**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

# Lab 12: Recursion-II

**CLO-03:** Design and develop solutions based on Software Construction principles.  
**CLO-04:** Use modern tools such as Eclipse, NetBeans etc. for software construction.

**Date: 09th Dec 2024**

**Time: 10:00 AM** **- 12:50 PM   
 02:30 PM – 04:50 PM**

**Instructor: Dr. Mehvish Rashid  
Lab Engineer: Mr. Aftab Farooq**

**Introduction:**

# Lab 12: Recursion-II

Students will have hands-on experience on designing, testing, and implementing recursive problems. Given a scenario, you will write the specifications and implement it by dividing into base case and recursive step. You may design helper methods to simplify your implementations. Write unit tests that check for compliance with the specifications.

## Lab Tasks

**Task 1: Recursive Binary Search on a Sorted Array**

**Objective:** Students will implement a recursive version of the binary search algorithm to practice recursion in a practical application and understand its benefits for searching in sorted arrays.

**Instructions:**

1. Create a Java program that takes a sorted array of integers and a target value to search for.
2. Implement a recursive method *binarySearchRecursive* that returns the index of the target value if found, or -1 if the target is not in the array.
3. The method should divide the search range into halves and recursively search in the appropriate half of the array.
4. Ensure the base case terminates the recursion when the search range is empty.
5. Include error handling for cases where the array is null or empty.

Analyze the time complexity and compare it to iterative binary search.

Important: Do not forget to write the specifications and unit tests for the code.

**Mandatory Enhancements:**

1. Modify the binary search to handle arrays of strings instead of integers.
2. Implement a variant that returns all indices of the target value if it appears multiple times in the array.

**Task 2: Recursive Parser for Mathematical Expressions**

**Objective:** Students will implement a recursive parser to evaluate simple mathematical expressions involving addition, subtraction, multiplication, and division.

**Instructions:**

1. Create a Java program that takes a string input representing a mathematical expression (e.g., "3 + 5 \* 2").
2. Implement a recursive function *evaluateExpression* that parses the expression and calculates its result.
3. Handle operator precedence (multiplication/division first, addition/subtraction second) and use recursion to break down the expression into simpler sub-expressions.
4. Follow good coding practices, including meaningful variable names, comments, and modular code.
5. Test with various expressions, including edge cases like parentheses and mixed operators.

**Mandatory Enhancements:**

1. Extend the parser to handle floating-point numbers.
2. Implement error handling for invalid expressions.

Important: Do not forget to write the specifications and unit tests for the code.

**Task 3: Recursive Sum of Digits**

**Objective:** Students will write a recursive function that calculates the sum of the digits of a given non-negative integer

**Instructions:**

1. Create a Java program that takes a non-negative integer as input.
2. Implement a recursive function *sumOfDigits* that computes the sum of its digits.
3. The base case should return 0 when the number is reduced to 0.
4. Test the program with various numbers, including edge cases like 0 and large integers.
5. Follow good coding practices, including meaningful variable names, comments, and modular code.
6. Test the program with various numbers, including edge cases like 0 and large integers.

**Mandatory Enhancements:**

1. Modify the function to handle negative numbers by converting them to positive before performing the sum.
2. Analyze the time complexity of the recursive algorithm for very large numbers.

Important: Do not forget to write the specifications and unit tests for the code.

**- Paste Code in word document**

**- Push Your Code on GitHub  
- Add Git Link in Document.**

**Compile a single word document by filling in the solution part and submit this Word file on LMS.**

**Solution**

### Task 1

### Code:

**package** test;

**import** java.util.List;

**import** java.util.ArrayList;

**public** **class** BinarySearch {

**public** **static** **int** binarySearch(**int**[] arr, **int** left, **int** right, **int** target) {

**if** (left > right) {

**return** -1;

}

**int** mid = left + (right - left) / 2;

**if** (arr[mid] == target) {

**return** mid + 1;

} **else** **if** (arr[mid] < target) {

**return** *binarySearch*(arr, mid + 1, right, target);

} **else** {

**return** *binarySearch*(arr, left, mid - 1, target);

}

}

**public** **static** **int** binarySearchString(String[] arr, **int** left, **int** right, String target){

**if** (left > right) {

**return** -1;

}

**int** mid = left + (right - left) / 2;

**int** comps = arr[mid].compareTo(target);

**if** (comps == 0) {

**return** mid + 1;

} **else** **if** (comps < 0) {

**return** *binarySearchString*(arr, mid + 1, right, target);

} **else** {

**return** *binarySearchString*(arr, left, mid - 1, target);

}

}

**public** **static** ArrayList<Integer> binarySearchAll(**int**[] arr, **int** left, **int** right, **int** target){

**int** ind = *binarySearch*(arr, left, right, target);

**int** lft = ind;

**int** rgt = ind;

ArrayList<Integer> sol = **new** ArrayList<>();

**if** (ind == -1)

**return** sol;

**while** (lft >= left && arr[lft] == target)

lft--;

**while** (rgt < right && arr[rgt] == target)

rgt++;

