Terence Henriod

Generated by Doxygen 1.7.6.1

Wed Nov 13 2013 17:44:06

Contents

Chapter 1

Class Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Greater< KeyType > Class Template Reference

Public Member Functions

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

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

4.1.1 Member Function Documentation

4.1.1.1 template<typename KeyType = int> bool Greater< KeyType >::operator() (const KeyType & *a*, const KeyType & *b*) const `[inline]`

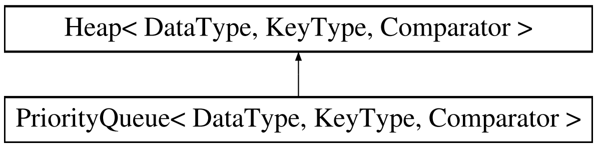
The documentation for this class was generated from the following file:

- test11.cpp

4.2 Heap< DataType, KeyType, Comparator > Class Template - Reference

`#include <Heap.h>`

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


Heap< DataType, KeyType, Comparator >
PriorityQueue< DataType, KeyType, Comparator >

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

Private Attributes

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

4.2.1 Detailed Description

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

A Heap ADT. The Heap is a data structure for storing data as though it were a tree whose requirements are that no parent is less than a child and that the tree must be completely full except for the bottom level. The data is stored in an array, granting fast access and easy reordering of data. Because the data is stored in an array, the bottom level of data fills up from left to right.

4.2.2 Constructor & Destructor Documentation

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

Heap

The default constructor for the heap. Instantiates an empty heap.

Parameters

maxNumber	The maximum capacity given to the heap. It is recommended that a size that is 2^n - 1 is chosen. This parameter defaults to DEFAULT_MAX_HEAP_SIZE (defined in Heap.h).

Precondition

1. A valid identifier for the Heap is given
2. The given template parameter DataType should support comparisons and some for key identification.
3. The given number for the parameter int maxNumber must be greater than zero.

Postcondition

1. An empty Heap of the given types and size (if it is specified) is instantiated.

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

Heap

The copy constructor for the heap. Instantiates a Heap that is a clone of the given other parameter.

Parameters

other	A Heap to be cloned into ∗this.

Precondition

1. A valid identifier for the Heap is given
2. The given template parameter DataType should support comparisons and some for key identification.

3. The given number for the parameter int maxNumber must be greater than zero.

Postcondition

1. A Heap that is a clone of the given other parameter is instantiated.

4.2.2.3 template$<$typename DataType , typename KeyType , typename Comparator $>$ Heap$<$ DataType, KeyType, Comparator $>$::$\sim$Heap ( )

$\sim$Heap

The destructor for the heap ADT. Ensures that all dynamically allocated memory is returned.

Precondition

1. There is a Heap to destruct.

Postcondition

1. The dynamic memory allocated for the array will be returned.
2. The Heap object ($*$this) will be destroyed.

1. The dynamically allocated array is deleted.
2. The rest of the heap is appropriately destroyed in the usual manner.

4.2.3 Member Function Documentation

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

clear

Empties the heap.

Precondition

1. A valid Heap instantiation exists.

Postcondition

1. Data items of the heap are "discarded" by setting the size member to zero.

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

insert

Inserts the given value into the appropriate place in the array if there is room. Throws an expression of type logic_error if the Heap is full.

Parameters

newDataItem	The new item to be inserted into the Heap.

Precondition

1. A valid Heap instantiation exists.
2. The given parameter newDataItem is of appropriate type.
3. The Heap must not be full for successful insertion.

Postcondition

1. If there was room in the Heap, the newDataItem will be placed in the Heap such that it will be less than any "parent" that it may have.
2. If the heap is full and insertion is attempted, an exception of type logic_error is thrown, indicating that the heap is full.

1. If the Heap is full, an exception is thrown, otherwise the DataType newDataItem is placed in the first available location.
2. The newDataItem is then percolated up to the appropriate level in the heap by comparing the newly inserted item with its current "parent" and swapping if necessary until the new item cannot rise any further.

Exceptions

logic_error	This exception is thrown if an attempt is made to insert into a full heap.

