Choose 4 tasks and return your source files (type is cpp) in Oma before deadline!

1. The principle of **RPN** calculator (Reverse Polish Notation calculator) has been described in address https://en.wikipedia.org/wiki/Reverse_Polish_notation (figure 1). Implement program **1.cpp** which works as **RPN** calculator. You have to use stack and you can assume that all numbers are one digit integers (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9). In figure 1 is example how it works in theory. When you enter integer you have to push it in stack. When you enter operand (+, - or *) then you have to pop two numbers from stack and make the arithmetic operation. The result you have to push in stack. When you enter character '=' you have to end and print the result of expression. Test your program with RPN expression **5 1 2 + 4 × + 3 –** which infix expression is "5 + ((1 + 2) × 4) − 3". Sample input and output is in figure 2.

Input	Action	Stack	Notes
5	Operand	5	Push onto stack.
1	Operand	1 5	Push onto stack.
2	Operand	2 1 5	Push onto stack.
+	Operator	3 5	Pop the two operands (1, 2), calculate (1 + 2 = 3) and push onto stack.
4	Operand	4 3 5	Push onto stack.
×	Operator	12 5	Pop the two operands (3, 4), calculate (3 * 4 = 12) and push onto stack.
+	Operator	17	Pop the two operands (5, 12), calculate (5 + 12 = 17) and push onto stack.
3	Operand	3 17	Push onto stack.
−	Operator	14	Pop the two operands (17, 3), calculate (17 - 3 = 14) and push onto stack.
	Result	14	

Figure 1. Principle and example of RPN calculator

```
Give RPN-expression vertically:

5
1
2
+
4
*
+
3
-
=
14
```

Figure 2. Sample input and output

2. Download http://users.metropolia.fi/~pasitr/2017-2018/TI00AA50-3011/exam/A.txt and implement a program **2.cpp** which counts how many times the largest and the smallest number occurs in the file **A.txt**. Sample output is in figure 3.

```
In file are 62 integers.
Minimum number -498 exists 2 times.
Maximum number 487 exists 1 times.
```

Figure 3. Sample print of program 1.cpp

3. **MVC pattern**. In link https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm you can see the principle of MVC pattern which is programmed in Java. In the address http://users.metropolia.fi/~pasitr/2017-2018/TI00AA50-3011/exam/1.cpp is one MVC solution with C++. Now you have to implement throwing dice program **3.cpp** where you use MVC pattern. In a dice is six symmetric size faces (figure 4). Probability of each faces is same (1/6). Use randomize function of C++. First you can enter how many times you throw dice. Sample print of program is in figure 5.

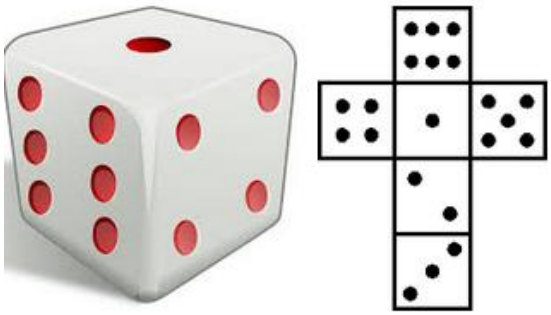

Figure 4. Dice contains 6 faces

```
How many times you want to throw dice: 10

Dice: 2
Dice: 2
Dice: 1
Dice: 2
Dice: 1
Dice: 5
Dice: 5
Dice: 4
Dice: 4
Dice: 1
```

Figure 5. Sample print of program 3.cpp

4. **Operator overloading**. Implement program **4.cpp** where you define class **Point**. Class **Point** contains two attributes **x** and **y**. This program contains four additional overloaded operators to the class **Point**. Those operators are +, –, = and ++. After you have implement these operators you have to implement main function. First in main program retrieves coordinates. First operation is **++point1** (figure 6, point 1). Second operation is **point2 = ++point1** (figure 6, point 2). Third operation is **point1 = point1+point3** (figure 6, point 3). Fourth operation is **point1 = point1-point3** (figure 6, point 4). Fifth operation is **point1 = point2 = point3** (figure 6, point 5). Sample input and output is in figure 6.

