Lista dwukierunkowa C++ | Fabryka | Iterator | Singleton

Wygenerowano za pomocą Doxygen 1.15.0

Rozdział 1

PZ_projekt1

1.1 Opis projektu

Celem projektu jest stworzenie dwukierunkowej listy, której elementy będą przechowywane na stercie. Implementacja ma wykorzystywać wzorce projektowe **Factory** oraz **Iterator**. Program umożliwi dodawanie, usuwanie i wyświetlanie elementów listy, a także przeglądanie jej elementów w obu kierunkach

1.2 Wymagania

1.2.1 Klasy i ich funkcjonalność

1. **Item** - reprezentuje pojedynczy element listy
    - Właściwości:
        - `data` - dane przechowywane w Item.
        - `next` - wskaźnik do następnego Item'a.
        - `prev` - wskaźnik do poprzedniego Item'a.

1. **ListTwoway** - klasa listy dwukierunkowej
    - **Właściwości:**
        - `head` - wskaźnik do pierwszego **Item** w liście, który zawiera dane oraz wskaźnik na następny element.
        - `tail` - wskaźnik do ostatniego **Item** w liście, który zawiera dane oraz wskaźnik na poprzedni element.
    - **Metody:**
        - **`unshift(data)`**
            - Dodaje element na początek listy. Tworzy nowy **Item** i ustawia go na `head` listy.
            - **Argumenty:** `data` – dane do zapisania w nowym **Item**.
            - **Działanie:** Jeśli lista jest pusta, nowy **Item** staje się zarówno `head`, jak i `tail`. W przeciwnym przypadku nowy **Item** jest ustawiany na początek listy.
        - **`push(data)`**

* Dodaje element na koniec listy. Tworzy nowy **Item** i ustawia go na `tail` listy.
* **Argumenty:** `data` – dane do zapisania w nowym **Item**.
* **Działanie:** Jeśli lista jest pusta, nowy **Item** staje się zarówno `head`, jak i `tail`. W przeciwnym przypadku nowy **Item** jest dodawany na koniec listy.

– **`insertAt(index, data)`**

* Dodaje element w określonym miejscu w liście na podstawie indeksu. Jeśli indeks jest poza zakresem, nic się nie dzieje.
* **Argumenty:**
  - `index` – indeks, pod którym ma zostać dodany nowy **Item**.
  - `data` – dane do zapisania w nowym **Item**.
* **Działanie:** Jeśli indeks jest 0, dodawany jest element na początek listy. Jeśli indeks jest równy liczbie elementów, dodawany jest element na koniec. W przeciwnym przypadku element jest wstawiany w odpowiednie miejsce w liście.

– **`removeAt(index)`**

* Usuwa element pod wskazanym indeksem. Jeśli indeks jest poza zakresem, nic się nie dzieje.
* **Argumenty:**
  - `index` – indeks **Item**∗∗**u, który ma zostać usunięty.**
* ∗∗**Działanie:** Usuwa **Item** o podanym indeksie i odpowiednio aktualizuje wskaźniki sąsiednich elementów.

– **`shift()`**

* Usuwa element z początku listy. Przesuwa wskaźnik `head` na następny **Item**.
* **Argumenty:** Brak.
* **Działanie:** Jeśli lista ma tylko jeden **Item**, zarówno `head`, jak i `tail` zostaną ustawione na `nullptr`. W przeciwnym przypadku wskaźnik `head` jest zmieniany na następny **Item**.

– **`pop()`**

* Usuwa element z końca listy. Przesuwa wskaźnik `tail` na poprzedni **Item**.
* **Argumenty:** Brak.
* **Działanie:** Jeśli lista ma tylko jeden **Item**, zarówno `head`, jak i `tail` zostaną ustawione na `nullptr`. W przeciwnym przypadku wskaźnik `tail` jest zmieniany na poprzedni **Item**.

– **`clear()`**

* Usuwa wszystkie **Item**∗∗**y listy, ustawiając wskaźniki `head` i `tail` na `nullptr`.**
* ∗∗**Argumenty:** Brak.
* **Działanie:** Usuwa wszystkie ∗∗Item∗∗y listy i resetuje wskaźniki `head` i `tail` na `nullptr`, co skutkuje opróżnieniem listy.

