Test KEN1120 Resit Introduction to Computer Science 1 - actual exam time 120 minutes -digital exam

Maastricht University

Number of questions: 12
Session period: 24 january 2022 09:00 - 11:15
Duration: 120 minutes

Instruction

Instructions to students:

• The exam consists of 12 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: 98600 (5 points)

Explain (shortly) how the program compilation in Java promotes platform independency (e.g. a Java program compiled on MacOS can also run on Windows).

Question 2 – Multiple choice – Question-ID: 96259 (2 points)

Which of the following variable declarations won't compile?

A `int num1, num2;`

B `double num1, int num2 = 0;`

C `int num1, num2 = 0;`

D `int num1 = 0, num2 = 0;`

Question 3 – Multiple choice – Question-ID: 98597 (2 points)

Which of the following statements won't increase the value of variable `x` by one? Assume that variable `x` is properly declared and initialized.

A `++x;`

B `x+=1;`

C `x+1;`

D `x++;`

Question 4 – Multiple choice – Question-ID: 98595 (2 points)

What will be the result of the following statement?

```
double a = 35 / 0;
```

A will store `0` to variable `a`

B will store `35` to variable `a`

C will store `35.0` to variable `a`

D will cause a syntax error

E will cause a runtime error

F will store infinity to variable `a`

Question 5 – Multiple choice – Question-ID: 98601 (2 points)

What is the meaning of the return data type `void`?

A An empty memory space is returned so that the developers can utilize it.

B `void` returns no data type.

C `void` is not supported in Java.

D none of the above.

Question 6 – Multiple choice – Question-ID: 98599 (5 points)

What is the value of variable `t` after the execution of the following statements?

```
int n=1024, t=0;
for (int i=1;i<=n;i*=2) {
   for (int j=1;j<=i;j++)
          t++;
          n--;
}
```

A 1023

B 1024

C 2047

D none of the above

Question 7 – Multiple choice – Question-ID: 89632 (5 points)

Which is the output printed after executing the following piece of code?

```
public class Recursion {

    public static int recursiveCall(int number, int[] arr) {
        if (arr.length==0)
            return 0;
        int[] arr2 = new int[arr.length-1];
        System.arraycopy(arr, 1, arr2, 0, arr.length-1);
        if (arr[0]>0)
            arr[0] = 0;
        return Math.abs(arr[0]) + recursiveCall(number, arr2);
    }
    public static void main(String[] args) {
        int[] arr = {6, -1, 10, 3, -9, -8, 0, 13};
        System.out.println(recursiveCall(0, arr));
    }
}
```

A -32

B -18

C 0

D 18

E 50

F 32

Question 8 – Fill in (multiple) – Question-ID: 87424 (5 points)

In the following code snippet, fill in the 5 gaps (with an expression, an operator, a statement, etc.) such that we can calculate the sum: $1^2+2^2+3^2+4^2+5^2$. You must assume we have no access to additional libraries (e.g. `Math`).

```
int i = [          ];
int sum = 0;

while (i [          ] [          ] ) {
   sum += [          ];
   i += [          ];
}
```

Question 9 – Fill in (multiple) – Question-ID: 86782 (7 points)

Fill in the array with the values that would be stored after the code executes:

```
int[] list = {-20, 33, 8, 20, 16, -15, 9};
int start=0;
int end=list.length-1;

while (start<end) {
  if (list[end]<list[start]) {
     int temp=list[start];
     list[start]=list[end];
     list[end]=temp;
     end--;
  }
  start++;
}
```

Index	0	1	2	3	4	5	6
Value							

Question 10 – Open-ended – Question-ID: 87378 (15 points)

Suppose that we are given an array of non-negative integers. We would like to change the elements of the array in such a way that the modified array is sorted, however, the only "allowed" operation on the array elements is to increase them by one. Given that this is the only allowed operation (e.g. you cannot reduce elements or increase them by more than one), you are asked to write a method called `makeIncreasing` that returns the *minimum* number of such operations (i.e. how many times do we need to increase elements by one) so that the array is sorted in a purely increasing order (i.e. equal values are not allowed). Therefore, when sorted, each element in the array must be larger than the previous one (see examples below).

For example, with the array `a={1,0,2},` the method call `makeIncreasing(a)` should return `3` and then we get the array `{1,2,3}` which is sorted (i.e. we increased `0` to `2` (2 operations) and `2` to `3` (1 operation)).

With array `b={2,4,11,3}` method call `makeIncreasing(b)` should return `9` and then we get the array `{2,4,11,12}` which is sorted (i.e. we increased `3` to `12` (9 operations)).

To complete this exercise, use the following method's signature:

```
public static int makeIncreasing (int[] array) {...}
```

Question 11 – Open-ended – Question-ID: 89744 (15 points)

Checkers is a board game with two players competing on an 8x8 checked board like the one shown below. Initially, each player has 12 pieces. Each player plays with piece of a specific colour (in the example, either black or white).

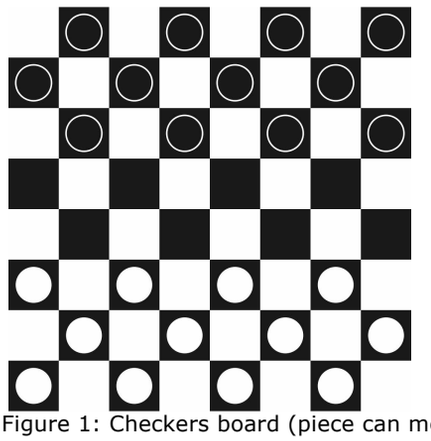

Figure 1: Checkers board (piece can move only in diagonal directions, using only the dark spots).

