

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](#).

```
00045     {
00046         if (index == 0) {
00047             addAtHead(value);
00048             return;
00049         }
00050
00051         Node* newNode = new Node(value);
00052         Node* temp = head;
00053         for (int i = 0; i < index - 1 && temp != nullptr; i++) {
00054             temp = temp->next;
00055         }
00056
00057         if (temp == nullptr || temp->next == nullptr) {
00058             addAtTail(value);
00059         }
00060         else {
00061             newNode->next = temp->next;
00062             newNode->prev = temp;
00063             temp->next->prev = newNode;
00064             temp->next = newNode;
00065         }
00066     }
```

3.1.3.2 addBack() [1/3]

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

Definition at line 33 of file [ConsoleApplication24.cpp](#).

```
00033     {
00034         Node* newNode = new Node(value);
00035         if (tail == nullptr) { // Gdy lista jest pusta
00036             head = tail = newNode;
00037         }
00038         else {
00039             newNode->prev = tail;
00040             tail->next = newNode;
00041             tail = newNode;
00042         }
00043     }
```

3.1.3.3 addBack() [2/3]

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

Definition at line 33 of file [projekt 1.cpp](#).

```
00033     {
00034         Node* newNode = new Node(value);
00035         if (tail == nullptr) { // Gdy lista jest pusta
00036             head = tail = newNode;
00037         }
00038         else {
00039             newNode->prev = tail;
00040             tail->next = newNode;
00041             tail = newNode;
00042         }
00043     }
```

3.1.3.4 addBack() [3/3]

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

Definition at line 34 of file [projekt.cpp](#).

```
00034     {
00035         Node* newNode = new Node(value);
00036         if (tail == nullptr) { // Gdy lista jest pusta
00037             head = tail = newNode;
00038         }
00039         else {
00040             newNode->prev = tail;
00041             tail->next = newNode;
00042             tail = newNode;
00043         }
00044     }
```

3.1.3.5 addFront() [1/3]

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

Definition at line 20 of file [ConsoleApplication24.cpp](#).

```
00020     {
00021         Node* newNode = new Node(value);
00022         if (head == nullptr) { // Gdy lista jest pusta
00023             head = tail = newNode;
00024         }
00025         else {
00026             newNode->next = head;
00027             head->prev = newNode;
00028             head = newNode;
00029         }
00030     }
```

3.1.3.6 addFront() [2/3]

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

Definition at line 20 of file [projekt 1.cpp](#).

```
00020     {
00021         Node* newNode = new Node(value);
00022         if (head == nullptr) { // Gdy lista jest pusta
00023             head = tail = newNode;
00024         }
00025         else {
00026             newNode->next = head;
00027             head->prev = newNode;
00028             head = newNode;
00029         }
00030     }
```

3.1.3.7 addFront() [3/3]

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

Definition at line 21 of file [projekt.cpp](#).

```
00021     {
00022         Node* newNode = new Node(value);
00023         if (head == nullptr) { // Gdy lista jest pusta
00024             head = tail = newNode;
00025         }
00026         else {
00027             newNode->next = head;
00028             head->prev = newNode;
00029             head = newNode;
00030         }
00031     }
```

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](#).

```
00081     {
00082         Node* current = head;
00083         if (current == nullptr) {
00084             std::cout << "Lista jest pusta.\n";
00085             return;
00086         }
00087         while (current != nullptr) {
00088             std::cout << current->data << " ";
00089             current = current->next;
00090         }
00091         std::cout << "\n";
00092     }
```

3.1.3.14 displayReverse() [1/3]

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

Definition at line 94 of file [ConsoleApplication24.cpp](#).

```
00094     {
00095         Node* current = tail;
00096         if (current == nullptr) {
00097             std::cout << "Lista jest pusta.\n";
00098             return;
00099         }
00100         while (current != nullptr) {
00101             std::cout << current->data << " ";
00102             current = current->prev;
00103         }
00104         std::cout << "\n";
00105     }
```

3.1.3.15 displayReverse() [2/3]

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

Definition at line 129 of file [projekt 1.cpp](#).

```
00129     {
00130         Node* current = tail;
00131         if (current == nullptr) {
00132             std::cout << "Lista jest pusta.\n";
00133             return;
00134         }
00135         while (current != nullptr) {
00136             std::cout << current->data << " ";
00137             current = current->prev;
00138         }
00139         std::cout << "\n";
00140     }
```

3.1.3.16 displayReverse() [3/3]

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

Definition at line 95 of file [projekt.cpp](#).

```
00095     {
00096         Node* current = tail;
00097         if (current == nullptr) {
00098             std::cout << "Lista jest pusta.\n";
00099             return;
00100         }
00101         while (current != nullptr) {
00102             std::cout << current->data << " ";
00103             current = current->prev;
00104         }
00105         std::cout << "\n";
00106     }
```