1. **ListIterator** - Iterator dla ListTwoway

- **Właściwości:**
  – **`current`**: Wskaźnik do bieżącego elementu w iteracji
  – **`list`**: Wskaźnik do listy, którą iterator przetwarza
- **Metody:**
  – **`hasNext()`**

* Zwraca `true`, jeśli istnieje kolejny element w iteracji

– **`next()`**

* Zwraca dane kolejnego elementu w liście i przesuwa iterator do tego elementu

  - **`clear()`**
    - Ustawia iterator na początek listy
  - **`printAll()`**
    - Wyświetla wszystkie elementy listy w kolejności od początku do końca
  - **`printAllReverse()`**
    - Wyświetla wszystkie elementy listy w odwrotnej kolejności (od końca do początku)

1. **Logger** - Singleton do loggowania wiadomości
   - **Metody:**
     - **`log(string msg)`**
       - Wyświetla wiadomość msg

1.2.2 Wzorce projektowe

- **Factory**: Wzorzec factory powinien umożliwiać tworzenie instancji listy. Dzięki temu użytkownik nie musi martwić się o szczegóły tworzenia obiektów
- **Iterator**: Iteratory pozwalają na łatwą nawigację po liście w obu kierunkach. Implementacja iteratora umożliwi łatwe przeglądanie elementów listy
- **Singleton**: Zapewnia, że dany obiekt klasy ma tylko jedną instancję w systemie i jest dostępny globalnie. Używany tam, gdzie potrzebny jest jeden, wspólny punkt dostępu, np. w przypadku loggera czy zarządcy konfiguracji. Dostęp do instancji uzyskiwany jest przez statyczną metodę, która tworzy obiekt tylko raz

1.3 Lista zadań (TODO)

1. **main.cpp**:
   - [OK] Zaimplementuj funkcję `main()`, która testuje wszystkie metody listy oraz iteratora.
   - ☐ Przetestuj wszystkie funkcjonalności listy (dodawanie, usuwanie, iteracja, wyświetlanie).
2. **Item**:
   - [OK] Stwórz klasę `Item` z odpowiednimi właściwościami: `data`, `next`, `prev`.
   - [OK] Zaimplementuj konstruktor oraz destruktor.
3. **ListTwoway**:
   - [OK] Stwórz klasę `ListTwoway` z odpowiednimi właściwościami: `head`, `tail`.
   - [OK] Stwórz **`unshift(data)`**.
   - [OK] Stwórz **`push(data)`**.
   - [OK] Stwórz **`insertAt(index, data)`**.
   - [OK] Stwórz **`shift()`**.
   - [OK] Stwórz **`pop()`**.
   - [OK] Stwórz **`removeAt(index)`**.
   - [OK] Stwórz **`clear()`**.
4. **Iterator**:
   - [OK] Zaimplementuj klasę `Iterator`, która będzie umożliwiać iterację po elementach listy.
5. **Factory**:
   - [OK] Stwórz klasę `Factory`, która będzie odpowiedzialna za tworzenie instancji listy.

Rozdział 2

Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 3

Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików wraz z ich krótkimi opisami:

Rozdział 4

Dokumentacja klas

4.1 Dokumentacja szablonu klasy Item< Type >

Klasa szablonowa reprezentująca węzeł listy dwukierunkowej.

`#include <Item.h>`

Metody publiczne

- Item (Type data)

  Konstruktor inicjalizujący element listy.

Atrybuty publiczne

- Type data

  Dane przechowywane w elemencie.
- Item * prev

  Wskaźnik na poprzedni element listy.
- Item * next

  Wskaźnik na następny element listy.

4.1.1 Opis szczegółowy

template<typename Type>
class Item< Type >

Klasa szablonowa reprezentująca węzeł listy dwukierunkowej.

Klasa przechowuje wartość typu `Type` oraz wskaźniki do poprzedniego i następnego elementu. Używana wewnętrznie w implementacji listy dwukierunkowej.

Parametry Szablonu

Type	Typ danych przechowywanych w elemencie listy.

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 Item()

```
template<typename Type>
Item< Type >::Item (
            Type data)  [inline]
```

Konstruktor inicjalizujący element listy.

Parametry

data	Wartość, którą ma przechowywać element.

Tworzy nowy węzeł listy z danymi `data`. Początkowo wskaźniki `prev` i `next` ustawione są na `nullptr`.

4.1.3 Dokumentacja atrybutów składowych

4.1.3.1 data

```
template<typename Type>
Type Item< Type >::data
```

Dane przechowywane w elemencie.

4.1.3.2 next

```
template<typename Type>
Item* Item< Type >::next
```

Wskaźnik na następny element listy.

4.1.3.3 prev

```
template<typename Type>
Item* Item< Type >::prev
```

Wskaźnik na poprzedni element listy.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Item.h

4.2 Dokumentacja szablonu klasy ListIterator< Type >

Klasa implementująca iterator dla listy dwukierunkowej.

`#include <ListIterator.h>`

Metody publiczne

- ListIterator (ListTwoway< Type > *list)
  Konstruktor klasy ListIterator.
- bool hasNext () const
  Sprawdza, czy istnieje następny element w liście.
- Type next ()
  Zwraca wartość następnego elementu i przesuwa iterator.
- void reset ()
  Resetuje iterator do początku listy.
- void printAll () const
  Wypisuje wszystkie elementy listy w kolejności od początku do końca.
- void printAllReverse () const
  Wypisuje wszystkie elementy listy w kolejności od końca do początku.

