Projekt Lista Dwukierunkowa

Generated by Doxygen 1.15.0

Chapter 1

redme

// PoVRay 3.7 Scene File " ... .pov" // author: Rafa^3 Curzyd^{3}o // date: 12-10-2025 //---— #version 3.7; global_settings{ assumed_gamma 1.0 } #default{ finish{ ambient 0.1 diffuse 0.9 }} //--— #include "colors.inc" #include "textures.inc" #include "glass.inc" #include "metals.inc" #include "golds.inc" #include "stones.inc" #include "woods.inc" #include "shapes.inc" #include "shapes2.inc" #include "functions.inc" #include "math.inc" #include "transforms.inc" //---— #declare Camera_0 = camera {/∗ultra_wide_angle∗/ angle 15 // front view location <0.0 , 1.0 ,-40.0> right x∗image_width/image_height look_at <0.0 , 1.0 , 0.0>} #declare Camera_1 = camera {/∗ultra_wide_angle∗/ angle 14 // diagonal view location <220.0 , 215.0 ,-220.0> right x∗image_width/image_height look_at <0.0 , 1 , 0.0>} #declare Camera_2 = camera {/∗ultra_wide_angle∗/ angle 90 //right side view location <3.0 , 1.0 , 0.0> right x∗image_width/image_height look_at <0.0 , 1.0 , 0.0>} #declare Camera_3 = camera {/∗ultra_wide_angle∗/ angle 90 // top view location <0.0 , 3.0 ,-0.001> right x∗image_width/image_height look_at <0.0 , 1.0 , 0.0>} camera{Camera_1} //---— // sun ---— light_source{<1500,2500,-2500> color White} // sky ---— sky_sphere{ pigment{ gradient <0,1,0> color_map{ [0 color rgb<1,1,1>]//White [0.4 color rgb<0.14,0.14,0.56>]//∼Navy [0.6 color rgb<0.14,0.14,0.56>]//∼Navy [1.0 color rgb<1,1,1>]//White } scale 2 } } // end of sky_sphere //---—

//----------------------------— the Axes -----------------------------— //--— #macro Axis_(AxisLen, Dark_Texture,Light_Texture) union{ cylinder { <0,-AxisLen,0>,<0,AxisLen,0>,0.05 texture{checker texture{Dark_Texture } texture{Light_Texture} translate<0.1,0,0.1>} } cone{<0,AxisLen,0>,0.2,<0,AxisLen+0.7,0>,0 texture{Dark_Texture} } } // end of union
#end // of macro "Axis()" //---— #macro AxisXYZ(AxisLenX, AxisLenY, AxisLenZ, Tex_Dark, Tex_Light) #end// of macro "AxisXYZ(...)" //--—

#declare Texture_A_Dark = texture { pigment{ color rgb<1,0.45,0>} finish { phong 1} } #declare Texture_A_Light = texture { pigment{ color rgb<1,1,1>} finish { phong 1} }

// ground ---— //-------------------------------—<<< settings of squared plane dimensions #declare RasterScale = 1.0; #declare RasterHalfLine = 0.035;
#declare RasterHalfLineZ = 0.035; //---— #macro Raster(RScale, HLine) pigment{ gradient x scale RScale color_map{[0.000 color rgbt<1,1,1,0>∗0.6] [0+HLine color rgbt<1,1,1,0>∗0.6] [0+HLine color rgbt<1,1,1,1>] [1-HLine color rgbt<1,1,1,1>] [1-HLine color rgbt<1,1,1,0>∗0.6] [1.000 color rgbt<1,1,1,0>∗0.6]} } #end// of Raster(RScale, HLine)-macro
//---—

plane { <0,1,0>, 0 // plane with layered textures texture { pigment{color White∗1.1} finish {ambient 0.45 diffuse 0.85}} texture { Raster(RasterScale,RasterHalfLine) rotate<0,0,0> } texture { Raster(RasterScale,RasterHalfLineZ) rotate<0,90,0>} rotate<0,0,0> } //--— end of squared plane XZ

