proj1

Generated by Doxygen 1.12.0

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 DoubleLinkedList< T > Class Template Reference

Implementation of a double linked list using the heap.

`#include <doublelinkedlist.hpp>`

Public Member Functions

- DoubleLinkedList (void)
  Default Constructor.
- ~DoubleLinkedList (void)
- void display (void)
  Print the entire list.
- void rdisplay (void)
  Print the entire list in reverse order.
- void append (const T &item)
  Append an item to the end of the list.
- void prepend (const T &item)
  Prepend an item to the beggining of the list.
- void insert (const T &item, size_t index)
  Insert an item at a certain index.
- void pop (void)
  Remove an element from the back of the list.
- void rpop (void)
  Remove an element from the head of the list.
- void pop_at (size_t index)
  Remove an element at a given index.
- T & operator[ ] (size_t index)
- void clear ()

3.1.1 Detailed Description

template<typename T>
class DoubleLinkedList< T >

Implementation of a double linked list using the heap.

Bottom Text

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DoubleLinkedList()

```
template<typename T >
DoubleLinkedList< T >::DoubleLinkedList (
            void )  [inline]
```

Default Constructor.

3.1.2.2 ~DoubleLinkedList()

```
template<typename T >
DoubleLinkedList< T >::~DoubleLinkedList (
            void )  [inline]
```

3.1.3 Member Function Documentation

3.1.3.1 append()

```
template<typename T >
void DoubleLinkedList< T >::append (
            const T & item)  [inline]
```

Append an item to the end of the list.

3.1.3.2 clear()

```
template<typename T >
void DoubleLinkedList< T >::clear ()  [inline]
```

3.1.3.3 display()

```
template<typename T >
void DoubleLinkedList< T >::display (
            void )  [inline]
```

Print the entire list.

3.1.3.4 insert()

```
template<typename T >
void DoubleLinkedList< T >::insert (
            const T & item,
            size_t index)  [inline]
```

Insert an item at a certain index.

3.1.3.5 operator[]()

```
template<typename T >
T & DoubleLinkedList< T >::operator[] (
            size_t index)  [inline]
```

3.1.3.6 pop()

```
template<typename T >
void DoubleLinkedList< T >::pop (
            void )  [inline]
```

Remove an element from the back of the list.

3.1.3.7 pop_at()

```
template<typename T >
void DoubleLinkedList< T >::pop_at (
            size_t index)  [inline]
```

Remove an element at a given index.

3.1.3.8 prepend()

```
template<typename T >
void DoubleLinkedList< T >::prepend (
            const T & item)  [inline]
```

Prepend an item to the beggining of the list.

3.1.3.9 rdisplay()

```
template<typename T >
void DoubleLinkedList< T >::rdisplay (
            void )  [inline]
```

Print the entire list in reverse order.

3.1.3.10 rpop()

```
template<typename T >
void DoubleLinkedList< T >::rpop (
            void )  [inline]
```

Remove an element from the head of the list.

Chapter 4

File Documentation

4.1 src/doublelinkedlist.hpp File Reference

```
#include <print>
```

Classes

- class DoubleLinkedList< T >

  Implementation of a double linked list using the heap.

4.2 doublelinkedlist.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <print>
00004
00008
00009 template<typename T>
00010 class DoubleLinkedList {
00011 private:
00012     struct Node {
00013         Node* previous;
00014         Node* next;
00015         T contents;
00016
00017         Node(Node* oPrevious, Node* oNext, T oContents):
00018             previous { oPrevious },
00019             next { oNext },
00020             contents { oContents } {}
00021
00022         ~Node(void) {
00023             delete next;
00024         }
00025     };
00026
00027     Node* head;
00028     Node* back;
00029
00030     template<typename F>
00031     Node* traverse_forward_if(F predicate) {
00032         if(head == nullptr) {
00033             return nullptr;
00034         }
00035
00036         Node* currentElement { head };
00037
```

```
00038         while(predicate(currentElement->next)) {
00039             currentElement = currentElement->next;
00040         }
00041 
00042         return currentElement;
00043     }
00044 
00045     Node* get_node_at(size_t index) {
00046         Node* currentNode { head };
00047         for(size_t i {}; i < index; i++) {
00048             currentNode = currentNode->next;
00049         }
00050 
00051         return currentNode;
00052     }
00053 
00054 public:
00056     DoubleLinkedList(void): head { nullptr }, back { nullptr } {}
00057     ~DoubleLinkedList(void) {
00058         delete head;
00059     }
00060 
00062     void display(void) {
00063         Node* currentElement { head };
00064         std::print("{{");
00065 
00066         while(true) {
00067             if(currentElement == nullptr) {
00068                 std::print("}}");
00069                 return;
00070             }
00071 
00072             std::print("{}, ", currentElement->contents);
00073 
00074             currentElement = currentElement->next;
00075         }
00076     }
00077 
00079     void rdisplay(void) {
00080         Node* currentElement { back };
00081         std::print("{{");
00082 
00083         while(true) {
00084             if(currentElement == nullptr) {
00085                 std::print("}}");
00086                 return;
00087             }
00088 
00089             std::print("{}, ", currentElement->contents);
00090 
00091             currentElement = currentElement->previous;
00092         }
00093     }
00094 
00096     void append(const T& item) {
00097         if(head == nullptr) {
00098             head = new Node { nullptr, nullptr, item };
00099             back = head;
00100             return;
00101         }
00102 
00103         back = new Node { back, nullptr, item };
00104         back->previous->next = back;
00105     }
00106 
00108     void prepend(const T& item) {
00109         if(head == nullptr) {
00110             head = new Node { nullptr, nullptr, item };
00111             back = head;
00112             return;
00113         }
00114 
00115         head = new Node { nullptr, head, item };
00116         head->next->previous = head;
00117     }
00118 
00120     void insert(const T& item, size_t index) {
00121         Node* atInsertion { get_node_at(index) };
00122         Node* beforeInsertion { atInsertion->previous };
00123 
00124         if(beforeInsertion == nullptr) {
00125             prepend(item);
00126             return;
00127         }
00128 
00129         beforeInsertion->next = atInsertion->previous = new Node(beforeInsertion, atInsertion, item);
00130     }
```

```
00131
00133     void pop(void) {
00134         Node* newBack { back->previous };
00135         delete back;
00136
00137         back = newBack;
00138     }
00139
00141     void rpop(void) {
00142         Node* newHead { head->next };
00143         head->next = nullptr;
00144         delete head;
00145
00146         head = newHead;
00147         head->previous = nullptr;
00148     }
00149
00151     void pop_at(size_t index) {
00152         Node* toPop { get_node_at(index) };
00153         if(toPop == head) {
00154             rpop();
00155             return;
00156         } else if(toPop == back) {
00157             pop();
00158             return;
00159         }
00160
00161         toPop->previous->next = toPop->next;
00162         toPop->next->previous = toPop->previous;
00163         toPop->next = nullptr;
00164
00165         delete toPop;
00166     }
00167
00168     T& operator[](size_t index) {
00169         return get_node_at(index)->contents;
00170     }
00171
00172     void clear() {
00173         delete head;
00174         head = nullptr;
00175     }
00176 };
```

4.3 src/main.cpp File Reference

```
#include <print>
#include "doublelinkedlist.hpp"
```

Functions

- int main (int argc, char *argv[])

4.3.1 Function Documentation

4.3.1.1 main()

```
int main (
            int argc,
            char * argv[])
```

Index