ProgramowanieZaawansowane

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 DoublyLinkedList Class Reference

Klasa reprezentuj¹ca dwukierunkow¹ listê wi¹zan¹.

`#include <DoublyLinkedList.hpp>`

Public Member Functions

- DoublyLinkedList ()
  Konstruktor domylny, inicjuje pust¹ listê.
- ~DoublyLinkedList ()
  Destruktor, który zwalnia zasoby klasy z pamiêci.
- void addToStart (int value)
  Dodanie elementu na pocz¹tek listy.
- void addToEnd (int value)
  Dodanie elementu na koniec listy.
- void addAtIndex (int index, int value)
  Dodanie elementu pod wskazany indeks.
- void removeFromStart ()
  Usuñ element z pocz¹tku listy.
- void removeFromEnd ()
  Usuñ element z koñca listy.
- void deleteAtIndex (int index)
  Usuñ element pod wskazanym indeksem.
- void read () const
  Wywietl ca³¹ listê od pocz¹tku do koñca.
- void readReverse () const
  Wywietl listê w odwrotnej kolejnoci.
- void printNextElement ()
  Wywietl nastêpny element.
- void printPrevElement ()
  Wywietl poprzedni element.
- void clearList ()
  Czyæ ca³¹ listê.

3.1.1 Detailed Description

Klasa reprezentuj¹ca dwukierunkow¹ listê wi¹zan¹.

Ta klasa przechowuje wêz^{3}y listy, umo¿liwia dodawanie, usuwanie elementów, oraz operacje takie jak wywietlanie listy.

Definition at line 19 of file DoublyLinkedList.hpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DoublyLinkedList()

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

Konstruktor domylny, inicjuje pust¹ listê.

Definition at line 30 of file DoublyLinkedList.hpp.
```
00030 : head(nullptr), tail(nullptr), current(nullptr) {}
```

3.1.2.2 ~DoublyLinkedList()

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

Destruktor, który zwalnia zasoby klasy z pamiêci.

Definition at line 34 of file DoublyLinkedList.hpp.
```
00034                           {
00035         head = nullptr;
00036         tail = nullptr;
00037         current = nullptr;
00038     }
```

3.1.3 Member Function Documentation

3.1.3.1 addAtIndex()

```
void DoublyLinkedList::addAtIndex (
            int index,
            int value)  [inline]
```

Dodanie elementu pod wskazany indeks.

Parameters

index	Indeks, pod który ma zostaæ dodany element.
value	Wartoæ elementu do dodania.

Definition at line 77 of file DoublyLinkedList.hpp.

```
00077                                                {
00078         if (index <= 0) {
00079             addToStart(value);
00080             return;
00081         }
00082
00083         Node* newNode = new Node(value);
00084         Node* temp = head;
00085         int currentIndex = 0;
00086
00087         while (temp && currentIndex < index) {
00088             temp = temp->next;
00089             currentIndex++;
00090         }
00091
00092         if (!temp) {
00093             addToEnd(value);
00094         }
00095         else {
00096             newNode->next = temp;
00097             newNode->prev = temp->prev;
00098             if (temp->prev) {
00099                 temp->prev->next = newNode;
00100             }
00101             temp->prev = newNode;
00102         }
00103     }
```

3.1.3.2 addToEnd()

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

Dodanie elementu na koniec listy.

Parameters

value	Wartoæ elementu do dodania na koniec listy.

Definition at line 60 of file DoublyLinkedList.hpp.

```
00060                                {
00061         Node* newNode = new Node(value);
00062         if (!tail) {
00063             head = tail = newNode;
00064         }
00065         else {
00066             tail->next = newNode;
00067             newNode->prev = tail;
00068             tail = newNode;
00069         }
00070     }
```

3.1.3.3 addToStart()

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

Dodanie elementu na pocz¹tek listy.

Parameters

value	Wartoæ elementu do dodania na pocz¹tek listy.

