Data Structures and Algorithms – Lab 5

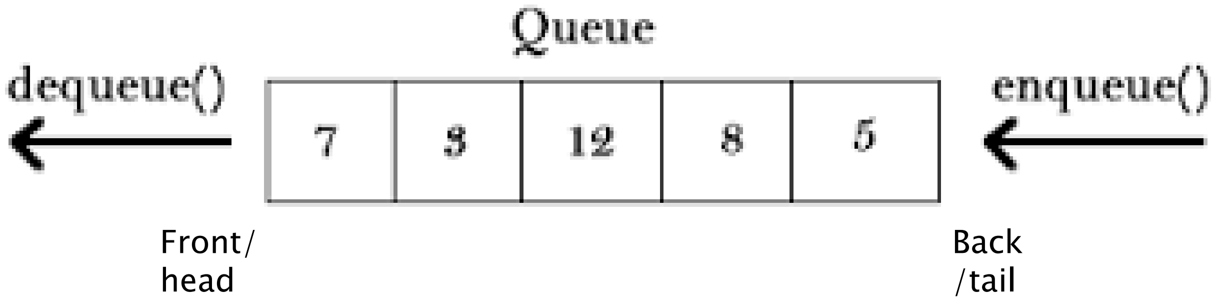
Iulia-Cristina Stanica
iulia.stanica@gmail.com

Roadmap

- Queue
- Queue vs stack
- Applications with queues
- First big homework

Queue

- Instance of an abstract data type (ADT)
- A collection of elements based on the FIFO model (first in, first out)



Applications of Queues

- Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur
- Handling requests on a server
- Printing queue of documents

Basic operations

- **enqueue(x):** (instead of push) – Adds the element x at the tail of the queue

- **dequeue():** (instead of pop()) – removes the element from the head of the queue and returns it; returns an error if the stack is empty

- **peek():** returns (but does not remove) the element at the head of the queue

- **isEmpty():** returns 1 if the queue is empty and 0 otherwise

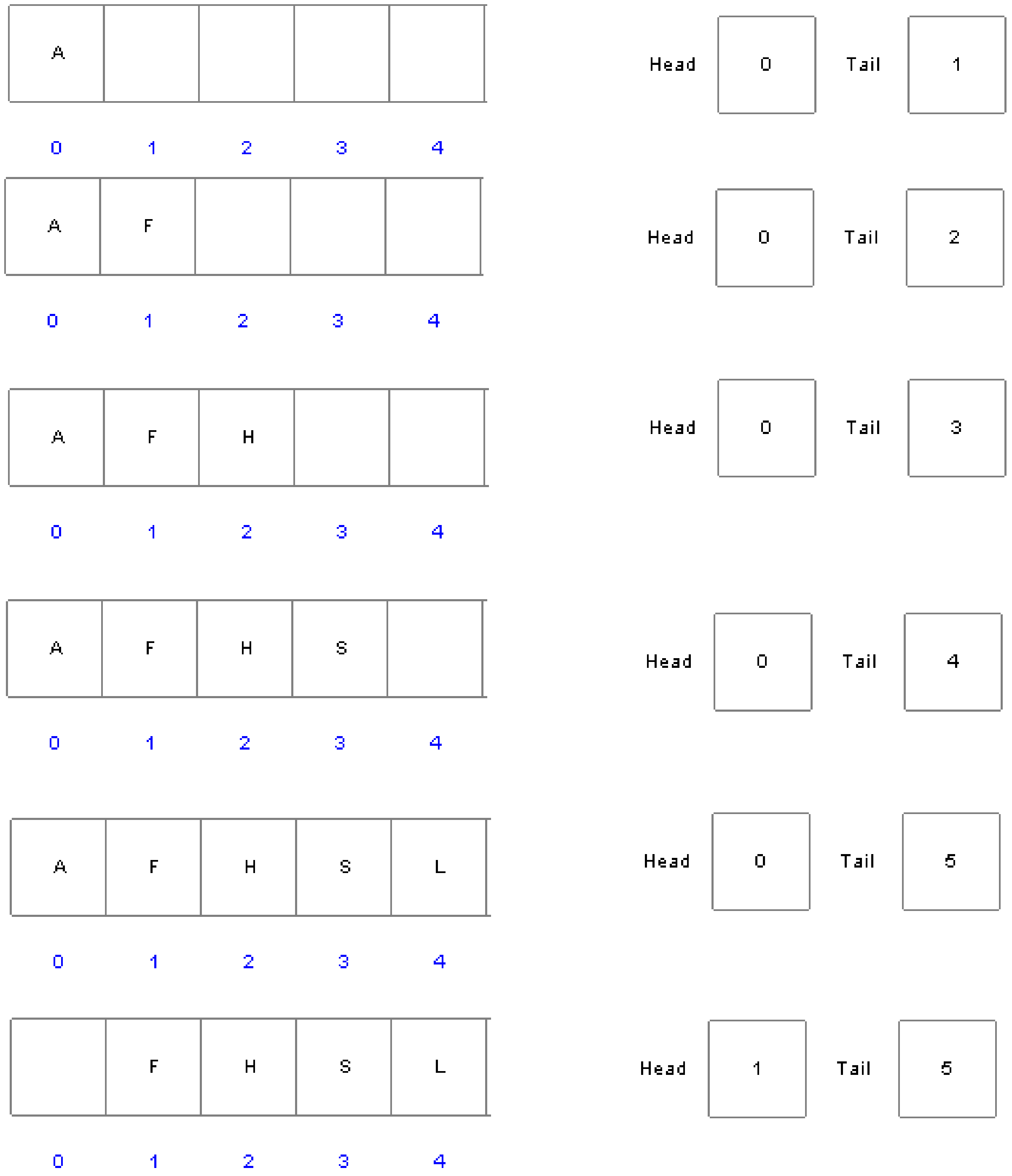
- OBS: <u>head</u> – index of the first element
  <u>tail</u> – index of the first empty position (after the last element)

