Estructura de Dades: Pràctica 2

Pràctica 2. Estructura de dades QUEUE

1. Introducció

Objectius: Familiaritzar-se amb les estructures lineals

Temes de teoria relacionats amb la pràctica: Tema 3 Estructures de dades lineals bàsiques

2. Enunciat

Una cua (queue en anglès) és una estructura de dades lineal on els elements continguts mantenen un ordre d'arribada.

Les dues operacions més importants són la inserció (enqueue) d'un element a la part posterior i l'eliminació (dequeue) d'un element a la part davantera. D'aquesta manera que diem que una cua és una estructura FIFO (First-in-First-Out) on el primer element afegit a la cua serà el primer en ser eliminat.

Una cua pot tenir les següents operacions:

OPERACIÓ	DESCRIPCIÓ
void enqueue(int key);	Inserir l'element al final de la cua
void dequeue();	Treure el primer element de la cua
bool isFull();	Retorna cert si l'estructura està plena
bool isEmpty();	Retorna cert si l'estructura està buida
void print();	Imprimeix tots els elements de l'estructura
int getFront();	Retorna el primer element de la cua

Exercici 1. Implementeu el TAD Queue amb una implementació amb array

A continuació teniu la definició de l'especificació de la classe ArrayQueue.

```
#ifndef ARRAYQUEUE_H
#define ARRAYQUEUE_H

#include <vector>


class ArrayQueue
{
    public:
        ArrayQueue(const int max_size);
        virtual ~ArrayQueue();
        void enqueue(const int key);
        void dequeue();
        bool isFull();
        bool isEmpty();
        void print();
        const int getFront();

    private:
        int _max_size;
        int _size;
        std::vector<int> _data;
};

#endif // ARRAYQUEUE_H
```

Estructura de Dades: Pràctica 2

Realitzeu la implementació del TAD ArrayQueue. Tot seguit, codifiqueu un *main.cpp* que tingui un *menu* amb les següents opcions:

1. Inserir element a la cua
2. Treure element de la cua
3. Consultar el primer element
4. Imprimir tot el contingut de l'ArrayQueue
5. Sortir

Quan tingueu el *menu* codificat, feu les següents proves:

Cas de prova 1:

Pas	**Entrada**	**Sortida**
1	Crear un ArrayQueue de mida 3	Estructura creada
2	Inserir element 10	Element 10 agregat
3	Inserir element 20	Element 20 agregat
4	Inserir element 30	Element 30 agregat
5	Inserir element 40	EXCEPTION: L'estructura està plena
6	Imprimir ArrayQueue	[10, 20, 30]
7	Treure element	Element 10 eliminat
8	Inserir element 40	Element 40 agregat
9	Imprimir ArrayQueue	[20, 30, 40]

Cas de prova 2:

Pas	**Entrada**	**Sortida**
1	Crear un ArrayQueue de mida 3	Estructura creada
2	Inserir element 10	Element 10 agregat
3	Consultar el primer element de la cua	10
4	Inserir element 20	Element 20 agregat
5	Inserir element 30	Element 30 agregat
6	Imprimir ArrayQueue	[10, 20, 30]
7	Treure element	Element 10 eliminat
8	Consultar el primer element de la cua	20
9	Treure element	Element 20 eliminat
10	Treure element	Element 30 eliminat
11	Treure element	EXCEPTION: L'estructura està buida
12	Imprimir ArrayQueue	[]

Estructura de Dades: Pràctica 2

Exercici 2. Implementeu el TAD Queue amb una estructura enllaçada

En aquest exercici es demana dissenyar i implementar l'estructura de dades LinkedQueue com a estructura dinàmica amb encadenaments.

Aquesta LinkedQueue està formada per nodes de tipus TAD Node. Aquest TAD Node conté com a mínim dos atributs: *element,* on es guarda l'element a inserir a la cua i *next*, l'apuntador al següent node.

L'especificació amb el mínim d'operacions necessàries del **TAD Node** és el següent:

- **constructor**: construeix el node amb l'element que rep com a paràmetre
- **getElement**: retorna l'element que hi ha guardat al node
- **getNext**: retorna l'adreça del següent node o *nullptr* en cas que no hi hagi següent
- **setNext**: modifica l'adreça del següent per l'adreça rebuda com a paràmetre

A continació es presenta l'especificació del TAD LinkedQueue.

```
#ifndef LINKEDQUEUE_H
#define LINKEDQUEUE_H

#include "Node.h"

class LinkedQueue
{
    public:
        LinkedQueue();
        virtual ~LinkedQueue();
        LinkedQueue(const LinkedQueue& q);
        void enqueue(const int key);
        void dequeue();
        bool isEmpty();
        void print();
        const int getFront();

    private:
        int _size;
        Node* _front;
        Node* _rear;
};

#endif // LINKEDQUEUE_H
```

Fixeu-vos en alguns detalls d'aquesta LinkedQueue:

- La cua s'ha definit amb nodes unidireccionals de tipus Node.
- Cal implementar el ***constructor de còpia***. Aquest ha de duplicar la cua encadenada de forma que els espais de memòria dels nodes de la cua des d'on es copia i de la cua del final siguin diferents. No n'hi ha prou amb copiar la direcció dels punters *_front* i *_rear*, sinó que cal fer-ne un de completament nou per cada un dels elements de la cua original.
- La funció ***isEmpty*** retorna *true* si la LinkedQueue està buida, *false* en cas contrari.
- El mètode ***print*** ha d'imprimir per consola tots els elements de la LinkedQueue.
- Es pot consultar el primer element de la cua amb ***getFront***.
- El destructor elimina tots els elements de la cua i la deixa buida.