Definition at line 44 of file DoublyLinkedList.hpp.

```
00044                                {
00045         Node* newNode = new Node(value);
00046         if (!head) {
00047             head = tail = newNode;
00048         }
00049         else {
00050             newNode->next = head;
00051             head->prev = newNode;
00052             head = newNode;
00053         }
00054     }
```

3.1.3.4 clearList()

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

Czyæ ca[31] listê.

Definition at line 212 of file DoublyLinkedList.hpp.

```
00212                 {
00213         while (head) {
00214             removeFromStart();
00215         }
00216         current = nullptr;
00217     }
```

3.1.3.5 deleteAtIndex()

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

Usuñ element pod wskazanym indeksem.

Parameters

index	Indeks elementu do usuniêcia.

Definition at line 141 of file DoublyLinkedList.hpp.

```
00141                                    {
00142         if (index < 0 || !head) return;
00143
00144         if (index == 0) {
00145             removeFromStart();
00146             return;
00147         }
00148
00149         Node* temp = head;
00150         int currentIndex = 0;
00151
00152         while (temp && currentIndex < index) {
00153             temp = temp->next;
00154             currentIndex++;
00155         }
00156
00157         if (!temp) return;
00158
00159         if (temp->prev) temp->prev->next = temp->next;
00160         if (temp->next) temp->next->prev = temp->prev;
00161
00162         if (temp == tail) tail = temp->prev;
00163
00164         delete temp;
00165     }
```

3.1.3.6 printNextElement()

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

Wywietl nastêpny element.

Definition at line 191 of file DoublyLinkedList.hpp.

```
00191                               {
00192         if (!current) current = head;
00193         else if (current->next) current = current->next;
00194         else current = head; // Powrót do pocz¹tku, jeli koniec zosta³ osi¹gniêty.
00195 
00196         if (current) cout « "Nastepny element: " « current->data « endl;
00197     }
```

3.1.3.7 printPrevElement()

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

Wywietl poprzedni element.

Definition at line 201 of file DoublyLinkedList.hpp.

```
00201                               {
00202         if (!current) current = tail;
00203         else if (current->prev) current = current->prev;
00204         else current = tail; // Powrót do koñca, jeli pocz¹tek zosta³ osi¹gniêty.
00205 
00206         if (current) cout « "Poprzedni element: " « current->data « endl;
00207     }
```

3.1.3.8 read()

```
void DoublyLinkedList::read () const  [inline]
```

Wywietl ca³¹ listê od pocz¹tku do koñca.

Definition at line 169 of file DoublyLinkedList.hpp.

```
00169                         {
00170         Node* temp = head;
00171         while (temp) {
00172             cout « temp->data « " ";
00173             temp = temp->next;
00174         }
00175         cout « endl;
00176     }
```

3.1.3.9 readReverse()

```
void DoublyLinkedList::readReverse () const  [inline]
```

Wywietl listê w odwrotnej kolejnoci.

Definition at line 180 of file DoublyLinkedList.hpp.

```
00180                                {
00181         Node* temp = tail;
00182         while (temp) {
00183             cout « temp->data « " ";
00184             temp = temp->prev;
00185         }
00186         cout « endl;
00187     }
```

3.1.3.10 removeFromEnd()

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

Usuñ element z koñca listy.

Definition at line 123 of file DoublyLinkedList.hpp.

```
00123                           {
00124         if (!tail) return;
00125 
00126         Node* temp = tail;
00127         tail = tail->prev;
00128         if (tail) {
00129             tail->next = nullptr;
00130         }
00131         else {
00132             head = nullptr;
00133         }
00134         delete temp;
00135     }
```

3.1.3.11 removeFromStart()

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

Usuñ element z pocz¹tku listy.

Definition at line 107 of file DoublyLinkedList.hpp.

```
00107                             {
00108         if (!head) return;
00109 
00110         Node* temp = head;
00111         head = head->next;
00112         if (head) {
00113             head->prev = nullptr;
00114         }
00115         else {
00116             tail = nullptr;
00117         }
00118         delete temp;
00119     }
```

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

- DoublyLinkedList.hpp

3.2 Node Struct Reference

Struktura reprezentuj¹ca wêze^3 listy.

```
#include <Node.hpp>
```

Public Member Functions

- Node (int value)

  Konstruktor tworz¹cy nowy wêze^3 z dan¹ wartoci¹.

