Projekt Listy Dwukierunkowej

Generated by Doxygen 1.14.0

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< T > Class Template Reference . . . 5

3.1.1 Constructor & Destructor Documentation . . . 5

3.1.1.1 DoublyLinkedList() . . . 5

3.1.1.2 ~DoublyLinkedList() . . . 6

3.1.2 Member Function Documentation . . . 6

3.1.2.1 addAtIndex() . . . 6

3.1.2.2 addToBack() . . . 6

3.1.2.3 addToFront() . . . 6

3.1.2.4 begin() . . . 6

3.1.2.5 clear() . . . 6

3.1.2.6 display() . . . 6

3.1.2.7 displayReverse() . . . 7

3.1.2.8 end() . . . 7

3.1.2.9 removeAtIndex() . . . 7

3.1.2.10 removeFromBack() . . . 7

3.1.2.11 removeFromFront() . . . 7

3.1.3 Member Data Documentation . . . 7

3.1.3.1 head . . . 7

3.1.3.2 size . . . 7

3.1.3.3 tail . . . 7

3.2 Iterator< T > Class Template Reference . . . 8

3.2.1 Constructor & Destructor Documentation . . . 8

3.2.1.1 Iterator() . . . 8

3.2.2 Member Function Documentation . . . 8

3.2.2.1 currentItem() . . . 8

3.2.2.2 isDone() . . . 9

3.2.2.3 next() . . . 9

3.2.2.4 prev() . . . 9

3.2.3 Member Data Documentation . . . 9

3.2.3.1 currentNode . . . 9

3.3 ListFactory< T > Class Template Reference . . . 10

3.3.1 Member Function Documentation . . . 10

3.3.1.1 createList() . . . 10

3.4 Node< T > Class Template Reference . . . 10

3.4.1 Constructor & Destructor Documentation . . . 10

3.4.1.1 Node() . . . 10

3.4.2 Member Data Documentation . . . 11
3.4.2.1 data . . . 11
3.4.2.2 next . . . 11
3.4.2.3 prev . . . 11

4 File Documentation **13**
4.1 fabryka.cpp File Reference . . . 13
4.2 fabryka.cpp . . . 13
4.3 fabryka.h File Reference . . . 13
4.3.1 Detailed Description . . . 13
4.4 fabryka.h . . . 14
4.5 iterator.cpp File Reference . . . 14
4.6 iterator.cpp . . . 14
4.7 iterator.h File Reference . . . 14
4.7.1 Detailed Description . . . 15
4.8 iterator.h . . . 15
4.9 list.cpp File Reference . . . 15
4.10 list.cpp . . . 15
4.11 list.h File Reference . . . 17
4.11.1 Detailed Description . . . 17
4.12 list.h . . . 18
4.13 main.cpp File Reference . . . 18
4.13.1 Function Documentation . . . 18
4.13.1.1 main() . . . 18
4.13.1.2 testIterator() . . . 18
4.14 node.h File Reference . . . 19
4.14.1 Detailed Description . . . 19
4.15 node.h . . . 19

Index **21**

Chapter 1

Class Index

1.1 Class List

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

DoublyLinkedList< T > 5
Iterator< T > 8
ListFactory< T > 10
Node< T > 10

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

fabryka.cpp 13
fabryka.h
    Definicja klasy ListFactory, implementuj¹cej wzorzec fabryki 13
iterator.cpp 14
iterator.h
    Definicja klasy Iterator do przechodzenia po licie 14
list.cpp 15
list.h
    Definicja klasy DoublyLinkedList, implementuj¹cej listê dwukierunkow¹ 17
main.cpp 18
node.h
    Definicja klasy Node, reprezentuj¹cej pojedynczy wêze[3] listy dwukierunkowej 19

Chapter 3

Class Documentation

3.1 DoublyLinkedList< T > Class Template Reference

`#include <list.h>`

Public Member Functions

- DoublyLinkedList ()

  Konstruktor domylny.
- ~DoublyLinkedList ()

  Destruktor, czyci pamiêæ.
- void addToFront (T data)
- void addToBack (T data)
- void addAtIndex (T data, int index)
- void removeFromFront ()
- void removeFromBack ()
- void removeAtIndex (int index)
- void display ()
- void displayReverse ()
- void clear ()
- Iterator< T > begin ()
- Iterator< T > end ()

Private Attributes

- Node< T > * head
- Node< T > * tail
- int size

3.1.1 Constructor & Destructor Documentation

3.1.1.1 DoublyLinkedList()

```
template<typename T>
DoublyLinkedList< T >::DoublyLinkedList ()
```

Konstruktor domylny.