Atrybuty prywatne

- Item< Type > * current
  Wskaźnik na aktualny element listy.
- ListTwoway< Type > * list
  Wskaźnik na listę, po której iterator się porusza.

4.2.1 Opis szczegółowy

template<typename Type>
class ListIterator< Type >

Klasa implementująca iterator dla listy dwukierunkowej.

Klasa umożliwia iterację po elementach listy w przód i w tył. Wykorzystuje wskaźnik do listy (`ListTwoway<Type>`) oraz bieżący element (`Item<Type>*`).

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 ListIterator()

```
template<typename Type>
ListIterator< Type >::ListIterator (
            ListTwoway< Type > * list)
```

Konstruktor klasy ListIterator.

Parametry

list	Wskaźnik na listę, po której iterator ma się poruszać.

Inicjalizuje iterator, ustawiając wskaźnik bieżącego elementu (`current`) na pierwszy element listy (`head`).

Parametry

list	Wskaźnik na listę, po której iterator ma się poruszać.

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 hasNext()

```
template<typename Type>
bool ListIterator< Type >::hasNext () const
```

Sprawdza, czy istnieje następny element w liście.

Sprawdza, czy istnieje kolejny element w liście.

Zwraca

true jeśli istnieje następny element, false w przeciwnym razie.

4.2.3.2 next()

```
template<typename Type>
Type ListIterator< Type >::next ()
```

Zwraca wartość następnego elementu i przesuwa iterator.

Zwraca wartość bieżącego elementu i przesuwa iterator na następny.

Zwraca

Wartość elementu typu Type.

Wartość aktualnego elementu typu Type.

Nota

Jeśli iterator jest ustawiony poza końcem listy, metoda zwraca `false` (co w przypadku typów liczbowych może być interpretowane jako `0`).

4.2.3.3 printAll()

```
template<typename Type>
void ListIterator< Type >::printAll () const
```

Wypisuje wszystkie elementy listy w kolejności od początku do końca.

Używa klasy Logger do wypisywania danych w konsoli lub pliku logu.

4.2.3.4 printAllReverse()

```
template<typename Type>
void ListIterator< Type >::printAllReverse () const
```

Wypisuje wszystkie elementy listy w kolejności od końca do początku.

Używa klasy Logger do wypisywania danych w konsoli lub pliku logu.

4.2.3.5 reset()

```
template<typename Type>
void ListIterator< Type >::reset ()
```

Resetuje iterator do początku listy.

Resetuje pozycję iteratora na początek listy.

4.2.4 Dokumentacja atrybutów składowych

4.2.4.1 current

```
template<typename Type>
Item<Type>* ListIterator< Type >::current  [private]
```

Wskaźnik na aktualny element listy.

4.2.4.2 list

```
template<typename Type>
ListTwoway<Type>* ListIterator< Type >::list  [private]
```

Wskaźnik na listę, po której iterator się porusza.

Dokumentacja dla tej klasy została wygenerowana z plików:

- ListIterator.h
- ListIterator.cpp

4.3 Dokumentacja szablonu klasy ListTwoway< Type >

Klasa szablonowa reprezentująca listę dwukierunkową.

```
#include <ListTwoway.h>
```

Metody publiczne

- ListTwoway ()

  Konstruktor domyślny klasy ListTwoway.
- ~ListTwoway ()

  Destruktor klasy ListTwoway.
- void unshift (Type data)

  Dodaje nowy element na początek listy.
- void push (Type data)

  Dodaje nowy element na koniec listy.
- void insertAt (int index, Type data)

  Wstawia element w określonym miejscu w liście.
- void removeAt (int index)

  Usuwa element z określonego miejsca w liście.
- void shift ()

  Usuwa pierwszy element listy.
- void pop ()

  Usuwa ostatni element listy.
- void clear ()

  Usuwa wszystkie elementy z listy i resetuje jej rozmiar.
- int getSize ()

  Zwraca aktualną liczbę elementów listy.
- Item< Type > * getHead ()

  Zwraca wskaźnik na pierwszy element listy.
- Item< Type > * getTail ()

  Zwraca wskaźnik na ostatni element listy.

Atrybuty prywatne

- Item< Type > * head

  Wskaźnik na pierwszy element listy.
- Item< Type > * tail

  Wskaźnik na ostatni element listy.
- int size = 0

  Aktualny rozmiar listy (liczba elementów).

4.3.1 Opis szczegółowy

template<typename Type>
class ListTwoway< Type >

Klasa szablonowa reprezentująca listę dwukierunkową.

Klasa przechowuje elementy typu `Item<Type>` połączone wskaźnikami w obu kierunkach. Umożliwia dodawanie elementów na początek i koniec listy, wstawianie elementów w dowolne miejsce, usuwanie elementów oraz czyszczenie całej listy.

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

4.3.2 Dokumentacja konstruktora i destruktora

4.3.2.1 ListTwoway()

```
template<typename Type>
ListTwoway< Type >::ListTwoway ()
```

Konstruktor domyślny klasy ListTwoway.

