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. Per tant, 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(const 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
`const int getFront();`	Retorna el primer element de la cua
`void printFrontRear();`	Imprimeix per pantalla els indexes del front i del rear

Exercici 1. Implementeu el TAD QueueStatic amb una implementació estàtica circular.

A sota teniu la definició de l'especificació de la classe QueueStatic.

- No podeu modificar el .h de la classe QueueStatic, ni afegir més mètodes o atributs.
- Com que estem treballant amb una implementació circular, afegirem un mètode printFrontRear() per veure els indexes del front i del rear i, així, podreu comprovar que la implementació és correcta.

Fixeu-vos bé en aquesta implementació i responeu les següents preguntes:

- Quina condició hem d'imposar en una cua estàtica no circular per comprovar si està plena? Aquesta condició és suficient en el cas de ser circular?
- La variable _max_elements és necessària en el cas de ser una cua no circular? Per a què utilitzem aquesta variable per les cues circulars?

Contesteu aquestes preguntes en un comentari al final del TAD.

```
1   #ifndef QUEUESTATIC_H
2   #define QUEUESTATIC_H
3
4   class QueueStatic {
5   public:
6       QueueStatic(const int max_size);
7       virtual ~QueueStatic();
8       void enqueue(const int key);
9       void dequeue();
10      bool isFull();
11      bool isEmpty();
12      void print();
13      const int getFront();
14      void printFrontRear();
15
16  private:
17      int _max_elements;
18      int _num_elements;
19      int _first;
20      int _last;
21      int* _content;
22  };
23
24  #endif /* QUEUESTATIC_H */
```

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

1. Inserir element a la QueueStatic
2. Treure element de la QueueStatic
3. Consultar el primer element de la QueueStatic
4. Imprimir tot el contingut de la QueueStatic
5. Imprimir les posicions del front i el rear
6. Sortir

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

Cas de prova 1:

Pas	Entrada	Sortida
1	Crear una QueueStatic de mida 3	Estructura creada
2	Inserir element 10	Element 10 agregat
3	Inserir element 20	Element 20 agregat
4	Imprimir front i rear	Front: 0, Rear: 2
5	Inserir element 30	Element 30 agregat

6	Inserir element 40	EXCEPTION: L'estructura està plena
7	Imprimir QueueStatic	[10, 20, 30]
8	Imprimir front i rear	Front: 0, Rear: 0
9	Treure element	Element 10 eliminat
10	Inserir element 50	Element 50 agregat
11	Imprimir QueueStatic	[20, 30, 50]
12	Imprimir front i rear	Front: 1, Rear: 1

Important:

- En aquest primer exercici no s'ha de fer el TAD QueueStatic amb templates
- Suposem que els elements de la QueueStatic seran de tipus enter
- El constructor crea un array dinàmic de mida max_elements
- El destructor elimina l'array dinàmic de memòria
- Les excepcions que retorneu seran de tipus string

Cas de prova 2:

Pas	Entrada	Sortida
1	Crear una QueueStatic 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 QueueStatic	[10, 20, 30]
7	Imprimir front i rear	Front: 0, Rear: 0
8	Treure element	Element 10 eliminat
9	Consultar el primer element de la cua	20
10	Treure element	Element 20 eliminat
11	Imprimir front i rear	Front: 2, Rear: 0
12	Treure element	Element 30 eliminat
13	Treure element	EXCEPTION: L'estructura està buida
14	Imprimir QueueStatic	[ ]
15	Imprimir front i rear	Front: 0, Rear: 0

Exercici 2. Implementeu el TAD Queue amb una implementació d'estructura enllaçada

En aquest exercici es demana dissenyar i implementar l'estructura de dades QueueLinked com a estructura dinàmica amb encadenaments simples sense sentinelles.
Aquesta QueueLinked 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

Estructura de Dades: Pràctica 2

A continuació es presenta l'especificació del TAD QueueLinked.

```
1   #ifndef QueueLinked_H
2   #define QueueLinked_H
3
4   #include <iostream>
5   #include "Node.h"
6
7   using namespace std;
8
9   template <class Type>
10  class QueueLinked {
11  public:
12      QueueLinked();
13      QueueLinked(const QueueLinked<Type>& q);
14      ~QueueLinked();
15      bool isEmpty();
16      void print();
17      void enqueue(const Type key);
18      void dequeue();
19      const Type getFront();
20      void printFrontRear();
21
22  private:
23      Node<Type>* _first;
24      Node<Type>* _last;
25  };
```

Fixeu-vos en alguns detalls d'aquesta QueueLinked:

- 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 *_first* i *_last*, 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 QueueLinked està buida, *false* en cas contrari.
- El mètode ***print*** ha d'imprimir per consola tots els elements de la QueueLinked.
- 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 QueueLinked codifiqueu un *main.cpp* que tingui un *menú* amb les mateixes opcions i casos de prova de l'exercici anterior.

