Programowanie zaawansowane

Generated by Doxygen 1.12.0

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 Class Reference . . . 5

3.1.1 Detailed Description . . . 5

3.1.2 Constructor & Destructor Documentation . . . 6

3.1.2.1 DoublyLinkedList() [1/3] . . . 6

3.1.2.2 ∼DoublyLinkedList() [1/3] . . . 6

3.1.2.3 DoublyLinkedList() [2/3] . . . 6

3.1.2.4 ∼DoublyLinkedList() [2/3] . . . 6

3.1.2.5 DoublyLinkedList() [3/3] . . . 6

3.1.2.6 ∼DoublyLinkedList() [3/3] . . . 6

3.1.3 Member Function Documentation . . . 7

3.1.3.1 addAtIndex() . . . 7

3.1.3.2 addBack() [1/3] . . . 7

3.1.3.3 addBack() [2/3] . . . 7

3.1.3.4 addBack() [3/3] . . . 8

3.1.3.5 addFront() [1/3] . . . 8

3.1.3.6 addFront() [2/3] . . . 8

3.1.3.7 addFront() [3/3] . . . 8

3.1.3.8 clear() [1/3] . . . 9

3.1.3.9 clear() [2/3] . . . 9

3.1.3.10 clear() [3/3] . . . 9

3.1.3.11 display() [1/3] . . . 9

3.1.3.12 display() [2/3] . . . 9

3.1.3.13 display() [3/3] . . . 10

3.1.3.14 displayReverse() [1/3] . . . 10

3.1.3.15 displayReverse() [2/3] . . . 10

3.1.3.16 displayReverse() [3/3] . . . 10

3.1.3.17 removeBack() [1/3] . . . 11

3.1.3.18 removeBack() [2/3] . . . 11

3.1.3.19 removeBack() [3/3] . . . 11

3.1.3.20 removeFromTail() . . . 12

3.1.3.21 removeFront() [1/3] . . . 12

3.1.3.22 removeFront() [2/3] . . . 12

3.1.3.23 removeFront() [3/3] . . . 13

3.2 Node Struct Reference . . . 13

3.2.1 Detailed Description . . . 13

3.2.2 Constructor & Destructor Documentation . . . 13

3.2.2.1 Node() [1/3] 13
3.2.2.2 Node() [2/3] 14
3.2.2.3 Node() [3/3] 14
3.2.3 Member Data Documentation 14
3.2.3.1 data 14
3.2.3.2 next 14
3.2.3.3 prev 14

4 File Documentation **15**
4.1 ConsoleApplication24.cpp File Reference 15
4.1.1 Function Documentation 15
4.1.1.1 main() 15
4.2 ConsoleApplication24.cpp 16
4.3 projekt 1.cpp File Reference 17
4.3.1 Function Documentation 18
4.3.1.1 main() 18
4.4 projekt 1.cpp 18
4.5 projekt.cpp File Reference 20
4.5.1 Function Documentation 20
4.5.1.1 main() 20
4.6 projekt.cpp 21

Index **23**

Chapter 1

Class Index

1.1 Class List

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

DoublyLinkedList 5
Node 13

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

ConsoleApplication24.cpp 15
projekt 1.cpp 17
projekt.cpp 20

Chapter 3

Class Documentation

3.1 DoublyLinkedList Class Reference

Public Member Functions

- DoublyLinkedList ()
- void addFront (int value)
- void addBack (int value)
- void removeFront ()
- void removeBack ()
- void display ()
- void displayReverse ()
- void clear ()
- ∼DoublyLinkedList ()
- DoublyLinkedList ()
- void addFront (int value)
- void addBack (int value)
- void addAtIndex (int index, int value)
- void removeFront ()
- void removeBack ()
- void removeFromTail ()
- void display ()
- void displayReverse ()
- void clear ()
- ∼DoublyLinkedList ()
- DoublyLinkedList ()
- void addFront (int value)
- void addBack (int value)
- void removeFront ()
- void removeBack ()
- void display ()
- void displayReverse ()
- void clear ()
- ∼DoublyLinkedList ()

3.1.1 Detailed Description

Definition at line 11 of file ConsoleApplication24.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DoublyLinkedList() [1/3]