Tworzy pustą listę, ustawiając wskaźniki `head` i `tail` na `nullptr`.

4.3.2.2 ~ListTwoway()

```
template<typename Type>
ListTwoway< Type >::~ListTwoway ()
```

Destruktor klasy ListTwoway.

W razie potrzeby może zostać rozszerzony o automatyczne czyszczenie listy.

Obecnie nie usuwa automatycznie elementów (należy wywołać clear()).

4.3.3 Dokumentacja funkcji składowych

4.3.3.1 clear()

```
template<typename Type>
void ListTwoway< Type >::clear ()
```

Usuwa wszystkie elementy z listy i resetuje jej rozmiar.

4.3.3.2 getHead()

```
template<typename Type>
Item< Type > * ListTwoway< Type >::getHead ()
```

Zwraca wskaźnik na pierwszy element listy.

Zwraca

Wskaźnik na głowę listy.

Wskaźnik na głowę listy (`head`).

4.3.3.3 getSize()

```
template<typename Type>
int ListTwoway< Type >::getSize ()
```

Zwraca aktualną liczbę elementów listy.

Zwraca liczbę elementów w liście.

Zwraca

Liczba elementów w liście.

Liczba elementów listy.

4.3.3.4 getTail()

```
template<typename Type>
Item< Type > * ListTwoway< Type >::getTail ()
```

Zwraca wskaźnik na ostatni element listy.

Zwraca

Wskaźnik na ogon listy.

Wskaźnik na ogon listy (`tail`).

4.3.3.5 insertAt()

```
template<typename Type>
void ListTwoway< Type >::insertAt (
            int index,
            Type data)
```

Wstawia element w określonym miejscu w liście.

Wstawia nowy element w określone miejsce w liście.

Parametry

index	Indeks miejsca, w którym element ma zostać wstawiony.
data	Wartość elementu do wstawienia.
index	Indeks, pod który ma zostać wstawiony element.
data	Wartość elementu do wstawienia.

4.3.3.6 pop()

```
template<typename Type>
void ListTwoway< Type >::pop ()
```

Usuwa ostatni element listy.

4.3.3.7 push()

```
template<typename Type>
void ListTwoway< Type >::push (
             Type data)
```

Dodaje nowy element na koniec listy.

Parametry

data	Wartość elementu do dodania.
data	Wartość, którą ma przechowywać nowy element.

4.3.3.8 removeAt()

```
template<typename Type>
void ListTwoway< Type >::removeAt (
            int index)
```

Usuwa element z określonego miejsca w liście.

Usuwa element z określonego indeksu.

Parametry

index	Indeks elementu do usunięcia.

4.3.3.9 shift()

```
template<typename Type>
void ListTwoway< Type >::shift ()
```

Usuwa pierwszy element listy.

4.3.3.10 unshift()

```
template<typename Type>
void ListTwoway< Type >::unshift (
            Type data)
```

Dodaje nowy element na początek listy.

Parametry

data	Wartość elementu do dodania.
data	Wartość, którą ma przechowywać nowy element.

4.3.4 Dokumentacja atrybutów składowych

4.3.4.1 head

```
template<typename Type>
Item<Type>* ListTwoway< Type >::head  [private]
```

Wskaźnik na pierwszy element listy.

4.3.4.2 size

```
template<typename Type>
int ListTwoway< Type >::size = 0  [private]
```

Aktualny rozmiar listy (liczba elementów).

4.3.4.3 tail

```
template<typename Type>
Item<Type>* ListTwoway< Type >::tail  [private]
```

Wskaźnik na ostatni element listy.

Dokumentacja dla tej klasy została wygenerowana z plików:

- ListTwoway.h
- ListTwoway.cpp

4.4 Dokumentacja szablonu klasy ListTwowayFactory< Type >

Klasa szablonowa implementująca wzorzec fabryki dla listy dwukierunkowej.

```
#include <ListTwowayFactory.h>
```

Statyczne metody publiczne

- static ListTwoway< Type > * createList ()
  Tworzy nową instancję listy dwukierunkowej.

4.4.1 Opis szczegółowy

template<typename Type>
class ListTwowayFactory< Type >

Klasa szablonowa implementująca wzorzec fabryki dla listy dwukierunkowej.

Zawiera statyczną metodę `createList()`, która tworzy i zwraca wskaźnik na nową instancję klasy ListTwoway<Type>.

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

4.4.2 Dokumentacja funkcji składowych

4.4.2.1 createList()

```
template<typename Type>
ListTwoway< Type > * ListTwowayFactory< Type >::createList ()  [inline], [static]
```

Tworzy nową instancję listy dwukierunkowej.

Zwraca

Wskaźnik na nowo utworzoną listę ListTwoway<Type>.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- ListTwowayFactory.h

4.5 Dokumentacja klasy Logger

Singleton do logowania wiadomości w projekcie.

```
#include <Logger.h>
```

Metody publiczne

- void log (string msg)
  Wypisuje komunikat.

