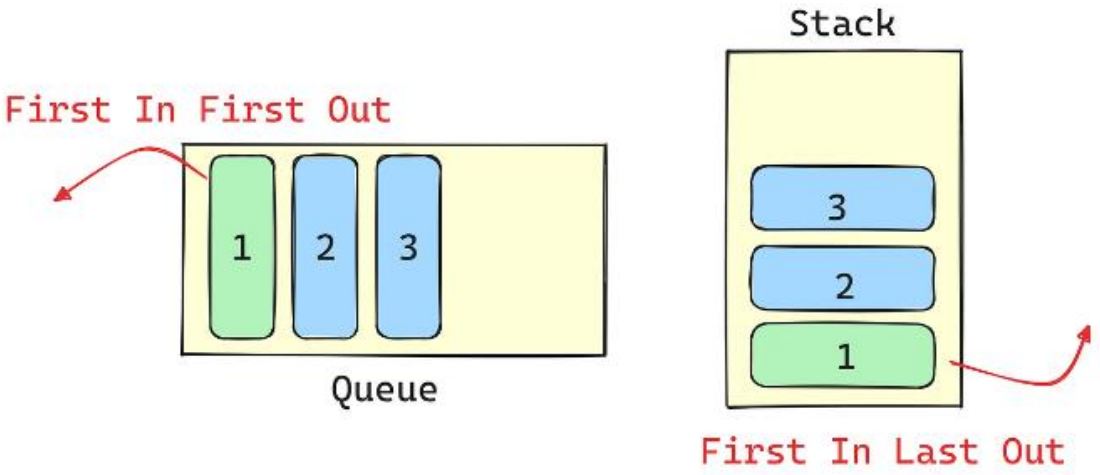
Structures de données et algorithmes – TP4

Andreea Geamanu

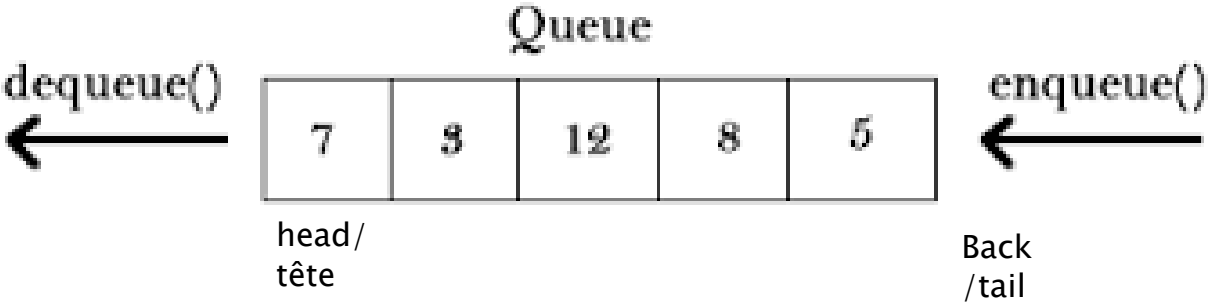
Objectifs pour aujourd'hui

- Queue (file d'attente)
- Queue vs stack
- Applications avec queues



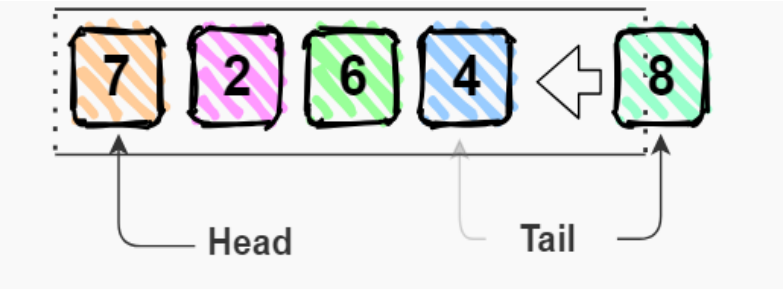
Queue (file d'attente)

- Instance d'un type de données abstraites (ADT)
- Une collection d'éléments, basée sur le modèle FIFO (first in, first out)