3.1.1.2 ~DoublyLinkedList()

```
template<typename T>
DoublyLinkedList< T >::~DoublyLinkedList ()
```

Destruktor, czyci pamiêæ.

3.1.2 Member Function Documentation

3.1.2.1 addAtIndex()

```
template<typename T>
void DoublyLinkedList< T >::addAtIndex (
            T data,
            int index)
```

3.1.2.2 addToBack()

```
template<typename T>
void DoublyLinkedList< T >::addToBack (
            T data)
```

3.1.2.3 addToFront()

```
template<typename T>
void DoublyLinkedList< T >::addToFront (
            T data)
```

3.1.2.4 begin()

```
template<typename T>
Iterator< T > DoublyLinkedList< T >::begin ()
```

3.1.2.5 clear()

```
template<typename T>
void DoublyLinkedList< T >::clear ()
```

3.1.2.6 display()

```
template<typename T>
void DoublyLinkedList< T >::display ()
```

3.1.2.7 displayReverse()

```
template<typename T>
void DoublyLinkedList< T >::displayReverse ()
```

3.1.2.8 end()

```
template<typename T>
Iterator< T > DoublyLinkedList< T >::end ()
```

3.1.2.9 removeAtIndex()

```
template<typename T>
void DoublyLinkedList< T >::removeAtIndex (
            int index)
```

3.1.2.10 removeFromBack()

```
template<typename T>
void DoublyLinkedList< T >::removeFromBack ()
```

3.1.2.11 removeFromFront()

```
template<typename T>
void DoublyLinkedList< T >::removeFromFront ()
```

3.1.3 Member Data Documentation

3.1.3.1 head

```
template<typename T>
Node<T>* DoublyLinkedList< T >::head  [private]
```

3.1.3.2 size

```
template<typename T>
int DoublyLinkedList< T >::size  [private]
```

3.1.3.3 tail

```
template<typename T>
Node<T>* DoublyLinkedList< T >::tail  [private]
```

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

- list.h
- list.cpp

3.2 Iterator< T > Class Template Reference

`#include <iterator.h>`

Public Member Functions

- Iterator (Node< T > *node)
  Konstruktor iteratora.
- Iterator< T > & next ()
  Przesuwa iterator do nastêpnego elementu.
- Iterator< T > & prev ()
  Przesuwa iterator do poprzedniego elementu.
- T currentItem ()
  Zwraca dane z aktualnego wêz[3]a.
- bool isDone ()
  Sprawdza, czy iterator zakoñczy[3] przechodzenie po licie.

Private Attributes

- Node< T > * currentNode

3.2.1 Constructor & Destructor Documentation

3.2.1.1 Iterator()

```
template<typename T>
Iterator< T >::Iterator (
            Node< T > * node)
```

Konstruktor iteratora.

Parameters

	node	Wêze[3], na który ma wskazywaæ iterator.

3.2.2 Member Function Documentation

3.2.2.1 currentItem()

```
template<typename T>
T Iterator< T >::currentItem ()
```

Zwraca dane z aktualnego wêz[3]a.

Returns

Dane typu T.

3.2.2.2 isDone()

```
template<typename T>
bool Iterator< T >::isDone ()
```

Sprawdza, czy iterator zakoñczy[3] przechodzenie po licie.

Returns

True, jeli iterator wskazuje na nullptr.

3.2.2.3 next()

```
template<typename T>
Iterator< T > & Iterator< T >::next ()
```

Przesuwa iterator do nastêpnego elementu.

Returns

Referencja do iteratora po przesuniêciu.

3.2.2.4 prev()

```
template<typename T>
Iterator< T > & Iterator< T >::prev ()
```

Przesuwa iterator do poprzedniego elementu.

cd ..

Returns

Referencja do iteratora po przesuniêciu.

3.2.3 Member Data Documentation

3.2.3.1 currentNode

```
template<typename T>
Node<T>* Iterator< T >::currentNode  [private]
```

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

- iterator.h
- iterator.cpp

3.3 ListFactory< T > Class Template Reference

`#include <fabryka.h>`

Static Public Member Functions

- static DoublyLinkedList< T > * createList ()

  Tworzy i zwraca now¹ instancjê listy dwukierunkowej.

3.3.1 Member Function Documentation

3.3.1.1 createList()

```
template<typename T>
DoublyLinkedList< T > * ListFactory< T >::createList ()  [static]
```

Tworzy i zwraca now¹ instancjê listy dwukierunkowej.

Returns

Wskanik na nowo utworzon¹ listê.

