Big Homework 1

Data Structures and Algorithms

Stacks, Queues & Lists

General mentions

• For this project you will work either in **teams of 2 people or alone.**

• The homework will be uploaded on the **Moodle** platform (curs.upb.ro), for the Hw1 assignment. If you encounter problems with the platform, contact your lab assistant.

• The homework must be submitted by **24.04.2023, at 08:00**. No late submissions will be accepted.

• You will be asked details about your solutions during the following lab. Projects **not presented at the following lab won't be graded!**

•The final submission will contain an archive named Student1FamilyName_Student1Name_Student2FamilyName_Student2Name_HW1 with:

· **the source files** of your project (.cpp and .h), **grouped in separate folders for each exercise** (and **not containing** the object files (.o), executables (.exe) or codeblocks project files (.cbp))

· **a README file** in which you will briefly specify all the functional sections of the project, together with instructions for the user; additionally, if you have parts of the homework that don't work, you may offer solution ideas for a partial score on these sections. You will have to explain your propositions at the presentation lab.

• For all questions regarding the project, communicate on Microsoft teams with your lab assistant (recommended to create a new post on the Teams group so that everyone can see the question & the answer). Don't forget to mention @General (to notify everyone) or @TeacherName.

• **Warning:** we will use plagiarism detection software on your submissions (Stanford's tool Moss). Copied homework will be marked with **0 points.**

! Observation: You can use the stacks, queues and lists that we used in class (they will be in headers (.h) and generic classes (template)). Alternatively, you can use their standard C++ implementations, BUT not other custom implementations from the internet.

Tasks:

1. Stack (3p)

You are Ethan Hunt and the IMF has sent you on a mission to protect the nuclear engineer specialist, Homer Simpson, who is in town for a very important presentation from a possible kidnap by the evil corporation called The Syndicate. You are gathering the information about your target but you soon discover that the IMF has no image of him, so you don't know what your target looks like. *What you do know is the fact that he is a reputable specialist and everyone will talk to him. What you also know is the fact that Homer is a really shy guy and he will not try to talk to anyone in return because he doesn't know anyone from the event.* Tonight, there is a formal dinner for all the participants and you are sent to gather information about the interaction between people in order to identify your target. After the event, you take your notes back to the base camp and you look over all the interactions that have happened between people under a matrix of N x N form, where 0 means that the person i has not talked to person j and 1 means that person i has talked to person j.

Using a Stack, find your target and save him from the kidnapers. (Extra info: the time complexity should be O(n))

P.S. It might be possible that Homer was so shy that he decided to skip dinner. If this happens, then you need to create another event in order to gather information again.

Obs: You can use the input matrix ONLY for gathering data, checking a certain element (i, j) OR at a maximum, traversing a particular row ± column. You MUST use the stack for getting the solution.

Input Example

```
MATRIX = { {0, 0, 1, 1, 0, 1},
           {0, 0, 1, 1, 0, 1},
           {0, 1, 0, 1, 0, 1},
           {0, 0, 0, 0, 0, 0},
           {1, 0, 1, 1, 0, 1},
           {0, 0, 0, 1, 0, 0},
```

Output example:

Homer is the person with the id 3

Input Example

```
MATRIX = { {0, 0, 1, 1, 0, 1},
           {0, 0, 1, 1, 0, 1},
           {0, 1, 0, 1, 0, 1},
           {0, 1, 0, 0, 0, 0},
           {1, 0, 1, 1, 0, 1},
           {0, 0, 0, 1, 0, 0},
```

Output example:

Everyone has talked to at least one other person. Homer is not present

Points:

- 0.5p reading input and building the matrix
- 2p main algorithm using stack
- 0.5p all validations, output, including impossible case

2. Queue (3p)

We have opened a restaurant, but we only have one chef. In this restaurant, we use queues to organize the orders that come in. Each order is represented by two numbers: the time t (at which the client comes in and places the order) and the duration d (time taken for our chef to prepare the dish). The orders received will be sorted by their arrival time t. When the dish is ready, we eliminate the order from our queue and the chef begins working on the next (if there is one). The restaurant closes at the time T.

a) (0.5p) Choose a convenient method of storing the orders (arrival time and duration) - e.g. a struct or a class. Read from the keyboard two numbers - N and T - and then the N orders.
b) (0.75p) Print all the times at which the queue is empty and our chef can take a breather. As an alternative, print these times as intervals. Only print values inside the working time of the restaurant (before time T).
c) (0.5p) Print the maximum duration our chef has to work on a single order.
d) (0.75p) For each order, print the theoretical completion time (that the client hoped for) and the actual completion time (when our chef will finally complete the client's dish).
e) (0.5p) Determine if there are orders that remain incomplete after the closing time of the restaurant OR orders that are completed after the closing time.

Ex:
N = 6, T = 20 and the orders are:
t = 0, d = 5
t = 1, d = 3
t = 10, d = 3
t = 11, d = 2
t = 12, d = 4
t = 18, d = 5

Order 1: expected completion time = 5, actual completion time = 5;
Order 2: expected completion time = 4, actual completion time = 8;
The chef takes a break (the queue is empty) between the times 8 and 10

We have T = 20. After computing the orders, we find out that the last one will be finished at t = 24. Therefore, we have orders which are completed after the closing of the restaurant. Obs: only orders that arrive AFTER the closing time won't be completed.

3. Lists (3p)

Rare polynomials are polynomials of large degrees and many coefficients equal to 0. Consider a rare polynomial. Each term of the polynomial can be represented by a structure/class which contains information about the coefficient and the exponent. The polynomial will be represented using a **list**.

Implement a program that calculates the value of the polynomial for a given number X, the sum and the multiplication of two rare polynomials. The assignment will be implemented in C++ using template classes.

a) (0.3p) Define the data structure needed to represent the term of the polynomial. This data structure (class or struct) will contain two fields:
   - the coefficient (numerical data type)
   - the exponent (integer data type)

b) (0.4p) Create a polynomial:
   - read the degree of the polynomial from the keyboard;
   - read all the terms (coefficient, exponent) from the keyboard.

Store each polynomial you use in a separate list. The list items will be of the type defined in point a). **Use a template class for the list**.

c) (0.3p) Display on the screen the content of a polynomial. Attention: terms with coefficients 0 will not be displayed!

d) (0.5p) Calculate the value of a given polynomial for a given number X. You must use the list and only simple variables for achieving the task (no array, additional lists, stacks etc.)

e) (0.5p) Implement the sum operation for two rare polynomials. Each polynomial is stored in a list. The result of the operation will be stored in a list as well. The resulting list will be displayed as a polynomial (like point c). Warning: the two polynomials can have different degrees!

f) (0.5p) Implement the multiplication operation. Each polynomial is stored in a list. The result of the operation will be stored in a list as well. Pay attention to the multiplication rules and display the final result to the standard console (like point c).

g) (0.5p) Consider polynomials with complex coefficients. Sum two polynomials using the method of point e).

1 extra point if all exercises run without compiling errors

Bonus (0.5p) - if using files for input in any of the exercises