//---— //--------------------------— objects in scene ---------------------------— //---—

difference {

union {

difference {
cylinder { <0,0,0>,<0,13,0>, 44/2

```
       texture { pigment { color Green }
               //normal  { bumps 0.5 scale <0.005,0.25,0.005>}
                 finish  { phong 0.5 reflection{ 0.00 metallic 0.00} }
               } // end of texture

       scale <1,1,1> rotate<0,0,0> translate<0,0,0>
     } // end of cylinder -------------------------------------
```

}

cylinder { <0,0,0>,<0,10,0>, 12/2

```
       texture { pigment { color Blue }
               //normal  { bumps 0.5 scale <0.005,0.25,0.005>}
                 finish  { phong 0.5 reflection{ 0.00 metallic 0.00} }
               } // end of texture

       scale <1,1,1> rotate<0,0,0> translate<0,13,0>
     } // end of cylinder -------------------------------------
```

} //end union

cylinder { <0,0,0>,<0,24,0>, 6/2

```
       texture { pigment { color Pink }
               //normal  { bumps 0.5 scale <0.005,0.25,0.005>}
                 finish  { phong 0.5 reflection{ 0.00 metallic 0.00} }
               } // end of texture

       scale <1,1,1> rotate<0,0,0> translate<0,0,0>
     } // end of cylinder -------------------------------------
```

cylinder { <-1,0,0>,<7,0,0>, 3/2 texture { pigment { color rgb<1,1,1>} //normal { bumps 0.5 scale <0.25, 0.↩
005,0.005>}
finish { phong 0.5 reflection{ 0.00 metallic 0.00} } } // end of texture scale <1,1,1> rotate<0,0,0> translate<0,17,0>
} // end of cylinder ---------------------------------—

difference {

cone{ <0,0,0>,33/2,<0,2.5,0>,38/2

```
  texture{ pigment{ color Orange}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<0,0,0> translate<0,10.8,0>
} // end of cone -----------------------------------

cone{ <0,2.5,0>,11.5/2,<0,0,0>,17/2

  texture{ pigment{ color Orange}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<0,0,0> translate<0,10.8,0>
} // end of cone -----------------------------------
```

} //end difference

difference {

cone{ $<$0,0,0$>$,33/2,$<$0,2.5,0$>$,38/2

```
  texture{ pigment{ color Orange}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<180,0,0> translate<0,2.4,0>
} // end of cone -----------------------------------

cone{ <0,2.5,0>,11.5/2,<0,0,0>,17/2

  texture{ pigment{ color Orange}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<180,0,0> translate<0,2.5,0>
} // end of cone -----------------------------------
```

} //end difference

box { $<$0,0,0$>$,$<$ 50, 50, 50$>$

```
  texture { pigment{ color rgb<1.00, 1.00, 1.00>*1.1}
            finish { phong 1 reflection{ 0.00 metallic 0.00} }
          } // end of texture

  scale <1,1,1> rotate<0,90,0> translate<0,0,0>
} // end of box --------------------------------------
```

#declare Ball = cylinder { $<$0,0,0$>$,$<$0,40,0$>$, 7/2

```
      texture { pigment { color Red }
              //normal  { bumps 0.5 scale <0.005,0.25,0.005>}
                finish  { phong 0.5 reflection{ 0.00 metallic 0.00} }
              } // end of texture

      scale <1,1,1> rotate<0,0,0> translate<0,0,12.5>
```

} //--—

union{ //--— #local Nr = 0; // start #local EndNr = 8; // end #while (Nr$<$ EndNr) object{ Ball translate$<$1,0.25,0$>$ rotate$<$0,Nr $*$ 360/EndNr,0$>$}