**for** (**int** i = lft + 1; i < rgt; i++){

sol.add(i + 1);

}

**return** sol;

}

**public** **static** **void** main(String[] args) {

**int**[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};

String[] arr2 = {"aaaaa", "abbbb", "acccc"};

**int** target = 5;

**int** result = *binarySearch*(arr, 0, arr.length - 1, target);

System.***out***.println(result);

result = *binarySearchString*(arr2, 0, arr2.length-1, "ball");

System.***out***.println(result);

**int**[] arr3 = {1, 1, 2, 2, 2, 2, 2, 2, 3, 3};

ArrayList<Integer> result2 = *binarySearchAll*(arr3, 0, arr3.length-1, 2);

System.***out***.println(result2);

}

}

### TestCases:

**package** test;

**import** org.junit.jupiter.api.Test;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** java.util.ArrayList;

**public** **class** BinarySearchTest {

// Test case for binarySearch with integers

@Test

**void** testBinarySearchIntegerFound() {

**int**[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};

**int** target = 5;

**int** result = BinarySearch.*binarySearch*(arr, 0, arr.length - 1, target);

*assertEquals*(5, result, "Expected index for 5 is 5");

}

@Test

**void** testBinarySearchIntegerNotFound() {

**int**[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};

**int** target = 10;

**int** result = BinarySearch.*binarySearch*(arr, 0, arr.length - 1, target);

*assertEquals*(-1, result, "Expected index for 10 is -1 (not found)");

}

// Test case for binarySearch with strings

@Test

**void** testBinarySearchStringFound() {

String[] arr = {"aaaaa", "abbbb", "acccc"};

String target = "abbbb";

**int** result = BinarySearch.*binarySearchString*(arr, 0, arr.length - 1, target);

*assertEquals*(2, result, "Expected index for 'abbbb' is 2");

}

@Test

**void** testBinarySearchStringNotFound() {

String[] arr = {"aaaaa", "abbbb", "acccc"};

String target = "ball";

**int** result = BinarySearch.*binarySearchString*(arr, 0, arr.length - 1, target);

*assertEquals*(-1, result, "Expected index for 'ball' is -1 (not found)");

}

// Test case for binarySearchAll (multiple occurrences)

@Test

**void** testBinarySearchAllFound() {

**int**[] arr = {1, 1, 2, 2, 2, 2, 2, 2, 3, 3};

ArrayList<Integer> result = BinarySearch.*binarySearchAll*(arr, 0, arr.length - 1, 2);

ArrayList<Integer> expected = **new** ArrayList<>();

expected.add(3);

expected.add(4);

expected.add(5);

expected.add(6);

expected.add(7);

expected.add(8);

*assertEquals*(expected, result, "Expected indices for target 2 are 3 to 8");

}

@Test

**void** testBinarySearchAllNotFound() {

**int**[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};

ArrayList<Integer> result = BinarySearch.*binarySearchAll*(arr, 0, arr.length - 1, 10);

ArrayList<Integer> expected = **new** ArrayList<>();

*assertEquals*(expected, result, "Expected no occurrences for target 10");

}

// Edge case for an empty array

@Test

**void** testBinarySearchEmptyArray() {

**int**[] arr = {};

**int** target = 5;

**int** result = BinarySearch.*binarySearch*(arr, 0, arr.length - 1, target);

*assertEquals*(-1, result, "Expected index for target 5 is -1 (not found) in an empty array");

}

// Edge case for a single element array

@Test

**void** testBinarySearchSingleElementFound() {

**int**[] arr = {5};

**int** target = 5;

**int** result = BinarySearch.*binarySearch*(arr, 0, arr.length - 1, target);

*assertEquals*(1, result, "Expected index for target 5 is 1");

}

@Test

**void** testBinarySearchSingleElementNotFound() {

**int**[] arr = {10};

**int** target = 5;

**int** result = BinarySearch.*binarySearch*(arr, 0, arr.length - 1, target);

*assertEquals*(-1, result, "Expected index for target 5 is -1 (not found) in a single element array");

}

}

### 

### **Time Complexity:** Time complexity of Binary Search is O(Log n)

### Both iterative and recursive have the same time complexity

### Task 2:

### Code:

**package** test;

**import** java.util.List;

**import** java.util.ArrayList;

**public** **class** Parser {

**public** **static** **int** multiply(String expression){

**int** num1 = expression.charAt(0) - '0';

**if** (expression.length() == 1)

**return** num1;

**char** oper = expression.charAt(1);

**if** (oper == '\*'){

**return** num1 \* *multiply*(expression.substring(2, expression.length()));

} **else** {

**return** num1 / *multiply*(expression.substring(2, expression.length()));

}

}

**public** **static** **int** addition(String expression){

**if** (expression.length() == 1){

**int** num1 = expression.charAt(0) - '0';

**return** num1;

}

**int** x = 0;

**while** (x < expression.length() && (expression.charAt(x) != '+' && expression.charAt(x) != '-')){

x++;

}

**if** (x == expression.length()){

**return** *multiply*(expression);

}

**if** (expression.charAt(x) == '+'){

**return** *addition*(expression.substring(0, x))

+ *addition*(expression.substring(x + 1, expression.length()));

