Projektowanie-zaawansowane

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 DoublyLinkedList Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	5
3.1.2.1 DoublyLinkedList()	5
$3.1.2.2 \sim DoublyLinkedList()$	6
3.1.3 Member Function Documentation	6
3.1.3.1 deleteAtIndex()	6
3.1.3.2 deleteFromBeginning()	6
3.1.3.3 deleteFromEnd()	7
3.1.3.4 deleteList()	7
3.1.3.5 displayBackward()	7
3.1.3.6 displayForward()	7
3.1.3.7 displayNext()	8
3.1.3.8 displayPrevious()	8
3.1.3.9 getNodeAtIndex()	8
3.1.3.10 insertAtBeginning()	8
3.1.3.11 insertAtEnd()	9
3.1.3.12 insertAtIndex()	9
3.2 Node Struct Reference	9
3.2.1 Detailed Description	10
3.2.2 Constructor & Destructor Documentation	10
3.2.2.1 Node()	10
	10
3.2.3.1 data	10
	10
	10
4 File Documentation	11
	11
	11
	11

Class Index

1.1 Class List

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

Doubly	'LinkedList		 																	5
Node			 												 					9

2 Class Index

File Index

2.1 File List

Here is a list of all	files	wi	th I	brie	ef d	esc	crip	tio	ns	:														
zadanie1.cpp											 													1

File Index

Class Documentation

3.1 DoublyLinkedList Class Reference

Public Member Functions

- DoublyLinkedList ()
- void insertAtBeginning (int value)
- void insertAtEnd (int value)
- void displayForward ()
- void deleteList ()
- void insertAtIndex (int value, int index)
- void deleteFromBeginning ()
- void deleteFromEnd ()
- void deleteAtIndex (int index)
- void displayBackward ()
- Node * getNodeAtIndex (int index)
- void displayNext (Node *node)
- void displayPrevious (Node *node)
- ∼DoublyLinkedList ()

3.1.1 Detailed Description

Definition at line 14 of file zadanie1.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DoublyLinkedList()

DoublyLinkedList::DoublyLinkedList () [inline]

Definition at line 21 of file zadanie1.cpp.

21 : head(nullptr), tail(nullptr) {}

6 **Class Documentation**

3.1.2.2 ∼DoublyLinkedList()

```
DoublyLinkedList::~DoublyLinkedList ( ) [inline]
Definition at line 191 of file zadanie1.cpp.
           deleteList(); // Usuwanie całej listy przy niszczeniu obiektu
```

3.1.3 Member Function Documentation

3.1.3.1 deleteAtIndex()

192 193

```
void DoublyLinkedList::deleteAtIndex (
            int index ) [inline]
```

Definition at line 126 of file zadanie1.cpp.

```
126
127
               if (index == 0) {
128
                    deleteFromBeginning();
129
130
131
              Node* temp = head;
for (int i = 0; i < index && temp != nullptr; ++i) {</pre>
132
133
134
                    temp = temp->next;
135
136
137
               if (temp == nullptr) return; // Indeks poza zakresem
138
139
               if (temp == tail) {
                    deleteFromEnd();
140
141
142
               else {
                    temp->prev->next = temp->next;
if (temp->next != nullptr) {
   temp->next->prev = temp->prev;
143
144
145
146
147
                    delete temp;
148
         }
149
```

3.1.3.2 deleteFromBeginning()

```
void DoublyLinkedList::deleteFromBeginning ( ) [inline]
```

Definition at line 98 of file zadanie1.cpp.

```
99
            if (head == nullptr) return; // Lista pusta
             Node* temp = head;
head = head->next;
100
101
102
             if (head != nullptr) {
                 head->prev = nullptr;
103
104
             else {
                tail = nullptr;
106
107
             delete temp;
108
109
```

3.1.3.3 deleteFromEnd()

```
void DoublyLinkedList::deleteFromEnd ( ) [inline]
```

Definition at line 112 of file zadanie1.cpp.

```
112
             if (tail == nullptr) return; // Lista pusta
113
             Node* temp = tail;
tail = tail->prev;
114
115
116
             if (tail != nullptr)
                  tail->next = nullptr;
117
118
119
             else {
120
                 head = nullptr;
122
             delete temp;
         }
123
```

3.1.3.4 deleteList()

```
void DoublyLinkedList::deleteList ( ) [inline]
```

Definition at line 60 of file zadanie1.cpp.

```
Node* current = head;
62
             while (current != nullptr) {
                  Node* next = current->next; // Zapisanie wskaźnika na następny węzel std::cout « "Usuwanie wezla o wartosci: " « current->data « std::endl;
63
64
                                                     // Zwalnianie obecnego węzła
                  delete current;
65
                                                     // Przechodzenie do następnego węzła
66
                  current = next;
68
             head = tail = nullptr;
                                                      // Resetowanie wskaźników po usunięciu listy
69
```

3.1.3.5 displayBackward()

void DoublyLinkedList::displayBackward () [inline]

Definition at line 152 of file zadanie1.cpp.

3.1.3.6 displayForward()

```
void DoublyLinkedList::displayForward ( ) [inline]
```

Definition at line 50 of file zadanie1.cpp.