The checkers board can be represented using a 2D-array with size 8x8. In our implementation, 0s represent empty spaces while the pieces are represented as 1s (for the white player) and 2s (black player). In the implementations, you must consider:

1. The board is represented as a 2D-array being the top left the position [0][0] and the bottom-right the [7][7].
2. White piece can only move upwards (decreasing the row number).
3. Each player will have, at least, a piece in the board, with a maximum of 12 per player.
4. To remove a piece from your opponent, it must be located in one of the next diagonals (either left or right) and there is an empty space in the next diagonal in the same direction (see Figure 2).

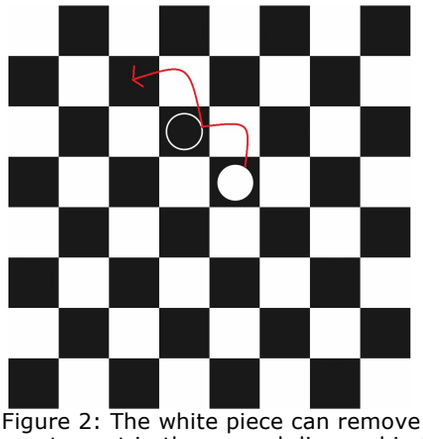

Figure 2: The white piece can remove the black one in the first diagonal (to the left) "jumping" to the empty spot in the second diagonal in the same direction.

Implement a method `checkers` that checks if, in the next movement, the player using white pieces can remove, at least, a black piece. To that end, you must check if at least one of the white pieces in the current `board` can remove a black one (see rule 4 above).

Example 1: Calling the method with the initial board, represented with the matrix below, the output must be `false` since there is no white piece that can remove a black one.

0	2	0	2	0	2	0	2
2	0	2	0	2	0	2	0
0	2	0	2	0	2	0	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0

Example 2: Calling the method with the board represented below, the output must be also `false` since, although the white piece has two pieces in the diagonals, one (in row 4) is behind (white pieces move "up") and the other one (in row 2) has not an empty space to "jump".

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	2
0	0	0	0	0	0	1	0
0	0	0	0	0	2	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Example 3: Calling the method with the board represented below, the output must be `true` since this time, the piece in row 2 can be removed (it is located in a valid diagonal and there is an empty space to "jump" in the same direction). Therefore, the white piece (represented with a 1) at position (3,6) will "eat" the black (represented with a 2) at (2,5) and land at (1,4).

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	2	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	2	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

To complete this exercise, use the following method signature:

```
public static boolean checkers(int[][] board) { ... }
```

Note: if you are familiar with checkers rules, you may remember game rules allow the "elimination" of multiple pieces if they are in consecutive diagonals (even if the diagonal is backwards, i.e. moving backwards on the board). To facilitate the implementation, we are not accounting for this types of movement.

Question 12 – Open-ended – Question-ID: 98771 (15 points)

We want to implement a method to choose the best option(s) for a set of products. The method receives the information about the products' offers in an array (`arr`) as follows. Each product's offer is represented by three values, a product identifier followed by the price of the product and its rate (quality of the product). The array `arr` is 1-dimensional and contains all the offers (therefore, the number of elements in `arr` is multiple of 3). An example of a valid array is as follows:

```
{1,10,2, 1,10,3, 2,12,5, 2,9,1}
```

representing:

```
{product_id1, price1, rate1, product_id2, price2, rate2, product_id3, price3, rate3,
product_id4, price4, rate4}
```

In this example, we have two offers for product type 1 with price 10 (both same price) and rates 2 and 3 respectively and two offers for product type 2 with prices 12 and 9 and rates 5 and 1 respectively.

Implement a method that returns the index (position in the initial array `arr`) of the **best offer for each product type**.

To decide which is the best offer, you must prioritize the price (the lower, the better) and then the rate (the higher, the better).

To facilitate the implementation, assume:

1. The method receives a parameter (called `products` in the signature below) that is the number of product types to be evaluated.
2. The product types are integers from `1` to `products`.
3. The array `arr` will always contain, at least, one group per product type.
4. You can assume that array `arr` always contain groups of products in ascending order by product type. E.g. {1,10,2, **3,3,5,** 2,12,5, 2,9,1, 3,3,1} will not be used as an input due to the group in bold letters which is located in an incorrect position).

Examples (spaces added to highlight the groups):

- **Example 1**: `selection({1,10,2, 1,10,3, 2,12,5, 2,9,1}, 2)`
  Using this input, the resulting array must be `{1,3}` since the lowest price for product type 1 is 10 in both possible cases but the rate for the second option is higher (3>2) and, for product 2, the lowest price is in index 3.
- **Example 2**: `selection({1,10,2, 2,12,5, 2,9,1, 3,2,5, 3,3,1}, 3)`
  Using this input, the resulting array must be `{0,2,3}`. The best (and only) offer for product type 1 is in the group in position 0, the best offer for product type 2 is the second one (in group 2, since it has a better price, index 2) and for type 3, the best offer is the first one (in group 3, both price and rate).
- **Example 3**: `selection({1,10,2, 2,12,5, 2,9,1, 3,3,1, 3,3,5}, 3)`
  Using this input, the resulting array must be `{0,2,4}`. The difference here is just related to type 3 which, having the same price for both offers, the second one (in position 4) is chosen due to the higher rate.

Use the following signature for the method to be implemented:

```
public static int[] selection(int[] arr, int products) { ... }
```

End of test KEN1120 Resit Introduction to Computer Science 1 - actual exam time 120 minutes -digital exam