CSA 1017
Data Structures and Algorithms 1
Assignment

matrices Mark Said Camilleri
B.Sc. (Hons) (Computing Science)
University of Malta

Friday 4th May 2016

Contents

Statement of Completion

The questions below were the ones that have been attempted:

Question 1 This question has been successfully completed.

Question 2 This question has been successfully completed.

Question 3 This question has been successfully completed.

Question 4 This question has been successfully completed.

Question 5 This question has been successfully completed.

Question 6 This question has been successfully completed.

Question 7 This question has been successfully completed.

Question 8 This question has been successfully completed.

Question 9 This question has been successfully completed.

Signature

Date

Task 1

Arabic to Roman Numeral Converter

Input	Expected Output	Actual Output
1	I	I
2	II	II
4	IV	IV
5	V	V
6	VI	IV
9	IX	IX
10	X	X
20	XX	XX
40	XL	XL
49	XLIX	XLIX
50	L	L
60	LX	LX
90	XC	XC
99	XCIX	XCIX
100	C	C
150	CL	CL
400	CD	CD
499	CDXCIX	CDXCIX
500	D	D
600	DC	DC
900	CM	CM
999	CMXCIX	CMXCIX
1000	M	M
1024	MXXIV	MXXIV
2000	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:
abc	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:

1.1 The source code for Task 1

```
/**
 * @author Mark Said Camilleri
 * @version 20160509
 */


import java.util.InputMismatchException;
import java.util.Scanner;


public class Question1 {

    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in);
        in.useDelimiter("\n");

        /*======================== WELCOME MESSAGE TO USER =======================*/
        System.out.println(" ----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println(" |---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri          |");
        System.out.println("|     Task 1: Arabic to Roman Numeral Converter     |");
        System.out.println(" |---------------------------------------------------|");
        System.out.print("| Please enter a number between 1 and 1024: ");


        int toConvert = 0; //value to be converted.
        boolean isError; // temporary boolean value used for error checking of the input.
        do {
            isError = false;
            try {
                toConvert = in.nextInt();
            } catch (InputMismatchException e) {
                isError = true;
                in.next(); //To clear the buffer
            }
            /*============ Makes sure input is a number is between 1 and 1024 =============*/
            if (isError || toConvert < 1 || toConvert > 1024) {
                /*==================== OUTPUT ERROR MESSAGE TO THE USER ==================*/
                System.out.println(" |---------------------------------------------------|");
                System.out.println("|The input was not a valid number
   between 1 and 1024|");
                 System.out.print("|Please try again and enter a number
   to convert: ");
            }
        } while (isError || toConvert < 1 || toConvert > 1024);

        System.out.printf("| %4d = %-24s in Roman Numerals |", toConvert
   , convert(toConvert));

    }

    /**
     * Takes an int decimal value and outputs a string of the same value
    in Roman Numerals.
     *
     * @param toConvert the decimal value to ve converted to Roman
   Numerals
     * @return The roman numeral equivalent of the input parameter
     */
    private static String convert(int toConvert) {

        //Defining the decimal and roman counterparts
        final int dec[] = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500,
   900, 1000};
        final String rom[] = {"I", "IV", "V", "IX", "X", "XL", "L", "XC"
   , "C", "CD", "D", "CM", "M"};

        /* Begins by checking the input paramerer against the largest
   roman numeral/numeral pair.
         * and works it's way down to the unit numeral.
         */
        for (int i = dec.length - 1; i >= 0; i--) {
            //If the value is larger, then the output is concatenated
   with the output of the difference.
            if (toConvert >= dec[i])
                return rom[i] + convert(toConvert - dec[i]);
        }
        return ""; //What to return at 0, the base case.
    }
}
```

Task 2

Reverse Polish Notation Evaluator

Input	Expected Output	Actual Output
4 5 +	9.0	9.0
12 6 -	6.0	6.0
3 2 /	1.5	1.5
7 2 *	14.0	14.0
23 85 + 92 *	9936.0	9936.0
43.5 3.2 - 4.5 * 3.24 +	184.59	184.59
matrices 34.8 62.11 * -76 /	-28.43984211	-28.439842105263157
282 -56 * 1024 - -55.6 *	934969.6	934969.6
Test	Your expression contained invalid characters. For input string ”T” Your expression is invalid. Evaluation failed	Your expression contained invalid characters. For input string ”T” Your expression is invalid. Evaluation failed
1 +	Stack is Empty. Your expression is invalid. Evaluation failed.	Stack is Empty. Your expression is invalid. Evaluation failed.
3 64 6 +	The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed.	The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed.
4 0 /	Infinity	Infinity

