Test 2021-2022-KEN1120- CS1; 2-hour exam

Maastricht University

Number of questions: 11
Session period: 20 october 2021 09:00 - 11:15
Duration: 120 minutes

Instruction

Instructions to students:

• The exam consists of 11 questions.
• Answer every question in the online form. Use the sheets provided for brainstorming and scratch space.
• Ensure that you properly motivate your answers (except for multiple choice, true/false, etc.).
• Provide clear explanations. Answers that cannot be easily understood may lower your grade.
• You are not allowed to have a communication device within your reach, nor to wear or use a watch.
• You have to return all pages of the exam (and the chromebook as well). You are not allowed to take any sheets, even blank, home.
• If you think a question is ambiguous, or even erroneous, and you cannot ask during the exam to clarify this, explain this in detail in the space reserved for the answer to the question.
• If you have not registered for the exam, your answers will not be graded, and thus handled as invalid.
• Success! Break a pencil (a keyboard?).

Question 1 – Open-ended – Question-ID: 85615 (6 points)

Analyse the algorithm below and:

1. Find an example of a potential ambiguous operation, describe why it can be considered as ambiguous and how to solve this issue.
2. Ensure that this pseudocode fits all the requirements of an algorithm (effectively computable operations, produce a result, halts in a finite amount of time, etc.). Propose modifications in case any of them is not met.

Changing the car's tires: algorithm to change the 4 tires of the car
Step 1 – Buy new tires
Step 2 – Change the tire
Step 3 – Inflate the wheel
Step 4 – Balance the wheel
Step 5 – Repeat

(See at the end)

Question 2 – Multiple choice – Question-ID: 87393 (4 points)

What will be the outcome of the following code snippet?

```
1   int x = 0;
2   int y = -10;
3
4   if ((y<0) || (y/x<0)) System.out.println("You did it!");
5   else System.out.println("You didn't!");
```

A Will not compile. There are syntax errors in lines 4 and 5

B Will print "You didn't!"

C Will print "You did it!"

D Will compile but cause a runtime error when executing line 4

Question 3 – Multiple choice – Question-ID: 87383 (2 points)

Bytecode is given as input to the _____

A Java Virtual Machine (JVM)

B Linker

C Assembler

D Compiler

Question 4 – Multiple choice – Question-ID: 87382 (2 points)

Java Virtual Machine (JVM) is a(n) _____.

A Assembler

B Compiler

C Debugger

D Interpreter

Question 5 – Multiple choice – Question-ID: 87375 (4 points)

Select the correct logical expression between variables `a` and `b`, such that the following loop terminates (i.e. does not run forever) without arithmetic overflow but also runs at least once?

```
while (a != b) {
    a--;
    b++;
}
```

Hint: `Math.abs(x)` gives the absolute value of expression `x`

A a=b

B a>b

C a>b && Math.abs(a-b)%2=0

D a>b && Math.abs(a-b)%2=1

E a<b

F a<b && Math.abs(a-b)%2=0

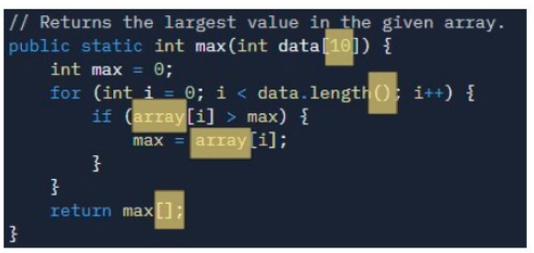
G a<b && Math.abs(a-b)%2=1

Question 6 – Hotspot: 5 hotspots – Question-ID: 86784 (5 points)

The following piece of code returns the largest value in the array `data` assuming that this array contains only positive values. Find the errors and mark them in the code.