Opérations de base

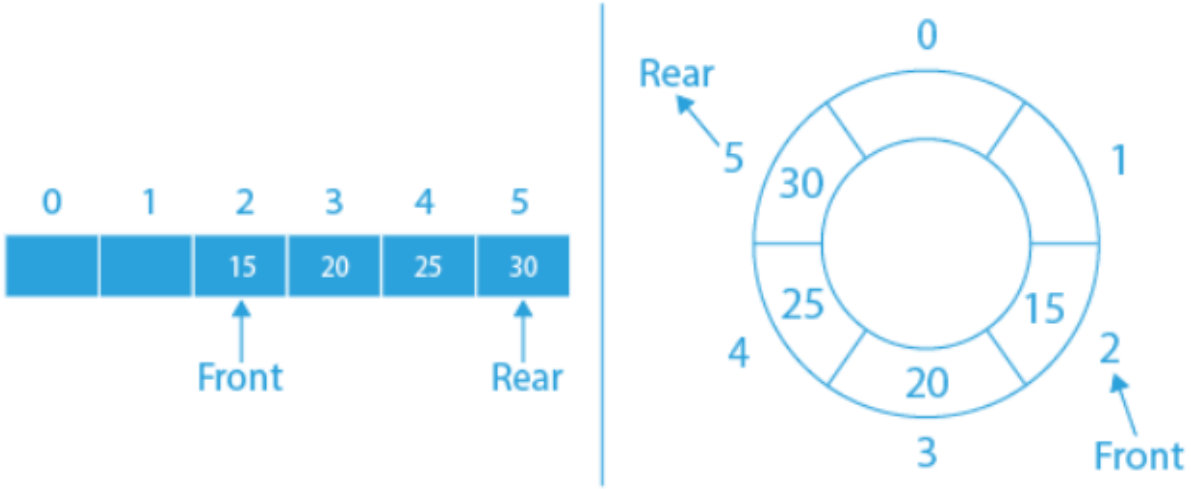
- enqueue(x): (à la place de push) – ajoute l'élément x à la queue (à l'extrémité back/tail)



- dequeue(): (à la place de pop()) – supprime l'élément qui se trouve en face de la queue (à l'extrémité front/head) et l'affiche; renvoie une erreur si la queue est vide
- peek(): renvoie (mais ne supprime pas) l'élément de la tête de la queue (l'extrémité front/head)
- isEmpty(): renvoie 1 si la queue est vide et 0 sinon
- OBS: head – indice du premier élément
  tail – indice de la première position libre (après le dernier élément)

Comment implementer la queue?

- a static data structure (array, circular array)
- a dynamic data structure (list)