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

Definition at line 17 of file ConsoleApplication24.cpp.

```
00017 : head(nullptr), tail(nullptr) {}
```

3.1.2.2 ~DoublyLinkedList() [1/3]

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

Definition at line 115 of file ConsoleApplication24.cpp.

```
00115                         {
00116         clear();
00117     }
```

3.1.2.3 DoublyLinkedList() [2/3]

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

Definition at line 17 of file projekt 1.cpp.

```
00017 : head(nullptr), tail(nullptr) {}
```

3.1.2.4 ~DoublyLinkedList() [2/3]

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

Definition at line 150 of file projekt 1.cpp.

```
00150                         {
00151         clear();
00152     }
```

3.1.2.5 DoublyLinkedList() [3/3]

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

Definition at line 18 of file projekt.cpp.

```
00018 : head(nullptr), tail(nullptr) {}
```

3.1.2.6 ~DoublyLinkedList() [3/3]

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

Definition at line 116 of file projekt.cpp.

```
00116                         {
00117         clear();
00118     }
```

3.1.3 Member Function Documentation

3.1.3.1 addAtIndex()

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

Definition at line 45 of file projekt 1.cpp.

```
                                      {
    if (index == 0) {
        addAtHead(value);
        return;
    }

    Node* newNode = new Node(value);
    Node* temp = head;
    for (int i = 0; i < index - 1 && temp != nullptr; i++) {
        temp = temp->next;
    }

    if (temp == nullptr || temp->next == nullptr) {
        addAtTail(value);
    }
    else {
        newNode->next = temp->next;
        newNode->prev = temp;
        temp->next->prev = newNode;
        temp->next = newNode;
    }
}
```

3.1.3.2 addBack() [1/3]

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

Definition at line 33 of file ConsoleApplication24.cpp.

```
                                  {
        Node* newNode = new Node(value);
        if (tail == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }
```

3.1.3.3 addBack() [2/3]

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

Definition at line 33 of file projekt 1.cpp.

```
                                  {
        Node* newNode = new Node(value);
        if (tail == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }
```

3.1.3.4 addBack() [3/3]

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

Definition at line 34 of file projekt.cpp.

```
                              {
        Node* newNode = new Node(value);
        if (tail == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }
```

3.1.3.5 addFront() [1/3]

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

Definition at line 20 of file ConsoleApplication24.cpp.

```
                               {
        Node* newNode = new Node(value);
        if (head == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
```

3.1.3.6 addFront() [2/3]

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

Definition at line 20 of file projekt 1.cpp.

```
                               {
        Node* newNode = new Node(value);
        if (head == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
```

3.1.3.7 addFront() [3/3]

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

Definition at line 21 of file projekt.cpp.

```
                               {
        Node* newNode = new Node(value);
        if (head == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
```

3.1.3.8 clear() [1/3]

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

Definition at line 108 of file ConsoleApplication24.cpp.

```
00108             {
00109         while (head != nullptr) {
00110             removeFront();
00111         }
00112     }
```

3.1.3.9 clear() [2/3]

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

Definition at line 143 of file projekt 1.cpp.

```
00143             {
00144         while (head != nullptr) {
00145             removeFront();
00146         }
00147     }
```

3.1.3.10 clear() [3/3]

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

Definition at line 109 of file projekt.cpp.

```
00109             {
00110         while (head != nullptr) {
00111             removeFront();
00112         }
00113     }
```

3.1.3.11 display() [1/3]

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

Definition at line 80 of file ConsoleApplication24.cpp.

```
00080               {
00081         Node* current = head;
00082         if (current == nullptr) {
00083             std::cout « "Lista jest pusta.\n";
00084             return;
00085         }
00086         while (current != nullptr) {
00087             std::cout « current->data « " ";
00088             current = current->next;
00089         }
00090         std::cout « "\n";
00091     }
```

3.1.3.12 display() [2/3]

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

Definition at line 115 of file projekt 1.cpp.

```
00115               {
00116         Node* current = head;
00117         if (current == nullptr) {
00118             std::cout « "Lista jest pusta.\n";
00119             return;
00120         }
00121         while (current != nullptr) {
00122             std::cout « current->data « " ";
00123             current = current->next;
00124         }
00125         std::cout « "\n";
00126     }
```