Statyczne metody publiczne

- static Logger * getInstance ()
  Zwraca wskaźnik na jedyną instancję Loggera.

Metody prywatne

- Logger ()
  Prywatny konstruktor, aby uniemożliwić tworzenie dodatkowych instancji.
- Logger (const Logger &)=delete
  Usunięcie konstruktora kopiującego.
- Logger & operator= (const Logger &)=delete
  Usunięcie operatora przypisania.

Statyczne atrybuty prywatne

- static Logger * instance = nullptr
  Wskaźnik na jedyną instancję singletona.

4.5.1 Opis szczegółowy

Singleton do logowania wiadomości w projekcie.

Umożliwia wypisywanie tekstu w konsoli lub pliku logu. Konstruktor prywatny wymusza użycie metody statycznej getInstance().

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 Logger() [1/2]

```
Logger::Logger ()  [private]
```

Prywatny konstruktor, aby uniemożliwić tworzenie dodatkowych instancji.

Konstruktor klasy Logger.

Prywatny, aby wymusić użycie singletona.

4.5.2.2 Logger() [2/2]

```
Logger::Logger (
            const Logger & )  [private], [delete]
```

Usunięcie konstruktora kopiującego.

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 getInstance()

```
Logger * Logger::getInstance ()  [static]
```

Zwraca wskaźnik na jedyną instancję Loggera.

Jeśli instancja nie istnieje, tworzy ją.

Zwraca

Wskaźnik na singleton Logger.

4.5.3.2 log()

```
void Logger::log (
            string msg)
```

Wypisuje komunikat.

Parametry

msg	Wiadomość do wypisania.

4.5.3.3 operator=()

```
Logger & Logger::operator= (
            const Logger & )  [private], [delete]
```

Usunięcie operatora przypisania.

4.5.4 Dokumentacja atrybutów składowych

4.5.4.1 instance

```
Logger * Logger::instance = nullptr  [static], [private]
```

Wskaźnik na jedyną instancję singletona.

Inicjalizacja wskaźnika singletona

Dokumentacja dla tej klasy została wygenerowana z plików:

- Logger.h
- Logger.cpp

Rozdział 5

Dokumentacja plików

5.1 Dokumentacja pliku Item.h

Definicja klasy szablonowej reprezentującej pojedynczy element (węzeł) listy dwukierunkowej.

Komponenty

- class Item< Type >

  Klasa szablonowa reprezentująca węzeł listy dwukierunkowej.

5.1.1 Opis szczegółowy

Definicja klasy szablonowej reprezentującej pojedynczy element (węzeł) listy dwukierunkowej.

Parametry Szablonu

Type	Typ danych przechowywanych w elemencie listy.

Autor

dmichura

5.2 Item.h

Idź do dokumentacji tego pliku.

```
00001
00007
00008 #ifndef ITEM_H
00009 #define ITEM_H
00010
00020 template <typename Type>
00021 class Item {
00022 public:
00023     Type data;
00024     Item* prev;
00025     Item* next;
00026
00034     Item(Type data) : data(data), prev(nullptr), next(nullptr) {}
00035 };
00036
00037 #endif // ITEM_H
```

5.3 Dokumentacja pliku ListIterator.cpp

Implementacja metod klasy ListIterator służącej do iteracji po liście dwukierunkowej.

```
#include "ListIterator.h"
#include "ListTwoway.h"
```

Definicje

- #define LIST_ITERATOR_CPP

5.3.1 Opis szczegółowy

Implementacja metod klasy ListIterator służącej do iteracji po liście dwukierunkowej.

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

Plik zawiera implementację metod klasy ListIterator, umożliwiającej przechodzenie po elementach listy dwukierunkowej w obu kierunkach, wypisywanie ich zawartości oraz resetowanie pozycji iteratora.

