TITLE

AUTHOR
Version
CREATEDATE

Table of Contents

Table of contents

Hierarchical Index

Class Hierarchy

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

Greater< KeyType > 5
Heap< DataType, KeyType, Comparator > 6
Heap< DataType > 6
PriorityQueue< DataType, KeyType, Comparator > 12

Less< KeyType > 11
TaskData 14
TestData 15
TestDataItem< KeyType > 16

Class Index

Class List

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

Greater< KeyType > 5
Heap< DataType, KeyType, Comparator > 6
Less< KeyType > 11
PriorityQueue< DataType, KeyType, Comparator > 12
TaskData 14
TestData 15
TestDataItem< KeyType > 16

File Index

File List

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

config.h ..... **Error! Bookmark not defined.**

Heap.cpp (This program will implement a Heap) 17

Heap.h ..... **Error! Bookmark not defined.**

ossim.cpp (This program will implement an operating system task scheduling system) 18

PriorityQueue.cpp (This program will implement a PriorityQueue) 20

PriorityQueue.h ..... **Error! Bookmark not defined.**

Class Documentation

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

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

Constructor & Destructor Documentation

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

Constructor for **Heap**. Creates an empty heap. Allocates enough memory for a heap containing maxNumber data items. Pseudocode: set maxSize equal to maxNumber, allocate memory for dataItems based on maxSize, and set size to 0

Parameters:

maxNumber	: The maximum number of elements that can be in this heap

Returns:

None

Precondition:

None

Postcondition:

This heap will have a maxSize and size defined, and our heap will be initialized

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

Copy constructor for **Heap**. Initializes the object to be an equivalent copy of other. Pseudocode:

1. Set maxSize equal to other's max size, size to other's max size, allocate memory to dataItems based on maxSize, and use a loop to traverse through our elements to set them equal to other's elements.

Parameters:

	return

Precondition:

None

Postcondition:

This heap will be equal to other

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

Destructor for **Heap**. Deallocates (frees) the memory used to store the heap. Pseudocode: Call the clear method

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

All memory being used by this heap will be deallocated

Member Function Documentation

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

Removes all the data items in the heap. Pseudocode: delete the array. Set size to 0.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Size will be set to 0 and all memory deallocated from the heap

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 bottom rightmost data item in the heap and moves it upward until the properties that define a heap are restored. Pseudocode:

1. Check if full. If so, throw logic error. Otherwise, go to 2.
2. Set dataItems[size] to parameter passed in
3. Increment size (so that the index we just set is now valid within the heap)
4. Call the reprioritize method

Parameters:

newDataItem	: new item to be inserted into the list

Returns:

None

Precondition:

Heap is not full

Postcondition:

The newDataItem will be inserted, and the rest of the heap will be sorted based on this new insertion

Exceptions:

logic_error	: Throws a logic error if the heap is full

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

Returns true if the heap is empty. Otherwise, returns false. Pseudocode: return size == 0

Parameters:

None	

Returns:

bool : true if heap is empty, false otherwise

Precondition:

None

Postcondition:

The elements of the heap will not be changed.

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

Returns true if the heap is full. Otherwise, returns false. Pseudocode: return size == maxSize

Parameters:

None	

Returns:

bool : True if heap is full, false otherwise

Precondition:

None

Postcondition:

The elements of the heap will not be changed.

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

Overloaded assignment operator for **Heap**. Sets the heap to be equivalent to the other **Heap** and returns a reference to this object. Pseudocode:

1. Check if this already equals the other. If it does, stop right there and just return
2. Otherwise, clear the heap.
3. Set maxSize equal to other's max size, size to other's max size, allocate memory to dataItems based on maxSize, and use a loop to traverse through our elements to set them equal to other's elements.
4. Returns this.

Parameters:

other	: Another heap of which this heap will become a copy of

Returns:

Heap : This

Precondition:

None

Postcondition:

This heap will be equal to other

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

Removes the data item with the highest priority (the root) from the 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. Pseudocode:

1. If empty, throw logic error. Else go to 2.
2. Swap root with last elements
3. Decrement size
4. Call reprioritize method
5. Return item at size (that item is technically out of bounds of our heap, but it still exists in memory so utilize this)

Parameters:

None	

Returns:

DataType Highest priority item / root

Precondition:

Heap is not empty

Postcondition:

Root will have been removed, and replaced, and everything will be sorted

Exceptions:

logic_error	: Throws an error if the heap is empty

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

Outputs the priorities of the data items in the heap in both array and tree form. The tree is output with its branches oriented from left (root) to right (leaves) - that is, the tree is output rotated counterclockwise ninety degrees from its conventional orientation. If the heap is empty, outputs "Empty heap". Note that this operation is intended for testing/debugging purposes only.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of this heap will not be changed.

