Projektowanie-zaawansowane

Generated by Doxygen 1.9.1

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 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) {}
```

3.1.2.2 ~DoublyLinkedList()

```
DoublyLinkedList::~DoublyLinkedList ( )  [inline]
```

Definition at line 191 of file zadanie1.cpp.

```
191                                 {
192         deleteList(); // Usuwanie całej listy przy niszczeniu obiektu
193     }
```

3.1.3 Member Function Documentation

3.1.3.1 deleteAtIndex()

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

Definition at line 126 of file zadanie1.cpp.

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

3.1.3.2 deleteFromBeginning()

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

Definition at line 98 of file zadanie1.cpp.

```
98                                      {
99         if (head == nullptr) return; // Lista pusta
100         Node* temp = head;
101         head = head->next;
102         if (head != nullptr) {
103             head->prev = nullptr;
104         }
105         else {
106             tail = nullptr;
107         }
108         delete temp;
109     }
```

3.1.3.3 deleteFromEnd()

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

Definition at line 112 of file zadanie1.cpp.

```
112                             {
113         if (tail == nullptr) return; // Lista pusta
114         Node* temp = tail;
115         tail = tail->prev;
116         if (tail != nullptr) {
117             tail->next = nullptr;
118         }
119         else {
120             head = nullptr;
121         }
122         delete temp;
123     }
```

3.1.3.4 deleteList()

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

Definition at line 60 of file zadanie1.cpp.

```
60                          {
61         Node* current = head;
62         while (current != nullptr) {
63             Node* next = current->next; // Zapisanie wskaźnika na następny węzeł
64             std::cout « "Usuwanie wezla o wartosci: " « current->data « std::endl;
65             delete current;             // Zwalnianie obecnego węzła
66             current = next;             // Przechodzenie do następnego węzła
67         }
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.

```
152                              {
153         Node* temp = tail;
154         while (temp != nullptr) {
155             std::cout « temp->data « " -> ";
156             temp = temp->prev;
157         }
158         std::cout « "nullptr" « std::endl;
159     }
```

3.1.3.6 displayForward()

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

Definition at line 50 of file zadanie1.cpp.

```
50                             {
51         Node* temp = head;
52         while (temp != nullptr) {
53             std::cout « temp->data « " -> ";
54             temp = temp->next;
55         }
56         std::cout « "nullptr" « std::endl;
57     }
```

3.1.3.7 displayNext()

```
void DoublyLinkedList::displayNext (
            Node * node )  [inline]
```

Definition at line 171 of file zadanie1.cpp.

```
171                                     {
172         if (node->next != nullptr) {
173             std::cout « "Nastepny element: " « node->next->data « std::endl;
174         }
175         else {
176             std::cout « "Brak nastepnego elementu." « std::endl;
177         }
178     }
```

3.1.3.8 displayPrevious()

```
void DoublyLinkedList::displayPrevious (
            Node * node )  [inline]
```

Definition at line 181 of file zadanie1.cpp.

```
181                                         {
182         if (node->prev != nullptr) {
183             std::cout « "Poprzedni element: " « node->prev->data « std::endl;
184         }
185         else {
186             std::cout « "Brak poprzedniego elementu." « std::endl;
187         }
188     }
```

3.1.3.9 getNodeAtIndex()

```
Node* DoublyLinkedList::getNodeAtIndex (
            int index )  [inline]
```

Definition at line 162 of file zadanie1.cpp.

```
162                                       {
163         Node* temp = head;
164         for (int i = 0; i < index && temp != nullptr; ++i) {
165             temp = temp->next;
166         }
167         return temp; // Zwraca wskaźnik na węzeł lub nullptr, jeśli indeks poza zakresem
168     }
```

3.1.3.10 insertAtBeginning()

```
void DoublyLinkedList::insertAtBeginning (
            int value )  [inline]
```

Definition at line 24 of file zadanie1.cpp.

```
24                                        {
25         Node* newNode = new Node(value); // Tworzenie nowego węzła
26         if (head == nullptr) {           // Jeśli lista jest pusta
27             head = tail = newNode;       // Head i tail wskazują na nowy węzeł
28         }
29         else {
30             newNode->next = head;        // Nowy węzeł wskazuje na obecny head
31             head->prev = newNode;        // Obecny head wskazuje na nowy węzeł jako prev
32             head = newNode;              // Aktualizacja head do nowego węzła
33         }
34     }
```

3.1.3.11 insertAtEnd()

```
void DoublyLinkedList::insertAtEnd (
            int value )  [inline]
```

Definition at line 37 of file zadanie1.cpp.

```
37                                  {
38         Node* newNode = new Node(value); // Tworzenie nowego węzła
39         if (tail == nullptr) {           // Jeśli lista jest pusta
40             head = tail = newNode;       // Head i tail wskazują na nowy węzeł
41         }
42         else {
43             tail->next = newNode;        // Ostatni węzeł wskazuje na nowy węzeł
44             newNode->prev = tail;        // Nowy węzeł wskazuje na stary tail
45             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.

```
72                                                  {
73         if (index == 0) {
74             insertAtBeginning(value);
75             return;
76         }
77
78         Node* newNode = new Node(value);
79         Node* temp = head;
80         for (int i = 0; i < index - 1 && temp != nullptr; ++i) {
81             temp = temp->next;
82         }
83
84         if (temp == nullptr || temp == tail) {
85             insertAtEnd(value);
86         }
87         else {
88             newNode->next = temp->next;
89             newNode->prev = temp;
90             if (temp->next != nullptr) {
91                 temp->next->prev = newNode;
92             }
93             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)

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()

```
Node::Node (
            int value )  [inline]
```

Definition at line 10 of file zadanie1.cpp.

```
10 : data(value), prev(nullptr), next(nullptr) {}
```

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

Chapter 4

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.

```
197           {
198     DoublyLinkedList list;
199
200     // Dodawanie elementów na początku listy
201     list.insertAtBeginning(33);
202     list.insertAtBeginning(27);
203     list.insertAtBeginning(14);
204
205
206     // Dodawanie elementów na końcu listy
207     list.insertAtEnd(99);
208     list.insertAtEnd(73);
209
210     // Dodawanie elemetu pod wskazany indeks
211     list.insertAtIndex(54, 2);
```

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