index to search and count
	 * @return: the number of occurrences of num in arr
	 */
	public static int countNumInArray(int[] arr, int num, int index) {
		//TODO: add code below
	}

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

3. **Complete the method countNumInArray(int[] arr, int num, int index)**: You need to use a recursive approach to implement this method. Use the following steps:

- If arr is null or index < 0, throw a new IllegalArgumentException with the message "Invalid input!"
- If index is greater than or equal to the length of arr, then return 0
- If the element of array arr at index equals to num: return 1 + countNumInArray(arr, num, index + 1)
- Else return 0 + countNumInArray(arr, num, index + 1)

4. Implement the **main()** method to test the newly implemented **countNumInArray** method. Test the method **countNumInArray** and print the value with the following array, input number and start index:

- Array = { 1, 6, 4 }, num = 6, index = 0 => should return 1
- Array = { 1, 4 }, num = 5, index = 0 => should return 0
- Array = { }, num = 3, index = 0 => should return 0
- Array = { 6, 2, 2 }, num = 2, index = 0 => should return 2

Part B2: Place hyphen character between two characters in a string

The second program that demonstrates the recursive algorithm is separated every two adjacent chars in a given string with a hyphen character. The examples of this program are displayed below:

"abc" => "a-b-c"

"" => ""

“a” => “a”

You can implement this method using the following suggestions:

- If the length of the input string is less than 2, simply return that str
- Otherwise, return the first character + the hyphen + the result of the recursive call on the tail of the string (substring from the index 1)

Implement this program using the following steps:

1. Create the second class PutHyphenBetweenChars in Lab11B

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

```
public class PutHyphenBetweenChars {
	/**
	 * This method returns a new string from the given string that puts a hyphen
	 *  between two characters
	 * @param str: the input string
	 * @return: a new string with a hyphen character is inserted between two
	 *		adjacent characters
	 */
	public static String putHyphenBetweenChars(String str) {
		//TODO: add code below
	}

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

3. **Complete the method putHyphenBetweenChars(String str)**: You need to use a recursive approach to implement this method. Use the following steps:

- If the length of the input string is less than 2, simply return that str
- Else, return the first character + the hyphen + the result of the recursive call on the tail of the string (Hint: call substring of str with the index 1)

4. Implement the **main()** method to test and print the result of the newly implemented **putHypheBetweenChars** method with the following case to make sure your program works:

- String = “” => should print “”
- String = “a” => should print “a”
- String = “abc” => should print “a-b-c”

Part B3: Remove duplicate adjacent characters in a string

In this problem, you need to create a new string from a given string that removes all adjacent and duplicated characters as the following examples:

- “hello” => “helo”

- “abbcddd” => “abcd”

- “xyyxzzz” => “xyxz”

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

1. Create a new class RemoveDuplicateChar in Lab11B project

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

```
public class RemoveDuplicateChar {
	/**
	 * This method removes all adjacent duplicate chars in the given string
	 * @param str: the given input string
	 * @return: the string with all adjacent similar characters have been removed
	 */
	public static String removeDuplicateChar(String str)	{
		//TODO: add code below
	}

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

3. **Complete the method removeDuplicateChar(String str)**: You need to use a recursive approach to implement this method. Use the following hints:

- If the length of the input string is less than 2, simply return that str
- Otherwise, if the first character equals the second character, return the result of the recursive call on the first character + the substring of str from the index 2.
- Otherwise, return the first character + the result of the recursive call on the substring of str from the index 1

Hint: use the String.charAt(index) method to get a specific character at *index* position. The method String.substring(index) will return the substring from the *index* position.

4. **Complete the main() method:** to test the removeDuplicateChar method with the following test cases to make sure it works:

- “” => “”
- “a” => “a”
- “hello” => “helo”
- “abbcddd” => “abcd”
- “xyyxzzz” => “xyxz”
- “112aa445” => “12a45”

Part B4: Subset sum

In the last problem of part B, you are given an array of integers arr and a number k. You need to develop a method to determine if there exists a subset of arr that sum of all elements in such a subset equals to k. Below are some examples:

arr = {2, 4, 6, 8}, k = 10 => The answer is yes because sum of {2,8} or sum of {4,6}, which are two subsets of arr, equals to k=10

arr = {5, 2, 6}, k = 7 => The answer is also yes because sum of {5, 2} = 7

arr = {1, 2, 3}, k = 7 => The answer is no there is no exist subset of arr that sum of the elements equals to k = 7

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

1. Create a new class SubsetSum in Lab11B project

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

```
public class SubsetSum {
	/**
	 * This method determines if there exists a subset of the given array arr
	 *    that sum of the elements equals to a specific number (k)
	 * @param arr: the given input array
	 * @param target: the input number to check
	 * @param index: the current index of the array
	 * @return: true or false - whether there exists a subset of
	 */
	public static boolean subsetSum(int[] arr, int target, int index) {
		//TODO: add code below
	}

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

```
        }
}
```

3. **Complete the method subsetSum(int[] arr, int target, int index)**: You need to use a recursive approach to implement this method. The idea to apply the recursive approach here is describe below:

Let index i from 0 to arr.length - 1:

- if arr[i] belongs to such a subset, then there is a subset A in arr[i+1] to the end of arr such that sum of elements in A = target - arr[i]

- if arr[i] does not belong to such a subset, there is a subset A in arr[i+1] to the end of arr such that sum of elements in A = target

Use the following hints:

- If index is greater than the length of arr, return true
- If length of the array is 0:
  - If target is 0, return true
  - Else return false
- Otherwise, if the index is the last position in arr
  - If the element at the index position equals the target or target is 0, return true
  - Else return false
- Otherwise:
  - Make a recursive call on with arr, target - arr[index], index + 1
  - If the previous call returns false, return the result of the recursive call with arr, target, index+1
  - Return the result of the last recursive call.

4. **Complete the main() method:** to test the subsetSum method with the following test cases to make sure it works:

- arr = {2, 4, 8}, k = 10, index = 0 => true
- arr = {2, 4, 8}, k = 14, index = 0 => true
- arr = {2, 4, 8}, k = 9, index = 0 => false
- arr = {2, 4, 8}, k = 8, index = 0 => true
- arr = { }, k = 0, index = 0 => true
- arr = {1}, k = 1, index = 0 => true
- arr = {2}, k = 1, index = 0 => false
- arr = {2}, k = 1, index = 1 => true

Lab Assignment Part B

Complete the following assignment of this lab:

1. Modify the program to add the hyphen character only between two similar adjacent characters in the string? Example:

“abbc” => “ab-bc”

“aa” => “a-a”

“aba” => “aba”

“a” => “a”

“” => “”

2. Write a recursive algorithm to check if a given string contains zero or more pairs of parentheses, i.e. “”, “()”, “(())”. The algorithm should return false for the following case: “a”, “(a)”, “((a))”, etc. Hint: check if the first and last characters are matched parentheses or not, then recursively repeat on the inside. Remember the base case of empty string.

Lab 11B 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