Hint: there are 5 errors in total.

```
// Returns the largest value in the given array.
public static int max(int data[10]) {
    int max = 0;
    for (int i = 0; i < data.length(); i++) {
        if (array[i] > max) {
            max = array[i];
        }
    }
    return max[];
}
```



Question 7 – Fill in (multiple) – Question-ID: 86783 (7 points)

Which will be the values stored in the array `values` after the execution of the following piece of code?

```
class Exam {
  public static void method(int[] data, int i) {
      int[] new_data;
      if(data[i] == 0)
          data[i] = i+1;
      else {
          new_data = data;
          new_data[i] = 10;
      }
  }
  public static void main(String args[]) {
    int[] values = new int[7];
    values[4] = 3;
    values[0] = -1;
    method(values, 1);
    values[2] = values[4];
    method(values, 0);
    values[3] = 1;
    values[6] = values[values[3]];
  }
}
```

Index	0	1	2	3	4	5	6
Value	10	2	3	1	3	0	2

Question 8 – Fill in (multiple) – Question-ID: 87961 (5 points)

Fill in the gaps in the code snippet below, such that the (recursive) method prints the digits of a number (one at a separate line), e.g. method call `printDigits(9691)` should print:

```
1
9
6
9
```

```
public static [void] printDigits (int x) {
  if ( [x==0] ) return;
  else {
    System.out.println( [x%10] );
    [printDigits] ( [x/10] );  //this should be your recursive call
  }
}
```

Question 9 – Open-ended – Question-ID: 85791 (15 points)

Given two non-empty strings `str` and `sub`, compute if exactly `n` copies of `sub` appear in `str`, possibly with overlapping. You can consider that `n` will always be non-negative and `str` and `sub` are always given in lowercase.

Example 1: `strCopies("applepen", "pen", 2)` returns `false` since the substring `pen` only appears once in `str`.

Example 2: `strCopies("penpineappleapplepen", "apple", 2))` returns also `true` since `apple` appears twice in `str`.

Example 3: `strCopies("songsongs", "songs", 2))` returns also `true` since `songs` appears twice in `str` (middle *s* is overlapped).

To complete this exercise, use the following method signature:

```
public boolean strCopies(String str, String sub, int n) { ... }
```

Hint: You may find the `substring` method helpful. See the documentation below:

substring

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:

`beginIndex` - the beginning index, inclusive.

`endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

(See at the end)

Question 10 − Open-ended − Question-ID: 85795 (15 points)

Checksum is a sequence of numbers whose purpose is to detect accidental changes in a data sequence with the goal to protect its integrity, usually after the transmission of this information in telecommunications. In this exercise you will implement a simple version of this function to verify the integrity of a matrix (2-dimensional array). Given a 2-dimensional array `arr` and two parity vectors `rowParity` and `columnParity`, the method must identify if the matrix became corrupted. To that end, you must check if the parity of the rows (i.e. the sum of all the elements in a row) matches what is indicated by `rowParity` and similarly, the parity of the columns (i.e. the sum of all the elements in a column) matches `columnParity`. In these vectors, `0` represents even (the addition of all the elements results in an even number) and `1` represents odd.

Example 1: The method returns `true`.
Row parity = [1, 1]
Column parity = [0, 0]
`Array =`

3	4
5	0

Explanation: Calling the method with the previous parameters returns `true` since all the elements in row 0 sum to 7 (an odd number) and those in row 1 sum to 5 (also odd) and the row parity vector is [1, 1] and similarly for the columns, the sum of elements (on each column) results in 8 (even) and 4 (even) respectively, being the column parity [0, 0].

Example 2: The method returns `true`.
Row parity = [0, 0, 0]
Column parity = [0, 1, 1]
`Array =`

1	2	15
6	4	10
3	1	0

Example 3: The method returns `false`.
Row parity = [0, 1, 0]
Column parity = [0, 1, 1]
`Array =`

3	4	1
2	10	8
1	17	6

Explanation: This example returns `false` because, although the column parity is correct (columns sum to 6, 31 and 15 respectively, matching [0, 1, 1]), the row parity is wrong (rows sum to 8, 20 and 24, that should be represented by [0, 0, 0] instead of [0, 1, 0]).