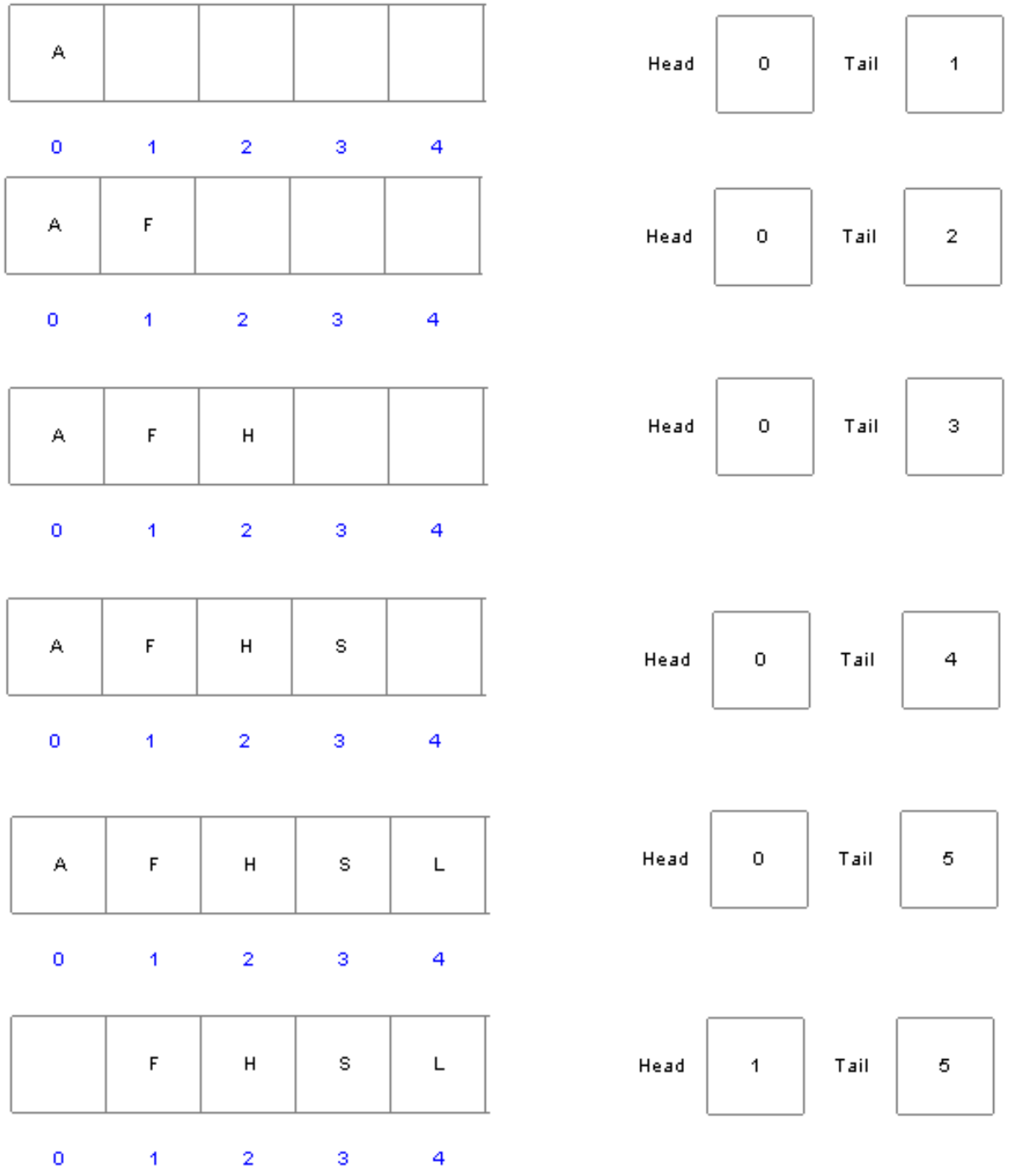


1. Queue –implementation avec tableau (Array)

Ex: Ecrivez la queue après chaque pas et les valeurs de “head” et “tail”:

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

1. Queue – implementation avec tableau (Array)



queue1.h

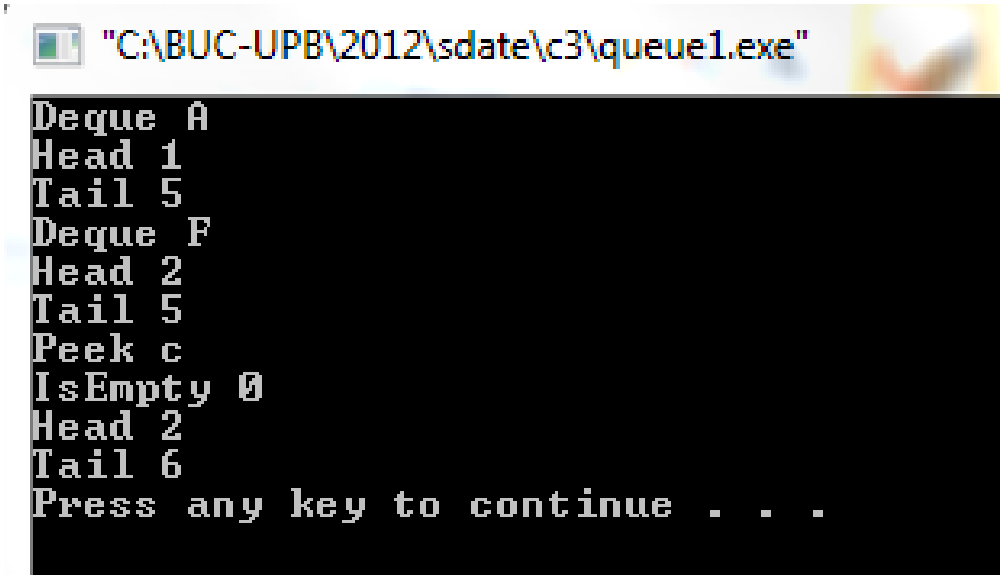
```
#define NMAX 100
template<typename T> class Queue {
  private:
    T queueArray[NMAX];
    int head, tail;
  public:

    void enqueue(T x) {
      if (tail == NMAX) { //on verifie si la queue est pleine
        cout<<"Error 101 - The queue is full!\n";
        return;
      }
      queueArray[tail] = x; //on ajoute l'element a la fin de la queue
      tail++; //on deplace le tail a droite
    }

    T dequeue() {
      if (isEmpty()) { //on verifie si la queue est vide
        cout<<"Error 102 - The queue is empty!\n";
        T x;
        return x;
      }
      T x = queueArray[head]; //on retourne l'element de la tete
      head++;                 //on deplace la tete a droite
      return x;   }

    T peek() {
      if (isEmpty()) {//on verifie si la queue est vide
        cout<<"Error 103 - The queue is empty!\n";
        T x;
        return x;
      }
      return queueArray[head]; //on retourne l'element situe dans la tete de la queue
    }

    int isEmpty() {
      return (head == tail); //si head et tail representent les memes indices, la queue est vide
    }

  Queue() {
    head = tail = 0; // la queue est vide au debut
  }
};
```