Important:

- En aquest segon exercici el TAD QueueLinked es farà amb templates
- No podeu modificar ni ampliar l'especificació de la classe QueueLinked
- Les excepcions que retorneu seran de tipus string o d'alguna de les classes que hi ha per controlar excepcions a C++, per exemple **out_of_range**.

Exercici 3. Recomanacions de mUBiesflix amb la QueueLinked

A **mUBiesflix** volem desenvolupar un sistema recomanador de pel·lícules, pel que necessitarem una estructura amb la qual gestionar les recomanacions. Farem servir una cua; concretament, el nostre TAD QueueLinked implementat a l'exercici 2. Per a aquest exercici us caldrà implementar una classe **Peli** que contindrà la informació de la pel·lícula recomanada. A continuació teniu la seva especificació i representació.

```
1   #ifndef PELI_H
2   #define PELI_H
3
4   #include <string>
5   #include <vector>
6   #include <iostream>
7   using namespace std;
8
9   class Peli {
10  private:
11      int peliId;
12      int directorId;
13      string titol;
14      int durada;
15      float valoracio;
16
17  public:
18      Peli (int, int, string, int, float);
19      Peli ();
20      ~Peli();
21      int getPeliId();
22      int getDirectorId();
23      string getTitol();
24      int getDurada();
25      float getValoracio();
26      void print();
27      string toString() const;
28      friend std::ostream& operator<<(std::ostream& os, const Peli& obj);
29  };
30
31  #endif // PELI_H
```

Heu de crear un programa que contingui les següents opcions de menú.

1. Llegir un fitxer amb les entrades de les pelis
2. Eliminar una peli
3. Inserir **n** entrades de pelis des de teclat (0 per finalitzar)
4. Imprimir per pantalla la cua de pelis
5. Sortir

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 peli i els camps estan separats per una coma. Teniu un exemple aquí:

```
862,7879,Toy Story,81,7.7
8844,4945,Jumanji,104,6.9
15602,26502,Grumpier Old Men,101,6.5
31357,2178,Waiting to Exhale,127,6.1
11862,56106,Father of the Bride Part II,106,5.7
```

Estructura de Dades: Pràctica 2

Important:

- En aquest tercer exercici el TAD QueueLinked seguirà sent amb templates
- No podeu modificar i/o ampliar l'especificació de la classe QueueLinked
- Per tal d'imprimir les pelis són necessaris dos mètodes a la classe Peli.
    1. Fer un mètode per transformar la peli en un string i que el retorni. L'especificació del mètode és: `string toString() const`
    2. Afegir la sobrecarrega del `operator<<`

    - A l'especificació posarem

```
friend std::ostream& operator<<(std::ostream& os, const Peli& obj);
```

    - A la implementació posarem:

```
std::ostream& operator<<(std::ostream& os, const Peli& obj){
    os << obj.toString();
    return os;
}
```

3. Lliurament

Implementar els exercicis en C++ i usar el **Visual Studio Code amb C++ versió 11**. Creareu un projecte de Visual Studio Code 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 compilar i executar en la seva totalitat. I, la pràctica ha de ser orientada a objectes. Si una part no us funciona, comenteu el codi.

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 tres setmanes. Els professors us proposem la següent planificació:

- **Setmana 1 (del 3 de març al 9 de març)** → LAB 4 amb el professor
    - **EXAMEN PRÀCTICA 1**
    - Implementació de l'exercici 1
- **Setmana 2 (del 10 de març al 16 de març)** → LAB 5 amb el professor
    - Implementació de l'exercici 2
- **Setmana 3 (del 17 de març al 23 de març)** → LAB 6 amb el professor
    - Implementació de l'exercici 3
- **Setmana 3 (del 24 de març al 30 de març)** → LAB 7 amb el professor
    - Implementació de l'exercici 3 i comentar el codi
    - Tasca al campus virtual per lliurar tota la pràctica

Estructura de Dades: Pràctica 2

El lliurament final de la pràctica serà el **dia 30 de març de 2025** al campus virtual. Recordeu que s'han de lliurar tots els exercicis en el **fitxer .zip** al campus virtual.

La Setmana del 31 de març al 4 d'abril teniu la prova pràctica de la pràctica 2 al LAB8.

- És obligatori lliurar la pràctica al campus virtual la setmana anterior
- La prova consistirà en una modificació o ampliació de la pràctica lliurada