2.1 The source code for Task 2

2.1.1 Stack Class

```java
import java.util.ArrayList;
import java.util.Collection;
import java.util.EmptyStackException;

/**
 * Created by mark on 14/02/16.
 *
 * A stack implemented as an ArrayList to have it dynamically increase its size.
 */
public class Stack<E> extends ArrayList {

    /**
     * Default constructor. Calls the ArrayList default constructor
     */
    public Stack() {
        super();
    }

    /**
     * Initialised a stack with the contents of the Collection in the parameter.
     *
     * @param c the contents to initialise the stack with.
     */
    public Stack(Collection<? extends E> c) {
        super(c);
    }

    /**
     * Pushes the data onto the stack
     *
     * @param data data to be pushed on the stack
     * @throws IndexOutOfBoundsException if not sucessfully pushed
     */
    public void push(E data) throws IndexOutOfBoundsException {
        int prevSize = this.size();
        this.add(data);

        //Condition to check if the data has been successfully added.
        if (!(this.size() == prevSize + 1))
            throw new IndexOutOfBoundsException("Failed to push to stack");
    }

    /**
     * Pops the topmost item from the stack.
     * @return the data from the top of the stack is not empty.
     * @throws EmptyStackException if the stack is empty.
     */
    public E pop() throws ArrayIndexOutOfBoundsException {
        if (this.size() == 0) throw new EmptyStackException();
        else return (E) this.remove(this.size() - 1);
    }

    /**
     * Returns the data at the top of the stack without popping it.
     * @return the data if the stack is not empty. null if it is empty.
     */
    public E peek() {
        if (this.size() == 0) return null;
        else return (E) this.get(this.size() - 1);
    }

    /**
     * Returns a string representation of the contents of the stack.
     * @overrides toString() in class AbstractCollection<E>
     * @return The string representation of the ArrayList if not empty. "Stack is empty" if it is empty.
     */
    public String toString() {
        if (this.size() == 0) return "Stack is empty";
        else return super.toString();
    }
}
```

2.1.2 Question 2 Main Class

```java
import java.util.EmptyStackException;
import java.util.Scanner;

/**
 * Created by mark on 09/02/16.
 * This answer assumes the RPN input is correct.
 */
public class Question2 {
    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in).useDelimiter("\n");

        /*======================= WELCOME MESSAGE TO USER ======================*/
        System.out.println("---------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println("|-------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri         |");
        System.out.println("|     Task 2: Reverse Polish Notation evaluator     |");
        System.out.println("|-------------------------------------------------|");
        System.out.println("|     Note: This program can only do +,-,* and /    |");
        System.out.print("| Please enter an expression to evaluate: "); //prompt for user input. Assumes it is rpn.

        /* Initialized a stack object (using the stack defined here).
         * Note, no importing of the Stack class.
         */
        Stack<Double> nums = new Stack<>();

        //Reads user input. Must be a valid RPN expression.
        String expression = in.next();
        System.out.println("
    |--------------------------------------------------|");
        System.out.println("| Contents of the stack at each step:
           |");//some message to user.

        boolean exceptionRaised = false; //used for error checking.
        try {
            //Iterates through the string inputted by the user.
            for (int i = 0; i < expression.length(); i++) {
                char cChar = expression.charAt(i);

                /* If the current character is a space,
                 * nothing needs to be done.
                 */
                if (Character.isWhitespace(cChar)) continue;

                /* If it’s a ’+’, then 2 numbers are popped, added and
                 * the answer is pushed onto the stack.
                 */
                else if (cChar == ’+’) {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 + num1);
                }
                /* If it’s a ’-’, then 2 numbers are popped,
                 * subtracted and the answer is pushed onto the stack.
                 * The conjunction is to make sure that it’s not
                 * detecting a negative number. The disjunction is
                 * true if the ’-’ is at the end of the string or
                 * there is a space after it. These both make sure
                 * that the ’-’ does not belong to a negative number
                 */
                else if (cChar == ’-’ && (i == expression.length() - 1
    || Character.isWhitespace(expression.charAt(i + 1)))) {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 - num1);
                }
                /* If it’s a ’*’, then 2 numbers are popped,
                 * multiplied and the answer is pushed onto the stack.
                 */
                else if (cChar == ’*’) {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 * num1);
                }
                /* If it’s a ’/’, then 2 numbers are popped,
                 * divided and the answer is pushed onto the stack.
                 */
                else if (cChar == '/') {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 / num1);
                }
            /*
             * Otherwise, assuming it's inputted correctly, the
             * character must be a number. In which case it is
             * converted ot a double (allowing for any real number
             * to be inputted) and pushed onto the stack.
             */
                else {
                    int start = i++;
                    while (Character.isDigit(expression.charAt(i)) ||
   expression.charAt(i) == '.')
                        i++;

                    nums.push(Double.parseDouble(expression.substring(
   start, i)));
                }
                System.out.printf("| %-49s |\n", nums.toString());
            }
            if (nums.size() != 1) {
                System.out.println("| The stack has not been emptied.
   There are too      |");
                System.out.println("| many operands in your expression.
                   |");
                exceptionRaised = true;
            }

        } catch (IndexOutOfBoundsException ioobe) {
            System.out.printf("| %-49s |\n", ioobe.getMessage());
            exceptionRaised = true;
        } catch (EmptyStackException ese) {
            System.out.println("| Stack is Empty.
               |");
            exceptionRaised = true;
        } catch (NumberFormatException nfe) {
            System.out.println("| Your expression contained invalid
   characters.     |");
            System.out.printf("| %-49s |\n", nfe.getMessage());
            exceptionRaised = true;
        } finally {
            if (exceptionRaised) {
                System.out.println("| Your expression is invalid.
   Evaluation failed.     |");
                System.out.println("

   ");
                System.exit(1);
            }
        }
        /*When the above iteration is complete, there should only be
         * one item on the stack which is the answer.
         */
```

```
        System.out.println("|--------------------------------------------------|");
        System.out.printf("| Answer of Evaluation = %-26s |\n", nums.pop().toString());
        System.out.println("

");
    }
}
```