#local Nr = Nr + 1; // next Nr #end // -------------— end of loop

rotate$<$0,0,0$>$ translate$<$0,0,0$>$ } // end of union ---—

// CSG difference, subtract intersections of shapes 2...N from Shape1 difference {
cone{ $<$0,0,0$>$,45/2,$<$0,1,0$>$,47/2

```
  texture{ pigment{ color rgbf<1.00,1,1,0.75>}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<0,0,0> translate<0,0.0001,0>
} // end of cone -----------------------------------
```

cone{ <0,0,0>,43/2,<0,1,0>,44/2

```
  texture{ pigment{ color rgbf<1.00,1,1,0.75>}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<0,0,0> translate<0,0.0001,0>
} // end of cone -----------------------------------
```

} //end diference

difference {

cone{ <0,0,0>,45/2,<0,1,0>,47/2

```
  texture{ pigment{ color Pink}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<180,0,0> translate<0,13.1,0>
} // end of cone -----------------------------------
```

cone{ <0,0,0>,43/2,<0,1,0>,44/2

```
  texture{ pigment{ color Yellow}
           // pigment{ color rgb<1.00,0.60,0.00>}
           finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
         } // end of texture
  scale <1,1,1> rotate<180,0,0> translate<0,13.1,0>
```

} //end diference

} //end difference

} // end of cone ---------------------------------—

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

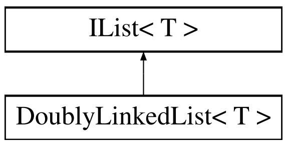
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 DoublyLinkedList< T > Class Template Reference

Inheritance diagram for DoublyLinkedList< T >:



Public Member Functions

- DoublyLinkedList ()
- ~DoublyLinkedList ()
- void push_front (const T &value) override
- void push_back (const T &value) override
- void insert_at (size_t index, const T &value) override
- void pop_front () override
- void pop_back () override
- void remove_at (size_t index) override
- void clear () override
- void print_forward () const override
- void print_backward () const override
- size_t size () const
- ListIterator< T > begin_iterator () const
- ListIterator< T > end_iterator () const

Public Member Functions inherited from IList< T >

- virtual ~IList ()=default

5.1.1 Constructor & Destructor Documentation

5.1.1.1 DoublyLinkedList()

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

Konstruktor - tworzy pusta liste

5.1.1.2 ∼DoublyLinkedList()

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

Destruktor - usuwa wszystkie elementy

5.1.2 Member Function Documentation

5.1.2.1 begin_iterator()

```
template<typename T>
ListIterator< T > DoublyLinkedList< T >::begin_iterator () const  [inline]
```

Zwraca iterator na pierwszy element

5.1.2.2 clear()

```
template<typename T>
void DoublyLinkedList< T >::clear ()  [override], [virtual]
```

Czysci cala liste

Implements IList< T >.

5.1.2.3 end_iterator()

```
template<typename T>
ListIterator< T > DoublyLinkedList< T >::end_iterator () const  [inline]
```

Zwraca iterator na ostatni element

5.1.2.4 insert_at()

```
template<typename T>
void DoublyLinkedList< T >::insert_at (
             size_t index,
             const T & value)  [override], [virtual]
```

Dodaje element na wskazany indeks

Implements IList< T >.

5.1.2.5 pop_back()

```
template<typename T>
void DoublyLinkedList< T >::pop_back ()  [override], [virtual]
```

Usuwa element z konca listy

Implements IList< T >.

5.1.2.6 pop_front()

```
template<typename T>
void DoublyLinkedList< T >::pop_front ()  [override], [virtual]
```

Usuwa element z poczatku listy

Implements IList< T >.

5.1.2.7 print_backward()

```
template<typename T>
void DoublyLinkedList< T >::print_backward () const  [override], [virtual]
```

Wyswietla liste od konca

Implements IList< T >.

5.1.2.8 print_forward()

```
template<typename T>
void DoublyLinkedList< T >::print_forward () const  [override], [virtual]
```

Wyswietla liste od poczatku

Implements IList< T >.

5.1.2.9 push_back()

```
template<typename T>
void DoublyLinkedList< T >::push_back (
             const T & value)  [override], [virtual]
```

Dodaje element na koniec listy

Implements IList< T >.

5.1.2.10 push_front()

```
template<typename T>
void DoublyLinkedList< T >::push_front (
            const T & value)  [override], [virtual]
```

Dodaje element na poczatek listy

Implements ILlist< T >.