5.3.2 Dokumentacja definicji

5.3.2.1 LIST_ITERATOR_CPP

```
#define LIST_ITERATOR_CPP
```

5.4 ListIterator.cpp

Idź do dokumentacji tego pliku.

```
00001
00010
00011 #ifndef LIST_ITERATOR_CPP
00012 #define LIST_ITERATOR_CPP
00013
00014 #include "ListIterator.h"
00015 #include "ListTwoway.h"
00016
00025 template<typename Type>
00026 ListIterator<Type>::ListIterator(ListTwoway<Type>* list)
00027 {
00028     this->list = list;
00029     current = list->getHead();
00030 }
00031
00036 template<typename Type>
00037 bool ListIterator<Type>::hasNext() const
00038 {
00039     return current != nullptr;
00040 }
00041
00049 template<typename Type>
00050 Type ListIterator<Type>::next()
```

```
00051 {
00052     if (current == nullptr) {
00053         return false;
00054     }
00055     Type data = current->data;
00056     current = current->next;
00057     return data;
00058 }
00059
00063 template<typename Type>
00064 void ListIterator<Type>::reset()
00065 {
00066     current = list->getHead();
00067 }
00068
00074 template<typename Type>
00075 void ListIterator<Type>::printAll() const {
00076     Item<Type>* currentItem = list->getHead();
00077     Logger::getInstance()->log("{ ");
00078     while (currentItem) {
00079         Logger::getInstance()->log(to_string(currentItem->data) + " ");
00080         currentItem = currentItem->next;
00081     }
00082     Logger::getInstance()->log("}\n");
00083 }
00084
00090 template<typename Type>
00091 void ListIterator<Type>::printAllReverse() const {
00092     Item<Type>* current = list->getTail();
00093
00094     Logger::getInstance()->log("{ ");
00095     while (current != nullptr) {
00096         Logger::getInstance()->log(to_string(current->data) + " ");
00097         current = current->prev;
00098     }
00099     Logger::getInstance()->log("}\n");
00100 }
00101
00102 #endif // LIST_ITERATOR_CPP
```

5.5 Dokumentacja pliku ListIterator.h

Definicja klasy szablonowej iterującej po elementach listy dwukierunkowej.

`#include "ListTwoway.h"`

Komponenty

- class ListIterator< Type >

  Klasa implementująca iterator dla listy dwukierunkowej.

5.5.1 Opis szczegółowy

Definicja klasy szablonowej iterującej po elementach listy dwukierunkowej.

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

Autor

dmichura

Data

2025-10-25 Klasa ListIterator pozwala na przechodzenie po liście w obu kierunkach, wypisywanie elementów oraz resetowanie pozycji iteratora. Współpracuje z klasą ListTwoway.

5.6 ListIterator.h

Idź do dokumentacji tego pliku.

```
00001
00011
00012 #ifndef LIST_ITERATOR_H
00013 #define LIST_ITERATOR_H
00014
00015 #include "ListTwoway.h"
00016
00026 template<typename Type>
00027 class ListIterator {
00028 private:
00029     Item<Type>* current;
00030     ListTwoway<Type>* list;
00031
00032 public:
00037     ListIterator(ListTwoway<Type>* list);
00038
00043     bool hasNext() const;
00044
00049     Type next();
00050
00054     void reset();
00055
00059     void printAll() const;
00060
00064     void printAllReverse() const;
00065 };
00066
00067 #endif // LIST_ITERATOR_H
```

5.7 Dokumentacja pliku ListTwoway.cpp

Implementacja klasy ListTwoway — listy dwukierunkowej w C++.

```
#include "Item.h"
#include "ListTwoway.h"
```

Definicje

- #define LIST_TWOWAY_CPP

5.7.1 Opis szczegółowy

Implementacja klasy ListTwoway — listy dwukierunkowej w C++.

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

Plik zawiera implementację wszystkich metod klasy ListTwoway, umożliwiającej dodawanie, usuwanie i przeszukiwanie elementów listy w obu kierunkach. Klasa wykorzystuje strukturę Item oraz klasę Logger.