Task 3

Prime Numbers

3.1 Prime Number Checker using divisibility check

Input	Expected Output	Actual Output
1	1 is not a prime number	1 is not a prime number
2	2 is a prime number	2 is a prime number
3	3 is a prime number	3 is a prime number
4	4 is not a prime number	4 is not a prime number
-2	-2 is not a prime number	-2 is not a prime number
a 677	677 is a prime number	677 is a prime number
34939	34939 is a prime number	34939 is a prime number
188737204	188737204 is not a prime number	188737204 is not a prime number
381165334	381165334 is not a prime number	381165334 is not a prime number
947396057	947396057 is a prime number	947396057 is a prime number
9223372036854775807	9223372036854775807 is not a prime number	9223372036854775807 is not a prime number
-9223372036854775808	-9223372036854775808 is not a prime number	-9223372036854775808 is not a prime number
9223372036854775808	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.
Test	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.

3.1.1 The source code for 3.1

```
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * @author Mark Said Camilleri
 * @version 20160511
 */

public class Question3a {

    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in).useDelimiter("\n");

        /*======================= WELCOME MESSAGE TO USER
    ======================*/
        System.out.println("
    ---------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and
    Algorithms 1    |");
        System.out.println("
    |-------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri
            |");
        System.out.println("|           Task 3.1: Prime Number Checker
            |");
        System.out.println("
    |-------------------------------------------------|");
        System.out.print("| Please enter an integer to check: ");

        try {
            //The number to check whether or not it's prime
            long prime = in.nextLong();

            System.out.println("
    |-------------------------------------------------|");
            if (isPrime(prime))
                System.out.printf("| %31d is a prime number |\n", prime)
    ;
            else
                System.out.printf("| %27d is not a prime number |\n",
    prime);

            System.out.println("

    ");
        } catch (InputMismatchException ime) {
            System.out.println("| Your input was not accepted. Please
    restart the   |");
            System.out.println("| program and try again with a valid
    input.        |");
            System.out.println("

    ");
            System.exit(1);
        }
    }

    /**
     * This method checks the input parameter to see if it is a prime
     * number or not.
     *
     * @param prime The number to check if it is prime
     * @return true if the number is prime, false if it isn't
     */
    private static boolean isPrime(long prime) {
        /*Firstly, if the number is 1, 0 or negative then it's not
         * prime
         */
        if (prime <= 1) return false;
            //If the number is 2, 2 is prime.
        else if (prime == 2) return true;
            //If the number is even and not 2, then it's not prime.
        else if (prime % 2 == 0) return false;
        /* Otherwise, divide this number by all odd numbers till the
         * square root of the number. If one is divisible then it's
         * not prime.
         */
        else {
            double root = Math.sqrt(prime);
            for (int i = 3; i <= root; i += 2) {
                if (prime % i == 0) return false;
            }
            return true;
        }
    }
}
```

3.2 Prime Number Checker using The Sieve of Eratosthenes

For this an `int` variable had to be used, due to it being the length of an array. As a result the range of numbers that can be checked was smaller.

Input	Expected Output	Actual Output
1	1 is not a prime number	1 is not a prime number
2	2 is a prime number	2 is a prime number
3	3 is a prime number	3 is a prime number
4	4 is not a prime number	4 is not a prime number
-2	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input
677	677 is a prime number	677 is a prime number
34939	34939 is a prime number	34939 is a prime number
188737204	188737204 is not a prime number	188737204 is not a prime number
381165334	381165334 is not a prime number	381165334 is not a prime number
947396057	947396057 is a prime number	947396057 is a prime number
2147483646	2147483646 is not a prime number	Requested array size exceeds VM limit Please restart the program and try again with a smaller input[1]
-2147483646	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input
9223372036854775807	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.
Test	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.