3.1.3.13 display() [3/3]

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

Definition at line 81 of file projekt.cpp.

```
                 {
        Node* current = head;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->next;
        }
        std::cout « "\n";
    }
```

3.1.3.14 displayReverse() [1/3]

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

Definition at line 94 of file ConsoleApplication24.cpp.

```
                        {
        Node* current = tail;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->prev;
        }
        std::cout « "\n";
    }
```

3.1.3.15 displayReverse() [2/3]

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

Definition at line 129 of file projekt 1.cpp.

```
                        {
        Node* current = tail;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->prev;
        }
        std::cout « "\n";
    }
```

3.1.3.16 displayReverse() [3/3]

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

Definition at line 95 of file projekt.cpp.

```
                        {
        Node* current = tail;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->prev;
        }
        std::cout « "\n";
    }
```

3.1.3.17 removeBack() [1/3]

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

Definition at line 63 of file ConsoleApplication24.cpp.

```
00063                       {
00064         if (tail == nullptr) {
00065             std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
00066             return;
00067         }
00068         Node* temp = tail;
00069         if (head == tail) { // Gdy w liście jest tylko jeden element
00070             head = tail = nullptr;
00071         }
00072         else {
00073             tail = tail->prev;
00074             tail->next = nullptr;
00075         }
00076         delete temp;
00077     }
```

3.1.3.18 removeBack() [2/3]

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

Definition at line 85 of file projekt 1.cpp.

```
00085                       {
00086         if (tail == nullptr) {
00087             std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
00088             return;
00089         }
00090         Node* temp = tail;
00091         if (head == tail) { // Gdy w liście jest tylko jeden element
00092             head = tail = nullptr;
00093         }
00094         else {
00095             tail = tail->prev;
00096             tail->next = nullptr;
00097         }
00098         delete temp;
00099     }
```

3.1.3.19 removeBack() [3/3]

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

Definition at line 64 of file projekt.cpp.

```
00064                       {
00065         if (tail == nullptr) {
00066             std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
00067             return;
00068         }
00069         Node* temp = tail;
00070         if (head == tail) { // Gdy w liście jest tylko jeden element
00071             head = tail = nullptr;
00072         }
00073         else {
00074             tail = tail->prev;
00075             tail->next = nullptr;
00076         }
00077         delete temp;
00078     }
```

3.1.3.20 removeFromTail()

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

Definition at line 101 of file projekt 1.cpp.

```
00101                       {
00102     if (tail == nullptr) return;
00103 
00104     Node* temp = tail;
00105     if (head == tail) {
00106         head = tail = nullptr;
00107     }
00108     else {
00109         tail = tail->prev;
00110         tail->next = nullptr;
00111     }
00112     delete temp;
00113 }
```

3.1.3.21 removeFront() [1/3]

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

Definition at line 46 of file ConsoleApplication24.cpp.

```
00046                         {
00047         if (head == nullptr) {
00048             std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
00049             return;
00050         }
00051         Node* temp = head;
00052         if (head == tail) { // Gdy w liście jest tylko jeden element
00053             head = tail = nullptr;
00054         }
00055         else {
00056             head = head->next;
00057             head->prev = nullptr;
00058         }
00059         delete temp;
00060     }
```

3.1.3.22 removeFront() [2/3]

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

Definition at line 68 of file projekt 1.cpp.

```
00068                         {
00069         if (head == nullptr) {
00070             std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
00071             return;
00072         }
00073         Node* temp = head;
00074         if (head == tail) { // Gdy w liście jest tylko jeden element
00075             head = tail = nullptr;
00076         }
00077         else {
00078             head = head->next;
00079             head->prev = nullptr;
00080         }
00081         delete temp;
00082     }
```

3.1.3.23 removeFront() [3/3]

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

Definition at line 47 of file projekt.cpp.

```
                             {
          if (head == nullptr) {
              std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
              return;
          }
          Node* temp = head;
          if (head == tail) { // Gdy w liście jest tylko jeden element
              head = tail = nullptr;
          }
          else {
              head = head->next;
              head->prev = nullptr;
          }
          delete temp;
      }
```

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

- ConsoleApplication24.cpp
- projekt 1.cpp
- projekt.cpp

3.2 Node Struct Reference

Public Member Functions

- Node (int value)
- Node (int value)
- Node (int value)

Public Attributes

- int data
- Node ∗ prev
- Node ∗ next

3.2.1 Detailed Description

Definition at line 3 of file ConsoleApplication24.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node() [1/3]

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

Definition at line 8 of file ConsoleApplication24.cpp.