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](#).

```
00047         {
00048             if (head == nullptr) {
00049                 std::cout << "Lista jest pusta, nie można usunąć elementu.\n";
00050                 return;
00051             }
00052             Node* temp = head;
00053             if (head == tail) { // Gdy w liście jest tylko jeden element
00054                 head = tail = nullptr;
00055             }
00056             else {
00057                 head = head->next;
00058                 head->prev = nullptr;
00059             }
00060             delete temp;
00061         }
```

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](#).

```
00008 : 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.](#)

```

00001 #include <iostream>
00002
00003 struct Node {
00004     int data;
00005     Node* prev;
00006     Node* next;
00007
00008     Node(int value) : data(value), prev(nullptr), next(nullptr) {}
00009 };
00010
00011 class DoublyLinkedList {
00012 private:
00013     Node* head;
00014     Node* tail;
00015
00016 public:
00017     DoublyLinkedList() : head(nullptr), tail(nullptr) {}
00018
00019     // Dodaj element na początek listy
00020     void addFront(int value) {
00021         Node* newNode = new Node(value);
00022         if (head == nullptr) { // Gdy lista jest pusta
00023             head = tail = newNode;
00024         }
00025         else {
00026             newNode->next = head;
00027             head->prev = newNode;
00028             head = newNode;
00029         }
00030     }
00031
00032     // Dodaj element na koniec listy
00033     void addBack(int value) {
00034         Node* newNode = new Node(value);
00035         if (tail == nullptr) { // Gdy lista jest pusta
00036             head = tail = newNode;
00037         }
00038         else {
00039             newNode->prev = tail;
00040             tail->next = newNode;
00041             tail = newNode;
00042         }
00043     }
00044
00045     // Usuń element z początku listy
00046     void removeFront() {
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     }
00061
00062     // Usuń element z końca listy
00063     void removeBack() {
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     }
00078
00079     // Wyświetl listę od początku
00080     void display() {
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 }
00092
00093 // Wyświetl listę w odwrotnej kolejności
00094 void displayReverse() {
00095     Node* current = tail;
00096     if (current == nullptr) {
00097         std::cout << "Lista jest pusta.\n";
00098         return;
00099     }
00100     while (current != nullptr) {
00101         std::cout << current->data << " ";
00102         current = current->prev;
00103     }
00104     std::cout << "\n";
00105 }
00106
00107 // Czyszczenie całej listy
00108 void clear() {
00109     while (head != nullptr) {
00110         removeFront();
00111     }
00112 }
00113
00114 // Destruktor, aby zwolnić pamięć
00115 ~DoublyLinkedList() {
00116     clear();
00117 }
00118 };
00119
00120 // Testowanie klasy w funkcji main
00121 int main() {
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 }

```

4.3 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.](#)

```
00001 #include <iostream>
00002
00003 struct Node {
00004     int data;
00005     Node* prev;
00006     Node* next;
00007
00008     Node(int value) : data(value), prev(nullptr), next(nullptr) {}
00009 };
00010
00011 class DoublyLinkedList {
00012 private:
00013     Node* head;
00014     Node* tail;
00015
00016 public:
00017     DoublyLinkedList() : head(nullptr), tail(nullptr) {}
00018
00019     // Dodaj element na początek listy
00020     void addFront(int value) {
00021         Node* newNode = new Node(value);
00022         if (head == nullptr) { // Gdy lista jest pusta
00023             head = tail = newNode;
00024         }
00025         else {
00026             newNode->next = head;
00027             head->prev = newNode;
00028             head = newNode;
00029         }
00030     }
00031
00032     // Dodaj element na koniec listy
00033     void addBack(int value) {
00034         Node* newNode = new Node(value);
00035         if (tail == nullptr) { // Gdy lista jest pusta
00036             head = tail = newNode;
00037         }
00038         else {
00039             newNode->prev = tail;
00040             tail->next = newNode;
00041             tail = newNode;
00042         }
00043     }
00044     // Dodaj element pod wskazany indeks
00045     void addAtIndex(int index, int value) {
00046         if (index == 0) {
00047             addAtHead(value);
```

```

00048         return;
00049     }
00050
00051     Node* newNode = new Node(value);
00052     Node* temp = head;
00053     for (int i = 0; i < index - 1 && temp != nullptr; i++) {
00054         temp = temp->next;
00055     }
00056
00057     if (temp == nullptr || temp->next == nullptr) {
00058         addAtTail(value);
00059     }
00060     else {
00061         newNode->next = temp->next;
00062         newNode->prev = temp;
00063         temp->next->prev = newNode;
00064         temp->next = newNode;
00065     }
00066 }
00067 // Usuń element z początku listy
00068 void removeFront() {
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 }
00083
00084 // Usuń element z końca listy
00085 void removeBack() {
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 }
00100 // Usuń element z końca listy
00101 void removeFromTail() {
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 }
00114 // Wyświetl listę od początku
00115 void display() {
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 }
00127
00128 // Wyświetl listę w odwrotnej kolejności
00129 void displayReverse() {
00130     Node* current = tail;
00131     if (current == nullptr) {
00132         std::cout << "Lista jest pusta.\n";
00133         return;
00134     }

```

