Heap ADT

AUTHOR

Version 1.00

11/04/2014

Table of Contents

Project 11 (Heap ADT)

This program will implement a Heap.

Author:

Saharath Kleips

The specifications of this project match those of the book C++ Data Structures - A Laboratory Course (3rd Edition) Project 11. A Heap forms a complete binary tree. For each data item E in the tree, all of E's descendants have priorities that are less than or equal to E's priority.

Class Index

Class Hierarchy

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

Class Index

Class List

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

File Index

File List

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

Class Documentation

Greater< KeyType > Class Template Reference

Public Member Functions

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

---

Detailed Description

template<typename KeyType = int>class Greater< KeyType >

Definition at line 46 of file test11.cpp.

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

1. test11.cpp

Heap< DataType, KeyType, Comparator > Class Template Reference

Public Member Functions

Heap (int maxNumber=DEFAULT_MAX_HEAP_SIZE)

The parameterized constructor that creates an empty Heap that allocates enough memory for a heap containing maxNumber data items.

Heap (const Heap &other)

The copy constructor that initializes this Heap to be equivalent to the other Heap.

Heap & operator= (const Heap &other)

The overloaded assignment operator that sets this Heap to be equivalent to the other Heap object parameter.

~Heap ()

The destructor that deallocates the memory used to store this Heap.

void insert (const DataType &newDataItem) throw (logic_error)

Inserts newDataItem into the heap.

DataType remove () throw (logic_error)

Removes the highest priority data item (the root) from this Heap and returns it.

void clear ()

Remove all data items from this Heap.

bool isEmpty () const

Return true of this Heap is empty.

bool isFull () const

Return true if the heap is full.

void **showStructure** () const

void writeLevels () const

Outputs the data items in level order.

Static Public Attributes

static const int **DEFAULT_MAX_HEAP_SIZE** = 10

Private Member Functions

void **showSubtree** (int index, int level) const

Private Attributes

int **maxSize**

int **size**

DataType * **dataItems**

Comparator **comparator**

Detailed Description

template<typename DataType, typename KeyType = int, typename Comparator = Less<KeyType>>class Heap< DataType, KeyType, Comparator >

Definition at line 24 of file Heap.h.

Constructor & Destructor Documentation

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

The parameterized constructor that creates an empty Heap that allocates enough memory for a heap containing maxNumber data items.

Postcondition:

This Heap will be a valid empty Heap with enough memory for maxNumber data items.

Parameters:

maxNumber	is how much memory to allocate for this Heap.

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

The copy constructor that initializes this Heap to be equivalent to the other Heap.

Precondition:

other is a valid Heap.

Postcondition:

This Heap will be a deep copy of the other Heap.

Parameters:

Definition at line 33 of file Heap.cpp.

other	is the Heap that this Heap will be made equivalent to.

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

The destructor that deallocates the memory used to store this Heap.

Postcondition:

This Heap will be an empty, deallocated, Heap.

Definition at line 85 of file Heap.cpp.

Member Function Documentation

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

Remove all data items from this Heap.

Postcondition:

This Heap will be an empty Heap.

Definition at line 209 of file Heap.cpp.

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

Inserts newDataItem into the heap.

Inserts this data item as the rightmost data item in the heap and moves it upward until the properties that define a heap are restored.

Precondition:

This Heap is not full.

Postcondition:

newDataItem will be inserted into the Heap.

Parameters:

Definition at line 48 of file Heap.cpp.

newDataItem	is the data item to be inserted into this Heap.

Exceptions:

This	Heap is full.

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

Return true of this Heap is empty.

Otherwise, returns false.

Returns:

True if this Heap is empty. False if this Heap is not empty.

Definition at line 219 of file Heap.cpp.

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

Return true if the heap is full.

Otherwise, returns false.

Returns:

True if this heap is full. False if this heap is not full.

Definition at line 229 of file Heap.cpp.

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

The overloaded assignment operator that sets this Heap to be equivalent to the other Heap object parameter.

Precondition:

other is a valid Heap.

Postcondition:

This Heap will be a deep copy of the other Heap.

Parameters:

Definition at line 100 of file Heap.cpp.

other	is the Heap that this Heap will be made equivalent to.

