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 tail 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)
  Returns the address of the contents at a specific index.
- void display_at (size_t index, size_t offset=0)
  Display an element at `index`, index can be offset by `offset`
- void clear ()
  Clear the memory allocated by the list.

3.1.1 Detailed Description

template<typename T>
class DoubleLinkedList< T >

Implementation of a double linked list using the heap.

As this is a template, the class can't be split into a header and implementation file.

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]
```

Clear the memory allocated by the list.

3.1.3.3 display()

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

Print the entire list.

3.1.3.4 display_at()

```
template<typename T >
void DoubleLinkedList< T >::display_at (
            size_t index,
            size_t offset = 0)  [inline]
```

Display an element at `index`, index can be offset by `offset`

3.1.3.5 insert()

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

Insert an item at a certain index.

3.1.3.6 operator[]()

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

Returns the address of the contents at a specific index.

3.1.3.7 pop()

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

Remove an element from the tail of the list.

3.1.3.8 pop_at()

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

Remove an element at a given index.

3.1.3.9 prepend()

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

Prepend an item to the beggining of the list.

3.1.3.10 rdisplay()

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

Print the entire list in reverse order.

3.1.3.11 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* tail;
00029
00030     Node* get_node_at(size_t index) {
00031         Node* currentNode { head };
00032         for(size_t i {}; i < index; i++) {
00033             currentNode = currentNode->next;
00034         }
00035
00036         return currentNode;
00037     }
```

```
00038
00039 public:
00041     DoubleLinkedList(void): head { nullptr }, tail { nullptr } {}
00042     ~DoubleLinkedList(void) {
00043         delete head;
00044     }
00045
00047     void display(void) {
00048         Node* currentElement { head };
00049         std::print("{{");
00050
00051         while(true) {
00052             if(currentElement == nullptr) {
00053                 std::print("}}");
00054                 return;
00055             }
00056
00057             std::print("{}, ", currentElement->contents);
00058
00059             currentElement = currentElement->next;
00060         }
00061     }
00062
00064     void rdisplay(void) {
00065         Node* currentElement { tail };
00066         std::print("{{");
00067
00068         while(true) {
00069             if(currentElement == nullptr) {
00070                 std::print("}}");
00071                 return;
00072             }
00073
00074             std::print("{}, ", currentElement->contents);
00075
00076             currentElement = currentElement->previous;
00077         }
00078     }
00079
00081     void append(const T& item) {
00082         if(head == nullptr) {
00083             head = new Node { nullptr, nullptr, item };
00084             tail = head;
00085             return;
00086         }
00087
00088         tail = new Node { tail, nullptr, item };
00089         tail->previous->next = tail;
00090     }
00091
00093     void prepend(const T& item) {
00094         if(head == nullptr) {
00095             head = new Node { nullptr, nullptr, item };
00096             tail = head;
00097             return;
00098         }
00099
00100         head = new Node { nullptr, head, item };
00101         head->next->previous = head;
00102     }
00103
00105     void insert(const T& item, size_t index) {
00106         Node* atInsertion { get_node_at(index) };
00107         Node* beforeInsertion { atInsertion->previous };
00108
00109         if(beforeInsertion == nullptr) {
00110             prepend(item);
00111             return;
00112         }
00113
00114         beforeInsertion->next = atInsertion->previous = new Node(beforeInsertion, atInsertion, item);
00115     }
00116
00118     void pop(void) {
00119         Node* newTail { tail->previous };
00120         delete tail;
00121
00122         tail = newTail;
00123     }
00124
00126     void rpop(void) {
00127         Node* newHead { head->next };
00128         head->next = nullptr;
00129         delete head;
00130
00131         head = newHead;
00132         head->previous = nullptr;
```

```
00133     }
00134
00136     void pop_at(size_t index) {
00137         Node* toPop { get_node_at(index) };
00138         if(toPop == head) {
00139             rpop();
00140             return;
00141         } else if(toPop == tail) {
00142             pop();
00143             return;
00144         }
00145
00146         toPop->previous->next = toPop->next;
00147         toPop->next->previous = toPop->previous;
00148         toPop->next = nullptr;
00149
00150         delete toPop;
00151     }
00152
00154     T& operator[](size_t index) {
00155         return get_node_at(index)->contents;
00156     }
00157
00159     void display_at(size_t index, size_t offset = 0) {
00160         std::print("{}", get_node_at(index + offset)->contents);
00161     }
00162
00164     void clear() {
00165         delete head;
00166         head = nullptr;
00167         tail = nullptr;
00168     }
00169 };
```

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