To complete this exercise, use the following method signature:

```
public boolean checksum(int[][] arr, int[] rowParity, int[] columnParity) { ... }
```

(See at the end)

Question 11 − Open-ended − Question-ID: 87867 (15 points)

Due to COVID-19, healthcare systems around the world are under pressure. Every day, many people that are severely sick due to COVID-19 are admitted to ICU units (and occupy a bed) and some others are recovering or unfortunately die (and free a bed). We are studying a time interval of `M` consecutive days (`M` can be as large as needed, COVID-19 is still ongoing) and for each day we observe the difference in occupied ICU beds since the previous day (obviously that number is an integer).

We use an array to store this information, e.g. if `M={1,10,20,-4,-7}` then that means that on day `0` (beginning of time period), there is an increase in ICU occupancies by `1`, on day `1` we have an increase in ICU occupancies by `10` since day 0 etc, on day `2` we have an increase of `20` since day `1` etc. To put it differently, over the first `3` days (days `0`, `1` and `2` in the array) there is an increase of `31 (=1+10+20)` in ICU occupied beds.

We assume that in every country there are `N` hospitals (`N` is an integer that cannot be smaller than 1). Speaking with the European Comissioner for Health, she told us that a time period of `k` consecutive days is considered "good" (in just political terms of course) for a country, if during this time period there is *at least 1* bed (*on average per hospital*) that is freed up.

For example, for a small country with just `3` hospitals (`N=3`) and the following daily ICU changes `M={42,-10,8,1,11,-6,-12,16,-15,-11,13}`:

- For a period of `k=4` days spanning from day `0` till day `3`, the sum of the daily differences is `42-10+8+1 = 41`. This is obviously not a "good" period for that country, since no beds were freed (on the contrary, the occupancy increased)!
- Similarly, for a period of `k=3` days spanning from day `5` till day `7`, the sum of the daily differences is: `-6-12+16 = -2`. This period is also not considered "good" for the country, since the average number of beds that are freed *per day* and *per hospital* is: `2/(N*K)=2/(3*3)=2/9 < 1` (the average is less than 1).
- However, for a period of k=`5` days spanning from day `5` till day `9`, the sum of the daily differences is: `-6-12+16-15-11 = -28`. This is a "good" period for that country, since the average number of beds that are freed *per day* and *per hospital* is: `28/(N*K)=28/(3*5)=28/15 > 1` (the average is greater than 1).

Your task is to help the EU Commissioner by writing a method called `findLongestGoodPeriod` that given the array `M` of daily differences in ICU beds and the number of hospitals in a country (`N`), computes which is the *longest* (in days) "good" period for that country. In the example presented above (i.e. with array `M={42,-10,8,1,11,-6,-12,16,-15,-11,13}` and `N=3`), then your method call `findLongestGoodPeriod(M,N)` should return `5` (since the longest "good" period of that country in days is the sequence `(-6, -12, 16, -15, -11)`).

To complete this exercise, use the following method signature:

```
public static int findLongestGoodPeriod(int[] M, int N) {…}
```

Hint: We think that a recursive solution might be the easiest, but it is not necessary to solve this exercise recursively.

End of test 2021-2022-KEN1120- CS1; 2-hour exam

(See at the end)

Indicative Answers to Open Ended Questions (Q1, Q9, Q10, Q11)

Q1

1. Find an example of a potential ambiguous operation, describe why it can be considered as ambiguous and how to solve this issue.

There are several issues which can be considered as ambiguous. Some examples (and a possible amendment) are:

- Step 1: The number and type of tires to be bought is not specified. This information must be included (4 tires from brand X, model Y).
- Step 2: it is not clear which is the tire to be replaced. This could lead to the user changing the same tire repeatedly. We can specify "Change one of the old tires with a new one".
- Step 3: Once again, more information is needed. In this case, we can provide information about the correct pressure to be included depending on the position of the wheel (rear or front)
- Step 4: as in Step 2, we can be balancing the same wheel all the time. We can re-phrase it as "Balance the wheel whose tire was just replaced".
- Step 5: this can serve as the answer for both questions (see below).

