PA09-Heap

Generated by Doxygen 1.8.11

Contents

1 Main Page

This project contains the following items -a Heap ADT implemented by using an array representation of a tree -a Heap which can have dataitems inserted and deleted from it with the proper function to rearrange the heap. -a writelevels function that outputs the data in a heap in level order, one level per line -a inheritance class called Priority queue to develop a simulation of an operating system's task schedule using a priority queue

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

5 Class Documentation

5.1 Greater< KeyType > Class Template Reference

Public Member Functions

- bool **operator()** (const KeyType &a, const KeyType &b) const

The documentation for this class was generated from the following file:

- test11.cpp

5.2 Heap< DataType, KeyType, Comparator > Class Template Reference

Public Member Functions

- Heap (int maxNumber=DEFAULT_MAX_HEAP_SIZE)
- Heap (const Heap &other)
- Heap & operator= (const Heap &other)
- ~Heap ()
- void insert (const DataType &newDataItem) throw (logic_error)
- DataType remove () throw (logic_error)
- void clear ()
- bool isEmpty () const
- bool isFull () const
- void showStructure () const
- void writeLevels () const

Static Public Attributes

- static const int **DEFAULT_MAX_HEAP_SIZE** = 10

Private Member Functions

- void showSubtree (int index, int level) const
- int Parent (int child)
- int LeftChild (int parent)
- int RightChild (int parent)
- void swap (int index1, int index2)

Private Attributes

- int **maxSize**
- int **size**
- DataType * **dataItems**
- Comparator **comparator**

5.2.1 Constructor & Destructor Documentation

5.2.1.1 template<typename DataType , typename KeyType , typename Comparator > Heap< DataType, KeyType, Comparator >::Heap (int *maxNumber* = DEFAULT_MAX_HEAP_SIZE)

This function is the param constructor for the Heap class.

This function will set the maxSize equal to the parameter maxNumber passed in. This function will set the size to 0 since we are not putting in any data yet. This function will dynamically allocate memory for the array of DataType with maxSize.

Parameters

int	maxNumber, which is the largest possible size for the heap.

Returns

This function does not return anything.

Precondition

none

Postcondition

The heap is empty and was just created.

5.2.1.2 template<typename DataType , typename KeyType , typename Comparator > Heap< DataType, KeyType, Comparator >::Heap (const Heap< DataType, KeyType, Comparator > & *other*)

This function is the copy constructor for the Heap class.

This function will set the maxSize equal to the other heap's maxSize. This function will set the size equal to the other heap's size. This function will dynamically allocate memory for the array of DataType with maxSize. This function will iterate a for loop for the maxSize until it copies the data of the other heap's dataItems array into this heap's dataItems array.

Parameters

const	Heap& other which is the other object to be copied to make this object

Returns

This function does not return anything.

Precondition

none

Postcondition

This heap is an exact copy of the other object that was passed in.

5.2.1.3 template<typename DataType , typename KeyType , typename Comparator > Heap< DataType, KeyType, Comparator >::~Heap ( )

This function is the destructor for the Heap class.

This function will call the clear function to clear the dataItems array. This function will call the delete operator on dataItems. This function will set dataItems to NULL.

Parameters

none	

Returns

none

Precondition

none

Postcondition

dataItems will be cleared and deleted and set to NULL.

5.2.2 Member Function Documentation

5.2.2.1 template<typename DataType , typename KeyType , typename Comparator > void Heap< DataType, KeyType, Comparator >::clear ( )

This function is the clear function for the Heap class.

This function calls the remove function while the isEmpty function is called. The function keeps calling the remove functio until the heap is empty.

Parameters

none	

Returns

This function does not return anything

Precondition

none

Postcondition

This function will clear the dataItems array in the heap class and make sure that the size is 0.

5.2.2.2 template<typename DataType, typename KeyType , typename Comparator > void Heap< DataType, KeyType, Comparator >::insert (const DataType & *newDataItem*) throw logic_error)

This function is the insert function for the Heap class.

