Project 02 - Stack, Queue, and Hash Table

June 5, 2016

Introduction

You are required to implement three data structures: a dynamic stack, a dynamic queue, and a dynamic hash table. 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 report that explains the design of your data structures.
- An implementation of a dynamic stack.
- An implementation of a dynamic queue.
- An implementation of a dynamic hash table.
- 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 = 13) (1 point) The constructor takes as an argument the initial size of the array and allocates memory for that array. The default number of entries is 13. 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() (2 points) The destructor deletes the memory allocated for the array.

Accessors

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

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

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

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

int display() const (1 point) Prints the content of the stack. (O(n))

Mutators

void push(Type const & data) (3 points) 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() (3 points) 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() (3 points) Removes all the elements in the stack. The array is resized to the initial size. (O(1))

int erase(Type const & data) (5 points) Removes the elements (from the top) in the structure that contains the element equal to the argument (use == to to test for equality with the retrieved element). If, after the element(s) is/are removed, the array is 1/4 full and the array size is greater than the initial size, the size of the array is halved. Return the number of elements that were deleted. This may throw a underflow exception.

You must pop every element in the stack to compare with the argument. You should use a temporary stack to hold the elements that are not equal to the argument. After the main stack is empty, return the elements that are in the temporary stack back to the main one. Calculate running time and include it in your report.

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 = 13) (1 point) 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 13. Other member variables are assigned as appropriate.

Destructor

~DynQueue() (2 points) The destructor deletes the memory allocated for the array.

Accessors

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

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

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

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

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

int display() const (1 point) Prints the content of the Queue. (O(n))

Mutators

void enqueue(Type const & data) (3 points) 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() (3 points) 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() (3 points) Removes all the elements in the queue. The array is resized to the initial size. (O(1))

int erase(Type const & data) (5 points) Remove the elements (from the front) in the structure that contains the element equal to the argument (use == to to test for equality with the retrieved element). If, after the element(s) is/are removed, the array is 1/4 full and the array size is greater than the initial size, the size of the array is halved. Return the number of elements that were deleted. This may throw a underflow exception.

You must dequeue every element in the queue to compare with the argument. You should use a temporary queue to hold the elements that are not equal to the argument. After the main queue is empty, return the elements that are in the temporary queue back to the main one. Calculate running time and include it in your report.

Friends

This class has no friends.

3 Hash Table

Design a hash table that provides the following basic operations: search (2 points), insert(5 points), and delete (3 points). The insert operation must provide dynamic resizing based on a load factor criterion (a threshold). The load factor is the ratio of the number of entries in the table and the number of buckets. The table must also provide collision control through chaining.

3.1 Hash Table Design

You are required to calculate the running time for all of your member functions, justify your hashing function (explain why you're choosing it.), and create UML diagrams for your class(es). The keys of the table can be: numbers (integers or reals), characters, or strings. The values the table stores can be of any type. You must use templates in your implementation. Finally, You need to document your design in the project report.

Your class(es) may include the following parts.

1. Constructors.
2. Destructor.
3. Data members.
4. Accessors, Mutators.
5. Friends.

3.2 Deliverables

1. Class(es) with UML diagrams.
2. Hash function description. Explain why you chose it.
3. Running-time function analysis. Provide a brief description of the analysis in the report.

4 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. Choose the *string* data type to create your Hash Table. So, when asked to create a Hash Table, both the keys and the table values must be strings. The TA will choose one member of your group to defend the demo, and the same grade will be assigned to all of the members of the group.

5 Rubric

This is how your project is going to be graded.

1. Report 40%.
2. Demo 60%. 20% for each structure. Stack (max score 22 points), Queue (max score 23 points), Hash Table (max score 10 points).

6 The Project Report

You must include everything you consider relevant in your design. For the Stack and Queue design, you should include UML diagrams and provide details of your erase function. The majority of the report will be on your hash table (UML diagrams, class(es), hash function, function running times.)

7 FAQ

These are some of the questions you may have.

Do I have to implement input validation?

No, you do not. Whatever your program requests will be entered.

Is each member supposed to submit the project?

No, just one submission per group. Specify members in the project report and in the submission box.

Can you give me an example of the menu program?

1. Stack.
2. Queue.
3. Hash Table.
0. Exit Select Option−− > 1 or 2 or 3 or 0.

Submenu 3.

0. Go back to previous menu.
1. Create table.
2. Insert.
3. Delete.

4. Search.
... Select option−− > 1 or 0 or 2 or 3 or 4.

Do I need to create multiple instances of my structures?
No, you do not have to. You are only required to instantiate a stack, a queue, and a table. If you create more is ok but that will not give you extra points for it (just personal satisfaction).

8 Acknowledgment

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