Estructura de Dades

Pràctica 1: Estructures enllaçades: pila, cua i cua circular

UNIVERSITAT DE BARCELONA

Blai Ras Jimenez i Albert Morales, Parella 5

Professor: Pere Hierro, Grup D

1. Exercici 1

En l'exercici 1 es demana un TAD de pila en Array, és a dir, un ArrayStack que contingui diferents caràcters de l'alfabet seguint les directius d'un Tipus Abstracte de Dades.

Es demanen els mètodes característics d'un TAD Pila:

1. ArrayStack(): Constructor de la nostra pila.
2. Bool empty(): booleà que em retornarà cert si la pila està buida, fals si no ho està.
3. Bool full(): booleà que em retornarà cert si la pila està plena, fals si no ho està.
4. Void push(const char item): mètode que inserta un element passat per paràmetre a la pila
5. Void pop(): mètode que treu l'últim element insertat
6. Char top(): mètode que retorna l'últim element insertat (en aquest cas de tipus char)
7. Void print(): mètode que imprimeix la informació de tota la pila per pantalla.

Tal i com hem vist en la teoria, un TAD Pila inserta i elimina segons l'esquema LIFO (last-in First-out), i ho fem gràcies als mètodes de push(char item) i pop().

Tot i això, pot ocorre algun error, per tant, necessitem gestionar aquestes "excepcions". En aquest cas, controlem els errors de pila buida i pila plena, és a dir, si intentem afegir caràcters a una pila plena o si intentem eliminar caràcters d'una pila buida. Per altra banda, els mètodes top() i print() també gestionen excepcions, ja que no podem retorna el primer element d'una pila buida o imprimir una pila buida!

Per fer-ho, ens ajudem dels mètodes empty() i full(), ambdós booleans, i dels administradors de excepcions “runtime_error”, EmptyStackException i FullStackException. **Hem fet servir els dos tipus d’excepcions per investigar una mica i aprendre els dos mètodes.**

La única diferència es que “runtime_error” no necessita classe pròpia en la capçalera, sinó que amb un “import” ja ens val. En canvi, les excepcions EmptyStackException() i FullStackException() son classes que hem hagut d’implementar en la capçalera però fan la mateixa i exacta funció.

Després de comprovar que la nostra pila afegeix i suprimeix caràcters, és a dir, testejant tots els mètodes i imprimint per veure si està tot correcte, ens disposem a fer la part dos del exercici: comprovar si els parèntesi, claus i claudàtors estan ben aparellats d’una expressió numèrica.

Per fer-ho, implementem en el main un codi que analitza un string passat per teclat per l’ususari. Per tant, fem us d’un iterador for (char c: <string>) on <string> es l’expressió entrada.

El primer que fem es un while, que farà un bucle sempre i quan l’string passat tingui caràcters. Un cop dins del while, si ens trobem un parèntesis del tipus (), {} o [ ] obert, l’afegim al stack amb el mètode push(). De totes formes, ho fem amb estructura try-catch per si sorgeix algun error, com per exemple de tipus pila plena:

```
while (!err && chars.length() > i){ // mentres no hi ha errors
    if(chars[i] == '(' || chars[i] == '[' || chars[i] == '{'){/
        try{
            stack->push(chars[i]);// introdueix el caràcter act
        }catch(FullStackException* exception){
            cout << exception->error;
            err = true;
        }
```

Un cop afegits, analitzarem, a dins del while, amb un else if, els parèntesis que tanquen. Eliminarem aquets parèntesis si compleixen la condició chars[i] == ‘), },]’ i que l’últim element afegit sigui el mateix parèntesis però tancat, tal i com hem fet abans.

Ho fem també amb estructura try-catch, per gestionar les excepcions de pila buida, la qual llençarà un missatge d’error. Un cop completat l’anàlisi del string passat, gestionem amb un simple if si l’expressió és correcte. Com que hem eliminat cada parèntesis si havíem trobat el seu corresponent, la pila ha de quedar buit. Per tant, si l’stack està buit, retornarem un missatge afirmatiu, sinó, retornarem un missatge del tipus “L’expressio esta ben aparellada":

```
if(err || !stack->empty()){// si la pila no està buida
    cout << "L'expressio esta mal aparellada" << endl;

}else{// sinó fer...
    cout << "L'expressio esta ben aparellada" << endl;
}

return 0;
```

Preguntes

1. ***Com has comprovat amb una pila que l'expressió està ben aparellada?***

   Explicat a dalt.

2. ***Quins errors controles i com ho fas?***

   En els mètodes de la cua, controlo:

   - A push(), que la pila no estigui plena, ja que sinó no puc afegir més caràcters.
   - A pop(), que la pila no estigui buida, ja que sinó no puc eliminar res.
   - A top(), que la pila no estigui buida, ja que sinó no puc retornar res.
   - A print(), que la pila no estigui buida, ja que sinó no imprimeixo res.

   En el codi per comprovar si una expressió està ben introduïda, al main, controlo les excepcions de:

   - Stack ple, quan intento afegir els parèntesis oberts.
   - Stack buit, quan intento eliminar els parèntesis tancats.