5.1.2.11 remove_at()

```
template<typename T>
void DoublyLinkedList< T >::remove_at (
            size_t index)  [override], [virtual]
```

Usuwa element z pod wskazanego indeksu

Implements IList< T >.

5.1.2.12 size()

```
template<typename T>
size_t DoublyLinkedList< T >::size () const  [inline]
```

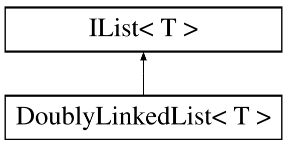
Zwraca liczbe elementow

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

- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/DoublyLinkedList.h

5.2 IList< T > Class Template Reference

Inheritance diagram for IList< T >:



Public Member Functions

- virtual ~IList ()=default
- virtual void push_front (const T &value)=0
- virtual void push_back (const T &value)=0
- virtual void insert_at (size_t index, const T &value)=0
- virtual void pop_front ()=0
- virtual void pop_back ()=0
- virtual void remove_at (size_t index)=0
- virtual void clear ()=0
- virtual void print_forward () const =0
- virtual void print_backward () const =0

5.2.1 Constructor & Destructor Documentation

5.2.1.1 ∼IList()

```
template<typename T>
virtual IList< T >::∼IList ()  [virtual], [default]
```

Wirtualny destruktor

5.2.2 Member Function Documentation

5.2.2.1 clear()

```
template<typename T>
virtual void IList< T >::clear ()  [pure virtual]
```

Czysci cala liste

Implemented in DoublyLinkedList< T >.

5.2.2.2 insert_at()

```
template<typename T>
virtual void IList< T >::insert_at (
            size_t index,
            const T & value)  [pure virtual]
```

Dodaje element na wskazany indeks

Implemented in DoublyLinkedList< T >.

5.2.2.3 pop_back()

```
template<typename T>
virtual void IList< T >::pop_back ()  [pure virtual]
```

Usuwa element z konca listy

Implemented in DoublyLinkedList< T >.

5.2.2.4 pop_front()

```
template<typename T>
virtual void IList< T >::pop_front ()  [pure virtual]
```

Usuwa element z poczatku listy

Implemented in DoublyLinkedList< T >.

5.2.2.5 print_backward()

```
template<typename T>
virtual void IList< T >::print_backward () const  [pure virtual]
```

Wyswietla liste od konca

Implemented in DoublyLinkedList< T >.

5.2.2.6 print_forward()

```
template<typename T>
virtual void IList< T >::print_forward () const  [pure virtual]
```

Wyswietla liste od poczatku

Implemented in DoublyLinkedList< T >.

5.2.2.7 push_back()

```
template<typename T>
virtual void IList< T >::push_back (
             const T & value)  [pure virtual]
```

Dodaje element na koniec listy

Implemented in DoublyLinkedList< T >.

5.2.2.8 push_front()

```
template<typename T>
virtual void IList< T >::push_front (
             const T & value)  [pure virtual]
```

Dodaje element na poczatek listy

Implemented in DoublyLinkedList< T >.

5.2.2.9 remove_at()

```
template<typename T>
virtual void IList< T >::remove_at (
             size_t index)  [pure virtual]
```

Usuwa element z pod wskazanego indeksu

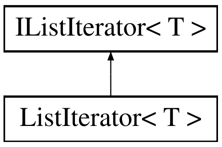
Implemented in DoublyLinkedList< T >.

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

- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IList.h

5.3 IListIterator< T > Class Template Reference

Inheritance diagram for IListIterator< T >:



Public Member Functions

- virtual ~IListIterator ()=default
- virtual bool has_next () const =0
- virtual bool has_prev () const =0
- virtual void next ()=0
- virtual void prev ()=0
- virtual T & value ()=0

5.3.1 Constructor & Destructor Documentation

5.3.1.1 ~IListIterator()

```
template<typename T>
virtual IListIterator< T >::~IListIterator ()  [virtual], [default]
```

Wirtualny destruktor

5.3.2 Member Function Documentation

5.3.2.1 has_next()

```
template<typename T>
virtual bool IListIterator< T >::has_next () const  [pure virtual]
```

Sprawdza, czy istnieje następny element

Implemented in ListIterator< T >.