} **else** {

**return** *addition*(expression.substring(0, x))

- *addition*(expression.substring(x + 1, expression.length()));

}

}

**public** **static** **int** evaluateExpression(String expression){

expression = expression.replaceAll("\\s+", "");

**return** *addition*(expression);

}

**public** **static** **void** main(String[] args) {

System.***out***.println(*evaluateExpression*("5+3\*5/5"));

}

}

### TestCases:

**package** test;

**import** org.junit.jupiter.api.Test;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**public** **class** ParserTest {

// Test case for multiplication in the expression

@Test

**void** testMultiply() {

*assertEquals*(15, Parser.*multiply*("3\*5"), "Expected result of 3\*5 is 15");

*assertEquals*(2, Parser.*multiply*("4/2"), "Expected result of 4/2 is 2");

}

// Test case for addition in the expression

@Test

**void** testAddition() {

*assertEquals*(8, Parser.*addition*("3+5"), "Expected result of 3+5 is 8");

*assertEquals*(1, Parser.*addition*("5-4"), "Expected result of 5-4 is 1");

*assertEquals*(0, Parser.*addition*("4-4"), "Expected result of 4-4 is 0");

}

// Test case for evaluateExpression (combination of addition, multiplication, and division)

@Test

**void** testEvaluateExpression() {

*assertEquals*(8, Parser.*evaluateExpression*("3+5"), "Expected result of 3+5 is 8");

*assertEquals*(8, Parser.*evaluateExpression*("3+5\*5/5"), "Expected result of 3+5\*5/5 is 7");

*assertEquals*(2, Parser.*evaluateExpression*("4/2"), "Expected result of 4/2 is 2");

*assertEquals*(8, Parser.*evaluateExpression*("5+3\*5/5"), "Expected result of 5+3\*5/5 is 11");

}

// Test case for expressions with spaces

@Test

**void** testEvaluateExpressionWithSpaces() {

*assertEquals*(8, Parser.*evaluateExpression*("3 + 5 \* 5 / 5"), "Expected result of 3 + 5 \* 5 / 5 is 7");

*assertEquals*(8, Parser.*evaluateExpression*("3 + 5"), "Expected result of 3 + 5 is 8");

}

// Test case for edge case: single number

@Test

**void** testSingleNumber() {

*assertEquals*(3, Parser.*evaluateExpression*("3"), "Expected result for single number 3 is 3");

*assertEquals*(1, Parser.*evaluateExpression*("1"), "Expected result for single number 10 is 10");

}

// Test case for complex expression

@Test

**void** testComplexExpression() {

*assertEquals*(18, Parser.*evaluateExpression*("5+3\*2+7"), "Expected result of 5+3\*2+7 is 16");

}

}

### 

### **Task 3:**

### **Code:**

**package** test;

**import** java.util.ArrayList;

**public** **class** SumOfDigits {

**public** **static** **int** sumofdigits(**int** num){

**if** (num == 0)

**return** 0;

**if** (num < 0) num \*= -1;

**int** num\_divide = num / 10;

**int** num\_mod = num % 10;

**return** (*sumofdigits*(num\_divide) + num\_mod);

}

**public** **static** **void** main(String[] args) {

System.***out***.println(*sumofdigits*(567));

}

}

### TestCases:

**package** test;

**import** org.junit.jupiter.api.Test;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**public** **class** SumOfDigitsTest {

// Test case for a positive number

@Test

**void** testPositiveNumber() {

*assertEquals*(18, SumOfDigits.*sumofdigits*(567), "Sum of digits of 567 should be 18");

}

// Test case for a single digit number

@Test

**void** testSingleDigit() {

*assertEquals*(5, SumOfDigits.*sumofdigits*(5), "Sum of digits of 5 should be 5");

}

// Test case for zero

@Test

**void** testZero() {

*assertEquals*(0, SumOfDigits.*sumofdigits*(0), "Sum of digits of 0 should be 0");

}

// Test case for a negative number

@Test

**void** testNegativeNumber() {

*assertEquals*(18, SumOfDigits.*sumofdigits*(-567), "Sum of digits of -567 should be 18");

}

// Test case for a very large number

@Test

**void** testLargeNumber() {

*assertEquals*(45, SumOfDigits.*sumofdigits*(123456789), "Sum of digits of 123456789 should be 45");

}

// Test case for a large negative number

@Test

**void** testLargeNegativeNumber() {

*assertEquals*(45, SumOfDigits.*sumofdigits*(-123456789), "Sum of digits of -123456789 should be 45");

}

// Test case for a single digit zero

@Test

**void** testSingleDigitZero() {

*assertEquals*(0, SumOfDigits.*sumofdigits*(0), "Sum of digits of 0 should be 0");

}

}

### Time Complexity: Timie complexity is O(Log10N)

### Deliverables:

Compile a single word document by filling in the solution part and submit this Word file on LMS.

In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to [aftab.farooq@seecs.edu.pk.](mailto:aftab.farooq@seecs.edu.pk.)