[1] This error depends on the amount of memory allocated for the program

3.2.1 The source code for 3.2

```
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * @author Mark Said Camilleri
 * @version 20160511
 */
public class Question3b {

    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in).useDelimiter("\n");

        /*======================== WELCOME MESSAGE TO USER
   =======================*/
        System.out.println("
   -----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and
   Algorithms 1    |");
        System.out.println("
   |---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri
           |");
        System.out.println("|          Task 3.2: Sieve of Eratosthenes
           |");
        System.out.println("
   |---------------------------------------------------|");
        System.out.print("| Please enter a positive integer to check: ")
   ;
        try {
        /*
         * Value to check if it's prime. Note that this is now an int
         * since the Sieve or Eratosthenes requires an array of this
         * amount of elements.
         */
            int prime = in.nextInt();

            System.out.println("
   |---------------------------------------------------|");
            if (sieve(prime))
                System.out.printf("| %31d is a prime number |\n", prime)
   ;
            else
                System.out.printf("| %27d is not a prime number |\n",
   prime);
            System.out.println("

   ");
        } catch (InputMismatchException ime) {
            System.out.println("| Your input was not accepted. Please
   restart the   |");
            System.out.println("| program and try again with a valid
   input.         |");
```

```
         System.out.println("

");
         System.exit(1);
     } catch (IllegalArgumentException iae) {
         System.out.printf("| %-49s |\n", iae.getMessage());
         System.out.println("| Please restart the program and try
again with a   |");
         System.out.println("| valid input.
            |");
         System.out.println("

");
         System.exit(1);
     } catch (OutOfMemoryError oome) {
         System.out.printf("| %-49s |\n", oome.getMessage());
         System.out.println("| Please restart the program and try
again with a   |");
         System.out.println("| smaller input.
            |");
         System.out.println("

");
         System.exit(1);
     }
 }

 /**
  * Runs the Sieve of Eratosthenes algorithm to check if the input
  * is a prime number or not.
  *
  * @param prime The number to check if it is prime or not.
  * @return true if the input is prime, false if it isn't.
  * @throws IllegalArgumentException if the input is <= 0
  */
 private static boolean sieve(int prime) throws
IllegalArgumentException, OutOfMemoryError {
     //Checks whether the input is valid or not
     if (prime <= 0)
         //Throws exception if input is not valid.
         throw new IllegalArgumentException("Sieve of Eratosthenes
only works on +ve integers.");
     else {
         //Each ith element is true if i-1 is prime. False otherwise.
         boolean[] nos = new boolean[prime];

         //We begin by assuming all the values are prime.
         Arrays.fill(nos, true);

         //Then, 1 is crossed out since it is a square.
         nos[0] = false;

     /*
      * After which each value is checked. i is assumed to be prime
      * unless marked as not prime. Multiples of i are not prime.
      * Therefore all the multiples of i are marked as not prime.
      * This keeps going on until the inputted value is set to not
      * prime or reached and still set to prime.
```

```
         */
            for (int i = 2; i <= prime; i++) {
                //Checks if i is marked as prime.
                if (nos[i - 1]) {
                    //If it is, all multiples of i are marked as not prime
                    for (int j = i + i; j <= prime; j += i) {
                        nos[j - 1] = false;
                    }
                }
            /*
             * If the inputted value is marked as prime, we can stop
             * there since we found out whether it's prime or not.
             * Otherwise, this program keeps going on till the end
             * (i = prime).
             */
                if (!nos[prime - 1]) break;
            }

            return nos[prime - 1];
        }
    }
}
```

Task 4

Shell Sort

Since the requirement of this task was for the program to generate an array containing 16,384 elements, all of which are random numbers, and sort them, it is impossible to write the contents of the array for a reliable amount of tests and keep this document to a reasonable size. As a result, the program itself checks whether the array that has been sorted, is in fact sorted. This is done using the algorithm `checkSortedAscending` as given in Algorithm 1 (also included in the source code).

```
Algorithm: Check if sorted in ascending order
matrices Input: int array
Output: boolean
for i = 0... length of array - 1 do
    if element at i > element at i+1 then
        return false
    end if
end for
return true
```

Algorithm 1: checkSortedAscending(int[] array)