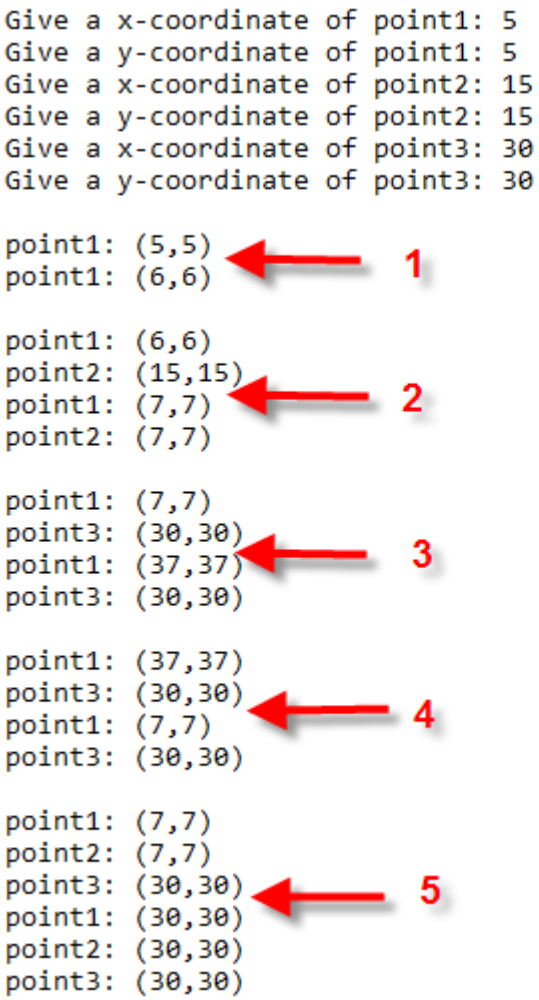

Figure 6. Sample input and output

5. **Arrays or linked lists**. As most school children know, Tic-Tac-Toe is a game played on a three-by-three board (figure 7). Two players, X and O, alternate in placing their respective marks in the cells of this board, starting with player X or O. If either player succeeds in getting three of his or her marks in a row, column, or diagonal, then that player wins. First you have to put value 0 to all cells in a board array. Then you have to simulate game and choose a player to start (X or Y). You have to make loop where you use randomize-function to choose the cell you put value -1 or 1. The game ends when X wins, O wins or it is tie. Sample output and output is in figure 6.

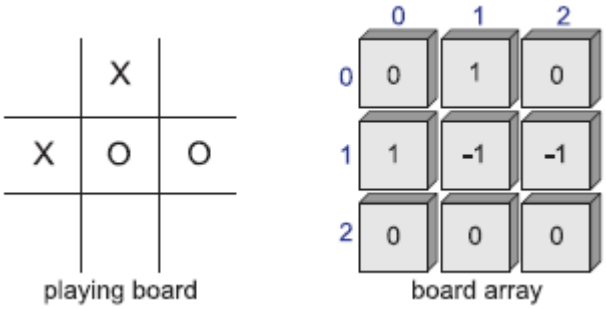

Figure 7. The idea of ***Tic-Tac-Toe***

```
X|X|O
-+-+-
X|O|O
-+-+-
X|O|X  X wins
```

Figure 6. Sample output

6. **Classes and objects**. Define class **Card**. **Suit** and **value** are the attributes of class Card. Suits are **Heart** (value 0), **Club** (value 1), Diamond (value 2) and Spade (value 3). Values of card are 1 (ess), 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 (jack), 12 (queen) and 13 (king). Suit and value of card get values in creation of card. Define methods **setSuit()**, **getSuit()**, **setValue()** and **getValue()**. Define also method **printCard()**, which prints the suit and value of the card. Create all 52 cards in function **main()**. Save all cards into array, which is an attribute of class **Deck**. The name of this array attribute is **card**. Another attribute of **Deck** is **size**, which tells how many cards are in the **Deck**. Define a method **shuffle()** which shuffles cards. Furthermore define method **getSize()**, which returns how many cards are in the deck. Define also method **dealCard()**, which returns top card (values) of the deck and remove the card from the deck. Furthermore define also a method **putCard()**, which puts a card into the deck. In function main() you have to create an empty deck and a full deck. After that you have to shuffle the full deck. Then you have to move a card form full deck to the empty deck. You have to print the value of this card. Do this 52 times. After that the program ends.

Figure 7. Cards

7. **Inheritance**. Define a class **Dice**. One property of dice is **count**, which tells you how many times you have thrown this dice. Another property is an array **counts** where you have information about that how many times you have got each number 1, 2, 3, 4, 5 and 6. One method of dice is **throw()** which returns number 1, 2, 3, 4, 5 or 6. All options are equal. Define also subclass **cheaterDice()** which gives number 6 in probability 0,25. You have to give this probability in one parametric constructor. In real life there are dices where are 4, 8, 10, 12, 20, 24, 30 and 100 faces. Modify you dice so that you give in a constructor how many faces are in your dice (8, 10, 12, 20, 24, 30 and 100 faces). Make a test program (main) where you create 3 dices. Then you throw 3 dices so long until the sum of faces is 18. You have to print how many times you thrown dices. You have also create 3 cheater dices and you throw 3 cheater dices so long until the sum of faces is 18. You have to print how many times you thrown cheater dices. Create also 3 dices with 8, 10, 12, 20, 24, 30 and 100 faces. Throw 3 dices so long until the sum of faces is 18. You have to print how many times you thrown dices.

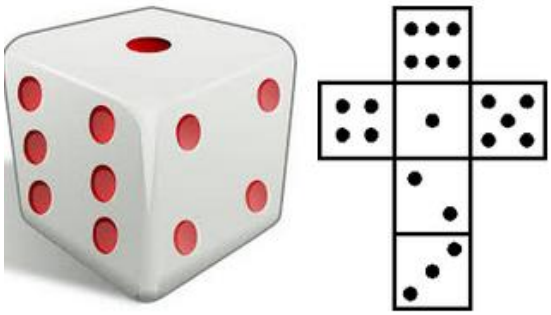

Figure 8. Normal dice contains 6 faces

Good luck for the exam!