1. Queue –implementation with array

Ex: Let’s check the values of “head” and “tail” after each of the following operations:

```
enqueue(‘A’);
enqueue(‘F’);
enqueue(‘H’);
enqueue(‘S’);
enqueue(‘L’);
dequeue();
```

1. Queue – implementation with array



queue1.h

```
#define NMAX 100
template<typename T> class Queue {
    private:
        T queueArray[NMAX];
        int head, tail;
    public:

        void enqueue(T x) {
            if (tail == NMAX) { //we check if it is full
                cout<<"The queue is full!\n";
                return;
            }
            queueArray[tail] = x; //we add the element on the tail position
            tail++; //we shift the tail to the right
        }

        T dequeue() {
            if (isEmpty()) { //we check if it is empty
                cout<<"The queue is empty!\n";
                T x;
                return x;
            }
            T x = queueArray[head]; //we return the first element
            head++;                 //we shift the head to the right
            return x;   }

        T peek() {
            if (isEmpty()) {//we check if it is empty
                cout<<"The queue is empty!\n";
                T x;
                return x;
            }
            return queueArray[head]; //we return the first element
        }

        int isEmpty() {
            return (head == tail); //if head and tail have the same values, the queue is empty
        }

    Queue() {
        head = tail = 0; // the queue is empty at the beginning
    }
};
```