On ajoute la taille (size=nombre total d' elements) pour savoir quand la queue est pleine ou pas.

```
#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<<"Error 101 - The queue is full!\n";
      return;
      }
      queueArray[tail] = x;
      tail = (tail + 1) % NMAX;
      size++;
    }

 T dequeue() {
      if (isEmpty()) {
        cout<<"Error 102 - The queue is empty!\n";
        T x;
        return x;
      }
      T x = queueArray[head];
      head = (head + 1) % NMAX;
      size--;
      return x;
}
 T peek() {
      if (isEmpty()) {
        cout<<stderr, "Error 103 - The queue is empty!\n";
        T x;
        return x;
      }
      return queueArray[head];
    }

    int  isEmpty()  {
      return (size == 0);
    }

  Queue() {
    head = tail = size = 0;
  }
};
```

Utilisation dans un fichier .cpp avec le main

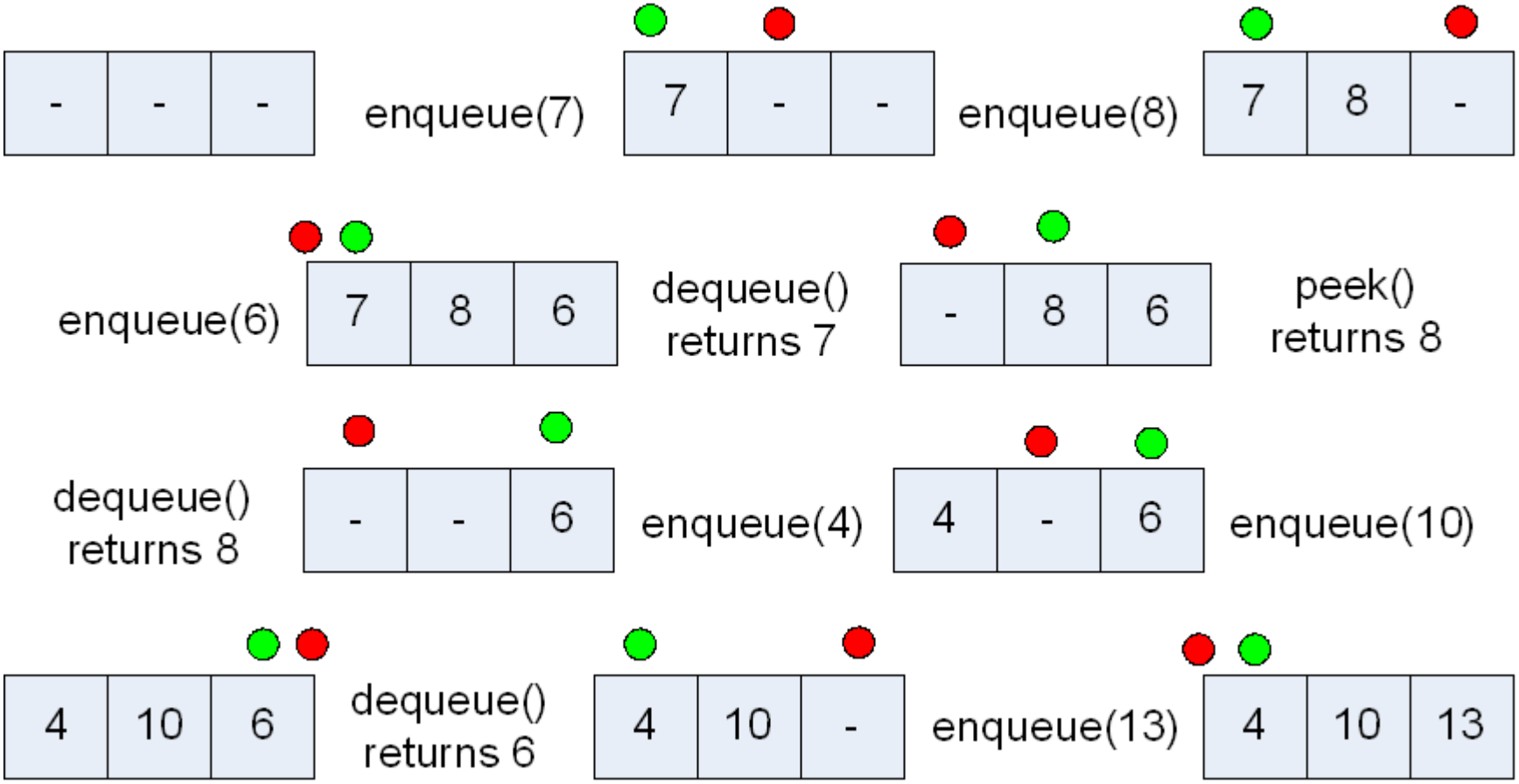
```
#include <iostream>
#include "queue1.h"

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;
}
```