Autor

dmichura

Data

2025-10-25

5.7.2 Dokumentacja definicji

5.7.2.1 LIST_TWOWAY_CPP

```
#define LIST_TWOWAY_CPP
```

5.8 ListTwoway.cpp

Idź do dokumentacji tego pliku.

```
00001
00013
00014 #ifndef LIST_TWOWAY_CPP
00015 #define LIST_TWOWAY_CPP
00016
00017 #include "Item.h"
00018 #include "ListTwoway.h"
00019
00024 template<typename Type>
00025 int ListTwoway<Type>::getSize()
00026 {
00027     return size;
00028 }
00029
00034 template<typename Type>
00035 Item<Type>* ListTwoway<Type>::getHead()
00036 {
00037     return head;
00038 }
00039
00044 template<typename Type>
00045 Item<Type>* ListTwoway<Type>::getTail()
00046 {
00047     return tail;
00048 }
00049
00055 template<typename Type>
00056 ListTwoway<Type>::ListTwoway()
00057 {
00058     head = nullptr;
00059     tail = nullptr;
00060 }
00061
00067 template<typename Type>
00068 ListTwoway<Type>::~ListTwoway()
00069 {
00070     // Można dodać clear(), jeśli chcesz automatycznie usuwać elementy.
00071 }
00072
00077 template<typename Type>
00078 void ListTwoway<Type>::unshift(Type data)
00079 {
00080     Item<Type>* newItem = new Item<Type>(data);
00081
00082     if(!head)
00083     {
00084         head = tail = newItem;
00085     }
00086     else
00087     {
00088         newItem->next = head;
00089         head->prev = newItem;
00090         head = newItem;
00091     }
00092     size++;
00093 }
00094
00099 template<typename Type>
00100 void ListTwoway<Type>::push(Type data)
00101 {
00102     Item<Type>* newItem = new Item<Type>(data);
00103
00104     if(!head)
00105     {
00106         head = tail = newItem;
00107     }
00108     else
00109     {
```

```
00110         tail->next = newItem;
00111         newItem->prev = tail;
00112         tail = newItem;
00113     }
00114     size++;
00115 }
00116
00122 template<typename Type>
00123 void ListTwoway<Type>::insertAt(int index, Type data)
00124 {
00125     if (index < 0 || index > size) {
00126         Logger::getInstance()->log("Index jest niepoprawny!\nIndex musi byc >= 0 i nie moze byc >
      size!\n");
00127         return;
00128     }
00129
00130     if (index == 0) {
00131         unshift(data);
00132         return;
00133     }
00134
00135     if (index == size) {
00136         push(data);
00137         return;
00138     }
00139
00140     Item<Type>* newItem = new Item<Type>(data);
00141     Item<Type>* current = head;
00142
00143     for (int i = 0; i < index - 1; ++i) {
00144         current = current->next;
00145     }
00146
00147     newItem->next = current->next;
00148     if (current->next) {
00149         current->next->prev = newItem;
00150     }
00151
00152     current->next = newItem;
00153     newItem->prev = current;
00154
00155     size++;
00156 }
00157
00162 template<typename Type>
00163 void ListTwoway<Type>::removeAt(int index)
00164 {
00165     if (index < 0 || index >= size) {
00166         Logger::getInstance()->log("Nieprawidłowy indeks!\n");
00167         return;
00168     }
00169
00170     if (index == 0) {
00171         shift();
00172         return;
00173     }
00174
00175     if (index == size - 1) {
00176         pop();
00177         return;
00178     }
00179
00180     Item<Type>* current = head;
00181     for (int i = 0; i < index; ++i) {
00182         current = current->next;
00183     }
00184
00185     current->prev->next = current->next;
00186     if (current->next) {
00187         current->next->prev = current->prev;
00188     }
00189
00190     delete current;
00191     size--;
00192 }
00193
00197 template<typename Type>
00198 void ListTwoway<Type>::shift()
00199 {
00200     if (!head) {
00201         Logger::getInstance()->log("Lista jest pusta!\n");
00202         return;
00203     }
00204
00205     if (head == tail) {
00206         delete head;
00207         head = tail = nullptr;
```

```
00208     }
00209     else {
00210         Item<Type>* temp = head;
00211         head = head->next;
00212         head->prev = nullptr;
00213         delete temp;
00214     }
00215 
00216     size--;
00217 }
00218 
00222 template<typename Type>
00223 void ListTwoway<Type>::pop()
00224 {
00225     if (!tail) {
00226         Logger::getInstance()->log("Lista jest pusta!\n");
00227         return;
00228     }
00229 
00230     if (head == tail) {
00231         delete tail;
00232         head = tail = nullptr;
00233     }
00234     else {
00235         Item<Type>* temp = tail;
00236         tail = tail->prev;
00237         tail->next = nullptr;
00238         delete temp;
00239     }
00240 
00241     size--;
00242 }
00243 
00247 template<typename Type>
00248 void ListTwoway<Type>::clear()
00249 {
00250     Item<Type>* current = head;
00251     while (current != nullptr) {
00252         Item<Type>* temp = current;
00253         current = current->next;
00254         delete temp;
00255     }
00256     head = tail = nullptr;
00257     size = 0;
00258 }
00259 
00260 #endif // LIST_TWOWAY_CPP
```

5.9 Dokumentacja pliku ListTwoway.h

Definicja klasy szablonowej ListTwoway — implementacja listy dwukierunkowej w C++.

```
#include "Item.h"
```

Komponenty

- class ListTwoway< Type >

  Klasa szablonowa reprezentująca listę dwukierunkową.

5.9.1 Opis szczegółowy

Definicja klasy szablonowej ListTwoway — implementacja listy dwukierunkowej w C++.

Klasa ListTwoway umożliwia dodawanie, usuwanie, wstawianie oraz czyszczenie elementów listy. Zawiera wskaźniki do pierwszego i ostatniego elementu listy oraz zmienną przechowującą jej rozmiar.

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

Autor

dmichura

Data

2025-10-25

5.10 ListTwoway.h

Idź do dokumentacji tego pliku.

```
00001
00012
00013 #ifndef LIST_TWOWAY_H
00014 #define LIST_TWOWAY_H
00015
00016 #include "Item.h"
00017
00028 template <typename Type>
00029 class ListTwoway {
00030 private:
00031     Item<Type>* head;
00032     Item<Type>* tail;
00033     int size = 0;
00034
00035 public:
00041     ListTwoway();
00042
00048     ~ListTwoway();
00049
00054     void unshift(Type data);
00055
00060     void push(Type data);
00061
00067     void insertAt(int index, Type data);
00068
00073     void removeAt(int index);
00074
00078     void shift();
00079
00083     void pop();
00084
00088     void clear();
00089
00094     int getSize();
00095
00100     Item<Type>* getHead();
00101
00106     Item<Type>* getTail();
00107 };
00108
00109 #endif // LIST_TWOWAY_H
```

5.11 Dokumentacja pliku ListTwowayFactory.h

Definicja klasy szablonowej ListTwowayFactory — fabryka tworząca obiekty ListTwoway.

```
#include "ListTwoway.cpp"
```

Komponenty

- class ListTwowayFactory< Type >

  Klasa szablonowa implementująca wzorzec fabryki dla listy dwukierunkowej.