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

- fabryka.h
- fabryka.cpp

3.4 Node< T > Class Template Reference

`#include <node.h>`

Public Member Functions

- Node (T data)

  Konstruktor tworz¹cy nowy wêze³.

Public Attributes

- T data
- Node< T > * next
- Node< T > * prev

3.4.1 Constructor & Destructor Documentation

3.4.1.1 Node()

```
template<typename T>
Node< T >::Node (
            T data)  [inline]
```

Konstruktor tworz¹cy nowy wêze³.

Parameters

	data	Wartoæ do przechowania w wêle.

3.4.2 Member Data Documentation

3.4.2.1 data

```
template<typename T>
T Node< T >::data
```

3.4.2.2 next

```
template<typename T>
Node<T>* Node< T >::next
```

3.4.2.3 prev

```
template<typename T>
Node<T>* Node< T >::prev
```

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

- node.h

Chapter 4

File Documentation

4.1 fabryka.cpp File Reference

```
#include "fabryka.h"
```

4.2 fabryka.cpp

Go to the documentation of this file.

```
00001 // fabryka.cpp
00002 #include "fabryka.h"
00003
00004 template <typename T>
00005 DoublyLinkedList<T>* ListFactory<T>::createList() {
00006     return new DoublyLinkedList<T>();
00007 }
00008
00009 // Jawne utworzenie instancji szablonu
00010 template class ListFactory<int>;
00011 template class ListFactory<double>;
00012 template class ListFactory<char>;
```

4.3 fabryka.h File Reference

Definicja klasy ListFactory, implementuj¹cej wzorzec fabryki.

```
#include "list.h"
```

Classes

- class ListFactory< T >

4.3.1 Detailed Description

Definicja klasy ListFactory, implementuj¹cej wzorzec fabryki.

4.4 fabryka.h

Go to the documentation of this file.

```
00001 // Fabryka.h
00002 #pragma once
00003 #include "list.h"
00004
00009
00010 template <typename T>
00011 class ListFactory {
00012 public:
00017     static DoublyLinkedList<T>* createList();
00018 };
```

4.5 iterator.cpp File Reference

```
#include "iterator.h"
```

4.6 iterator.cpp

Go to the documentation of this file.

```
00001 // iterator.cpp
00002 #include "iterator.h"
00003
00004 template <typename T>
00005 Iterator<T>::Iterator(Node<T>* node) : currentNode(node) {}
00006
00007 template <typename T>
00008 Iterator<T>& Iterator<T>::next() {
00009     if (currentNode) {
00010         currentNode = currentNode->next;
00011     }
00012     return *this;
00013 }
00014
00015 template <typename T>
00016 Iterator<T>& Iterator<T>::prev() {
00017     if (currentNode) {
00018         currentNode = currentNode->prev;
00019     }
00020     return *this;
00021 }
00022
00023 template <typename T>
00024 T Iterator<T>::currentItem() {
00025     return currentNode->data;
00026 }
00027
00028 template <typename T>
00029 bool Iterator<T>::isDone() {
00030     return currentNode == nullptr;
00031 }
00032
00033 // Jawne utworzenie instancji szablonu
00034 template class Iterator<int>;
00035 template class Iterator<double>;
00036 template class Iterator<char>;
```

4.7 iterator.h File Reference

Definicja klasy Iterator do przechodzenia po licie.

```
#include "node.h"
```

Classes

- class Iterator< T >

4.7.1 Detailed Description

Definicja klasy Iterator do przechodzenia po licie.

4.8 iterator.h

Go to the documentation of this file.

```
00001 // Iterator.h
00002 #pragma once
00003 #include "node.h"
00004
00009
00010 template <typename T>
00011 class Iterator {
00012 private:
00013     Node<T>* currentNode; // Wskanik na aktualny wêze[3]
00014
00015 public:
00020     Iterator(Node<T>* node);
00021
00026     Iterator<T>& next();
00027
00032     Iterator<T>& prev();
00033
00038     T currentItem();
00039
00044     bool isDone();
00045 };
```