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

3.2.2.2 Node() [2/3]

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

Definition at line 8 of file projekt 1.cpp.

```
00008 : data(value), prev(nullptr), next(nullptr) {}
```

3.2.2.3 Node() [3/3]

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

Definition at line 9 of file projekt.cpp.

```
00009 : data(value), prev(nullptr), next(nullptr) {}
```

3.2.3 Member Data Documentation

3.2.3.1 data

```
int Node::data
```

Definition at line 4 of file ConsoleApplication24.cpp.

3.2.3.2 next

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

Definition at line 6 of file ConsoleApplication24.cpp.

3.2.3.3 prev

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

Definition at line 5 of file ConsoleApplication24.cpp.

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

- ConsoleApplication24.cpp
- projekt 1.cpp
- projekt.cpp

Chapter 4

File Documentation

4.1 ConsoleApplication24.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 121 of file ConsoleApplication24.cpp.

```
00121            {
00122     DoublyLinkedList list;
00123 
00124     list.addFront(10);
00125     list.addFront(20);
00126     list.addBack(30);
00127     list.display();         // Powinno wyświetlić: 20 10 30
00128 
00129     list.removeFront();
00130     list.display();         // Powinno wyświetlić: 10 30
00131 
00132     list.removeBack();
00133     list.display();         // Powinno wyświetlić: 10
00134 
00135     list.clear();
00136     list.display();         // Powinno wyświetlić: Lista jest pusta.
00137 
00138     return 0;
00139 }
```

4.2 ConsoleApplication24.cpp

Go to the documentation of this file.

```
#include <iostream>

struct Node {
    int data;
    Node* prev;
    Node* next;

    Node(int value) : data(value), prev(nullptr), next(nullptr) {}
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Dodaj element na początek listy
    void addFront(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    // Dodaj element na koniec listy
    void addBack(int value) {
        Node* newNode = new Node(value);
        if (tail == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }

    // Usuń element z początku listy
    void removeFront() {
        if (head == nullptr) {
            std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
            return;
        }
        Node* temp = head;
        if (head == tail) { // Gdy w liście jest tylko jeden element
            head = tail = nullptr;
        }
        else {
            head = head->next;
            head->prev = nullptr;
        }
        delete temp;
    }

    // Usuń element z końca listy
    void removeBack() {
        if (tail == nullptr) {
            std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
            return;
        }
        Node* temp = tail;
        if (head == tail) { // Gdy w liście jest tylko jeden element
            head = tail = nullptr;
        }
        else {
            tail = tail->prev;
            tail->next = nullptr;
        }
        delete temp;
    }

    // Wyświetl listę od początku
    void display() {
        Node* current = head;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->next;
        }
        std::cout « "\n";
    }

    // Wyświetl listę w odwrotnej kolejności
    void displayReverse() {
        Node* current = tail;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->prev;
        }
        std::cout « "\n";
    }

    // Czyszczenie całej listy
    void clear() {
        while (head != nullptr) {
            removeFront();
        }
    }

    // Destruktor, aby zwolnić pamięć
    ~DoublyLinkedList() {
        clear();
    }
};

// Testowanie klasy w funkcji main
int main() {
    DoublyLinkedList list;

    list.addFront(10);
    list.addFront(20);
    list.addBack(30);
    list.display();         // Powinno wyświetlić: 20 10 30

    list.removeFront();
    list.display();         // Powinno wyświetlić: 10 30

    list.removeBack();
    list.display();         // Powinno wyświetlić: 10

    list.clear();
    list.display();         // Powinno wyświetlić: Lista jest pusta.

    return 0;
}
```

4.3 projekt 1.cpp File Reference

```
#include <iostream>
```

Classes

- struct Node
- class DoublyLinkedList

Functions

- int main ()