5.11.1 Opis szczegółowy

Definicja klasy szablonowej ListTwowayFactory — fabryka tworząca obiekty ListTwoway.

Klasa ListTwowayFactory umożliwia wygodne tworzenie nowych instancji listy dwukierunkowej bez bezpośredniego wywoływania konstruktora.

Parametry Szablonu

Type	Typ danych przechowywanych w liście.

Autor

dmichura

Data

2025-10-25

5.12 ListTwowayFactory.h

Idź do dokumentacji tego pliku.

```
00001
00012
00013 #ifndef LIST_TWOWAY_FACTORY_H
00014 #define LIST_TWOWAY_FACTORY_H
00015
00016 #include "ListTwoway.cpp"
00017
00027 template<typename Type>
00028 class ListTwowayFactory {
00029 public:
00034     static ListTwoway<Type>* createList() {
00035         return new ListTwoway<Type>();
00036     }
00037 };
00038
00039 #endif // LIST_TWOWAY_FACTORY_H
```

5.13 Dokumentacja pliku Logger.cpp

Implementacja klasy Logger — prosty singleton do logowania wiadomości.

```
#include "Logger.h"
```

Definicje

- #define LOGGER_CPP

5.13.1 Opis szczegółowy

Implementacja klasy Logger — prosty singleton do logowania wiadomości.

Klasa Logger umożliwia wypisywanie komunikatów w konsoli lub pliku logu. W projekcie służy do wypisywania elementów listy oraz informacji diagnostycznych.

Autor

Data

2025-10-25

5.13.2 Dokumentacja definicji

5.13.2.1 LOGGER_CPP

```
#define LOGGER_CPP
```

5.14 Logger.cpp

Idź do dokumentacji tego pliku.

```
00001
00011
00012 #ifndef LOGGER_CPP
00013 #define LOGGER_CPP
00014
00015 #include "Logger.h"
00016
00018 Logger* Logger::instance = nullptr;
00019
00025 Logger::Logger() {
00026 }
00027
00034 Logger* Logger::getInstance() {
00035     if (instance == nullptr)
00036         instance = new Logger();
00037     return instance;
00038 }
00039
00044 void Logger::log(string msg) {
00045     cout « msg;
00046 }
00047
00048 #endif // LOGGER_CPP
```

5.15 Dokumentacja pliku Logger.h

Definicja klasy Logger — prosty singleton do logowania wiadomości.

```
#include <iostream>
#include <string>
```

Komponenty

- class Logger

  Singleton do logowania wiadomości w projekcie.

5.15.1 Opis szczegółowy

Definicja klasy Logger — prosty singleton do logowania wiadomości.

Klasa Logger umożliwia wypisywanie komunikatów w konsoli lub pliku logu. Singleton zapewnia jedną globalną instancję klasy w całym projekcie.

Autor

dmichura

Data

2025-10-25

5.16 Logger.h

Idź do dokumentacji tego pliku.

```
00001
00011
00012 #ifndef LOGGER_H
00013 #define LOGGER_H
00014
00015 #include <iostream>
00016 #include <string>
00017 using namespace std;
00018
00026 class Logger {
00027 private:
00028     static Logger* instance;
00029
00031     Logger();
00032
00034     Logger(const Logger&) = delete;
00035
00037     Logger& operator=(const Logger&) = delete;
00038
00039 public:
00046     static Logger* getInstance();
00047
00052     void log(string msg);
00053 };
00054
00055 #endif // LOGGER_H
```

5.17 Dokumentacja pliku main.cpp

Przykładowe użycie listy dwukierunkowej z klas Logger i ListIterator.

```
#include <iostream>
#include "Logger.cpp"
#include "ListIterator.cpp"
#include "ListTwowayFactory.h"
```

Funkcje

- int main ()

5.17.1 Opis szczegółowy

Przykładowe użycie listy dwukierunkowej z klas Logger i ListIterator.

Plik demonstruje:

- tworzenie listy dwukierunkowej za pomocą ListTwowayFactory,
- dodawanie i usuwanie elementów (push, unshift, insertAt, pop, shift, clear),
- iterację po liście przy pomocy ListIterator,
- wypisywanie listy w kolejności normalnej i odwrotnej,
- logowanie komunikatów przy użyciu singletona Logger.

Autor

dmichura

Data

2025-10-25

5.17.2 Dokumentacja funkcji

5.17.2.1 main()

```
int main ()
```

5.18 Dokumentacja pliku README.md

Skorowidz