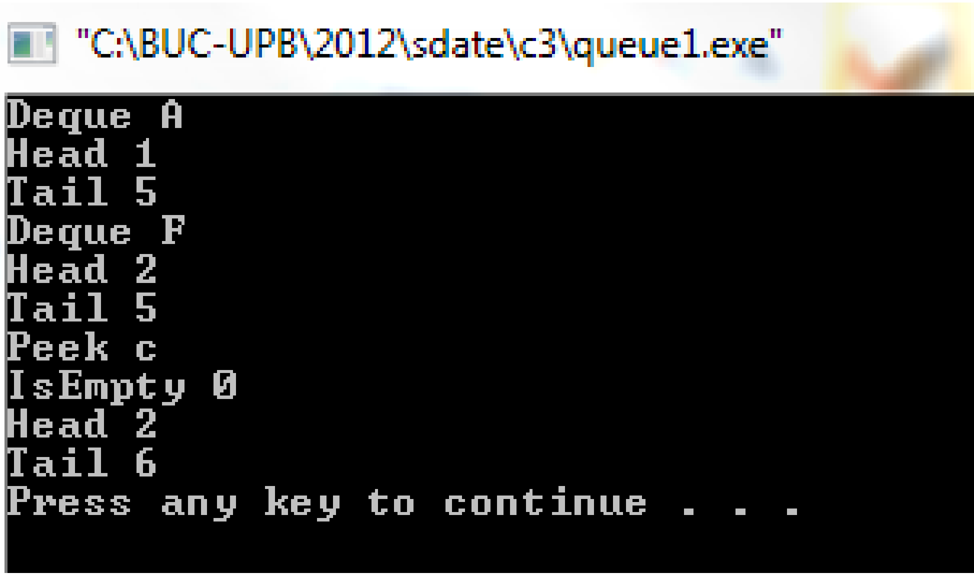
Using the queue in a .cpp file with the main function

```
#include <iostream>
#include "queue1.h"

int main() {

    Queue<char> q;

    q.enqueue('A');
    q.enqueue('F');
    q.enqueue('H');
    q.enqueue('S');
    q.enqueue('L');
    cout<<"Dequeue "<<q.dequeue()<<endl;
    cout<<"Head "<<q.getHead()<<endl;
    cout<<"Tail "<<q.getTail()<<endl;
    cout<<"Dequeue "<<q.dequeue()<<endl;
    cout<<"Head "<<q.getHead()<<endl;
    cout<<"Tail "<<q.getTail()<<endl;
    cout<<"Peek "<<q.peek()<<endl;
    cout<<"IsEmpty "<<q.isEmpty()<<endl;
    q.enqueue('X');
    cout<<"Head "<<q.getHead()<<endl;
    cout<<"Tail "<<q.getTail()<<endl;
    return 0;
}
```