The function first checks to see if the heap is full, if it is, the function throws a logic error. Otherwise the function creates an index called child that is equal to the size of the array. The Function then inserts the dataItem passed into the child index of the array. The Function then incremenets the size. The Function runs in a while loop where child is greater than 0 and by calling the comparator function which checks to see if the value of the newDataItem is smaller than the parent of the newly inserted dataItem. If that statement is true, the while loop runs and calls swap with the Parent of the child and child as the parameters. The function then sets child equal to the Parent of the current index since they were swapped. The while loop runs until one of the statements is false, which will finish rearranging the heap.

Parameters

const	DataType &newDataItem which is the new item to be inserted into the heap

Returns

This function does not return anything

Precondition

A heap so that items may be inserted into.

Postcondition

This function will insert the newDataItem and rearrange the heap accordingly.

5.2.2.3 template<typename DataType , typename KeyType , typename Comparator > bool Heap< DataType, KeyType, Comparator >::isEmpty ( ) const

This function is the isEmpty function for the Heap class.

This function checks to see if size is equal to 0. If it is, the function returns true. Otherwise the function returns false.

Parameters

none	

Returns

This function returns if the heap is empty or not.

Precondition

none

Postcondition

This function will return if the heap is empty or not.

5.2.2.4 template<typename DataType , typename KeyType , typename Comparator > bool Heap< DataType, KeyType, Comparator >::isFull ( ) const

This function is the isFull function for the Heap class.

This function checks to see if size is equal to the maxSize. If it is, the function returns true. Otherwise the function returns false.

Parameters

none	

Returns

This function returns if the heap is full or not.

Precondition

none

Postcondition

This function will return if the heap is fullor not.

5.2.2.5 template<typename DataType , typename KeyType , typename Comparator > int Heap< DataType, KeyType, Comparator >::LeftChild (int *parent*) [private]

This function is the LeftChild function for the Heap class.

This function will take the parent and multiply the parent by 2 and add 1. This algorithm can be found in the C++ datastructure course textbook.

Parameters

int	parent which is used to find the left child of the parent

Returns

This function will return the index of the left child of the parent

Precondition

none

Postcondition

This function will return the left child index from the parent index that was passed in.

5.2.2.6 template<typename DataType , typename KeyType , typename Comparator > Heap< DataType, KeyType, Comparator > & Heap< DataType, KeyType, Comparator >::operator= (const Heap< DataType, KeyType, Comparator > & *other*)

This function is the overloaded assignment operator for the Heap class.

This function will first check to see that the other object is not the same as this object. This function will then clear the current object. This function will set the maxSize equal to the other heap's maxSize. This function will set the size equal to the other heap's size. This function will dynamically allocate memory for the array of DataType with maxSize. This function will iterate a for loop for the maxSize until it copies the data of the other heap's dataItems array into this heap's dataItems array.

Parameters

const	Heap& other which is the other object to be copied to make this object

Returns

This function returns a pointer to this object.

Precondition

none

Postcondition

This heap is an exact copy of the other object that was passed in.

5.2.2.7 template<typename DataType , typename KeyType , typename Comparator > int Heap< DataType, KeyType, Comparator >::Parent (int *child*) [private]

This function is the parent function for the Heap class.

This function will take the child and subtract one from it and then divide 2 from that value. This algorithm can be found in the C++ datastructure course textbook.

Parameters

int	child which is the child index that we use to find the parent of the child.

Returns

This function will return the parent of the child index that was passed in.

Precondition

none

Postcondition

This function will return the parent index from the child index that was passed in.

5.2.2.8 template<typename DataType , typename KeyType , typename Comparator > DataType Heap< DataType, KeyType, Comparator >::remove ( ) throw logic_error)

This function is the remove function for the Heap class.