2. Ensure that this pseudocode fits all the requirements of an algorithm (effectively computable operations, produce a result, halts in a finite amount of time, etc.). Propose modifications in case any of them is not met.

To be effectively computable and halt in a finite amount of time we need to correct the problem in Step 5. This step is ambiguous since it is not clear which steps must be repeated. Moreover, it creates an infinite loop. It can be re-phrased as "Repeat steps from 2 to 4 until the 4 tires have been replaced".

Q9

A recursive version could be:

```
    public static boolean strCopies(String str, String sub, int n) {
          if (checkStr(str, sub) == n) return true;
          else return false;
        }

    static int checkStr(String str, String sub) {
          int strlen = str.length();
          int sublen = sub.length();

          if (strlen < sublen) return 0;
          if (str.substring(0, sublen).equals(sub))
            return 1 + checkStr(str.substring(1), sub);
          else
            return checkStr(str.substring(1), sub);
    }
```

A non-recursive version could be as follows:

```
public static boolean strCopies (String str, String sub, int n) {
      int counter = 0;
      int sublen = sub.length();

     for(int i=0; i<str.length()-sublen+1; i++) {
         if (str.substring(i, sublen+i).equals(sub))
              counter++;
      }

      if(counter == n)
          return true;
      return false;
    }
```

Q10

```
public static boolean checksum (int[][] arr, int[] rowParity, int[] columnParity) {
        int numRows = rowParity.length;
        int numColumns = columnParity.length;

        int[] rowPar = new int[numRows];
        int[] columnPar = new int[numColumns];

        for(int i=0; i<numRows; i++) {
            int sumRow = 0;
            for(int j = 0; j < numColumns; j++){
              sumRow = sumRow + arr[i][j];
            }
            rowPar[i] = sumRow%2;
        }

        for(int i=0; i<numColumns; i++) {
            int sumColumn = 0;
            for(int j = 0; j < numRows; j++){
                sumColumn = sumColumn + arr[j][i];
            }
            columnPar[i] = sumColumn%2;
        }

        if((Arrays.equals(rowParity,rowPar)) && (Arrays.equals(columnParity,columnPar)))
          return true;
        else
            return false;
    }
```

Q11

A non-recursive solution could be as follows:

```
public static int findLongestGoodPeriod(int[] M, int N) {

    int K=0; // the length of the sequence we are looking for
    for (int i=0;i<M.length;i++) {
        double sum = M[i];
        for (int j=i+1;j<M.length;j++) { //double loop to check all combinations
            sum+=M[j]; //step 1: i sum all elements

            if (-sum/(N*(j-i+1))>1) { //step 2: check the formula
                K=Math.max(K,j-i+1); //if correct, then step 3: update the K
            }
        }
        }

    return K;
}
```

An example of a recursive solution would be:

```
public static int findLongestGoodPeriod (int[] M, int N) {
// One possible recursive solution
// Base case: The whole array is a good period
double sum = 0;
for (int i=0; i<M.length; i++) {
    sum += M[i];
}
if (-sum/N >= M.length) {
    return M.length;
}

// Otherwise, repeat the process for all sub-sequences
// For this purpose, take a sub-array removing the first element (left)
// and a sub-array removing the last element (right)...
// and let the recursion do the dirty job
int[] leftM = new int[M.length-1];
int[] rightM = new int[M.length-1];
for (int i=0; i<M.length-1; i++) {
    leftM[i] = M[i];
    rightM[i] = M[i+1];
}
return Math.max(findLongestGoodPeriod(leftM, N), findLongestGoodPeriod(rightM, N));

}
```