Project 02 - Stack and Queue

Sunday, June 11 at 11:59pm

Introduction

You are required to implement two data structures: a dynamic stack and a dynamic queue. You are also required to create UML diagrams for each of the classes that you implement. Finally, you have to provide the means to test your system by developing a menu program that allows the user to manipulate your structures.

Deliverables

- A 1 page pdf report with your UML diagrams and a brief explanation. Please include what each teammate did and approximate hours spent on the project.
- An implementation of a dynamic stack.
- An implementation of a dynamic queue.
- A menu program to test the implemented data structures.

1 Stack

In this part of the project, you need to implement one class, and create its respective UML diagram. Your functions must meet the required running times or no points will be granted.

1.1 Description

A stack stores elements in an ordered list and allows insertions and deletions at one end of the list in O(1) time.

The elements in this stack are stored in an array. The size of the array may be changed depending on the number of elements currently stored in the array, according to the following two rules:

If an element is being inserted into a stack where the array is already full, the size of the array is doubled. If, after removing an element from a stack

where the number of elements is 1/4 the size of the array, then the size of the array is halved. The size of the array may not be reduced below the initially specified size.

1.2 Data Members

1. A pointer to an instance of type, Type *array, to be used as an array.
2. A counter, int count.
3. The initial size of the array, int initialSize.
4. The current size of the array, int arraySize.

1.3 Member Functions

Constructors

DynStack(int n = 15) The constructor takes as an argument the initial size of the array and allocates memory for that array. The default number of entries is 15. If the argument is either 0 or a negative integer, set the initial capacity of the array to 1. Other class members are assigned as appropriate.

Destructor

~DynStack() The destructor deletes the memory allocated for the array.

Accessors

Type top() const Returns the object at the top of the stack. It may throw a underflow exception. (O(1))

int size() const Returns the number of elements currently stored in the stack. (O(1))

bool empty() const Returns true if the stack is empty, false otherwise. (O(1))

int capacity() const Returns the current size of the array. (O(1))

void display() Prints the content of the stack. (O(n))

Mutators

void push(Type const & data) Inserts the new element at the top of the stack. If the array is full, the size of the array is doubled. (O(1) on average)

Type pop() Removes the element at the top of the stack. If, after the element is removed, the array is 1/4 full and the array size is greater than the initial size, the size of the array is halved. This may throw a underflow exception. (O(1) on average)

void clear() Removes all the elements in the stack. The array is resized to the initial size. (O(1))

Friends

This class has no friends.

2 Queue

In this part of the project, you need to implement one class, and create its respective UML diagram. Your functions must meet the required running times or no points will be granted.

2.1 Description

A queue stores objects in an ordered list and allows insertions at one end and deletions from the other end of the list in O(1) time.

The objects in this queue are stored in an array. The capacity of the array may be changed depending on the number of objects currently stored in the array, according to the following two rules:

If an object is being inserted into a queue where the array is already full, the capacity of the array is doubled. If, after removing an object from a queue where the number of objects is one-quarter (1/4) the capacity of the array, then the capacity of the array is halved. The capacity of the array may not be reduced below the initially specified capacity.

2.2 Data Members

1. A pointer to an instance of type, Type *array, to be used as an array.
2. A head index, int iHead.
3. A tail index, int iTail.
4. A counter, int count.
5. The initial size of the array, int initialSize.
6. The current size of the array, int arraySize.

2.3 Member Functions

Constructors

DynQueue(int n = 15) The constructor takes as an argument the initial capacity of the array and allocates memory for that array. If the argument is either 0 or a negative integer, set the initial capacity of the array to 1. The default initial capacity of the array is 15. Other member variables are assigned as appropriate.

Destructor

~DynQueue() The destructor deletes the memory allocated for the array.

Accessors

Type front() const Returns the object at the front of the queue. It may throw a underflow exception. (O(1))

Type back() const Returns the object at the back of the queue. It may throw a underflow exception. (O(1))

int size() const Returns the number of elements currently stored in the queue. (O(1))

bool empty() const Returns true if the queue is empty, false otherwise. (O(1))

int capacity() const Returns the current size of the array. (O(1))

void display() Prints the content of the Queue. (O(n))

Mutators

void enqueue(Type const & data) Insert the new element at the back of the queue. If the array is full, the size of the array is first doubled. (O(1) on average)

Type dequeue() Removes the element at the front of the queue. If, after the element is removed, the array is 1/4 full and the array size is greater than the initial size, the size of the array is halved. This may throw a underflow exception. (O(1) on average)

void clear() Removes all the elements in the queue. The array is resized to the initial size. (O(1))

Friends

This class has no friends.

3 The Menu Program

In order to test your program, you are required to implement a menu program that provides the means to run each of the functions in your classes. Please choose the *string* data type to create your Queue and Stack. So, when asked to create a Queue or a Stack, its array cells should hold string elements. The TA will choose one group to demo the project.

Format for the Menu Program

First, take a character, 's' or 'q' (lowercase) that specifies whether we will be working with a stack or queue. Next, give the options (please have them in this **EXACT** order for grading purposes):

1. Return Capacity (items actually in the data structure)
2. Return Size of the data structure
3. View first item ("top" for Stack, "front" for Queue)
4. Insert item ("push" for Stack, "enqueue" for Queue)
5. Delete item ("pop" for Stack, "dequeue" for Queue)
6. Display
7. Clear

Submit the following files to Canvas (named **EXACTLY** as shown below **without zipping** for grading purposes: (you may also include implementation files with the header files if you would like)

1. menu.cpp
2. stack.h
3. queue.h
4. makefile
5. project2.pdf

The rubric is as follows:

1. A program that does not compile will result in a zero
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Report 10%
5. Functionality 70% (functions were declared and implemented as required)

4 Acknowledgment

This project was created based on the work shared by the University of Waterloo.