1. Homework 2

Posted: May/10/2018
Due: September/10/2018 24.00

All homework solutions are due September/10/2018 24.00. I recommend to submit at least one version of all homework solutions long before due date.

1.1. Homework 2.1 (10 Points)

Objective: Modification of an existing algorithm.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:
Given a set of sticks S, with | S | = *n*. The set of the lengths of the sticks are:

$$sl = \{s_{1,} \cdots s_n\}$$

Given is a stick *s_new*, with length *l*. Does a combination of elements of

$$\{s_{1,} \cdots s_n\}$$

exist so such

$$l = \sum_{i=1}^{k} s_i ; 1 \leq k \leq n$$

If such a set exist print out this set. If more than one set exist print out the smallest set.

Explanation:
Assume the set of sticks is: { 1, 2, 3, 5 }, then
if *s_new* == 5 then the set { 5 } would be printed.
if *s_new* == 4 then the set { 1, 3 } would be printed.
if *s_new* == 8 then the set { 3, 5 } would be printed.

Your Work:
Write a program to solve the described problem for all cases. You can hard code the sticks in your program. See the code snippet below.

Requirements:
Create a test environemnt which covers all boundery cases.

You have to name the file: Sticks.java

- You can use arrays, and you have to use an iterative algorithm.
- You can hardcode all values in your program.
- You can use basic types and arrays.
- You can not use any publicly available class or library which can determine if a particular set of sticks matches the length.

A snippet of the code might look like this:

```
public class Sticks {
 static int[] stickLengths = { 1, 2, 3, 5, 8 };
    static int soManySticks = stickLengths.length;
    static int[] unknowStickLengths = { 1, 5, 6, 7, 8, 9 };
```

```
  public static void main( String[] arguments ) {
    for ( int index = 0; index < unknowStickLengths.length; index ++ )
        doTestLength(unknowStickLengths[index]);
  }
}
```

Example:

An example of a solution execution:

```
% java Sticks
1 inch: yes; used stickLengths = 1 inch
5 inch: yes; used stickLengths = 5 inch
6 inch: yes; used stickLengths = 5 inch 1 inch
7 inch: yes; used stickLengths = 5 inch 2 inch
8 inch: yes; used stickLengths = 8 inch
9 inch: yes; used stickLengths = 8 inch 1 inch
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab2-1 ’All files required’
# you can see if your submission was successful:
# try -q hpb-grd lab2-1
```

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
public class Sticks {
  static int[] stickLengths = { 1, 2, 3, 4, 8, 9 };
  // static int[] stickLengths = { 1, 2, 3, 4, 8, 9 };
  static int[] unknowStickLengths = { 5 };


  private static String theFollowingSticksAreUsed (int value) {
    String returnValue = "";
    for ( int index = stickLengths.length; index >= 0 ; index --)   {
        if ( ( ( 1 << index ) & value ) == ( 1 << index ) )
            returnValue += stickLengths[index] + " inch ";

    }
    returnValue = returnValue.trim();        // remove trailing ’ ’
    if ( returnValue == "" )
        returnValue = "empty set";
    return returnValue;
  }

  private static int calculteLengthForThisSet(int value) {
    int sum = 0;
    for ( int index = stickLengths.length; index >= 0 ; index --)    {
```

```
                    if ( ( ( 1 << index ) & value ) == ( 1 << index ) )     {
                            sum += stickLengths[index];
                    }
            }
            return sum;
        }
        private static int countOnBits (int value) {
            int soManyBitsOn = 0;
            for ( int index = stickLengths.length; index >= 0 ; index --)   {
                    if ( ( ( 1 << index ) & value ) == ( 1 << index ) )
                            soManyBitsOn++;
            }
            return soManyBitsOn;
        }
        private static void  doTestLength(int thisLength)   {
            int setSize         = (int)Math.pow(2, stickLengths.length);
            boolean foundAset  = false;
            int smallestSet     = ˜0;       // -1 and modify if in loop
            String theLongestSet = null;

            int index = 0;

            while ( ( index < setSize ) ) {
                    if (  thisLength == calculteLengthForThisSet(index) )   {
                            if ( countOnBits(index) == countOnBits(smallestSet) )   {
                                    theLongestSet = theLongestSet + "; " + theFollowin
                            }
                            if ( countOnBits(index) < countOnBits(smallestSet) )    {
                                    smallestSet = index;
                                    theLongestSet = theFollowingSticksAreUsed(index);
                            }
                    }
                    index ++;
            }
            if ( theLongestSet != null )    {
                    System.out.println(thisLength + " inch: "       +
                                      "\tyes; used stickLengths = " +
                                      theLongestSet);
            } else
                    System.out.println(thisLength + " inch: \tno");
        }
        public static void main( String[] arguments ) {
            for ( int index = 0; index < unknowStickLengths.length; index ++ )
                    doTestLength(unknowStickLengths[index]);
        }
    }
```

Source Code: Src/22_sol/Sticks.java

1.2. Homework 2.2 (10 Points)

Objective: Modification of an existing algorithm.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed

Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

Modify the original shown program HW 1.3 so such the numerical expression can include '(', ')', '[', ']', '{', '}'. All operations are performed on integers with integer results.

The original calculator code:

```
import java.util.Vector;
import java.util.Stack;

public class Calculator {

        // See https://docs.oracle.com/javase/10/docs/api/java/util/Stack.html
    static Stack<Double> numberStack = new Stack<Double>();
    static Stack<String> operatorStack = new Stack<String>();
        // See https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
    static String operators =  "+-%*/^" ;

    public static void main (String args []) {
        performCalculation("2", "+", "3");
        performCalculation("2", "+", "3", "*", "3");
        performCalculation("2", "*", "3", "+", "3");
        performCalculation("2", "+", "3", "^", "4");
        performCalculation("2", "^", "3", "+", "4");
        performCalculation("2", "^", "3", "^", "4");
    }

    // See https://docs.oracle.com/javase/8/docs/technotes/guides/language/varargs
    public static void performCalculation (String ... valuesAndOperators)       {
        Vector<String> aLine = new Vector<String>();
        for ( String valuesAndOperator: valuesAndOperators )    {
                aLine.add(valuesAndOperator);
                System.out.print(valuesAndOperator + " ");
        }
        System.out.println(" =  " + calculate(aLine) );
    }
    /** drives the calculation and returns the result
     */
    public static double calculate (Vector<String> inputLine) {
        while ( inputLine.size() >= 1 ) {
                if ( operator( inputLine.firstElement() )       )
                        performOperator(inputLine.firstElement());
                else
                        performNumber(inputLine.firstElement());

                inputLine.removeElementAt(0);
        }
        while ( !  operatorStack.empty() )      {
                if ( numberStack.size() > 1 )
                        evaluate();
                else    {
                        System.out.println("dangling operand ....");
```

```
                    System.out.println(numberStack);
                    System.exit(1);

            }
    }

    return numberStack.pop();
}

/** perform the required operation based on precedence of the operators on the
 */
public static boolean operator (String op) {
    return ( operators.indexOf(op) >= 0 );
}

/** deteremence a precedence level for the operator
 */
public static int precedence (String op) {
    return operators.indexOf(op);
}

/** perform the required operation based on precedence on the stack
 */
public static void performOperator (String op) {
            while (! operatorStack.empty()  &&
                    (  precedence(op) < precedence(operatorStack.peek() ) )
                  )
                            evaluate();
            operatorStack.push(op);
}

/** pushes the number on the number stack
 */
public static void performNumber (String number) {
            numberStack.push(Double.valueOf(number));
}

/** get the number of the stack, if a number is available, else RIP
 */
public static double  getNumber () {
    if ( numberStack.empty() ){
            System.out.println("not enough numbers ...");
            System.exit(2);
    }
    return numberStack.pop();
}

/** perform the required ovperation based on the operator in the stack
 */
public static void evaluate () {
            String currentOp = operatorStack.pop();
            double right = getNumber();
            double left = getNumber();
            if ( currentOp.equals("+") )
```

```
                numberStack.push( left + right );
            else if ( currentOp.equals("-") )
                numberStack.push( left - right );
            else if ( currentOp.equals("*") )
                numberStack.push( left * right );
            else if ( currentOp.equals("%") )
                numberStack.push( left % right );
            else if ( currentOp.equals("/") )
                numberStack.push( left / right );
            else if ( currentOp.equals("^") )
                numberStack.push( Math.pow(left , right ) );
            else
                System.out.println("Unknow Operator");
      }
    }
```

Source Code: Src/21/Calculator.java

My main program looks like:

```
public class Calculator {

        // See https://docs.oracle.com/javase/10/docs/api/java/util/Stack.html
    static Stack<Double> numberStack = new Stack<Double>();
    static Stack<String> operatorStack = new Stack<String>();
        // See https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
    static String operators =  "+-%*/^" ;
    static String openingP[] = { "(", "[", "{" };
    static String closingP[] = { ")", "]", "}" };

    public static void main (String args []) {
        performCalculation("1", "*", "{", "2", "+", "3", "-",
                           "[", "1", "*", "(", "2", "-", "1", ")", "]", "+", "3", "}");
        performCalculation("2", "+", "[", "(", "3", "-", "6", ")", "/", "5", "]");
        performCalculation("1", "+", "(", "2", "+", "3", ")", "*", "3");
        performCalculation("2", "^", "3","^", "4");
        performCalculation("(", "2", "^", "3", ")", "^", "4");
    }
    ...
```

Explanation:
The paranthesis of the numerical exporessions have to match pairwise. You can assume all expresstion are valid. A few examples and their stepwise calculation:

- 2 * (3 + 5) = 2 * 8 = 16
- 2 + [(11 - 6) / 5] = 2 + [-5 / 5] = 2 - 1 = 1
- 2 ^ 3 ^ 4 = 2 ^ 81 = 2417851639229258349412352

1 - (2 + [+ 2 -) - 3] would be an illegal expression.

Requirements:

- You have to name your code: Calculator.java
- You can hard code all expression as shown in the main program.
- You can assume all numerical expressions are valid.
- How can you show that your code is correct? Which kind of test cases did you implement?

Example:

An example of a solution execution:

```
% java Calculator

1 * { 2 + 3 - [ 1 * ( 2 - 1 ) ] + 3 }  =  7.0
2 + [ ( 3 - 6 ) / 5 ]  =  2.0
1 + ( 2 + 3 ) * 3  =  16.0
2 ˆ 3 ˆ 4  =  2.4178516392292583E24
( 2 ˆ 3 ) ˆ 4  =  4096.0
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab2-2 ’All files required’
# you can see if your submission was successful:
# try -q hpb-grd lab2-2
```

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
import java.util.Vector;
import java.util.Stack;

public class Calculator {

        // See https://docs.oracle.com/javase/10/docs/api/java/util/Stack.html
    static Stack<Double> numberStack = new Stack<Double>();
    static Stack<String> operatorStack = new Stack<String>();
        // See https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
    static String operators =  "+-%*/ˆ" ;
    static String openingP[] = { "(", "[", "{" };
    static String closingP[] = { ")", "]", "}" };

    public static void main (String args []) {
        performCalculation("2", "+", "3");
        performCalculation("1", "*", "{", "2", "+", "3", "-",
                               "[", "1", "*", "(", "2", "-", "1", ")", "]", "+", "
        performCalculation("2", "+", "[", "(", "3", "-", "6", ")", "/", "5", "]");
        performCalculation("1", "+", "(", "2", "+", "3", ")", "*", "3");
        performCalculation("2", "*", "3", "+", "3");
        performCalculation("2", "*", "(", "3", "+", "3", ")");
        System.out.println();
    }

    public static String printExpression(Vector<String> theExpression)  {
        String returnV = "";
        for ( int index = 0; index < theExpression.size(); index ++ )
                returnV = returnV + " " + theExpression.elementAt(index);
```

```
            return returnV;
        }
        public static int findRightP(Vector<String> expression, String right )      {
            int index = -1;         // first index ++ in loop
            boolean rValue = false;
            while ( ( index < expression.size() - 1 ) && ! rValue ) {
                    index ++;
                    rValue = expression.elementAt(index).equals(right) ;

            }
            return ( rValue ? index : -1 );
        }
        public static int findLeftP(Vector<String> expression, String left )        {
            int index = expression.size();          // first index -- in loop
            boolean rValue = false;
            while ( ( index > 0 ) && ! rValue )     {
                    index --;
                    rValue = expression.elementAt(index).equals(left) ;

            }
            return ( rValue ? index : -1 );
        }
        public static void replaceMostInnerP(Vector<String> expression, String left, S
            Vector<String> localExpression = new Vector<String>();
            int leftPosition  = findLeftP(expression, left);
            int rightPosition = findRightP(expression, right);

            for ( int index = leftPosition + 1; index < rightPosition ; index ++ )  {
                    localExpression.add(expression.elementAt(leftPosition + 1));
                    expression.removeElementAt(leftPosition + 1);
            }
            expression.removeElementAt(leftPosition);               // remaing opening
            expression.removeElementAt(leftPosition);               // remaing closing

            int value = (int)calculate(localExpression);
            expression.add(leftPosition, "" + value);
        }
        // See https://docs.oracle.com/javase/8/docs/technotes/guides/language/varargs
        public static void performCalculation (String ... valuesAndOperators)       {
            Vector<String> expression = new Vector<String>();
            int left;
            int right;
            System.out.println();
            for ( String valuesAndOperator: valuesAndOperators )    {
                    expression.add(valuesAndOperator);
                    System.out.print(valuesAndOperator + " " );
            }
            for ( int index = 0; index < openingP.length; index ++ )        {
                    while ( ( left = findLeftP( expression, openingP[index] ) ) >= 0)
                            replaceMostInnerP(expression, openingP[index], closingP[in
                    }
            }
            System.out.print(" =  " + calculate(expression) );
        }
        /** drives the calculation and returns the result
         */
        public static double calculate (Vector<String> expression ) {
            while ( expression.size() >= 1 )        {
                    if ( operator( expression.firstElement() )      )
                            performOperator(expression.firstElement());
                    else
                            performNumber(expression.firstElement());

                    expression.removeElementAt(0);
            }
            while ( !  operatorStack.empty() )      {
                    if ( numberStack.size() > 1 )
                            evaluate();
                    else    {
                            System.out.println("dangling operand ....");
                            System.out.println(numberStack);
                            System.exit(1);

                    }
            }

            return numberStack.pop();
        }

        /** perform the required operation based on precedence of the operators on the
         */
        public static boolean operator (String op) {
            return ( operators.indexOf(op) >= 0 );
        }

        /** deteremence a precedence level for the operator
         */
        public static int precedence (String op) {
            return operators.indexOf(op);
        }

        /** perform the required operation based on precedence on the stack
         */
        public static void performOperator (String op) {
                    while (! operatorStack.empty()  &&
                            (  precedence(op) < precedence(operatorStack.peek() ) )
                          )
                                    evaluate();
                    operatorStack.push(op);
        }

        /** pushes the number on the number stack
         */
        public static void performNumber (String number) {
                    numberStack.push(Double.valueOf(number));
        }

        /** get the number of the stack, if a number is available, else RIP
```

```
     */
    public static double  getNumber () {
        if ( numberStack.empty() ){
                System.out.println("not enough numbers ...");
                System.exit(2);
        }
        return numberStack.pop();
    }

    /** perform the required ovperation based on the operator in the stack
     */
    public static void evaluate () {
                String currentOp = operatorStack.pop();
                double right = getNumber();
                double left = getNumber();
                if ( currentOp.equals("+") )
                        numberStack.push( left + right );
                else if ( currentOp.equals("-") )
                        numberStack.push( left - right );
                else if ( currentOp.equals("*") )
                        numberStack.push( left * right );
                else if ( currentOp.equals("%") )
                        numberStack.push( left % right );
                else if ( currentOp.equals("/") )
                        numberStack.push( left / right );
                else if ( currentOp.equals("^") )
                        numberStack.push( Math.pow(left , right ) );
                else
                        System.out.println("Unknow Operator");
    }
}
```

