Brandon Perry

Coding Assignment #2

CS 221

Dr. Troung Tran

Documentation

**QueType**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Requirements**

QueType is required to create a queue with a max size of 5. It will create function to add elements into the queue as well as remove elements. It is intended to also test whether the queue is full or empty with Boolean functions. QueType must also create a function that will make the queue into an empty state.

***CRC Card***

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name: QueType | Superclass: | | Subclasses: |
| Primary Responsibility: Create a Queue and Perform Alterations on it | | | |
| Responsibilities | | Collaborations | |
| Create a queue | |  | |
| Initialize queue to empty state | |  | |
| Return true if queue is full | |  | |
| Return true if queue is empty | |  | |
| Puts newExpense into rear of queue | | ExpenseType | |
| Removes newExpense from front on queue | | ExpenseType | |
| Deletes queue | |  | |

***ADT Specs***

Structure: Elements are added to the rear and removed from the front of the queue

Definition: provided by user

maxQue: maximum number of items that might be on the queue

Operations:

QueType QueType (int max)

Function: parameterized class constructor

Pre: none

Post: maxQue, front, rear have been initialized the array to hold the queue elements has been dynamically allocated

QueType QueType ()

Function: default class constructor

Pre: none

Post: maxQue, front, rear have been initialized the array to hold the queue elements has been dynamically allocated

QueType ~QueType ()

Function: class deconstructor

Pre: Queue is initialized

Post: Queue is deleted

Void MakeEmpty

Function: Initializes the queue to an empty state

Precondition: None

Postcondition: Queue is empty

Bool IsEmpty

Function: Determines whether the queue is empty

Precondition: Queue has been initialized

Postcondition: Function value = (queue is empty)

Bool IsFull

Function: Determines whether the queue is full

Precondition: Queue has been initialized

Postcondition: Function value = (queue is full)

Void Enqueue (ExpenseType newExpense)

Function: Adds newExpense to the rear of the queue

Precondition: Queue has been initialized

Postcondition: If (queue is full), FullQue exception is thrown, else newExpense is at rear of queue

Void Dequeue (ExpenseType& newExpense)

Function: Removes front item form the queue and returns it in item

Precondition: Queue has been initialized

Postcondition: If (queue is empty), EmptyQueue exception is thrown and Expense is undefined, else front element has been removed from queue and expense is a copy of the removed element

***UML DIAGRAM***

|  |
| --- |
| QueType |
| -front: int  -rear: int  -expense: ExpenseType\*  -maxQue: int |
| +QueType (max: int)  +QueType ()  +~QueType ()  +MakeEmpty (): void  +IsEmpty () const: bool  +IsFull () const: bool  +Enqueue (newExpense: ExpenseType): void  + Dequeue (newExpense: ExpenseType&): void |

**CountedQueType**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Requirements**

CountedQueType must do all of the same responsibilities as QueType by using its derived functions. CountedQueType must keep track of the length of the queue when adding an element or removing one. It must also keep track of the length when making the list empty. Another function that must be implemented in gaining access to the elements of the queue and adding up all of the items to produce a sum.

***CRC Card***

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name: CountedQueType | Superclass: QueType | | Subclasses: |
| Primary Responsibility: Keep count of length of queue | | | |
| Responsibilities | | Collaborations | |
| When Enqueue is called, increase length by one | |  | |
| When Dequeue is called, decrease length by one | |  | |
| Return total length of queue as an int | |  | |
| Return total of all expense as a double | |  | |
| When MakeEmpty is called, decrease length by one | |  | |

***ADT Specs***

Structure: Elements are added to the rear and removed from the front of the queue

Definition: provided by user

Operations:

QueType Enqueue (ExpenseType newExpense)

Function: implements Enqueue from QueType and increases the length

Pre: Queue has been initialized

Post: QueType enqueue is implemented and length is increased by one

void Dequeue (ExpenseType& newExpense)

Function: implements Dequeue from QueType and reduces the length

Pre: Queue has been initialized queue is not empty

Post: Quetype Dequeue is implemented and length is reduced by one

int GetLength () const

Function: gets the length of the queue

Pre: Queue has been initialized

Post: returns length

CountedQueType (int max)

Function: paramterized class constructor

Pre: None

Post: initializes length

CountedQueType()

Function: default class constructor

Pre: none

Post: initializes length

double GetTotalExpense()

Function: adds up all of the expenses within the queue

Pre: queue is initialized

Post: returns total of all amounts added

void MakeEmpty()

Function: Initializes the queue to an empty state.

Precondition: None

Postcondition: Queue is empty.

***UML DIAGRAM***

|  |
| --- |
| CountedQueType |
| -length: int |
| +CountedQueType (max: int)  +CountedQueType  +Enqueue (newExpense: ExpenseType): QueType  +Dequeue (newExpense: &ExpenseType): void  +GetLength () const: int  +GetTotalExpense (): double  +MakeEmpty(): void |

**Test Plan**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Intentions:**

This test plan is made in order to test the derived CountedQueType class from the QueType class. It will test MakeEmpty, Enqueue, Dequeue, IsFull, IsEmpty, GetLength, and GetTotalExpense. IsFull and IsEmpty return Boolean values to test the condition of the queue. These functions are called through CountedQueType but are defined in the QueType class. A maximum of five elements are allowed into the queue.

**Input File:**

|  |
| --- |
| Input File: Queueinput |
| IsEmpty  GetLength  Enqueue food 300.50  Enqueue gas 60..35  Enqueue book 89.99  Enqueue computer 699.00  GetLength  GetTotalExpense  Dequeue  Dequeue  GetLength  GetTotalExpense  Enqueue rent 850.0  Enqueue internet 29.99  Enqueue utility 178.30  Enqueue game 50.00  IsFull  GetLength  GetTotalExpense  MakeEmpty  IsEmpty  GetLength  Quit |

**Output File:**

|  |
| --- |
| OutPut File: QueueOutput |
| Queue is empty.  Length of queue is: 0  food 300.5 was added to queue  gas 60 was added to queue  book 89.99 was added to queue  computer 699 was added to queue  Length of queue is: 4  The total expenses are: 1149.49  food 300.5 was removed from queue  gas 60 was removed from queue  Length of queue is: 2  The total expenses are: 788.99  rent 850 was added to queue  internet 29.99 was added to queue  utility 178.3 was added to queue  game 50 was added to queue  Queue is full.  Length of queue is: 5  The total expenses are: 1847.28  Queue is empty.  Length of queue is: 0 |

**Results:**

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

**Explanation:**

The program ran successfully. I am not positive on whether or not the FullQueue exception executed properly as seen in the photo above. I had to press “Continue” in Visual Studio in order to continue the program which executed the rest of the code fully.

I did have some issues throughout deleing with accessing the queue in the GetTotalExpense function as well as having a memory exception with the ~QueType deconstructor. I believe these had to do with accessing newExpense. I added a parameter for GetTotalExpense in order to access the queue elements properly.