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

isEmpty

Indicates if the Heap is full by returning true if it is empty and false otherwise.

Returns

empty A boolean containing the truth of the emptiness of the Heap.

Precondition

1. A valid Heap instantiation exists.

Postcondition

1. The Heap will remain unchanged.
2. If the Heap is empty, true is returned, and false otherwise.

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

isFull

Indicates if the Heap is full by returning true if it is full, and false otherwise.

Returns

full A boolean containing the truth of the fullness of the heap

Precondition

1. A valid Heap instantiation exists.

Postcondition

1. The Heap will remain unchanged.
2. If the Heap is full, true is returned, and false otherwise.

Exceptions

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

operator=

The overloaded assignment operator for the Heap ADT. Clones the given Heap other parameter into ∗this.

Parameters

other	A Heap whose data will be cloned into this one.

Returns

∗this ∗this is returned by reference for multi-line assignments.

Precondition

1. Both ∗this and other are valid Heaps

Postcondition

1. The contents of other will be cloned into ∗this. The original data will be more.

1. If ∗this is not being assigned to ∗this, the current data is abandoned.
2. If other has a different maxSize than ∗this, the dataItems array is re-sized
3. The dataItems array is made equivalent to the one in other

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

remove

Removes the item at the top ("root") of the Heap. The heap is then reorderd appropriately to maintain the properties of a heap. The removed item is returned. An exception is thrown if removal is attempted on an empty heap.

Returns

removedItem An item of type DataType. This item was the item at the top of the heap.

Precondition

1. A valid Heap has been instantiated.
2. The heap has an item to be removed, otherwise, and excpetion will be thrown.

Postcondition

1. If an item at the top of the Heap, it is removed from the heap and returned.
2. If the Heap is empty, an exception of type logic_error is thrown with a message indicating that removal cannot be performed.

1. If the Heap is empty, an exception of type logic_error is thrown. If the Heap is not empty, the top item is stored for returning.
2. The bottom-right-most item is placed at the top of the Heap and the size is reduced.
3. The newly placed top item is worked downward until it sits in the appropriate place. This is accomplished by swapping the item with any child that compares greater than the new top item.

Exceptions

logic_error	This exception is thrown if removal on an empty heap is attempted.

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

showStructure

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.

Precondition

1. A valid Heap instantiation exists.
2. The type DataType supports operator<<

Postcondition

1. The Heap will remain unchanged.
2. The Heap will be displayed on the screen, first in array form, then in tree form.

1. The array elements are iteratively displayed.

2. The showSubtree helper is then called to display the Heap as a tree

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

FunctionName

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.

Precondition

1. A valid Heap instantiation exists.
2. The type DataType supports operator<<
3. recursive calls to this function may have been previously made.

Postcondition

1. The Heap will remain unchanged.
2. The Heap subtree will be displayed in a right-ward growing tree on the screen.

1. A reversed in-order traversal is used to display the items as a tree.
2. Children are found using the formula 2∗parentIntex + (1, 2)

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

writeLevels

Writes the priorities (keys) of the contents of the Heap wo the screen, one level at a time, beginning at the top.

Precondition

1. A valid instantiation of the Heap exists.
2. Type DataType must support a getPriority() method.

Postcondition

1. The Heap will remain unchanged.
2. The priorities (keys) of the Heap will be listed by level, from top to bottom. If the Heap is empty, it is reported.

1.

4.2.4 Member Data Documentation

4.2.4.1 template$<$typename DataType, typename KeyType = int, typename Comparator = Less$<$ KeyType $>>$ Comparator **Heap**$<$ DataType, KeyType, Comparator $>$::**comparator** `[private]`

4.2.4.2 template$<$typename DataType, typename KeyType = int, typename Comparator = Less$<$ KeyType $>>$ DataType$*$ **Heap**$<$ DataType, KeyType, Comparator $>$::**dataItems** `[private]`