Source Code: Src/22_sol/Calculator.java

1.3. Homework 2.3 (10 Points)

Objective: Understanding strings

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

This program

```
class StringThing {
  public static void method(String id, String literal, String aNewString)       {
        System.out.println(id + ".      " + (literal == aNewString ));
  }
  public static void main( String args[] ) {
        String aString = "123";
        String bString = "123";
```

```
            String cString = "1" + "23";
            int number = 3;
            System.out.println("a.  " +     "123" == aString   );
            System.out.println("b.  " +   ( "12" + number == aString ) );
            System.out.println("c.  " +   aString  + 1 * 23 );
            System.out.println("d.  " +   123 + number + aString  );
            System.out.println("e.  " +   ( 123 + number ) + aString   );
            System.out.println("f.  " +   ( 123 - 2 + "" +  number + aString )  );
            System.out.println("g.  " +   123 * number + aString  );
            System.out.println("h.  " +   123 / number + aString  );
            System.out.println("i.  " +   ( 123 - number )  + aString  );
            System.out.println("j.  " +     ( "123" == aString )   );
            System.out.println("g.  " +     ( "a" + "a" == "aa"  )   );
            System.out.println("h.  " +     ( "123" == cString )   );
            System.out.println("1." + "x" == "x");

            method("1", "xyz", "x" + "yz");
            method("2", "xyz", new String("x") + "yz" );
            method("3", "xyz", "x" + "y" +"z");
            method("4", "1" + "2" + "3", "1" + 2 * 1 + 3);
            method("5", "1" + "2" + "3", "1" + 2 + 3);
            method("6", "1" + "2" + "3", "1" + (3 - 1)  + 3);
      }
    }
```

` Source Code: Src/22/StringThing.java`

produces the following output:

```
false
b.  false
c.  12323
d.  1233123
e.  126123
f.  1213123
g.  369123
h.  41123
i.  120123
j.  true
g.  true
h.  true
1.  true
2.  false
3.  true
4.  true
5.  true
6.  true
```

Explanation:
You should not run the code in order to conclude the answer. You should use the knowledge you have, without executing the code. Java evaluates expressions strictly from left to right. Keep this in mind when you think about line a). *"1." + "x" == "x"* becomes *"1.x" == "x"*, which is false. The compiler will create literals if possible.

Your Work:
You have to answer the following questions by adding comments to the code.

- Explain the output.
- How many strings will be generated in the lines marked a) ... j) and 1. ... 6? You need to be able to explain your answer.
- When is the earliest moment the garbage collector can free up memory for each of the lines marked a) ... j) and 1. ... 6? You need to be able to explain your answer.

Requirements:
You have to name the file *StringThing.java*. Add your answers to the program and submit the modified program as your solution.

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab2-3 'All files required'
# you can see if your submission was successful:
# try -q hpb-grd lab2-3
```