Create a pair of classes that store a series of values. Your classes should be named miniqueue and ministack. The data type of the values will be flexible, using a template. The classes should be able to store up to 10 values in an array. Your classes will allow the user to add and remove values from the list, depending on one of the two methods shown below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Queue: | Input → | Value | Value | Value | Value | → Output |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Stack: | Input →  Output ← | Value | Value | Value | Value |  |

A Queue uses a First In First Out (FIFO) method for storing data. Values are stored in the order they are entered, like a pipeline. The first value into the list is the first to be removed.

A Stack uses a Last In First Out (LIFO) method for storing data. Values are stored in the order they are entered, like a stack of boxes. The first value is on the bottom of the stack, and is the last to be removed.

Test your program with the code files below:

<https://markbowman.org/231/Program04.zip>

Each class should have the following data elements:

* Array of values
* Number of values being used

The miniqueue class should have the following functions:

* Constructor
* Push - add a value to the end of the queue
* Pop - remove a value from the front
* Front - return the value at the front
* Size - return count of values

The ministack class should have the following functions:

* Constructor
* Push - add a value to the top of the stack
* Pop - remove a value from the top
* Top - return the value at the top
* Size - return count of values

Along with your class, use the sample main() program to verify that the class functions correctly. Do not change the Main.cpp file. Include testing for underflow and overflow with appropriate error messages. Demonstrate your template works with each of the following data types:

* Integer
* String

MiniStack.h:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MiniStack.h

\* MiniStack Header File

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef MINISTACK\_H

#define MINISTACK\_H 10 //maximum values that can be in the array

// ministack template

template <class T>

class ministack

{

private:

T\* stackArray; // Pointer to the stack array

int numOfItems; // number of items in the stack array

public:

ministack(); //Constructor function

// ministack operations

void push(T); //push function

void pop(); //pop function

T top(); //top function

int size(); //size function

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Constructor

template <class T>

ministack<T>::ministack()

{

stackArray = new T[MINISTACK\_H];

numOfItems = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Push Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// pushes the argument onto the stack.

template <class T>

void ministack<T>::push(T item)

{

// check for overflow

if (numOfItems >= MINISTACK\_H)

{

cout << "Stack overflow(no pun intended)!\n";

}

else

{

// put the value at the end of the array

stackArray[numOfItems] = item;

numOfItems++;

return;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Pop Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// removes the value at the top of the stack

template <class T>

void ministack<T>::pop()

{

// check for underflow

if (numOfItems == 0)

{

cout << "Stack underflow!\n";

}

else

{

// remove the last element

numOfItems--;

return;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Top Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// returns the value at the top of the stack

template <class T>

T ministack<T>::top()

{

// check for underflow

if (numOfItems == 0)

{

cout << "Stack underflow!\n";

}

else

{

// return the last element of the stack

return stackArray[numOfItems - 1];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Size Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// returns the size of the stack

template <class T>

int ministack<T>::size()

{

return numOfItems;

}

#endif

MiniQueue.h:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MiniQueue.h

\* MiniQueue Header File

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef MINIQUEUE\_H

#define MINIQUEUE\_H 10

// MiniQueue template

template <class T>

class miniqueue

{

private:

T\* queueArray; // Pointer to the queue array

int numOfItems; // Number of items in the queue

public:

// Constructor

miniqueue(); //Constructor function

// miniqueue operations

void push(T item); //push function

void pop(); //pop function

T front(); //front function

int size(); //size function

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// This constructor creates an empty queue

template <class T>

miniqueue<T>::miniqueue()

{

queueArray = new T[MINIQUEUE\_H];

numOfItems = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Push Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// inserts a value at the end of the queue.

template <class T>

void miniqueue<T>::push(T item)

{

// check for overflow

if (numOfItems >= MINIQUEUE\_H)

{

cout << "Queue overflow!\n";

}

else

{

// Insert new item

queueArray[numOfItems] = item;

// Update item count

numOfItems++;

return;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Pop Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// removes the value at the start of the queue

template <class T>

void miniqueue<T>::pop()

{

int i = 0;

// check for underflow

if (numOfItems == 0)

cout << "Queue underflow!\n";

else

{

// shift the array by 1 position to left to remove the first element

for (i; i < (numOfItems - 1); i++)

{

queueArray[i] = queueArray[i + 1];

}

// Update item count

numOfItems--;

return;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Front Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// returns the front element of the queue

template <class T>

T miniqueue<T>::front()

{

// check for underflow

if (numOfItems == 0)

{

cout << "Queue underflow!\n";

}

else

{

return queueArray[0];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Size Function

\* Created by Safford, Twymun

\* Last Updated: 25-Oct-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// returns the current size of the queue

template <class T>

int miniqueue<T>::size()

{

return numOfItems;

}

#endif