DSA – Seminar 5

1. Written test

Allocated time: 50 minutes (until the break).

2. Projects:

- There are 33 projects. The list with the projects can be found at the course webpage.
- Most projects are composed of an ADT and a representation for the given ADT. Your task is to:
  - Implement and test the given ADT (all operations from the ADT, even if they will not be used later).
  - Find a problem that can be solved with the given ADT.
  - Implement the solution of the problem using the ADT implemented.
- There are projects were the problem is given as well, for those projects a solution to the given problem has to be implemented.
- Realization of a project consists of the implementation of an application and a documentation (on paper).
  - The application has to be implemented in C++, and it does not have to be generic (you can implement the ADT directly with the elements that will be stored in it for solving the problem). If the application is not working, the project grade will be 4.
    - Every operation from the interface of the ADT has to be tested. Write tests to achieve at least 95% coverage for all the operations from the container (try to use some tool that shows the coverage of the tests, for example: OpenCPPCoverage).
  - The documentation has to contain the following elements:
    - ADT – specification and interface (independent of the representation). If the container can be iterated, than the interface of the iterator has to be written as well.
    - Representation of the container and implementation of the operations in PSEUDOCODE (iterator as well).
    - Tests for the container.
    - Complexity of the operations from the interface, for one operation (that does not have constant complexity) include the computations done for determining the complexity.
    - The statement of the problem chosen to be solved using the given ADT. Describe why the given ADT is suitable for solving the selected problem.
    - The solution for the proposed problem: PSEUDOCODE for every function from the solution.
    - Complexity of the operations from the solution.

- Stage of the project is a **PDF document**, part of the documentation, consisting of the ADT Specification and interface, representation of the container and iterator (but no implementation) and the statement of the selected problem. Students having projects without trees (binary heap is not a tree) will have to send this document, by mail, to the seminar teacher, until the end of the day (11:59 pm) when you have the 6th Seminar.

- Students having projects with trees will have to send this document, by mail, to the seminar teacher until the end of the day (11:59 pm) when you have the 7th Seminar.
- Projects will be presented in the first three days of the exam session: 12-14 June. The schedule for the project presentations is available at the course webpage.

Project grading:

- 1p - Start
- 1p – ADT Specification and interface
- 3p – ADT Representation + Operation implementation in pseudocode
- 1p – ADT Tests
- 1p – ADT Complexities
- 1p – Problem statement
- 2p – Application implementation and complexities

Project example: *ADT Priority Queue*

Selected Problem: Scholarship allocation based on the average grade. We have *n* students and for each student we know his/her average grade. Scholarships will be given to the first *m* students with the highest average grade. Simulate the process of scholarship allocation, and display the students that will receive a scholarship.

ADT Priority Queue

Domain: PQ = {*pq* | *pq* is a priority queue with elements (e,p), of type TElem x TPriority}

Operations (interface):

init (pq, R)
push(pq, e, p)
pop(pq, e, p)
top(pq, e, p)
isEmpty(pq)
isFull(pq) – possibly, if we want pq with fixed capacity
destroy(pq)

Possible reprezentation:

- Sorted Linked List
- Sorted Dynamic Array
- Binary Heap

```
Student:
Name:string
Grade: real

Algorithm AllocateScholarship is:
      CreatePQWithStudents(pq)
```

```
      @ read m
      DisplayScholarship(pq, m)
End-algorithm

Subalgorithm CreatePQWithStudents(pq) is:
      @read n
      init (pq, „>=”)
      for i ← 1, n execute
            @ read student and average grade
            push(pq, student, average)
      end-for
end-subalgorithm

Subalgorithm DisplayScholarship(pq, m) is:
      While m > 0 and ¬ isEmpty(pq) execute
            pop(pq, student, average)
            @ print student, average
            m ← m – 1
      end-while
      destroy(pq)
end-subalgorithm
```