Problèmes!

- HEAD et TAIL sont en constante augmentation
- Pendant que les éléments sont retirés de la file d'attente, la partie du tableau qui est effectivement utilisée est décalée vers la droite
- Nous pouvons arriver à la fin du tableau et ne pas pouvoir mettre tous les éléments dans la queue (avec enqueue), même si une grande partie du tableau (sa partie gauche) est vide.

2. Queue –implémentation avec tableau circulaire



vert = HEAD; rouge = TAIL; NMAX=3

Ex1.

a) Testez l'implémentation avec tableau de la queue (queue2.h), en utilisant un fichier de type header.
b) Ajoutez les getters pour tail et head.

Ex 2.

- Mettez en œuvre une classe appelée QueuedStack pour créer une pile (stack) avec deux queues circulaires. (header « queue2.h »)
- La classe peut stocker des valeurs arbitraires d'un type T (utiliser class template pour le type). La classe a deux variables internes (attributs):

  QueueCirc <T> q1, q2;
- La classe QueuedStack a:
  - Un constructeur vide
  - Les méthodes:
    - void push(T x);
    - T pop();
    - int isEmpty();

HINT: (une des plusieurs méthodes)

- Pour push utiliser seulement q1;
- Pour pop faire des opérations entre q1 et q2, car on doit retourner l'élément situé au "tail" de la queue (tandis que « dequeue » retourne l'element du « head »)

Ex 3.

- Mêmes demandes que pour l'ex 2, mais cette fois-ci créer une queue avec deux piles (stack). (méthodes enqueue, dequeue, isEmpty)
  - Quel est l'algorithme?

- Vous devez inclure:
  - 2 attributs Stack
  - Un constructeur
  - Un destructeur vide
  - Une méthode enqueue
  - Une méthode dequeue

Exo extra: faire un système de messagerie avec QUEUES (Files d'attente)

- Les messages sont reçus dans l'ordre où ils sont envoyés
- Les classes concernées sont les suivantes:
  - Message
  - MessageSender
  - MessageReceiver

  On a une QUEUE q des objets de type Message, utilisée dans ces classes.
- Un objet Message a: un expéditeur, un destinataire, le contenu et une date (faire un struct pour la date)
- Un message est placé dans une file d'attente par un objet MessageSender qui a une méthode « putMessage » (enqueue)
- Un message est supprimé de la file d'attente (dequeue) par un objet de la classe MessageReceiver.
- Votre classe file d'attente (QUEUE) peut recevoir tous types d'objets (TEMPLATE CLASS)
- Testez vos classes dans une fonction main.