Project- Deques, Stacks & Queues

General Description:

Design and develop array based and linked list-based implementations for the Dequeue ADT. Your implementation must support generic data types using C++ templates. Develop Adapter Files to provide Stack and Queue functionality for the Deques.

Definitions:

You should implement the ADTs precisely as described in the following partial header files.

*Deque.h*

*template <typename E>*

*class Deque {*

*public:*

*Deque(); //constructor*

*~Deque(); //destructor*

*void insertFront(const E& e);*

*void insertBack(const E& e);*

*const E& eraseFront(); // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& eraseBack(); // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& front() const; // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& back() const; // throw(DequeEmpty); //note that the use of exceptions is optional*

*int size () const;*

*bool empty() const;*

*}*

You will also need header files for the Stack and Queue ADT’s for each implementation:

*Stack.h*

*template <typename E>*

*class Stack {*

*public:*

*Stack(); //constructor*

*~Stack(); //destructor*

*void push(const E& e);*

*const E& pop(); // throw(StackEmpty); //note that the use of exceptions is optional*

*const E& top() const; // throw(StackEmpty); //note that the use of exceptions is optional*

*int size () const;*

*bool empty() const;*

*}*

*Queue.h*

*template <typename E>*

*class Queue {*

*public:*

*Queue(); //constructor*

*~Queue(); //destructor*

*void enqueue(const E& e);*

*const E& dequeue(); //throw(QueueEmpty); //note that the use of exceptions is optional*

*const E& front() const; //throw(QueueEmpty); //note that the use of exceptions is optional*

*int size () const;*

*bool empty() const;*

*}*

You will need to write two versions of the Deque.cpp file (one for the array-based implementation, one for the linked list based implementation). Both Stack and Queue must be implemented by use of an Adapter applied to the Deque ADT. As a result, you will need 2 additional C++ files, Stack and Queue. Your Stack and Queue cpp files will be much much smaller though. See pages 220 – 222 of your text for a discussion of this topic along with partially provided code.

Requirements:

Develop a C++ software solution that includes an array based and linked-list based implementation for the Deque ADT. Develop C++ Adapter Classes for both the Stack and Queue ADT’s.

You should also provide formal algorithms for each of the methods of your Deque data structures (2 versions, one each for your array and linked list implementations), an efficiency analysis for each of your algorithms, and all of the necessary .cpp and .h files.  Be sure to include expand files so that a new data type for the template can easily be added.

Deliverable:

You should provide two separate zip files, one for the array-based implementation and one for the linked-list based implementation. Each zip file should contain 3 header files (Deque.h, Stack.h, Queue.h), 6 cpp files (Deque.cpp, Stack.cpp, Queue.cpp, DequeExpand.cpp, StackExpand.cpp, and QueueExpand.cpp). In addition, you should include a document that contains formal algorithms for each function you have implemented along with a Big O efficiency analysis of the algorithms. Finally, you should include in your document an overall statement regarding the efficiency of each type of implementation (StackArray, StackList, QueueArray and QueueList).

It is always a good idea to include a readme file as well that includes any instructions for compiling and using your software. Although it is not strictly required, It is also advisable to develop a test file that instantiates and tests your stack and queue implementations.

Note on Exceptions:

Implementing your Deque, Stack and Queue to use Exceptions for indicated operations is optional. If you do it, you will receive extra credit.

Grading Criteria:

* Your code must conform to the standardized header files presented in this document.
* Your code must work! Obviously it has to compile, but it must also work with test code that will create both versions of the stack and queue and test the public functions.
* Your code will be evaluated for STYLE (as in whitespace, alignment of code blocks, comments that match the code, and appropriate names of variables.)
* Your efficiency analysis must be performed for every function for each implementation.
* Your efficiency analysis must include a summary of the overall efficiency of all implementations.