4.2.4.3 template$<$typename DataType, typename KeyType = int, typename Comparator = Less$<$ KeyType $>>$ const int **Heap**$<$ DataType, KeyType, Comparator $>$::**DEFAULT_MAX_HEAP_SIZE** = 10 `[static]`

4.2.4.4 template$<$typename DataType, typename KeyType = int, typename Comparator = Less$<$ KeyType $>>$ int **Heap**$<$ DataType, KeyType, Comparator $>$::**maxSize** `[private]`

4.2.4.5 template$<$typename DataType, typename KeyType = int, typename Comparator = Less$<$ KeyType $>>$ int **Heap**$<$ DataType, KeyType, Comparator $>$::**size** `[private]`

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

- Heap.h
- Heap.cpp

4.3 Less$<$ KeyType $>$ Class Template Reference

```
#include <Heap.h>
```

Public Member Functions

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

4.3.1 Detailed Description

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

A class with an overloaded operator() (function operator) for use as the comparator of a Heap object. Subtle manipulations make this class act more as a function, rather than a class.

4.3.2 Member Function Documentation

4.3.2.1 template<typename KeyType = int> bool Less< KeyType >::operator() (const KeyType & *a*, const KeyType & *b*) const `[inline]`

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

- Heap.h

4.4 PriorityArrivalCompare< TaskType > Class Template Reference

Public Member Functions

- bool operator() (const TaskType &first, const TaskData &second)

4.4.1 Detailed Description

template<typename TaskType = TaskData>class PriorityArrivalCompare< TaskType >

A class that acts as a function to compare both priority and arrival time to create a "fair queue."

CURRENTLY NOT IN USE.

4.4.2 Member Function Documentation

4.4.2.1 template<typename TaskType = TaskData> bool PriorityArrivalCompare< TaskType >::operator() (const TaskType & *first*, const TaskData & *second*) `[inline]`

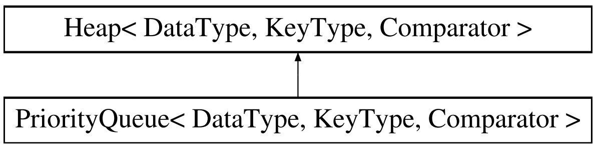
The documentation for this class was generated from the following file:

- ossim.cpp

4.5 PriorityQueue< DataType, KeyType, Comparator > Class - Template Reference

`#include <PriorityQueue.h>`

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


Heap< DataType, KeyType, Comparator >
PriorityQueue< DataType, KeyType, Comparator >

Public Member Functions

- PriorityQueue (int maxNumber=defMaxQueueSize)
- PriorityQueue (const Heap< DataType, KeyType, Comparator > &other)
- void enqueue (const DataType &newDataItem)
- DataType dequeue ()

4.5.1 Detailed Description

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

The Priority Queue ADT. Inherits an array based Heap ADT in order to provide a priority first functionality. This class is really just a practice for inheritance and provides new function names to mask the Heap's functionality.

4.5.2 Constructor & Destructor Documentation

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

PriorityQueue

The default constructor for the PriorityQueue ADT. Calls the Heap constructor with the given size parameter value.

Parameters

maxNumber	The size value for the PriorityQueue. Defaults to the constant value defMaxQueueSize specified in PriorityQueue.h.

Precondition

1. A valid identifier is selected for the PriorityQueue.

Postcondition

1. An empty PriorityQueue will be instantiated.

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

PriorityQueue

The copy constructor for the PriorityQueue ADT. Calls the Heap copy constructor with the given Heap parameter value in order to create a clone of other into this.

Parameters

other	A Heap to be cloned into ∗this.

Precondition

1. A valid identifier is selected for the PriorityQueue.
2. Heap other is a valide Heap instantiation.

Postcondition

1. An empty PriorityQueue will be instantiated.

4.5.3 Member Function Documentation

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

dequeue

Dequeues the highest priority item. The item is returned along with its removal.

Returns

dequeuedItem The item of type DataType that was removed from the Heap, and therefore, the PriorityQueue.