4.3.1 Function Documentation

4.3.1.1 main()

```
int main ()
```

Definition at line 156 of file projekt 1.cpp.

```
00156            {
00157     DoublyLinkedList list;
00158 
00159     list.addFront(10);
00160     list.addFront(20);
00161     list.addBack(30);
00162     list.display();         // Powinno wyświetlić: 20 10 30
00163 
00164     list.removeFront();
00165     list.display();         // Powinno wyświetlić: 10 30
00166 
00167     list.removeBack();
00168     list.display();         // Powinno wyświetlić: 10
00169 
00170     list.clear();
00171     list.display();         // Powinno wyświetlić: Lista jest pusta.
00172 
00173     return 0;
00174 }
```

4.4 projekt 1.cpp

Go to the documentation of this file.

```
#include <iostream>

struct Node {
    int data;
    Node* prev;
    Node* next;

    Node(int value) : data(value), prev(nullptr), next(nullptr) {}
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Dodaj element na początek listy
    void addFront(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    // Dodaj element na koniec listy
    void addBack(int value) {
        Node* newNode = new Node(value);
        if (tail == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }
    // Dodaj element pod wskazany indeks
void addAtIndex(int index, int value) {
    if (index == 0) {
        addAtHead(value);
        return;
    }

    Node* newNode = new Node(value);
    Node* temp = head;
    for (int i = 0; i < index - 1 && temp != nullptr; i++) {
        temp = temp->next;
    }

    if (temp == nullptr || temp->next == nullptr) {
        addAtTail(value);
    }
    else {
        newNode->next = temp->next;
        newNode->prev = temp;
        temp->next->prev = newNode;
        temp->next = newNode;
    }
}
    // Usuń element z początku listy
    void removeFront() {
        if (head == nullptr) {
            std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
            return;
        }
        Node* temp = head;
        if (head == tail) { // Gdy w liście jest tylko jeden element
            head = tail = nullptr;
        }
        else {
            head = head->next;
            head->prev = nullptr;
        }
        delete temp;
    }

    // Usuń element z końca listy
    void removeBack() {
        if (tail == nullptr) {
            std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
            return;
        }
        Node* temp = tail;
        if (head == tail) { // Gdy w liście jest tylko jeden element
            head = tail = nullptr;
        }
        else {
            tail = tail->prev;
            tail->next = nullptr;
        }
        delete temp;
    }
    // Usuń element z końca listy
void removeFromTail() {
    if (tail == nullptr) return;

    Node* temp = tail;
    if (head == tail) {
        head = tail = nullptr;
    }
    else {
        tail = tail->prev;
        tail->next = nullptr;
    }
    delete temp;
}
    // Wyświetl listę od początku
    void display() {
        Node* current = head;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->next;
        }
        std::cout « "\n";
    }

    // Wyświetl listę w odwrotnej kolejności
    void displayReverse() {
        Node* current = tail;
        if (current == nullptr) {
            std::cout « "Lista jest pusta.\n";
            return;
        }
        while (current != nullptr) {
            std::cout « current->data « " ";
            current = current->prev;
        }
        std::cout « "\n";
    }

    // Czyszczenie całej listy
    void clear() {
        while (head != nullptr) {
            removeFront();
        }
    }

    // Destruktor, aby zwolnić pamięć
    ~DoublyLinkedList() {
        clear();
    }
};

// Testowanie klasy w funkcji main
int main() {
    DoublyLinkedList list;

    list.addFront(10);
    list.addFront(20);
    list.addBack(30);
    list.display();         // Powinno wyświetlić: 20 10 30

    list.removeFront();
    list.display();         // Powinno wyświetlić: 10 30

    list.removeBack();
    list.display();         // Powinno wyświetlić: 10

    list.clear();
    list.display();         // Powinno wyświetlić: Lista jest pusta.

    return 0;
}

12345
```

4.5 projekt.cpp File Reference

`#include <iostream>`

Classes

- struct Node
- class DoublyLinkedList

Functions

- int main ()

4.5.1 Function Documentation

4.5.1.1 main()