Public Attributes

- int data
- Node * next
- Node * prev

3.2.1 Detailed Description

Struktura reprezentuj¹ca wêze[3] listy.

Ka¿dy wêze[3] przechowuje dane (data) oraz wskaniki na poprzedni (prev) i nastêpny (next) element listy.

Definition at line 8 of file Node.hpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node()

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

Konstruktor tworz¹cy nowy wêze[3] z dan¹ wartoci¹.

Parameters

value	Wartoæ do przechowania w wêle.

Definition at line 17 of file Node.hpp.

```
00017 : data(value), next(nullptr), prev(nullptr) {}
```

3.2.3 Member Data Documentation

3.2.3.1 data

```
int Node::data
```

Definition at line 9 of file Node.hpp.

3.2.3.2 next

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

Definition at line 10 of file Node.hpp.

3.2.3.3 prev

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

Definition at line 11 of file Node.hpp.

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

- Node.hpp

Chapter 4

File Documentation

4.1 DoublyLinkedList.hpp File Reference

```
#include <iostream>
#include "Node.hpp"
```

Classes

- class DoublyLinkedList

  Klasa reprezentuj¹ca dwukierunkow¹ listê wi¹zan¹.

4.2 DoublyLinkedList.hpp

Go to the documentation of this file.

```
00001
00008 #include <iostream>
00009 #include "Node.hpp"
00010 using namespace std;
00011
00019 class DoublyLinkedList {
00020 private:
00021
00022     Node* head;
00023     Node* tail;
00024     Node* current; // wskanik do poruszania siê po licie
00025
00026 public:
00030     DoublyLinkedList() : head(nullptr), tail(nullptr), current(nullptr) {}
00034     ~DoublyLinkedList() {
00035         head = nullptr;
00036         tail = nullptr;
00037         current = nullptr;
00038     }
00044     void addToStart(int value) {
00045         Node* newNode = new Node(value);
00046         if (!head) {
00047             head = tail = newNode;
00048         }
00049         else {
00050             newNode->next = head;
00051             head->prev = newNode;
00052             head = newNode;
00053         }
00054     }
00060     void addToEnd(int value) {
00061         Node* newNode = new Node(value);
```

```
00062         if (!tail) {
00063             head = tail = newNode;
00064         }
00065         else {
00066             tail->next = newNode;
00067             newNode->prev = tail;
00068             tail = newNode;
00069         }
00070     }
00077     void addAtIndex(int index, int value) {
00078         if (index <= 0) {
00079             addToStart(value);
00080             return;
00081         }
00082 
00083         Node* newNode = new Node(value);
00084         Node* temp = head;
00085         int currentIndex = 0;
00086 
00087         while (temp && currentIndex < index) {
00088             temp = temp->next;
00089             currentIndex++;
00090         }
00091 
00092         if (!temp) {
00093             addToEnd(value);
00094         }
00095         else {
00096             newNode->next = temp;
00097             newNode->prev = temp->prev;
00098             if (temp->prev) {
00099                 temp->prev->next = newNode;
00100             }
00101             temp->prev = newNode;
00102         }
00103     }
00107     void removeFromStart() {
00108         if (!head) return;
00109 
00110         Node* temp = head;
00111         head = head->next;
00112         if (head) {
00113             head->prev = nullptr;
00114         }
00115         else {
00116             tail = nullptr;
00117         }
00118         delete temp;
00119     }
00123     void removeFromEnd() {
00124         if (!tail) return;
00125 
00126         Node* temp = tail;
00127         tail = tail->prev;
00128         if (tail) {
00129             tail->next = nullptr;
00130         }
00131         else {
00132             head = nullptr;
00133         }
00134         delete temp;
00135     }
00141     void deleteAtIndex(int index) {
00142         if (index < 0 || !head) return;
00143 
00144         if (index == 0) {
00145             removeFromStart();
00146             return;
00147         }
00148 
00149         Node* temp = head;
00150         int currentIndex = 0;
00151 
00152         while (temp && currentIndex < index) {
00153             temp = temp->next;
00154             currentIndex++;
00155         }
00156 
00157         if (!temp) return;
00158 
00159         if (temp->prev) temp->prev->next = temp->next;
00160         if (temp->next) temp->next->prev = temp->prev;
00161 
00162         if (temp == tail) tail = temp->prev;
00163 
00164         delete temp;
00165     }
```

```
00169     void read() const {
00170         Node* temp = head;
00171         while (temp) {
00172             cout « temp->data « " ";
00173             temp = temp->next;
00174         }
00175         cout « endl;
00176     }
00180     void readReverse() const {
00181         Node* temp = tail;
00182         while (temp) {
00183             cout « temp->data « " ";
00184             temp = temp->prev;
00185         }
00186         cout « endl;
00187     }
00191     void printNextElement() {
00192         if (!current) current = head;
00193         else if (current->next) current = current->next;
00194         else current = head; // Powrót do pocz¹tku, jeli koniec zosta³ osi¹gniêty.
00195 
00196         if (current) cout « "Nastepny element: " « current->data « endl;
00197     }
00201     void printPrevElement() {
00202         if (!current) current = tail;
00203         else if (current->prev) current = current->prev;
00204         else current = tail; // Powrót do koñca, jeli pocz¹tek zosta³ osi¹gniêty.
00205 
00206         if (current) cout « "Poprzedni element: " « current->data « endl;
00207     }
00208 
00212     void clearList() {
00213         while (head) {
00214             removeFromStart();
00215         }
00216         current = nullptr;
00217     }
00218 };
```