Precondition

1. The PriorityQueue must not be empty, otherwise an exception will be thrown.

Postcondition

1. If the PriorityQueue is not empty, the highest priority item is removed from the queue and returned. If dequeueing is attempted on an empty PriorityQueue, an exception of type logic_error is thrown.

1. The remove method of the Heap base class is utilized.

Exceptions

logic_error	This exception is thrown by the Heap base class if removal is attempted when the PriorityQueue is empty.

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

enqueue

Adds the data item to the Priority Queue. The item's releative position will depend on it's priority.

Parameters

newDataItem	The new DataItem to be inserted into the PriorityQueue

Precondition

1. DataType must support a valid getPriority() method.

Postcondition

1. If there is room in the PriorityQueue, the new item will be stored in the appropriate position. If there is no room, an exception of type logic_error is thrown.

1. The Heap's insert method is called.

Exceptions

logic_error	The Heap base class may throw an exception if insertion is attempted when the data array is full.

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

- PriorityQueue.h
- PriorityQueue.cpp

4.6 TaskData Struct Reference

Public Member Functions

- int getPriority () const
- int getArrival () const

Public Attributes

- int priority
- int arrived

4.6.1 Detailed Description

A struct used to simulate a task an operating system might have to schedule. Contains members that contain data pertaining to priority and arrival time. Supports a getPriority method to maintain compatibility with the Heap ADT.

4.6.2 Member Function Documentation

4.6.2.1 int TaskData::getArrival ( ) const [inline]

4.6.2.2 int TaskData::getPriority ( ) const [inline]

4.6.3 Member Data Documentation

4.6.3.1 int TaskData::arrived

4.6.3.2 int TaskData::priority

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

- ossim.cpp

4.7 TestDataItem< KeyType > Class Template Reference

Public Member Functions

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

Private Attributes

- KeyType priority

template<typename KeyType> class TestDataItem< KeyType >

4.7.1 Constructor & Destructor Documentation

4.7.1.1 template<typename KeyType > TestDataItem< KeyType >::TestDataItem ( ) `[inline]`

4.7.2 Member Function Documentation

4.7.2.1 template<typename KeyType > KeyType TestDataItem< KeyType >::getPriority ( ) const `[inline]`

4.7.2.2 template<typename KeyType > void TestDataItem< KeyType >::setPriority (KeyType *newPty*) `[inline]`

4.7.3 Member Data Documentation

4.7.3.1 template<typename KeyType > KeyType TestDataItem< KeyType >::priority `[private]`

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

- test11.cpp

Chapter 5

File Documentation

5.1 config.h File Reference

Defines

- #define LAB11_TEST1 1

5.1.1 Define Documentation

5.1.1.1 #define LAB11_TEST1 1

Heap class configuration file. Activate test #N by defining the corresponding LAB11_T-ESTN to have the value 1.

5.2 Heap.cpp File Reference

Class implementations declarations for the Heap ADT.

`#include "Heap.h" #include <stdexcept> #include <iostream> ×`

5.2.1 Detailed Description

Class implementations declarations for the Heap ADT.

Author

Terence Henriod

Lab 10: Heap

Version

Original Code 1.00 (11/8/2013) - T. Henriod

5.3 Heap.h File Reference

Class declarations for the Heap ADT.

`#include <stdexcept>` `#include <iostream>`

Classes

- class Less< KeyType >
- class Heap< DataType, KeyType, Comparator >

5.3.1 Detailed Description

Class declarations for the Heap ADT. Class declaration for the Heap implementation of the Priority Queue ADT -- inherits the array implementation of the Heap ADT.

Author

Terence Henriod

Lab 10: Heap

Version

Original Code 1.00 (11/8/2013) - T. Henriod

5.4 heapsort.cpp File Reference

Functions

- template<typename DataType >
  void moveDown (DataType dataItems[ ], int root, int size)
- template<typename DataType >
  void heapSort (DataType dataItems[ ], int size)

5.4.1 Function Documentation

5.4.1.1 template<typename DataType > void **heapSort** (DataType *dataItems[ ],* int *size*)

heapSort

Heap sort routine. Sorts the data items in the array in ascending order based on priority.