The reference to this Heap.

Definition at line 62 of file Heap.cpp.

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

Removes the highest priority data item (the root) from this Heap and returns it.

Replaces the root data item with the bottom rightmost data item and moves this data item downward until the properties that define a heap are restored.

Precondition:

This Heap is not empty.

Postcondition:

This Heap will no longer contain the highest priority data item.

Returns:

The highest priority data item.

Exceptions:

Returns:

This	Heap is empty.

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

Outputs the data items in level order.

One level per line. Only outputs each data item's priority. If the heap is empty, then outputs "Empty heap".

Postcondition:

The level order of this Heap will be outputted to the console.

Definition at line 240 of file Heap.cpp.

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

2 Heap.h
3 Heap.cpp
4 show11.cpp

Less< KeyType > Class Template Reference

Public Member Functions

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

---

Detailed Description

template<typename KeyType = int>class Less< KeyType >

Definition at line 18 of file Heap.h.

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

5 Heap.h

PriorityQueue< DataType, KeyType, Comparator > Class Template Reference

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

IMAGE

Public Member Functions

PriorityQueue (int maxNumber=defMaxQueueSize)

The parameterized constructor that creates an empty Priority Queue that allocates enough memory for maxNumber data items.

void enqueue (const DataType &newDataItem)

Inserts newDataItem into this Priority Queue.

DataType dequeue ()

Removes the highest priority item from this Priority Queue and returns it.

Detailed Description

template<typename DataType, typename KeyType = int, typename Comparator = Less<KeyType>>class PriorityQueue< DataType, KeyType, Comparator >

Definition at line 24 of file PriorityQueue.h.

Constructor & Destructor Documentation

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

The parameterized constructor that creates an empty Priority Queue that allocates enough memory for maxNumber data items.

Postcondition:

This Priority Queue will be a valid empty Priority Queue with enough memory for maxNumber data items.

Parameters:

Definition at line 135 of file Heap.cpp.

maxNumber	is how much memory to allocate for this Priority Queue.

Member Function Documentation

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

Removes the highest priority item from this Priority Queue and returns it.

Precondition:

This Priority Queue is not empty.

Postcondition:

This Priority Queue will no longer contain the highest priority item.

Returns:

The highest priority data item.

Definition at line 56 of file PriorityQueue.cpp.

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

Inserts newDataItem into this Priority Queue.

Precondition:

This Priority Queue is not full.

Postcondition:

newDataItem will be inserted into the Priority Queue.

Parameters:

Definition at line 32 of file PriorityQueue.cpp.

newDataItem	is the data item to be inserted into the Priority Queue.

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

6 PriorityQueue.h
7 PriorityQueue.cpp

TaskData Struct Reference

Public Member Functions

int **getPriority** () const

Public Attributes

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

Detailed Description

Definition at line 23 of file ossim.cpp.

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

8 ossim.cpp

TestData Class Reference

Public Member Functions

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

Private Attributes

int **priority**

Detailed Description

Definition at line 22 of file test11pq.cpp.

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

9 test11pq.cpp

TestDataItem< KeyType > Class Template Reference

Public Member Functions

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

Private Attributes

KeyType **priority**

Detailed Description

template<typename KeyType>class TestDataItem< KeyType >

Definition at line 29 of file test11.cpp.

The documentation for this class was generated from the following file:
10 test11.cpp

File Documentation

Heap.cpp File Reference

```
#include "Heap.h"
#include "show11.cpp"
#include <stdexcept>
#include <iostream>
```

Detailed Description

Definition in file Heap.cpp.

PriorityQueue.cpp File Reference

This program will implement a Priority Queue.

```
#include "PriorityQueue.h"
#include <stdexcept>
#include <iostream>
```

Detailed Description

This program will implement a Priority Queue.

Author:

Saharath Kleips

The specifications of this project match those of the book C++ Data Structures - A Laboratory Course (3rd Edition) Project 11. A Priority Queue is a linear data structure in which data items are maintained in descending order based on priority. You can only access data at the front of the queue, examining this data item entails removing it from the queue.

Definition in file PriorityQueue.cpp.

Index

INDE