4.1 The source code for Task 4

```
import java.util.Arrays;

/**
 * @author Mark Said Camilleri
 * @version 20160516
 */
public class Question4 {

    public static final int SIZE = 16384;

    public static void main(String[] args) {

        /*======================= WELCOME MESSAGE TO USER
   ======================*/
        System.out.println("
   ----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and
   Algorithms 1    |");
        System.out.println("
   |--------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri
           |");
        System.out.println("|          Task 4: Shell Sorting Algorithm
           |");
        System.out.println("
   |--------------------------------------------------|");
        System.out.println("| The array to be sorted is:
           |");

        //The array, which  will eventially be sort
        int[] arr = new int[SIZE];

        //Populating the array with random data.
        for (int i = 0; i < arr.length; i++) {
            arr[i] = (int) (Math.random() * SIZE);
        }

        //Displays the unsorted array to the user.
        System.out.println("Unsorted Array: " + Arrays.toString(arr));

        //Sorts the array
        int[] sorted = shellSort(arr);

        //Displays the sorted array to the user
        System.out.println("Sorted Array:   " + Arrays.toString(sorted))
   ;

        //Displays whether the array is indeed sorted or not.
        System.out.println("Array is sorted: " + checkSortedAscending(
   sorted));
    }

    /**
     * Sorts the inputted array using the shell sort algorithm.
     *
     * @param unsorted The array to be sorted
     * @return The sorted version of the array
     */
    private static int[] shellSort(int[] unsorted) {

        /*
         * First, the program begins my making a deep copy of the
         * array to make sure the original one isn't affected.
         */
        int[] sorted = Arrays.copyOf(unsorted, SIZE);

        /*
         * This loop begins with the iterator being set at half the
         * size of the array, - 1. This then iterates downwards by
         * incrementing the iterator, dividing it by 2 and subtracting
         * 1 from it, until it performs it's final iteration at a
         * value of 1.
         */
        for (int sep = (unsorted.length / 2) - 1; sep > 0; sep = (sep + 1) / 2 - 1) {
            /*
             * This for loop goes through the items between sep and
             * the end of the array. To then use an bu sort
             * algorithm to switch it
             */
            for (int i = sep; i < SIZE; i++) {
                /*
                 * The algorthm begins by storing the value in a
                 * temporary variable.
                 */
                int temp = sorted[i];

                //iterator for next for loop. Used also outside loop.
                int j;
                /*
                 * Finds good position of temp in steps of sep, sort
                 * of like insertion sort.
                 */
                for (j = i; j >= sep && sorted[j - sep] > temp; j -= sep) {
                    /*
                     * Moves the (j-sep)th by a step of sep to
                     * leave space for temp at it's proper place.
                     */
                    sorted[j] = sorted[j - sep];
                }
                //Once found temp is stored in it's proper place.
                sorted[j] = temp;
            }
        }
        return sorted;

    }

    /**
     * Method that checks whether the elements in the array in the
     * parameter are all in ascending order.
     *
     * @param array The array to be checked.
     * @return true if all elements are in ascending order, false otherwise.
     */
    public static boolean checkSortedAscending(int[] array) {

         /* Iterates through all elements except for the last one.
         * If an unsorted element is found for loop stops.
         */
        for (int i = 0; i < array.length - 1; i++) {
            /*
             * If the ith element is larger than the i-1th element
             * then the array is not sorted. Method ends returning
             * false
             */
            if (array[i] > array[i + 1])
                return false;
        }
        /* If the condition is never satisfied, then the method returns
         * true.
         */
        return true;
    }
}
```

Task 5

Approximation of a Square Root

For this task, the Newton-Raphson algorithm was used, with the formula shown in Equation 5.2, which is basically a specific case of the general formula for the Newton-Raphson algorithm whcih is given in Equation 5.1.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5.1}$$

$$x_{n+1} = x_n - \frac{x_n^2 - \text{input}}{2x_n} \tag{5.2}$$

Input	Expected Output	Actual Output
1	1	1
2	1.414213562	1.414213562373095
4	2	2
100	10	10
1000	31.6227766	31.622776601683793
9999	99.99499987	99.99499987499375
29535083406040468068234	$1.718577418 \times 10^{11}$	$1.7185774176929147 \times 10^{11}$
45.3	6.730527468	6.730527468185536
5.6	2.366431913	2.3664319132398464
0.25	0.5	0.5
0.125	0.3535533906	0.35355339059327373
-4	Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root:	Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root:
0	Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root:	Your input was not accepted since it was ≥ 0 Please enter a POSITIVE number to find its square root:
Test	Your input was not accepted. Please try again:	Your input was not accepted. Please try again:

5.1 The source code for Task 5

```java
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * @author Mark Said Camilleri
 * @version 20160516
 */
public class Question5 {
    public static void main(String[] args) {

        /*=========================== WELCOME USER ================================*/
        System.out.println(" ---------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println(" |-------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri         |");
        System.out.println("|       Task 5: Approximation of a Square Root      |");
        System.out.println(" |-------------------------------------------------|");
        System.out.print("| Please enter a positive number to find its square root: ");

        //variable to state whether input should be accepted or not.
        boolean isWrong = false;

        //input variable. This shall be x1 and the first value of xn,
        double input = 1;
        do {
            try {
                //accpets user input.
                input = new Scanner(System.in).nextDouble();

                /*
                 * If the value is not positive, then it will raise an
                 * error
                 */
                if (input <= 0) {
                    isWrong = true;
                    System.out.println("| Your input was not accepted since it was \u2264 0");
                    System.out.print("| Please enter a POSITIVE number to find its square root: ");
                }

                //It will also raise an error if it's not a number.
            } catch (InputMismatchException imme) {
                isWrong = true;
                System.out.print("| Your input was not accepted. Please try again: ");
            }
            //This is repeated until a correct value was inputted.
        } while (isWrong);

        //Setting the first value for x_n
        double curr_x = Math.random() * input + 1;
        //Initializing x_(n-1)
        double prev_x;

        //Counts the number of steps taken.
        int steps = 0;

        do {
            steps++;
            prev_x = curr_x;

            //Newton-Raphson formula on y=sqrt(x)
            curr_x = prev_x - (Math.pow(prev_x, 2) - input) / (2 * prev_x);
            /*
             * This will keep on, either until there has been no change
             * or until it has done Integer.MAX_VALUE number of steps.
             */
        }
        while (curr_x != prev_x && curr_x != Double.NaN && steps < Integer.MAX_VALUE);

        System.out.println("|---------------------------------------------------|");

        /*
         * If the value is not the same as the sqrt computed by the
         * built-in sqrt method, then it's approximately equal to,
         * which is denoted by \u2248.
         */
        char sign = (curr_x == Math.sqrt(input)) ? '=' : '\u2248';

        System.out.println("The square root of " + input + " " + sign + " " + curr_x);
        System.out.println("This was done in " + steps + " steps");
    }
}
```

Task 6

Matrix Multiplication

The methods used in this programme were created to work for any $n \times n$ matrices, with n being specified by a constant in the program called `SIZE`. Since writing the tests for 32×32 matrices in this document would be too cumbersome, 3×3 matrices shall be tested instead.

It is worth noting that the matrices used to test the output are the matrices outputted by the program. Since the program rounds each matrix (matrices **A**, **B** and **AB**) to 2 d.p. (and the expected output is also being rounded to 2 d.p) the expected output and the actual output will not be exactly the same, but they should be approximately equal to each other.