4.3 Node.hpp File Reference

Classes

- struct Node

  Struktura reprezentuj¹ca wêze³ listy.

4.4 Node.hpp

Go to the documentation of this file.

```
00001 
00008 struct Node {
00009     int data;
00010     Node* next;
00011     Node* prev;
00012 
00017     Node(int value) : data(value), next(nullptr), prev(nullptr) {}
00018 };
```

4.5 PZ1.cpp File Reference

```
#include <iostream>
#include "DoublyLinkedList.hpp"
```

Functions

- int main ()

4.5.1 Function Documentation

4.5.1.1 main()

```
int main ()
```

Definition at line 5 of file PZ1.cpp.

```
00005             {
00006         DoublyLinkedList list;
00007
00008         list.addToStart(10);
00009         list.addToEnd(20);
00010         list.addAtIndex(1, 15);
00011         list.read();
00012
00013         list.removeFromStart();
00014         list.read();
00015
00016         list.removeFromEnd();
00017         list.read();
00018
00019         list.addToEnd(30);
00020         list.addToEnd(40);
00021         list.deleteAtIndex(1);
00022         list.read();
00023
00024         list.readReverse();
00025
00026         list.printNextElement();
00027         list.printNextElement();
00028
00029         list.printPrevElement();
00030         list.printPrevElement();
00031
00032         list.clearList();
00033         list.read();
00034
00035         return 0;
00036 }
```

4.6 PZ1.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "DoublyLinkedList.hpp"
00003
00004
00005 int main() {
00006         DoublyLinkedList list;
00007
00008         list.addToStart(10);
00009         list.addToEnd(20);
00010         list.addAtIndex(1, 15);
00011         list.read();
00012
00013         list.removeFromStart();
00014         list.read();
00015
00016         list.removeFromEnd();
00017         list.read();
00018
00019         list.addToEnd(30);
00020         list.addToEnd(40);
00021         list.deleteAtIndex(1);
00022         list.read();
00023
00024         list.readReverse();
00025
00026         list.printNextElement();
```

```
00027         list.printNextElement();
00028
00029         list.printPrevElement();
00030         list.printPrevElement();
00031
00032         list.clearList();
00033         list.read();
00034
00035         return 0;
00036 }
00037
```

Index