T-201-GSKI, GAGNASKIPAN

Vor 2018

INHERITANCE - QUEUE/STACK

Assignment grading. For full marks your solution needs to be accepted by Mooshak. Points may be deducted for

- not implementing the solution according to the specifications,
- redundant or repeated code.
- messy code,
- code that is difficult to understand and not explained in comments.

If your solution is not accepted by Mooshak, it will receive a maximum grade of 7.

QUEUE/**STACK** using **LINKED LIST** and **BUFFER** (dynamic array)

Your assignment is to make abstract data types (ADT) for **Queue** and **Stack** end then implement them using both a **singly-linked list** and a **dynamic array**. The classes Queue and Stack should be abstract classes with no implementation, while the classes LinkedList and BufferList should inherit them both and implement the functionality to actually store the data entered into them.

Under “other material” you can find an implementation for ***“Lausn við QueueStack með Templates (verkefni frá 2017)”*** which uses a LinkedList implementation with templates encapsulated in the classes Queue and Stack.

*The assignment will be in **four different parts** on Mooshak:*

- Part 1 only needs an implementation of Queue, Stack and LinkedList. **50%**
- Part 2 only needs an implementation of Stack and BufferList. **20%**
- Part 3 only needs an implementation of Queue and BufferList. **20%**
- Part 4 needs an implementation of Queue, Stack, LinkedList and BufferList. **10%**

You get three attempts to return each part of the assignment.

The Mooshak tests will ***not*** call the stream operator << or your print functions, so their output can be however you can best read it, and is only used for your own tests. The simplest way to print the contents may in some cases be a reverse order in LinkedList compared to BufferList. This is fine, so long as you understand the output.

Part 1:

Edit this implementation (or your own) so that Queue and Stack are ADTs and LinkedList inherits both and implements the pure virtual functions push, pop, add and remove. The overridden function can then either call base function on the LinkedList class itself (like the encapsulation) or you can implement the functionality within the overridden functions. Just remember that two of those overridden functions use exactly the same functionality.

All classes, Queue, Stack and LinkedList are implemented as template classes, so they can be tested for different data types. Remember that the simplest way to get template classes to compile correctly is to implement them entirely within the .h header files.

Queue and Stack are both abstract classes, with only pure virtual functions.

There is one exception to this, which is the stream << overload which is a friend function and can't be virtual. In order to make that functionality virtual we implement a pure virtual print() function which the ostream from the << overload is sent into. The print function can now be implemented in derived classes like any other virtual function.

The class LinkedList must now inherit both classes, Stack and Queue (make sure you forward the template <T> correctly) and implement their pure virtual functions. Queue has the functions add() and remove() and Stack has the functions push() and pop() and they both have the function print(). All of these must be implemented in LinkedList.

The functionality behind them can be as simple as calling functions you already have on the linked list, push_front, push_back and pop_front (or headInsert, tailInsert and headRemove).

Part 2:

Now take the implementation base for BufferList and make it inherit Stack in the same way as LinkedList does, and implement/override its virtual functions. The implementation must accept and return values in exactly the same way, so the outside functionality is the same, but the implementation itself must be done using a dynamic array. When you return this, make sure

you initialize your buffer (dynamic array) to the size INITIAL_CAPACITY in the constructor. This way the tests can make sure the memory reallocation (resize) functionality is implemented.

Part 3:

Now take the implementation base for BufferList and make it inherit Queue in the same way as LinkedList does, and implement/override its virtual functions. The implementation must accept and return values in exactly the same way, so the outside functionality is the same, but the implementation itself must be done using a dynamic array. When you return this, make sure you initialize your buffer (dynamic array) to the size INITIAL_CAPACITY in the constructor. This way the tests can make sure the memory reallocation (resize) functionality is implemented. ***The best impementation does not need to move items around in the array when something is added or removed from the front of the list.*** Instead you can keep track of the start and end of the actual contents of the array. The implementation should also not reallocate memory (resize) as soon as the end of the array is reached, if there is room at the front (because items have been remove off the front of the queue). The room at front should be used until the added items reach the start of the valid data. Look at the concept ***"circle buffer"*** for more information.

Part 4:

Now make BufferList inherit both Queue and Stack and make sure the implementation handles the functionality of both ADTs. In the same way as in LinkedList, there are only three base operations needed, to take off two end and add to one, or vice versa, to get full functionality, but realize that they are not necessarity the same operations as work best with a linked list. The implementation of Queue is likely more complex than Stack so this part should be ready when that is done, but make sure everything connects correctly and that all the tests in main() can be run without interfering with each other, before trying to return the assignment.

RÖÐ/STAFLI útfært með TENGDUM LISTA og KVIKLEGU FYLKI

Verkefnið er að útfæra hugræn gagnatög (ADT) fyrir **röð** og **stafla** og útfæra þau svo á tvenna vegu, með því að nota **eintengdan lista** og kviklegt fylki. Klasarnir Queue og Stack eiga að vera abstract klasar með engar útfærslur, á meðan klasarnir LinkedList og BufferList eiga að erfa þá báða og útfæra virknina til að geyma gögnin sem bætt er í þá.
Undir "annað efni" finnið þið útfærslu á ***"Lausn við QueueStack með Templates (verkefni frá 2017)"*** sem notar tengdan lista með templates sem er hjúpuð í klösunum Queue og Stack.