A	**B**	Expected **AB**	Actual **AB**
$\begin{bmatrix} 6.72 & 1.99 & 1.73 \\ 3.85 & 0.00 & 3.82 \\ 2.02 & 8.81 & -2.13 \end{bmatrix}$	$\begin{bmatrix} -1.92 & 2.21 & -2.42 \\ -0.18 & -5.64 & 0.03 \\ -0.70 & -3.86 & -6.60 \end{bmatrix}$	$\begin{bmatrix} -14.47 & -3.05 & -27.62 \\ -10.07 & -6.24 & -34.53 \\ -3.97 & -37.00 & 9.43 \end{bmatrix}$	$\begin{bmatrix} -14.48 & -3.01 & -27.66 \\ -10.07 & -6.24 & -34.60 \\ -3.96 & -36.99 & 9.43 \end{bmatrix}$
$\begin{bmatrix} 0.98 & 8.13 & -1.21 \\ 2.36 & -3.08 & -3.51 \\ 9.54 & -7.90 & -8.48 \end{bmatrix}$	$\begin{bmatrix} 3.67 & -0.20 & -7.32 \\ -4.25 & -7.69 & 9.94 \\ 8.91 & 3.91 & -0.75 \end{bmatrix}$	$\begin{bmatrix} -41.74 & -67.45 & 74.55 \\ -9.52 & 9.49 & -45.26 \\ -6.97 & 25.69 & -142.00 \end{bmatrix}$	$\begin{bmatrix} -41.75 & -67.48 & 74.53 \\ -9.47 & 9.52 & -45.27 \\ -6.97 & 25.64 & -141.96 \end{bmatrix}$
$\begin{bmatrix} -5.62 & 0.99 & 4.84 \\ 3.69 & 2.17 & 9.55 \\ -5.38 & -3.36 & -6.33 \end{bmatrix}$	$\begin{bmatrix} -6.78 & 6.98 & 3.41 \\ 8.55 & 3.79 & -1.39 \\ -8.44 & -1.03 & 4.98 \end{bmatrix}$	$\begin{bmatrix} 5.72 & -40.46 & 3.56 \\ 25.36 & 76.56 & 77.40 \\ 49.00 & 137.66 & 149.57 \end{bmatrix}$	$\begin{bmatrix} 5.79 & -40.48 & 3.51 \\ -87.04 & 24.11 & 57.07 \\ 61.22 & -43.76 & -45.17 \end{bmatrix}$
$\begin{bmatrix} -9.31 & 8.19 & 2.23 \\ -2.87 & 7.49 & 1.03 \\ 9.20 & -5.98 & 7.96 \end{bmatrix}$	$\begin{bmatrix} -1.38 & -4.24 & -7.42 \\ 0.23 & 1.28 & 9.99 \\ 8.33 & 5.48 & 6.01 \end{bmatrix}$	$\begin{bmatrix} 33.31 & 62.18 & 164.30 \\ 14.26 & 27.40 & 102.31 \\ 52.24 & -3.04 & -80.16 \end{bmatrix}$	$\begin{bmatrix} 33.29 & 62.25 & 164.22 \\ 14.25 & 27.42 & 102.29 \\ 52.21 & -3.09 & -80.15 \end{bmatrix}$
$\begin{bmatrix} -6.14 & 7.85 & 6.00 \\ -8.71 & 3.36 & -1.96 \\ -4.86 & 8.48 & 4.84 \end{bmatrix}$	$\begin{bmatrix} -0.47 & -2.13 & 3.90 \\ -4.35 & -5.06 & 6.65 \\ -3.94 & -2.29 & -8.39 \end{bmatrix}$	$\begin{bmatrix} -54.90 & -40.38 & -22.08 \\ -2.80 & 6.04 & 4.82 \\ -53.67 & -43.64 & -3.17 \end{bmatrix}$	$\begin{bmatrix} -54.91 & -40.38 & -22.12 \\ -2.77 & 6.07 & 4.74 \\ -53.71 & -43.67 & -3.19 \end{bmatrix}$
$\begin{bmatrix} -4.13 & 4.79 & -8.23 \\ -6.02 & 0.78 & -1.33 \\ -5.98 & -2.59 & 5.39 \end{bmatrix}$	$\begin{bmatrix} 6.72 & 5.18 & 0.65 \\ 7.19 & 6.84 & 2.82 \\ 6.72 & 8.95 & -2.87 \end{bmatrix}$	$\begin{bmatrix} -48.62 & -62.29 & 34.44 \\ -43.78 & -37.75 & 2.10 \\ -22.59 & -0.45 & -26.66 \end{bmatrix}$	$\begin{bmatrix} -48.66 & -62.29 & 34.42 \\ -43.76 & -37.74 & 2.07 \\ -22.59 & -0.51 & -26.67 \end{bmatrix}$

6.1 The source code for Task 6

```
/**
 * @author Mark Said Camilleri
 * @version 20160517
 */
public class Question6 {

    /**
     * Stores the size of the matrices
     */
    public static final int SIZE = 32;

    public static void main(String[] args) {

        /*=========================== WELCOME USER
   ================================*/
        System.out.println("
   -----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and
   Algorithms 1    |");
        System.out.println("
   |---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri
           |");
        System.out.println("|           Task 6: Matrix Multiplication
           |");
        System.out.println("
   |---------------------------------------------------|");

        /*
         * The 2 matricies. Matrix A will be multuplied with matrix B
         * to from matrix AB.
         */
        double[][] matrixA = new double[SIZE][SIZE];
        double[][] matrixB = new double[SIZE][SIZE];

        //Put random values from 1 to 10 in matrixA
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                matrixA[i][j] = Math.random() * 20 - 10;
            }
        }

        //Put random values from 1 to 10 in matrixB
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                matrixB[i][j] = Math.random() * 20 - 10;
            }
        }

        //Multiplied matrix
        double[][] result = new double[SIZE][SIZE];


       /*
        * Transposes MatrixB. This is done to make the multiplication
```

```java
        * easier since the method would accept 2 arrays, one for the
        * row and the other for the column.
        */
        double[][] matBtranspose = matTranspose(matrixB);

       /*
        * Multiplies each row of Matrix A with each column of Matrix B
        * and adds them up to get the value of each cell.
        */

        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                result[i][j] = cellMult(matrixA[i], matBtranspose[j]);
            }
        }
        //Outputs Matrix A
        System.out.print("Matrix A = ");
        displayMatrix(matrixA);


        System.out.println();
        System.out.print("Matrix B = ");
        displayMatrix(matrixB);

        System.out.println();
        System.out.print("  A * B =  ");
        displayMatrix(result);

    }

    /**
     * Performs a matrix multiplication for 1 cell only.
     *
     * @param row The row of the first matrix to be multiplied
     * @param col The colum of the second matrix to be multiplied
     * @return The value of the cell that resides in both the row and column that were inputted, for the resultant matrix.
     */
    public static double cellMult(double[] row, double[] col) {
        //Result to be returned when finished.
        double result = 0;

        for (int i = 0; i < SIZE; i++) {
            result += (row[i] * col[i]);
        }

        return result;
    }

    /**
     * Transposes a square matrix of n x n size
     *
     * @param matrix The matrix to be transposed
     * @return matrix transposed
     */
    public static double[][] matTranspose(double[][] matrix) {

        double[][] transposed = new double[matrix.length][matrix[0].
```

```
   length];

        for (int i = 0; i < transposed.length; i++) {
            for (int j = 0; j < transposed[0].length; j++) {
                transposed[i][j] = matrix[j][i];
            }
        }

        return transposed;
    }

    /**
     * Displays the matrix to the user.
     *
     * @param matrix The matrix to be displayed to the user.
     */
    public static void displayMatrix(double[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {

            for (int j = 0; j < matrix[i].length; j++) {
                System.out.printf("%7.2f ", matrix[i][j]);
            }
            System.out.println();
            System.out.print("           ");
        }
    }
}
```