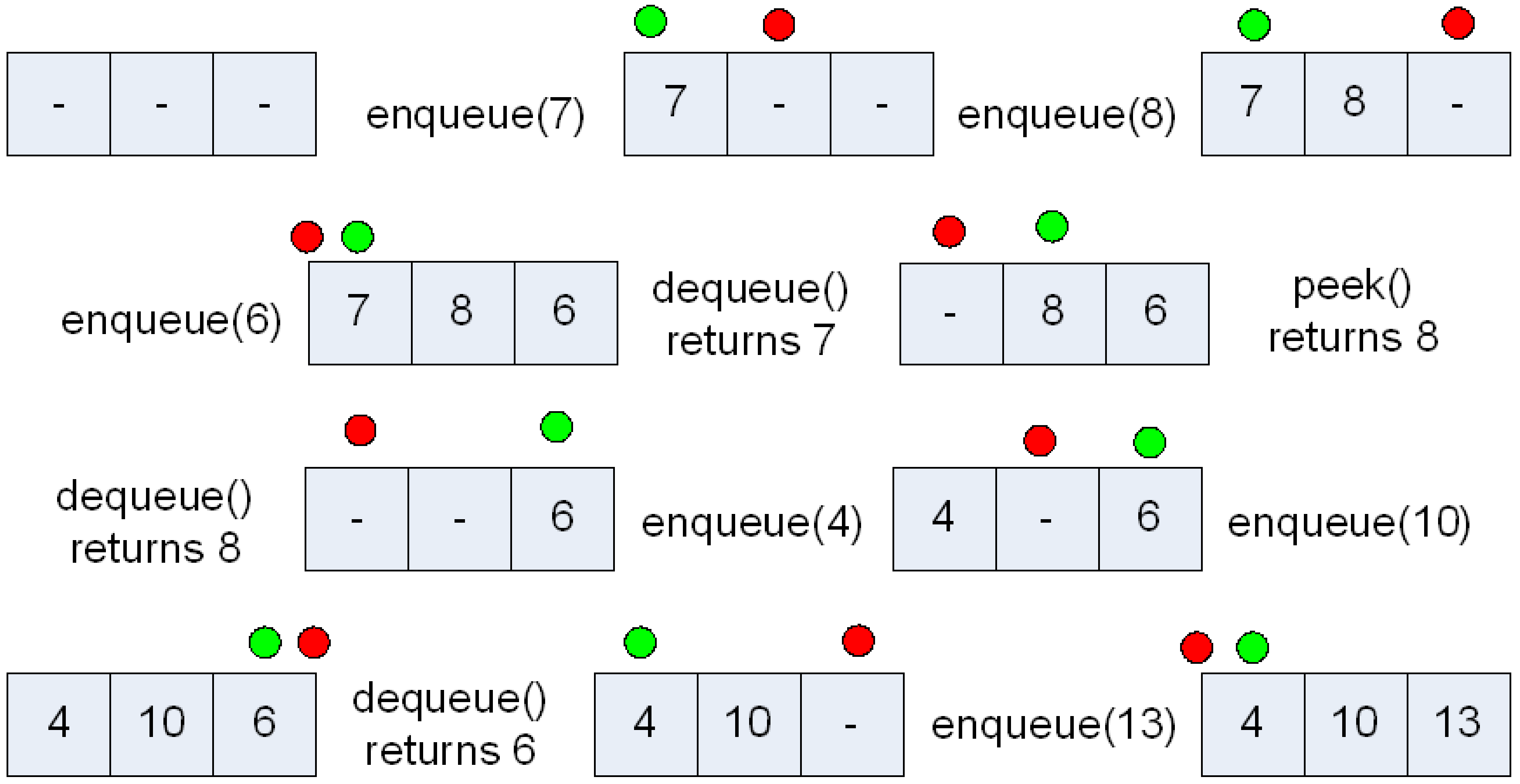

"C:\BUC-UPB\2012\sdate\c3\queue1.exe"

```
Deque A
Head 1
Tail 5
Deque F
Head 2
Tail 5
Peek c
IsEmpty 0
Head 2
Tail 6
Press any key to continue . . .
```

Problems!

- HEAD and TAIL **increase constantly**
- While we remove elements from the queue, the used part of the queue is **shifted to the right**
- We can **get to the end of the queue** and not be able to add all the remaining elements (using enqueue), even if a large part of the array (the left part) is empty
- We want to be always able to store NMAX elements.

2. Queue – circular array implementation



green = HEAD; red = TAIL; NMAX=3

Ex 1 – queue2.h

We add the size (size=total number of element) to know if the queue is full or not.

Add the code for the constructor and the methods dequeue(), peek() and isEmpty(). Test the header.

```
#define NMAX 10
template<typename T> class Queue {
    private:
        T queueArray[NMAX];
        int head, tail, size;

    public:

        void enqueue(T x) {
            if (size == NMAX) {
                cout<<"The queue is full!\n";
              return;
            }
            queueArray[tail] = x;
            tail = (tail + 1) % NMAX;
            size++;
        }
```

```
T dequeue() {
  // TO DO
}

T peek() {
  // TO DO
}

int isEmpty() {
// TO DO
}

Queue() {
 // TO DO
}
};
```

Ex 2.

- Create a class called QueuedStack **to** implement a stack using two queues (header « queue2.h »)
- The class can store values of type T (use template classes). The class has two members:

  Queue <T> q1, q2;
- The class QueuedStack has:
  - An empty constructor
  - The methods:
    - **void push(T x);**
    - **T pop();**
    - **int isEmpty();**

HINT: (one possible method)

- For **push** use the queue q1;
- For **pop** use both q1 et q2, because we should display the element situated at the "tail" of the queue (while « dequeue » returns the element situated at the « head »)

Ex 3.

- Same requests from ex 2, but we want to create a queue with 2 stacks (methods enqueue, dequeue, isEmpty).

Homework: make a messaging system using queues

- Messages are received in the order they are sent
- The classes involved are:
  - **Message**
  - **MessageSender**
  - **MessageReceiver**
- An object of type Message has a: a sender, recipient, content and a date (*struct or class* for representing dates)
- A message is placed in a queue by an object of type MessageSender
- A message is removed from the queue (dequeued) by an object of type MessageReceiver.
- Your queue class can receive any types of objects, including Message Objects (template class)
- Test your program in the main function.

Big Homework 1.

- You can find the homework on moodle fils.curs.pub.ro.
- You must use the same platform to upload your solutions.
- Deadline: 04.04.2018, 23:55. No late submission will be accepted!