The function first checks to see if the heap is empty by calling the isEmpty function, if it iself, the function throws a logic error. Otherwise the function creates a temp DataType that is of index 0 of the dataItems array, which is the DataType to be returned. The function then sets the index 0 of the array to be the last index in the array. The function then sets the parent to be 0, since thats the index since we will rearrange the array. The function will use a while loop to run until the parent is smaller than the size, so that way the function runs until it traverses the array or a return is performed. The function will then check to see if the RightChild of the parent is smaller than the size. We check the right child first because if a right child exists, then we know for a fact that a left child must exist so we can compare the values of the left and right child. The function then does a comparison to check to see if the left child is smaller than the right child, if it is, the function swaps the parent with the right child, and then sets the parent equal to the right child of the parent since they were swapped. Otherwise the function does a comparison to check to see if the right child is smaller than the left child, if it is, the function swaps the parent with the left child, and then sets the parent equal to the left child of the parent since they were swapped. Other wise the function does a comparison to check to see if the right child is equal to the left child, if it is, the function swaps it with the left child since it wont matter, and then sets the parent equal to the left child of the parent since they were swapped. If both of those are false, the function just returns temp since The function then check to see if the left child of the parent is smaller than the size of the array. If it is, we just check to see if the the dataItem of the left item is larger than the parent's dataItem and if it is we swap them and set the parent as the new parent left child since they were swapped. If that isn't true, the function returns temp since there is no right child to compare with since we first checked to see if a right child exists or not.

Parameters

none	

Returns

This function returns the DataType item that was removed from the heap.

Precondition

A heap so that the item may be removed from the heap.

Postcondition

This function removes the max heap of the heap and returns the DataType item that was removed from the heap. h

5.2.2.9 template<typename DataType , typename KeyType , typename Comparator > int Heap< DataType, KeyType, Comparator >::RightChild (int *parent*) [private]

This function is the RightChild function for the Heap class.

This function will take the parent and multiply the parent by 2 and add 2. This algorithm can be found in the C++ datastructure course textbook.

Parameters

int	parent which is used to find the right child of the parent

Returns

This function will return the index of the right child of the parent

Precondition

none

Postcondition

This function will return the right child index from the parent index that was passed in.

5.2.2.10 template<typename DataType , typename KeyType , typename Comparator > void Heap< DataType, KeyType, Comparator >::showStructure ( ) const

This function is the showStructure function for the Heap class

Outputs the priorities of the data items in a heap in both array and tree form. If the heap is empty, outputs "Empty heap". This operation is intended for testing/debugging purposes only.

Parameters

none	

5.2.2.11 template<typename DataType , typename KeyType , typename Comparator > void Heap< DataType, KeyType, Comparator >::showSubtree (int *index,* int *level*) const [private]

This function is the showStructure function for the Heap class

The function is the Helper function for the showStructure() function. Outputs the subtree (subheap) whose root is stored in dataItems[index]. Argument level is the level of this dataItems within the tree.

Parameters

none	

5.2.2.12 template<typename DataType , typename KeyType , typename Comparator > void Heap< DataType, KeyType, Comparator >::swap (int *index1,* int *index2*) [private]

This function is the swap function for the Heap class.

This function will take in two indexes and swap the data in them. This function creates a local variable DataType temp and initializes it to be of dataItems[index1]; The function assigns the value of that index to be the value of the second index passed. The function then assigns the value of the second index to be the temp variable that was created.

Parameters

int	index1 which is the first index to be swapped
int	index2 which is the second index to be swapped

Returns

This function does not return anything

Precondition

none

Postcondition

This function will swap the values of the two indexes that were passed in as the parameters.

5.2.2.13 template<typename DataType , typename KeyType , typename Comparator > void Heap< DataType, KeyType, Comparator >::writeLevels ( ) const

This function is the writeLevels function for the Heap class.

The function creates two int local variables called curr_Node and level and sets them to 0 and 1 respectively. The function checks to see if the array is empty by calling the isEmpty function and if it is, the function couts "Empty Heap." The function outputs the data items in a heal in level order, one level per line. The function does this by iterating through the size of the array. The function then sets if the curr_Node is equal to the level, it prints a new line and then multiplies the level by 2 and sets the curr_Node to 0. The function then prints the dataItems in current index and increments curr_Node;

Parameters

none	

Returns

This function does not return anything.

Precondition

An object to write the levels of the heap.

Postcondition

This function does not return anything.