4.9 list.cpp File Reference

```
#include "list.h"
```

4.10 list.cpp

Go to the documentation of this file.

```
00001 // list.cpp
00002 #include "list.h"
00003
00004 template <typename T>
00005 DoublyLinkedList<T>::DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}
00006
00007 template <typename T>
00008 DoublyLinkedList<T>::~DoublyLinkedList() {
00009     clear();
00010 }
00011
00012 template <typename T>
00013 void DoublyLinkedList<T>::addToFront(T data) {
00014     Node<T>* newNode = new Node<T>(data);
00015     if (!head) {
00016         head = tail = newNode;
00017     }
00018     else {
00019         newNode->next = head;
00020         head->prev = newNode;
00021         head = newNode;
```

```
00022     }
00023     size++;
00024 }
00025
00026 template <typename T>
00027 void DoublyLinkedList<T>::addToBack(T data) {
00028     Node<T>* newNode = new Node<T>(data);
00029     if (!tail) {
00030         head = tail = newNode;
00031     }
00032     else {
00033         tail->next = newNode;
00034         newNode->prev = tail;
00035         tail = newNode;
00036     }
00037     size++;
00038 }
00039
00040 template <typename T>
00041 void DoublyLinkedList<T>::addAtIndex(T data, int index) {
00042     if (index < 0 || index > size) {
00043         std::cerr « "Nie ma takiego indexu." « std::endl;
00044         return;
00045     }
00046     if (index == 0) {
00047         addToFront(data);
00048         return;
00049     }
00050     if (index == size) {
00051         addToBack(data);
00052         return;
00053     }
00054
00055     Node<T>* current = head;
00056     for (int i = 0; i < index; ++i) {
00057         current = current->next;
00058     }
00059
00060     Node<T>* newNode = new Node<T>(data);
00061     newNode->next = current;
00062     newNode->prev = current->prev;
00063     current->prev->next = newNode;
00064     current->prev = newNode;
00065     size++;
00066 }
00067
00068 template <typename T>
00069 void DoublyLinkedList<T>::removeFromFront() {
00070     if (!head) return;
00071     Node<T>* temp = head;
00072     if (head == tail) {
00073         head = tail = nullptr;
00074     }
00075     else {
00076         head = head->next;
00077         head->prev = nullptr;
00078     }
00079     delete temp;
00080     size--;
00081 }
00082
00083 template <typename T>
00084 void DoublyLinkedList<T>::removeFromBack() {
00085     if (!tail) return;
00086     Node<T>* temp = tail;
00087     if (head == tail) {
00088         head = tail = nullptr;
00089     }
00090     else {
00091         tail = tail->prev;
00092         tail->next = nullptr;
00093     }
00094     delete temp;
00095     size--;
00096 }
00097
00098 template <typename T>
00099 void DoublyLinkedList<T>::removeAtIndex(int index) {
00100     if (index < 0 || index >= size) {
00101         std::cerr « "Nie ma takiego indexu." « std::endl;
00102         return;
00103     }
00104     if (index == 0) {
00105         removeFromFront();
00106         return;
00107     }
00108     if (index == size - 1) {
```

```
00109         removeFromBack();
00110         return;
00111     }
00112
00113     Node<T>* current = head;
00114     for (int i = 0; i < index; ++i) {
00115         current = current->next;
00116     }
00117
00118     current->prev->next = current->next;
00119     current->next->prev = current->prev;
00120     delete current;
00121     size--;
00122 }
00123
00124 template <typename T>
00125 void DoublyLinkedList<T>::display() {
00126     Node<T>* current = head;
00127     while (current) {
00128         std::cout << current->data << " <-> ";
00129         current = current->next;
00130     }
00131     std::cout << "nullptr" << std::endl;
00132 }
00133
00134 template <typename T>
00135 void DoublyLinkedList<T>::displayReverse() {
00136     Node<T>* current = tail;
00137     while (current) {
00138         std::cout << current->data << " <-> ";
00139         current = current->prev;
00140     }
00141     std::cout << "nullptr" << std::endl;
00142 }
00143
00144 template <typename T>
00145 void DoublyLinkedList<T>::clear() {
00146     while (head) {
00147         removeFromFront();
00148     }
00149 }
00150
00151 template <typename T>
00152 Iterator<T> DoublyLinkedList<T>::begin() {
00153     return Iterator<T>(head);
00154 }
00155
00156 template <typename T>
00157 Iterator<T> DoublyLinkedList<T>::end() {
00158     return Iterator<T>(tail);
00159 }
00160
00161 // Jawne utworzenie instancji szablonu
00162 template class DoublyLinkedList<int>;
00163 template class DoublyLinkedList<double>;
00164 template class DoublyLinkedList<char>;
```

4.11 list.h File Reference

Definicja klasy DoublyLinkedList, implementuj¹cej listê dwukierunkow¹.

```
#include "node.h"
#include "iterator.h"
#include <iostream>
```

Classes

- class DoublyLinkedList< T >

4.11.1 Detailed Description

Definicja klasy DoublyLinkedList, implementuj¹cej listê dwukierunkow¹.