Per provar el vostre TAD LinkedQueue codifiqueu un *main.cpp* que tingui un *menu* amb les mateixes opcions i casos de prova de l'exercici anterior.

Exercici 3. Utilitzeu el TAD LinkedQueue per a resoldre el següent problema.

Es vol realitzar la gestió d'una cua de sortida de vols d'un aeroport. Els vols tenen un identificador, un origen, un destí i una hora de sortida. Per exemple:

RYR1013,Barcelona,Londres-Stansted,dom 12:42PM WET

RYR1025,Barcelona,Dublin,dom 12:56PM WET

RYR103,Barcelona,Londres-Gatwick,dom 13:26PM WET

RYR1062,Barcelona,Madrid-Barajas,dom 15:03PM WET

Utilitzarem el nostre TAD LinkedQueue implementat a l'exercici 2. Per a aquest exercici us caldrà implementar una classe **Flight** que contindrà la informació del vol descrita anteriorment. A més a més, caldrà implementar un tipus de node específic per a poder emmagatzemar els vols. L'especificació del TAD **FlightNode** és la següent:

```
#ifndef FLIGHTNODE_H
#define FLIGHTNODE_H

#include "Flight.h"

class FlightNode
{
    public:
        FlightNode(Flight& f);
        virtual ~FlightNode();
        Flight& getElement();
        FlightNode* getNext();
        void setNext(FlightNode& n);

    private:
        Flight* _element;
        FlightNode* _next;
};

#endif // FLIGHTNODE_H
```

Adoneu-vos que per fer funcionar la LinkedQueue únicament caldrà que canvieu el tipus de node pel **FlightNode**. La lògica d'encuar i treure elements de la LinkedQueue hauria de romandre igual.

Utilitzeu el TAD LinkedDeque implementat a l'exercici 2 per crear un programa que contingui les següents opcions de menú.

1. Llegir un fitxer amb les entrades de vols
2. Eliminar un vol
3. Inserir **n** entrades de vols des de teclat (0 per finalizar)
4. Imprimir la cua de vols

Fixeu-vos que la primera opció del menú és per introduir les dades des d'un fitxer de text. El format del fitxer és una línia per cada vol i els camps estan separats per una coma. Teniu un exemple a l'inici de l'explicació de l'exercici.

Estructura de Dades: Pràctica 2

3. Lliurament

Implementar els exercicis en C++ i usar el **NetBeans 8.2 amb C++ versió 11**. Creareu un projecte NetBeans per a cada exercici. Lliurar el codi C++ corresponent als vostres exercicis en una carpeta anomenada codi, amb una subcarpeta per a cada exercici. Els comentaris de cada exercici (observacions, decisions preses i resposta a les preguntes plantejades, si n'hi ha) els fareu com a comentari al fitxer main.cpp de cada exercici.

Com a màxim el dia del lliurament es penjarà en el campus virtual un fitxer comprimit en format ZIP amb el nom del grup (A, B, C, D, E o F), el nom i cognoms de la persona que ha fet la pràctica i el número de la pràctica com a nom de fitxer. Per exemple, **GrupA_BartSimpson_P2.zip**, on P2 indica que és la "pràctica 2". El fitxer ZIP inclourà: la carpeta amb tot el codi.

Els criteris per acceptar la pràctica són:

- La pràctica ha de funcionar en la seva totalitat.
- La pràctica ha de ser orientada a objectes.

IMPORTANT: La còpia de la implementació de la pràctica implica un zero a la nota de pràctiques de l'assignatura per les persones implicades (tant la que ha copiat com la que ha deixat copiar).

4. Planificació

Per aquesta pràctica disposeu de dues setmanes però tingueu en compte que només teniu una sessió de laboratori amb el vostre professor de pràctiques.

Els professors us proposem la següent planificació:

- **Setmana 1 (del 25 de febrer al 1 de març)** →laboratori amb el professor
  - Implementació de l'exercici 1
- **Setmana 2 (del 4 de març al 8 de març)** →laboratori amb el professor
  - EXAMEN PRÀCTICA 1
  - Implementació de l'exercici 2
  - Tasca al campus virtual per lliurar exercici 1 i 2
- **Setmana 2 (del 11 de març al 15 de març)** →laboratori amb el professor
  - Implementació de l'exercici 3
  - Tasca al campus virtual per lliurar exercici 1, 2 i 3

El lliurament final de la pràctica serà el **dia 17 de març de 2019** al campus virtual. Recordeu que s'han de lliurar tots els exercicis en el fitxer .zip al campus virtual.