T-201-GSKI, GAGNASKIPAN

Vor 2017

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.

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

QUEUE/STACK

Your assignment is to implement a **Queue** class and a **Stack** class using a **Single-Linked List** data structure. Each class, Queue and Stack should use *the same implementation* of LinkedList to store the data entered into them. The only difference is the way in which Queue and Stack allow access to that data.

THE IMPLEMENTATION

Implement a single-linked list as a **template** class so that it can be used to store different data types. Remember that the simplest way to get template classes to compile correctly is to implement the entirely within the .h header files. You should in the end not need to have any .cpp files apart from main.cpp, only three .h header files for the linked list, the stack and the queue.

Then implement the data structures Queue and Stack using your linked list implementation, also using templates, forwarding the template type into the linked list.

Note that the implementations of Queue and Stack should be very small, as they simply use functionality of the linked list implementation.

Each class should overload the **'<<'** operator. The linked list should write out each element value in the list, followed by a single space, ' '. The Stack and Queue can simple call the implementation on the linked list.

The implementation file for your **linked list** should define the exception **EmptyException**. This should be thrown when attempting to remove an element from an empty list.

STACK 50%

A stack is a data structure similar to a stack of dishes. When an element is added to the stack it is added to the top of it. The same is true when a dish is added to a stack of dishes, the dish is not added to the bottom it is added to the top. When an element is removed from the stack it is removed from the top. The dish example still applies to the removal of an element. When you remove a dish from a stack of dishes you, in general, don't remove a dish from the middle of the stack, you remove a dish from the top of the stack.
Elements in the stack adhere to the **LIFO** or last in first out principle, where the last element to be added to the stack is the first to get removed from it.
The Stack data structure should implement the **push()** and **pop()** methods to add and remove elements.
The Stack class must be implemented in the file "**stack.h**" and be named **Stack**.
The main file initializes it as Stack<int>, Stack<double> and Stack<string>.

QUEUE 50%

A queue is a data structure that can be thought of as a line in a shop. When a new element is added to the queue it is added to the end, that is like when a person enters the line at the shop, they go to the end.
When an element is removed from the queue it is taken from the front just like when the person at the front of the line is serviced and leaves the shop. The element that is behind the formerly first element then becomes the first element.
Elements in the queue adhere to the **FIFO** or first in first out principle, where the first element to be added to the queue is the first element to be removed from it.
The Queue data structure should implement the **add()** and **remove()** methods to add and remove elements.
The Queue class must be implemented in the file "**queue.h**" and be named **Queue**.
The main file initializes it as Queue<int>, Queue<double> and Queue<string>.

RÖÐ/STAFLI

Verkefnið er að útfæra **röð** og **stafla** sem bæði nota útfærslu á **eintengdum lista**. Hvor klasi fyrir sig, röð (Queue) og stafli (Stack) á að *nota sömu útfærsluna* á eintengdum lista (LinkedList) til að geyma gögnin sem bætt er í þá. Eini munurinn felst í því hvernig röðin og staflinn leyfa aðgengi að þeim gögnum.

ÚTFÆRSLA

Byrjið á því að útfæra eintengdan lista sem notar **sniðmát** til þess að geta geymt ýmsar gagnatýpur.
Athugið að einfaldast er að útfæra **template** klasa í heild sinni inni í .h skrám. Það ættu því á endanum ekki að þurfa að vera neinar .cpp skrár nema main.cpp, einungis þrjár .h skrár, fyrir tengda listann, staflann og röðina.
Útfærið röð og stafla sem hvort um sig notar sömu listaútfærsluna. Þær gagnagrindur nota einnig sniðmát og framlengja gagnatagið til listaútfærslunnar.
Athugið að útfærslurnar á Queue og Stack ættu að vera mjög umfangslitlar þar sem þær nota einfaldlega virkni tengda listans.
Hver klasi þarf að yfirskrifa **'<<'** virkjann. Tengdi listinn þarf að skrifa út gildi hvers staks í listanum, með einföldu bili (' ') á milli og á eftir. Staflinn og Röðin geta einfaldlega kallað á útfærslu tengda listans.
Skráin sem þið útfærið **tengda listann** í þarf að skilgreina villuna **EmptyException**. Henni á að vera kastað ef reynt er að fjarlægja stak úr tómum lista.

STAFLI 50%

Stafli er gagnagrind sem má líkja við stafla af diskum. Þegar staki er bætt á staflann er því bætt efst á hann. Það er líka gert þegar diskum er bætt á stafla, diskunum er bætt efst á staflann ekki í hann miðjan eða neðst. Þegar stak er tekið af staflanum þá er er það tekið efst af honum, þetta er líkt því þegar þú tekur disk af stafla af diskum, þú tekur efsta diskinn, ekki disk sem er í miðlum staflanum.
Stökum sem bætt er við staflann fara eftir **LIFO** eða last in first out, það þýðir að síðasta stakið sem bætt var við staflann er það fyrsta sem fer af honum.
Staflagagnagrindin þarf að útfæra aðgerðirnar **push()** og **pop()** til að bæta við og fjarlægja stök.

Staflaklasinn verður að vera útfærður í skránni “**stack.h**” og bera heitið **Stack**.
Aðalkeyrsluskráin frumstillir staflann sem Stack<int>, Stack<double> og Stack<string>.

RÖÐ 50%

Röð er gagnagrind sem má líkja við biðröð í verslun. Þegar nýju staki er bætt við röðina þá er því bætt við aftast, rétt eins og þegar manneskja fer í biðröð þá fer hún aftast. Þegar stak er tekið af röðinni er fremsta stakið tekið, það sama gerist í biðröð, sá sem hefur beðið lengst fær afgreiðslu og fer úr búðinni. Stakið sem var fyrir aftan fremsta stakið verður því nýja fremsta stakið.
Stökum sem bætt er við röðina fara eftir **FIFO** eða first in first out, það þýðir að það stak sem hefur verið lengst í röðinni er það stak sem tekið er af fyrst.
Biðraðargagnagrindin þarf að útfæra aðgerðirnar **add()** og **remove()** til að bæta við og fjarlægja stök.
Biðraðarklasinn verður að vera útfærður í skránni “**queue.h**” og bera heitið **Queue**.
Aðalkeyrsluskráin frumstillir röðina sem Queue<int>, Queue<double> og Queue<string>.