8 Class Documentation

3.1.3.7 displayNext()

3.1.3.8 displayPrevious()

3.1.3.9 getNodeAtIndex()

3.1.3.10 insertAtBeginning()

```
void DoublyLinkedList::insertAtBeginning (
                int value ) [inline]
Definition at line 24 of file zadanie1.cpp.
            Node* newNode = new Node(value); // Tworzenie nowego węzła
25
                                                 // Jeśli lista jest pusta
26
            if (head == nullptr) {
                head = tail = newNode;
                                                 // Head i tail wskazują na nowy węzeł
28
            else {
29
                newNode->next = head;
                                                 // Nowy węzeł wskazuje na obecny head
30
                                                 // Obecny head wskazuje na nowy węzeł jako prev
// Aktualizacja head do nowego węzła
                head->prev = newNode;
head = newNode;
31
32
```

3.2 Node Struct Reference 9

3.1.3.11 insertAtEnd()

```
void DoublyLinkedList::insertAtEnd (
              int value ) [inline]
Definition at line 37 of file zadanie1.cpp.
          Node* newNode = new Node(value); // Tworzenie nowego węzła
38
          if (tail == nullptr) {
                                           // Jeśli lista jest pusta
39
               head = tail = newNode;
40
                                           // Head i tail wskazują na nowy węzeł
41
42
             tail->next = newNode;
                                           // Ostatni węzeł wskazuje na nowy węzeł
43
              newNode->prev = tail;
                                           // Nowy węzeł wskazuje na stary tail
44
              tail = newNode;
                                           // Aktualizacja tail do nowego węzła
46
47
```

3.1.3.12 insertAtIndex()

```
void DoublyLinkedList::insertAtIndex (
               int value,
                int index ) [inline]
Definition at line 72 of file zadanie1.cpp.
73
            if (index == 0) {
                insertAtBeginning(value);
75
                return;
76
77
78
           Node* newNode = new Node(value);
           Node* temp = head;
for (int i = 0; i < index - 1 && temp != nullptr; ++i) {</pre>
79
                temp = temp->next;
83
           if (temp == nullptr || temp == tail) {
   insertAtEnd(value);
84
85
86
            newNode->next = temp->next;
88
89
               newNode->prev = temp;
                if (temp->next != nullptr) {
90
91
                    temp->next->prev = newNode;
92
                temp->next = newNode;
94
95
```

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

· zadanie1.cpp

3.2 Node Struct Reference

Public Member Functions

• Node (int value)

10 Class Documentation

Public Attributes

- int data
- Node * prev
- Node * next

3.2.1 Detailed Description

Definition at line 4 of file zadanie1.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node()

3.2.3 Member Data Documentation

3.2.3.1 data

```
int Node::data
```

Definition at line 5 of file zadanie1.cpp.

3.2.3.2 next

```
Node* Node::next
```

Definition at line 7 of file zadanie1.cpp.

3.2.3.3 prev

```
Node* Node::prev
```

Definition at line 6 of file zadanie1.cpp.

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

zadanie1.cpp

File Documentation

4.1 zadanie1.cpp File Reference

```
#include <iostream>
```

Classes

- struct Node
- · class DoublyLinkedList

Functions

• int main ()

4.1.1 Function Documentation

4.1.1.1 main()

```
int main ( )
```

Definition at line 197 of file zadanie1.cpp.

```
DoublyLinkedList list;
199
          // Dodawanie elementów na początku listy
list.insertAtBeginning(33);
list.insertAtBeginning(27);
list.insertAtBeginning(14);
200
201
202
203
204
205
206
          // Dodawanie elementów na końcu listy
207
          list.insertAtEnd(99);
list.insertAtEnd(73);
209
210
          // Dodawanie elemetu pod wskazany indeks
           list.insertAtIndex(54, 2);
```

12 File Documentation

```
212
213
          // Wyświetlanie listy od początku
          std::cout « "Lista dwukierunkowa: ";
214
          list.displayForward();
215
216
          // Test funkcji displayNext oraz displayPrevious
217
218
          Node* nodeAtIndex1 = list.getNodeAtIndex(1); // Pobieramy wezeł o indeksie 1
219
          if (nodeAtIndex1 != nullptr) {
        std::cout « "Wywolanie funkcji displayNext dla wezla o indeksie 1 (wartosc: " « nodeAtIndex1->data « "):" « std::endl; list.displayNext(nodeAtIndex1); // Wyświetla następny element po węźle
220
221
222
223
               std::cout « "Wywolanie funkcji displayPrevious dla wezla o indeksie 1 (wartosc: " «
        nodeAtIndex1->data « "): " « std::endl;
224
              list.displayPrevious(nodeAtIndex1); // Wyświetla poprzedni element przed węzłem
225
226
          // Usuwanie elementu z początku listy std::cout « "Usuwanie elementu z poczatku" « std::endl;
227
228
229
          list.deleteFromBeginning();
230
          list.displayForward();
231
          // Usuwanie elementu z końca listy std::cout « "Usuwanie elementu z konca:" « std::endl;
2.32
233
234
          list.deleteFromEnd();
235
          list.displayForward();
236
          // Usuwanie elementu z danego indexu
std::cout « "Usuwanie elementu z danego indeksu:" « std::endl;
237
238
          list.deleteAtIndex(2);
239
240
          list.displayForward();
241
242
          // Wyświetlanie listy od końca
          std::cout « "Wyswietlanie listy od komca:" « std::endl;
list.displayBackward();
243
244
245
          // Usuwanie całej listy
std::cout « "Usuwanie calej listy:" « std::endl;
246
247
248
          list.deleteList();
249
250
          return 0;
251 }
```