`int main ()`

Definition at line 122 of file projekt.cpp.

```
00122            {
    DoublyLinkedList list;

    list.addFront(10);
    list.addFront(20);
    list.addBack(30);
    list.display();         // Powinno wyświetlić: 20 10 30

    list.removeFront();
    list.display();         // Powinno wyświetlić: 10 30

    list.removeBack();
    list.display();         // Powinno wyświetlić: 10

    list.clear();
    list.display();         // Powinno wyświetlić: Lista jest pusta.

    return 0;
}
```

4.6 projekt.cpp

Go to the documentation of this file.

```
#include <iostream>


struct Node {
    int data;
    Node* prev;
    Node* next;

    Node(int value) : data(value), prev(nullptr), next(nullptr) {}
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Dodaj element na początek listy
    void addFront(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    // Dodaj element na koniec listy
    void addBack(int value) {
        Node* newNode = new Node(value);
        if (tail == nullptr) { // Gdy lista jest pusta
            head = tail = newNode;
        }
        else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }

    // Usuń element z początku listy
    void removeFront() {
        if (head == nullptr) {
            std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
            return;
        }
        Node* temp = head;
        if (head == tail) { // Gdy w liście jest tylko jeden element
            head = tail = nullptr;
        }
        else {
            head = head->next;
            head->prev = nullptr;
        }
```

```
            delete temp;
        }

        // Usuń element z końca listy
        void removeBack() {
            if (tail == nullptr) {
                std::cout « "Lista jest pusta, nie można usunąć elementu.\n";
                return;
            }
            Node* temp = tail;
            if (head == tail) { // Gdy w liście jest tylko jeden element
                head = tail = nullptr;
            }
            else {
                tail = tail->prev;
                tail->next = nullptr;
            }
            delete temp;
        }

        // Wyświetl listę od początku
        void display() {
            Node* current = head;
            if (current == nullptr) {
                std::cout « "Lista jest pusta.\n";
                return;
            }
            while (current != nullptr) {
                std::cout « current->data « " ";
                current = current->next;
            }
            std::cout « "\n";
        }

        // Wyświetl listę w odwrotnej kolejności
        void displayReverse() {
            Node* current = tail;
            if (current == nullptr) {
                std::cout « "Lista jest pusta.\n";
                return;
            }
            while (current != nullptr) {
                std::cout « current->data « " ";
                current = current->prev;
            }
            std::cout « "\n";
        }

        // Czyszczenie całej listy
        void clear() {
            while (head != nullptr) {
                removeFront();
            }
        }

        // Destruktor, aby zwolnić pamięć
        ~DoublyLinkedList() {
            clear();
        }
    };

    // Testowanie klasy w funkcji main
    int main() {
        DoublyLinkedList list;

        list.addFront(10);
        list.addFront(20);
        list.addBack(30);
        list.display();          // Powinno wyświetlić: 20 10 30

        list.removeFront();
        list.display();          // Powinno wyświetlić: 10 30

        list.removeBack();
        list.display();          // Powinno wyświetlić: 10

        list.clear();
        list.display();          // Powinno wyświetlić: Lista jest pusta.

        return 0;
    }
```

Index

∼DoublyLinkedList
    DoublyLinkedList, 6

addAtIndex
    DoublyLinkedList, 7
addBack
    DoublyLinkedList, 7
addFront
    DoublyLinkedList, 8

clear
    DoublyLinkedList, 8, 9
ConsoleApplication24.cpp, 15
    main, 15

data
    Node, 14
display
    DoublyLinkedList, 9
displayReverse
    DoublyLinkedList, 10
DoublyLinkedList, 5
    ∼DoublyLinkedList, 6
    addAtIndex, 7
    addBack, 7
    addFront, 8
    clear, 8, 9
    display, 9
    displayReverse, 10
    DoublyLinkedList, 6
    removeBack, 10, 11
    removeFromTail, 11
    removeFront, 12

main
    ConsoleApplication24.cpp, 15
    projekt 1.cpp, 18
    projekt.cpp, 20

next
    Node, 14
Node, 13
    data, 14
    next, 14
    Node, 13, 14
    prev, 14

prev
    Node, 14
projekt 1.cpp, 17
    main, 18
projekt.cpp, 20
    main, 20

removeBack
    DoublyLinkedList, 10, 11
removeFromTail
    DoublyLinkedList, 11
removeFront
    DoublyLinkedList, 12