4.12 list.h

Go to the documentation of this file.

```
00001 // list.h
00002 #pragma once
00003 #include "node.h"
00004 #include "iterator.h"
00005 #include <iostream>
00006
00011
00012 template <typename T>
00013 class DoublyLinkedList {
00014 private:
00015     Node<T>* head; // Wskanik na pocz¹tek listy
00016     Node<T>* tail; // Wskanik na koniec listy
00017     int size;      // Rozmiar listy
00018
00019 public:
00023     DoublyLinkedList();
00024
00028     ~DoublyLinkedList();
00029
00030     // --- Metody z zadania ---
00031     void addToFront(T data);
00032     void addToBack(T data);
00033     void addAtIndex(T data, int index);
00034     void removeFromFront();
00035     void removeFromBack();
00036     void removeAtIndex(int index);
00037     void display();
00038     void displayReverse();
00039     void clear();
00040
00041     // --- Metody dla iteratora ---
00042     Iterator<T> begin();
00043     Iterator<T> end();
00044 };
```

4.13 main.cpp File Reference

```
#include <iostream>
#include "fabryka.h"
#include "list.cpp"
#include "iterator.cpp"
#include "fabryka.cpp"
```

Functions

- void testIterator (DoublyLinkedList< int > &list)
- int main ()

4.13.1 Function Documentation

4.13.1.1 main()

```
int main ()
```

4.13.1.2 testIterator()

```
void testIterator (
            DoublyLinkedList< int > & list)
```

4.14 node.h File Reference

Definicja klasy Node, reprezentuj¹cej pojedynczy wêze^3 listy dwukierunkowej.

Classes

- class Node< T >

4.14.1 Detailed Description

Definicja klasy Node, reprezentuj¹cej pojedynczy wêze^3 listy dwukierunkowej.

4.15 node.h

Go to the documentation of this file.

```
00001 // Node.h
00002 #pragma once // Zabezpiecza przed wielokrotnym do$^3$¹czaniem tego samego pliku
00003
00008
00009 template <typename T>
00010 class Node {
00011 public:
00012     T data;         // Dane przechowywane w wêle
00013     Node<T>* next;  // Wskanik na nastêpny wêze$^3$
00014     Node<T>* prev;  // Wskanik na poprzedni wêze$^3$
00015
00020     Node(T data) : data(data), next(nullptr), prev(nullptr) {}
00021 };
```

Index

~DoublyLinkedList
    DoublyLinkedList< T >, 5

addAtIndex
    DoublyLinkedList< T >, 6
addToBack
    DoublyLinkedList< T >, 6
addToFront
    DoublyLinkedList< T >, 6

begin
    DoublyLinkedList< T >, 6

clear
    DoublyLinkedList< T >, 6
createList
    ListFactory< T >, 10
currentItem
    Iterator< T >, 8
currentNode
    Iterator< T >, 9

data
    Node< T >, 11
display
    DoublyLinkedList< T >, 6
displayReverse
    DoublyLinkedList< T >, 6
DoublyLinkedList
    DoublyLinkedList< T >, 5
DoublyLinkedList< T >, 5
    ~DoublyLinkedList, 5
    addAtIndex, 6
    addToBack, 6
    addToFront, 6
    begin, 6
    clear, 6
    display, 6
    displayReverse, 6
    DoublyLinkedList, 5
    end, 7
    head, 7
    removeAtIndex, 7
    removeFromBack, 7
    removeFromFront, 7
    size, 7
    tail, 7

end
    DoublyLinkedList< T >, 7

fabryka.cpp, 13
fabryka.h, 13

head
    DoublyLinkedList< T >, 7

isDone
    Iterator< T >, 8
Iterator
    Iterator< T >, 8
Iterator< T >, 8
    currentItem, 8
    currentNode, 9
    isDone, 8
    Iterator, 8
    next, 9
    prev, 9
iterator.cpp, 14
iterator.h, 14

list.cpp, 15
list.h, 17
ListFactory< T >, 10
    createList, 10

main
    main.cpp, 18
main.cpp, 18
    main, 18
    testIterator, 18

next
    Iterator< T >, 9
    Node< T >, 11
Node
    Node< T >, 10
Node< T >, 10
    data, 11
    next, 11
    Node, 10
    prev, 11
node.h, 19

prev
    Iterator< T >, 9
    Node< T >, 11

removeAtIndex
    DoublyLinkedList< T >, 7
removeFromBack
    DoublyLinkedList< T >, 7

removeFromFront
DoublyLinkedList< T >, 7

size
DoublyLinkedList< T >, 7

tail
DoublyLinkedList< T >, 7
testIterator
main.cpp, 18