Parameters

dataItems[]	The array to be heapified.
size	The number of items in the array as a whole.

Precondition

1. The array should contain elements arranged as in a binary search tree.

Postcondition

1. The items in the array will be heapified.

5.4.1.2 template<typename DataType > void moveDown (DataType *dataItems[ ],* int *root,* int *size*)

moveDown

Converts a binary search tree (array implementation) subtree into a heap. Assumes any lower subtrees are alrady heaps. Restores the binary tree that is rooted at root to a heap by moving dataItems[root] downward until the tree satisfies the heap property. Parameter size is the number of data items in the array.

Parameters

dataItems[]	The array to be heapified.
root	The index of the array indicating the root of a sub-tree.
size	The number of items in the array as a whole.

Precondition

1. All parameters are valid.
2. Any subtrees are already heaps (assumed, not checked)

Postcondition

1. The sub-array will be heapified.

5.5 ossim.cpp File Reference

Contains definitions for cunctions that together comprise a heap sort utility. The heapSort() shell function is provided by the lab manual package, while the moveDown helper function was written by T. Henriod. This heapSort is for arrays.

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

Classes

- struct TaskData
- class PriorityArrivalCompare< TaskType >

Functions

- TaskData addTask (int arrivalTime, int numPriorities)
- int main ()

5.5.1 Detailed Description

Contains definitions for cunctions that together comprise a heap sort utility. The heap-Sort() shell function is provided by the lab manual package, while the moveDown helper function was written by T. Henriod. This heapSort is for arrays. A shell program that utilizes the PriorityQueue ADT to simulatean operating system's use of a priority queue to regulate access to a system resource (printer, disk, etc.).

Author

Terence Henriod

Lab 10: Heap Sort

Version

Original Code 1.00 (11/8/2013) - T. Henriod

Author

Terence Henriod

Lab 10: Operating System Scheduling Simulator

Version

Original Code 1.00 (11/8/2013) - T. Henriod

5.5.2 Function Documentation

5.5.2.1 TaskData addTask (int *arrivalTime,* int *numPriorities*)

addTask

Generates a new task for the simulation using the arrival time of the task and a random priority level.

Parameters

arrivalTime	The time the task is simulated to arrive into the PriorityQueue.
num-Priorities	The number of different priority levels being used in the simulation.

Returns

newTask A new task of type TaskData. Contains information relevant to the simulation.

Precondition

1. Ideally, int arrivalTime and numPriorities should be passed logical values.

Postcondition

1. A new task will be generated with a simulation arrival time and a random priority.

5.5.2.2 int main ( )

main

The driving function of the program. A number of differing priority levels and a length of time to run the simulation are prompted for, and then the simulation is run, randomly adding 0, 1, or 2 tasks of random priority each time. Each time a task is "processed," its summary statistics are reported.

Returns

0 This return value indicates error free execution.

Precondition

1. None.

Postcondition

1. A operating system task scheduling simulation will have been run.

5.6 PriorityQueue.cpp File Reference

Class implementations declarations for the PriorityQueue ADT (inherits from the array based Heap ADT).

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

5.6.1 Detailed Description

Class implementations declarations for the PriorityQueue ADT (inherits from the array based Heap ADT).

Author

Terence Henriod

Project Name

Version

Original Code 1.00 (11/8/2013) - T. Henriod

5.7 PriorityQueue.h File Reference

```
#include <stdexcept> #include <iostream> #include "Heap.-
cpp"
```

Classes

- class PriorityQueue< DataType, KeyType, Comparator >

Variables

- const int defMaxQueueSize = 10

5.7.1 Variable Documentation

5.7.1.1 const int **defMaxQueueSize** = 10

5.8 test11.cpp File Reference

```
#include <iostream> #include <string> #include <cctype>×
#include "Heap.cpp" #include "config.h"
```

Classes

- class TestDataItem< KeyType >
- class Greater< KeyType >

Functions

- void printHelp ()
- int main ()

5.8.1 Function Documentation

5.8.1.1 int main ( )

5.8.1.2 void printHelp ( )