     D'aquesta manera, si l'expressió està mal introduïda, no ens sortirà un missatge d'error i "petarà", sinó que ens retornarà el motiu del error juntament amb el missatge "L'expressio esta ben aparellada".

   Les excepcions per tant son dues: EmptyStackException() i FullStackException(), i estan implementades en la capçalera i en els source files, com a classe. D'aquesta manera, jo puc cridar una excepció fent "throw new EmptyStackException();" o "EmptyStackException* exception".

2. Exercici 2

En l'exercici dos es demana implementar un TAD de cua en Array d'enters. La capacitat màxima d'aquest serà 100 i es demanen els següents mètodes:

1. Void enqueue(int x): mètode que afegeix un enter passat per paràmetre al final de la cua

2. Int dequeue(): mètode que elimina el primer element de la cua i el retorna

3. Int front(): aquest mètode no el demanen. L'he implementat per investigar i perquè en l'exercici 1 es demana. Retorna el primer element de la cua.

4. Int size(): mètode que em retorna el número d'elements afegits a la cua.

5. Bool empty(): booleà que em retorna cert si la cua està buida, fals si no ho està.

6. Bool full(): booleà que em retorna cert si la cua està plena, fals si no ho està.

7. Void print(): mètode que m'imprimeix la informació de la cua per pantalla.

Per tal de realitzar tots aquests mètodes, ens ajudarem de tres variables: primer, últim i n. “Primer” correspon al índex del primer element, “últim” correspon al índex del últim element, mentre que “n” és la mida, el nombre d’elements afegits a la cua. Últim i Primer són inicialitzats a -1 en el constructor per gestionar problemes.

Tal i com hem vist a teoria, un TAD Cua insereix per darrera i elimina per davant, és a dir, segueix l’esquema FIFO (first-in first-out). Els mètodes encarregats de fer-ho són enqueue(int x) i dequeue().

Com a l’exercici anterior, pot ser que intentem afegir un enter a una cua plena o que intentem eliminar un enter d’una cua plena, així que també necessitarem gestionar excepcions. En aquest exercici, son totes de tipus “runtime_error”, i controlen amb l’ajuda dels mètodes full() i empty() que no es facin cap de les operacions anteriors.

Paral·lelament, els mètodes print() i front() també faran ús d’aquests mètodes i de les excepcions de tipus “Cua buida”, ja que no podem retornar el primer element d’una cua buida o imprimir-la.

Preguntes

1. ***Quina és la sortida del main?***

La sortida del main és la següent:

```
Mida actual de la cua: 0
Encuem 3 elements a la cua...
1 2 3
Cua plena (0:no, 1:si): 0
Treiem 1er element de la cua: 1
2 3
Treiem 2on element de la cua: 2
Encuem 2 elements a la cua...
Treiem 3er element de la cua: 3
Mida actual de la cua: 2
Cua buida (0:no, 1:si): 0

RUN SUCCESSFUL (total time: 1s)
```

Es demana la mida de la cua, zero, no em afegit res.

S'encuen 3 elements, els enters u, dos i tres.

Es comprova la seva inserció amb el mètode print().

Es comprova si la cua està plena. Un booleà al ser cridat per imprimir per pantalla retorna 1 si és cert i 0 si no ho és. Retorna zero, la cua no està plena!

Es treu el primer element de la cua. Per tant, es retorna i s'elimina el 1. Es crida el mètode print() per comprovar aquesta eliminació. S'imprimeix 2 i 3, la cua restant.

Es treu el segon element de la cua, el dos. Es retorna un dos. S'encuen dos enters més, 4 i 5.

Es treu el tercer element de la cua, el tres. Es retorna un tres, per tant.

Es demana la mida de la cua. Es retorna un dos, corresponent als números 4 i 5.

Es comprova si la cua està buida. Es retorna un zero, no, no està buida.

2. ***Quins errors controleu dels que es poden produir quan es demana afegir o eliminar elements de la cua?***

Explicat anteriorment, en el desenvolupament.

3. ***Què feu per controlar els errors i com ho feu?***

També ho he explicat en el desenvolupament. Només explicaré com són les excepcions de tipus "runtime_error". Aquesta excepció no necessita implementar una classe ja que ja es troba dins de la classe "stdexcept".

Per tant, fem un "include" d'aquesta classe, de manera que per gestionar una excepció qualsevol programarem un "throw" de runtime_error("missatge d'error);.

I d'aquesta manera no fa falta implementar classes de tipus excepcions de cues buides i plenes en la capçalera i en el .cpp per després ser cridades en els mètodes. Ho podem veure per exemple en el mètode enqueue(int x):

```
void ArrayQueue::enqueue(int x) { //Em pasen el numero per parametre
    if (!full()) { //Si la pila no esta plena...
        ultim++; //Incrementa l'index
        data[ultim%MAX_QUEUE] = x; //Afageixlo
        n++; //Incrementa la mida
    }else {
        throw runtime_error("La cua esta plena!"); //Excepcio
    }

}
```