The documentation for this class was generated from the following files:

- Heap.h
- Heap.cpp
- show11.cpp
- test.cpp

5.3 Less< KeyType > Class Template Reference

Public Member Functions

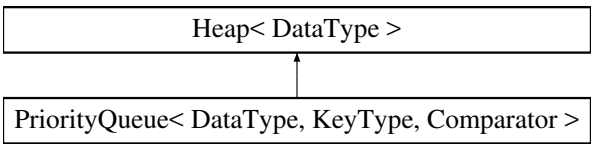
- bool **operator()** (const KeyType &a, const KeyType &b) const

The documentation for this class was generated from the following file:

- Heap.h

5.4 PriorityQueue< DataType, KeyType, Comparator > Class Template Reference

Inheritance diagram for PriorityQueue< DataType, KeyType, Comparator >:



Public Member Functions

- PriorityQueue (int maxNumber=defMaxQueueSize)
- void enqueue (const DataType &newDataItem)
- DataType dequeue ()

Additional Inherited Members

5.4.1 Constructor & Destructor Documentation

5.4.1.1 template<typename DataType , typename KeyType , typename Comparator > PriorityQueue< DataType, KeyType, Comparator >::PriorityQueue (int *maxNumber* = `defMaxQueueSize`)

This function is the constructor for the PriorityQueue class.

This function calls the base class implementation of the heap constructor, with the pass parameter maxNumber as the parameter.

Parameters

int	maxNumber, which is the largest possible size for the priorityqueue

Returns

This function does not return anything.

Precondition

none

Postcondition

The priotity queue is empty and has been dynamically allocated to have the maxNumber equal to the size.

5.4.2 Member Function Documentation

5.4.2.1 template<typename DataType , typename KeyType , typename Comparator > DataType PriorityQueue< DataType, KeyType, Comparator >::dequeue ( )

This function is the dequeue function for the PriorityQueue class.

This function calls the base class implementation of the remove function and returns return of the remove function.

Parameters

none	

Returns

This function returns true or false depending on if the item was dequeued.

Precondition

none

Postcondition

This function dequeues the top item of the priorityqueue and returns true or false depending on if the item was dequeued.

5.4.2.2 template<typename DataType , typename KeyType , typename Comparator > void PriorityQueue< DataType, KeyType, Comparator >::enqueue (const DataType & *newdataItem*)

This function is the insert function for the PriorityQueue class.

This function calls the base class implementation of the insert function, with the passed parameter newdataItem as the parameter.

Parameters

const	DataType &newDataItem, the dataItem to be inserted.

Returns

This function does not return anything.

Precondition

none

Postcondition

Inserts a new item into the priorityQueue.

The documentation for this class was generated from the following files:

- PriorityQueue.h
- PriorityQueue.cpp

5.5 TaskData Struct Reference

Public Member Functions

- int **getPriority** () const

Public Attributes

- int **priority**
- int **arrived**

The documentation for this struct was generated from the following file:

- ossim.cpp

5.6 TestData Class Reference

Public Member Functions

- void **setPriority** (int newPriority)
- int **getPriority** () const
- void **setPriority** (int newPriority)
- int **getPriority** () const

Private Attributes

- int **priority**

The documentation for this class was generated from the following files:

- test11hs.cpp
- test11pq.cpp

5.7 TestDataItem< KeyType > Class Template Reference

Public Member Functions

- void **setPriority** (KeyType newPty)
- KeyType **getPriority** () const

Private Attributes

- KeyType **priority**

The documentation for this class was generated from the following file:

- test11.cpp

6 File Documentation

6.1 Heap.cpp File Reference

This program will implement a HashTable using a Binary Search Tree ADT.

```
#include "Heap.h"
```

6.1.1 Detailed Description

This program will implement a HashTable using a Binary Search Tree ADT.

Author

Kripash Shrestha

Version

1.0

The specifications of the program are instructed and documented on Lab 11 C++ Data Structures: A Laboratory Course Third Edition by Brandle, Geisler, Roberge and Whittington

Date

Wednesday, November 15, 2017

6.2 PriorityQueue.cpp File Reference

This program will implement a inheritance class called Priority queue to develop a simulation of an operating system's task schedule using a priority queue.

```
#include "PriorityQueue.h"
```

6.2.1 Detailed Description

This program will implement a inheritance class called Priority queue to develop a simulation of an operating system's task schedule using a priority queue.

Author

Kripash Shrestha

Version

1.0

The specifications of the program are instructed and documented on Lab 11 C++ Data Structures: A Laboratory Course Third Edition by Brandle, Geisler, Roberge and Whittington

Date

Wednesday, November 15, 2017

Index