```

00135         while (current != nullptr) {
00136             std::cout << current->data << " ";
00137             current = current->prev;
00138         }
00139         std::cout << "\n";
00140     }
00141
00142     // Czyszczenie całej listy
00143     void clear() {
00144         while (head != nullptr) {
00145             removeFront();
00146         }
00147     }
00148
00149     // Destruktor, aby zwolnić pamięć
00150     ~DoublyLinkedList() {
00151         clear();
00152     }
00153 };
00154
00155 // Testowanie klasy w funkcji main
00156 int main() {
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 }
00175
00176 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     {
```



```

00123     DoublyLinkedList list;
00124
00125     list.addFront(10);
00126     list.addFront(20);
00127     list.addBack(30);
00128     list.display();           // Powinno wyświetlić: 20 10 30
00129
00130     list.removeFront();
00131     list.display();           // Powinno wyświetlić: 10 30
00132
00133     list.removeBack();
00134     list.display();           // Powinno wyświetlić: 10
00135
00136     list.clear();
00137     list.display();           // Powinno wyświetlić: Lista jest pusta.
00138
00139     return 0;
00140 }

```

4.6 projekt.cpp

[Go to the documentation of this file.](#)

```

00001 #include <iostream>
00002
00003
00004 struct Node {
00005     int data;
00006     Node* prev;
00007     Node* next;
00008
00009     Node(int value) : data(value), prev(nullptr), next(nullptr) {}
00010 };
00011
00012 class DoublyLinkedList {
00013 private:
00014     Node* head;
00015     Node* tail;
00016
00017 public:
00018     DoublyLinkedList() : head(nullptr), tail(nullptr) {}
00019
00020     // Dodaj element na początek listy
00021     void addFront(int value) {
00022         Node* newNode = new Node(value);
00023         if (head == nullptr) { // Gdy lista jest pusta
00024             head = tail = newNode;
00025         }
00026         else {
00027             newNode->next = head;
00028             head->prev = newNode;
00029             head = newNode;
00030         }
00031     }
00032
00033     // Dodaj element na koniec listy
00034     void addBack(int value) {
00035         Node* newNode = new Node(value);
00036         if (tail == nullptr) { // Gdy lista jest pusta
00037             head = tail = newNode;
00038         }
00039         else {
00040             newNode->prev = tail;
00041             tail->next = newNode;
00042             tail = newNode;
00043         }
00044     }
00045
00046     // Usuń element z początku listy
00047     void removeFront() {
00048         if (head == nullptr) {
00049             std::cout << "Lista jest pusta, nie można usunąć elementu.\n";
00050             return;
00051         }
00052         Node* temp = head;
00053         if (head == tail) { // Gdy w liście jest tylko jeden element
00054             head = tail = nullptr;
00055         }
00056         else {
00057             head = head->next;
00058             head->prev = nullptr;
00059         }

```

```
00060         delete temp;
00061     }
00062
00063     // Usuń element z końca listy
00064     void removeBack() {
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     }
00079
00080     // Wyświetl listę od początku
00081     void display() {
00082         Node* current = head;
00083         if (current == nullptr) {
00084             std::cout << "Lista jest pusta.\n";
00085             return;
00086         }
00087         while (current != nullptr) {
00088             std::cout << current->data << " ";
00089             current = current->next;
00090         }
00091         std::cout << "\n";
00092     }
00093
00094     // Wyświetl listę w odwrotnej kolejności
00095     void displayReverse() {
00096         Node* current = tail;
00097         if (current == nullptr) {
00098             std::cout << "Lista jest pusta.\n";
00099             return;
00100         }
00101         while (current != nullptr) {
00102             std::cout << current->data << " ";
00103             current = current->prev;
00104         }
00105         std::cout << "\n";
00106     }
00107
00108     // Czyszczenie całej listy
00109     void clear() {
00110         while (head != nullptr) {
00111             removeFront();
00112         }
00113     }
00114
00115     // Destruktor, aby zwolnić pamięć
00116     ~DoublyLinkedList() {
00117         clear();
00118     }
00119 };
00120
00121 // Testowanie klasy w funkcji main
00122 int main() {
00123     DoublyLinkedList list;
00124
00125     list.addFront(10);
00126     list.addFront(20);
00127     list.addBack(30);
00128     list.display();           // Powinno wyświetlić: 20 10 30
00129
00130     list.removeFront();
00131     list.display();           // Powinno wyświetlić: 10 30
00132
00133     list.removeBack();
00134     list.display();           // Powinno wyświetlić: 10
00135
00136     list.clear();
00137     list.display();           // Powinno wyświetlić: Lista jest pusta.
00138
00139     return 0;
00140 }
```

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](#)