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

Writes each level of the tree. Pseudocode:

1. Use loop to go through entire tree (condition being that the "currentNode" is less than size).
2. Check if the currentNode index is less than the index of the next level.
3. If it is, output the currentNode and increment currentNode.
4. Otherwise, output an endl and increase the nextLevel to the appropriate length (2*currentNode + 1)

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

None

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

- Heap.h
- **Heap.cpp**
- show11.cpp

Less< KeyType > Class Template Reference

Public Member Functions

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

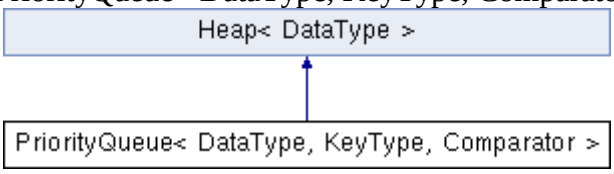
---

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

- Heap.h

PriorityQueue< DataType, KeyType, Comparator > Class Template Reference

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



Public Member Functions

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

Additional Inherited Members

Constructor & Destructor Documentation

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

Constructor for **PriorityQueue**. Calls constructor for heap. Set maxNumber of items this PriorityHeap can hold.

Parameters:

maxNumber	: maximum number of items allowed in this heap.

Returns:

None

Precondition:

None

Postcondition:

This PriorityHeap will be initialized.

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

Destructor for PriorityHeap. Deallocates all memory in this heap. Calls **Heap** clear method.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

All memory of this heap will be deallocated.

Member Function Documentation

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

Removes highest priority item in the PriorityHeap. Uses base heap's remove method.

Parameters:

None	

Returns:

DataType : highest priority item that was removed from the PriorityHeap

Precondition:

None

Postcondition:

HIghest priority item will be removed from the PriorityHeap.

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

Enqueues an item into the **PriorityQueue**. Uses heap insert method.

Parameters:

newDataItem	: New item to insert into the PriorityHeap.

Returns:

None

Precondition:

None

Postcondition:

Item will be inserted into the heap.

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

- PriorityQueue.h
- **PriorityQueue.cpp**

TaskData Struct Reference

Public Member Functions

- int **getPriority** () const
- int **getPriority** () const

Public Attributes

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

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

- **ossim.cpp**
- ossim.cs

TestData Class Reference

Public Member Functions

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

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

- test11hs.cpp
- test11pq.cpp

TestDataItem< KeyType > Class Template Reference

Public Member Functions

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

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

- test11.cpp

File Documentation

Heap.cpp File Reference

This program will implement a **Heap**.
`#include "Heap.h"`

Detailed Description

This program will implement a **Heap**.

Author:
Tim Kwist

Version:
1.0

The specifications of this program are defined by C++ Data Structures: A Laboratory Course (3rd edition) by Brandle, JGeisler, Roberge, Whittington, lab 11.

Date:
Wednesday, November 5, 2014

ossim.cpp File Reference

This program will implement an operating system task scheduling system.

```
#include <iostream>
#include <cstdlib>
#include "PriorityQueue.cpp"
```

Classes

- struct **TaskData**

Functions

- int **main** ()

Detailed Description

This program will implement an operating system task scheduling system.

Author:

Tim Kwist

Version:

1.0

The specifications of this program are defined by C++ Data Structures: A Laboratory Course (3rd edition) by Brandle, JGeisler, Roberge, Whittington, lab 11.

Date:

Wednesday, November 5, 2014

Function Documentation

int main ()

Programming Exercise #1 for Lab 11. (Shell) Operating system task scheduling simulation Pseudocode:

1. Get number of priority levels
2. Get length of time to run the simulator
3. Run loop for length of time to run the simulator. 3a. Dequeue item in **PriorityQueue** if it exists. Output the item dequeue'd. 3b. Determine number of items to add to the list based on a random number (50% no items, 25% 1 item, 25% 2 items) 3c. Insert previously determined number of items into **PriorityQueue**.

Parameters:

None	

Returns:

int : default return for main function

Precondition:

None

Postcondition:

None

PriorityQueue.cpp File Reference

This program will implement a **PriorityQueue**.

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

Detailed Description

This program will implement a **PriorityQueue**.

Author:

Tim Kwist

Version:

1.0

The specifications of this program are defined by C++ Data Structures: A Laboratory Course (3rd edition) by Brandle, JGeisler, Roberge, Whittington, lab 11.

Date:

Wednesday, November 5, 2014

Index

INDEX