*Verkefnið verður í **fjórum hlutum** inni á Mooshak:*

- Hluti 1 þarf bara útfærslu á Queue, Stack og LinkedList. Hann gildir **50%**
- Hluti 2 þarf bara útfærslu á Stack og BufferList. Hann gildir **20%**
- Hluti 3 þarf bara útfærslu á Queue og BufferList. Hann gildir **20%**
- Hluti 4 þarf útfærslu á Queue, Stack, LinkedList og BufferList. Hann gildir **10%**

Þið fáið þrjár tilraunir til að skila hverjum hluta verkefnisins inn í Mooshak.

Mooshak prófanirnar munu ***ekki*** kalla á straumvirkjann << eða print föllin ykkar, þannig að úttakið úr þeim má vera eins og ykkur hentar best, og einungis notað fyrir ykkar eigin prófanir. Það getur til dæmis verið að þægilegasta leiðin til að skrifa út innihaldið sé í sumum tilfellum í öfugri röð á LinkedList miðað við BufferList. Það er í góðu lagi, svo fremi þið skiljið úttakið.

Hluti 1:
Breytið þessari útfærslu (eða ykkar eigin) á þann hátt að Queue og Stack séu hugræn gagnatög og að LinkedList erfi báða og útfæri sýndarföllin push, pop, add og remove. Yfirskrifuðu föllin geta annað hvort kallað í önnur grunnföll á LinkedList klasanum eða þá að þið getið útfært virknina beint í föllin. Athugið bara að tvær þessara aðgerða framkvæma nákvæmlega sömu virknina.

Allir klasarnir, Queue, Stack og LinkedList eru template klasar, þannig að þeir geta verið prófaðir með mörgum gagnatýpum. Athugið að einfaldast er að útfæra **template** klasa í heild sinni inni í .h skrám.
Queue og Stack eru báðir abstract klasar, sem þýðir að þeir eru einungis með "pure virtual" sýndarföllum.
Eina undantekningin er yfirskriftin á straumoperatornum <<, sem er friend fall og getur því ekki verið virtual. Til þess að sú virkni sé virtual kallar operator fallið á fall sem heitir print(), sendir

strauminn áfram í það. print() fallið er skilgreint pure virtual og því getur sú virkni verið útfærð inni í undirklösum eins og hvert annað virtual fall.
Klasinn LinkedList verður að erfa báða klasana, Stack og Queue (athugið að senda template <T> rétt áfram) og útfæra sýndarföllin þeirra. Queue hefur föllin add() og remove() og Stack hefur föllin push() og pop() og þeir hafa báðir fallið print(). Öll þessi föll verða að vera útfærð í LinkedList.
Virknin á bak við þau getur einfaldlega verið að kalla á föll sem eru þegar til á tengda listanum, push_front, push_back og pop_front (eða headInsert, tailInsert og headRemove).

Hluti 2:
Takið nú grunnútfærsluna fyrir BufferList og látið hana erfa Stack á sama hátt og LinkedList og útfæra/yfirskrifa sýnarföllin. Útfærslan verður að taka við og skila gildum á sama hátt, þannig að virknin út á við er sú sama, en útfærslan verður nú að vera gerð með kviklegu fylki. Þegar þessu er skilað verður að passa að fylkið sé frumstillt í stærð INITIAL_CAPACITY í smiðnum. Þannig geta prófanirnar athugað að endurúthlutum minnis fari örugglega fram.

Hluti 3:
Takið nú grunnútfærsluna á BufferListog látið hana erfa Queue á sama hátt og LinkedList og útfæra/yfirskrifa sýnarföllin. Útfærslan verður að taka við og skila gildum á sama hátt, þannig að virknin út á við er sú sama, en útfærslan verður nú að vera gerð með kviklegu fylki. Þegar þessu er skilað verður að passa að fylkið sé frumstillt í stærð INITIAL_CAPACITY í smiðnum. Þannig geta prófanirnar athugað að endurúthlutum minnis fari örugglega fram.
Besta útfærsla þarf ekki á því að halda að flytja öll stök til í fylkinu þegar hlutum er bætt inn eða þeir teknir út fremst í fylkinu. Í staðinn er hægt að halda utan um byrjun og endi á raunverulegu innihaldi fylkisins. Einnig ætti útfærslan ekki að fara í endurúthlutun minnis um leið og endirinn er kominn í enda fylkisins, ef pláss hefur losnað fremst (því stök hafa verið tekin framan af röðinni) heldur ætti að nýta plássið fremst alveg þangað til viðbættu stökin ná í skottið á fyrstu stökunum sem enn eru gild. Skoðið hugtakið ***"circle buffer"*** til glöggvunar.

Hluti 4:
Látið nú BufferList klasann erfa bæði Queue og Stack og gangið úr skugga um að útfærslan höndli virkni beggja hugrænu gagnataganna. Á sama hátt og í LinkedList ættu í raun bara að vera þrjú grunnföll sem þarf að útfæra, þ.e. taka af tveimur endum og bæta á einn, eða öfugt, til

þess að ná allri virkninni, en athugið að það eru ekki endilega sömu þrjú föllin eins og henta best á LinkedList. Útfærslan á Queue er að líkindum flóknari en Stack og því ætti þessi hluti að liggja fyrir um leið og hún er tilbúin, en gangið vel úr skugga um að allt tali saman, og að hægt sé að keyra öll test föllin í main() skránni, áður en reynt er að skila.