Task 7

Maximum Number in an Array

Test 1

Input 1 2 3 4 5 6

Expected Output 6

Actual Output 6

Test 2

Input 2 0 -3 10 3

Expected Output 10

Actual Output 10

Test 3

Input -9 -10 -5 -2 5

Expected Output 5

Actual Output 5

Test 4

Input -70210333 620267078 832297266 -242136666 -759786803 89498046 390942259 -616830953 707423547 190501876

Expected Output 832297266

Actual Output 832297266

Test 5

Input 680932039 798137906 1362068 -482386095 976166649 54797596 264771471 906347353 -674021294 -491689938

Expected Output 976166649

Actual Output 976166649

Test 6

Input 463847872 970322047 994450928 475574164 639319683 -942622068 -319105910 -453518765 -167026456 579529276

Expected Output 994450928

Actual Output 994450928

Test 7

Input 1 2 3 4 e

Expected Output There was an error processing your input. Please try again.

Actual Output There was an error processing your input. Please try again.

Test 8

Input Test

Expected Output There was an error processing your input. Please try again.

Actual Output There was an error processing your input. Please try again.

7.1 The source code for Task 7

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * @author Mark Said Camilleri
 * @version 20160517
 */
public class Question7 {
    public static void main(String[] args) {

        /*=========================== WELCOME USER ================================*/
        System.out.println("-----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println("|---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri         |");
        System.out.println("|               Task 7: Largest Number              |");
        System.out.println("|---------------------------------------------------|");

        Scanner in = new Scanner(System.in);

        //This will store the numbers before formatted as numbers.
        String[] text_nos;
        int[] numbers = {0};

        /*
         * This is to ask the user to input the numbers again if the
         * user has provided us with a bad input.
         */
        boolean inputIsCorrect;
        do {
            try {
                inputIsCorrect = true;

                //Ask user for input.
                System.out.println("| Please enter a list of integers. Seperate them    |");
                System.out.println("| with a space. Press Enter at the end of the list: |");

                //Reads input and splits it to string array.
                text_nos = in.nextLine().split("\\s");

                //Creates int array to store them.
                numbers = new int[text_nos.length];

                //Parses the strings in the string array to integers.
                for (int i = 0; i < text_nos.length; i++) {
                    numbers[i] = Integer.parseInt(text_nos[i]);
                }
            } catch (NumberFormatException nme) {

                //Flags program to ask again for input.
                inputIsCorrect = false;

                //Tells user there was an error
                System.out.println("|---------------------------------------------------|");
                System.out.println("|    There was an error processing your input.      |");
                System.out.println("|                 Please try again.                 |");
                System.out.println("|---------------------------------------------------|");
            }
        } while (!inputIsCorrect);

        System.out.println("|===================================================|");
        System.out.printf("| The maximum number from your list was: %10d |\n", findMax(numbers, Integer.MIN_VALUE));
        System.out.println("|===================================================|");
    }

    /**
     * Recursive function to find the maximum number in an array
     *
     * @param nums       The array to return it's max value
     * @param max        Current maximum value.
     *                   This should start with Integer.MIN_VALUE.
     * @return the maximum value.
     */
    public static int findMax(int[] nums, int max) {

        //Base case. If array is empty.
        if (nums.length == 0) {
            return max;
        } else {

            /*
             * If the element at startIndex > max, then it is the
             * current maximum value.
             */
            if (nums[0] > max) {
                max = nums[0];
            }
            /*
             * Recursive call. Checks this max with the array starting
             * from the next element
             */
            return findMax(Arrays.copyOfRange(nums, 1, nums.length), max);

        }
    }
}
```

Task 8

Task 9