5.3.2.2 has_prev()

```
template<typename T>
virtual bool IListIterator< T >::has_prev () const  [pure virtual]
```

Sprawdza, czy istnieje poprzedni element

Implemented in ListIterator< T >.

5.3.2.3 next()

```
template<typename T>
virtual void IListIterator< T >::next ()  [pure virtual]
```

Przesuwa iterator na następny element

Implemented in ListIterator< T >.

5.3.2.4 prev()

```
template<typename T>
virtual void IListIterator< T >::prev ()  [pure virtual]
```

Przesuwa iterator na poprzedni element

Implemented in ListIterator< T >.

5.3.2.5 value()

```
template<typename T>
virtual T & IListIterator< T >::value ()  [pure virtual]
```

Zwraca referencję do wartości bieżącego elementu

Implemented in ListIterator< T >.

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

- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IListIterator.h

5.4 ListFactory Class Reference

Static Public Member Functions

- template< typename T>
  static DoublyLinkedList< T > * create ()

  Tworzy nową listę dwukierunkową na stercie.

5.4.1 Member Function Documentation

5.4.1.1 create()

```
template<typename T>
DoublyLinkedList< T > * ListFactory::create ()  [inline], [static]
```

Tworzy nową listę dwukierunkową na stercie.

Template Parameters

T	Typ elementów listy

Returns

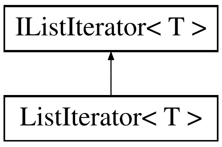
Wskaźnik do nowo utworzonej listy

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

- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListFactory.h

5.5 ListIterator< T > Class Template Reference

Inheritance diagram for ListIterator< T >:



Public Member Functions

- ListIterator (Node< T > ∗start)

  Konstruktor iteratora.
- bool has_next () const override
- bool has_prev () const override
- void next () override
- void prev () override
- T & value () override
- Node< T > ∗ get_node () const

Public Member Functions inherited from IListIterator< T >

- virtual ∼IListIterator ()=default

5.5.1 Constructor & Destructor Documentation

5.5.1.1 ListIterator()

```
template<typename T>
ListIterator< T >::ListIterator (
            Node< T > ∗ start)  [inline], [explicit]
```

Konstruktor iteratora.

Parameters

start	Wskaznik na element, od ktorego zaczyna iteracje

5.5.2 Member Function Documentation

5.5.2.1 get_node()

```
template<typename T>
Node< T > * ListIterator< T >::get_node () const  [inline]
```

Zwraca surowy wskaznik na aktualny wezel (uzywane przez liste)

5.5.2.2 has_next()

```
template<typename T>
bool ListIterator< T >::has_next () const  [inline], [override], [virtual]
```

Sprawdza, czy istnieje nastepny element

Implements IListIterator< T >.

5.5.2.3 has_prev()

```
template<typename T>
bool ListIterator< T >::has_prev () const  [inline], [override], [virtual]
```

Sprawdza, czy istnieje poprzedni element

Implements IListIterator< T >.

5.5.2.4 next()

```
template<typename T>
void ListIterator< T >::next ()  [inline], [override], [virtual]
```

Przesuwa iterator na nastepny element

Implements IListIterator< T >.

5.5.2.5 prev()

```
template<typename T>
void ListIterator< T >::prev ()  [inline], [override], [virtual]
```

Przesuwa iterator na poprzedni element

Implements IListIterator< T >.

5.5.2.6 value()

```
template<typename T>
T & ListIterator< T >::value ()  [inline], [override], [virtual]
```

Zwraca wartosc aktualnego elementu

Implements IListIterator< T >.

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

- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListIterator.h

5.6 Node< T > Struct Template Reference

Public Member Functions

- Node (const T &value)

    Konstruktor wezla.

Public Attributes

- T **data**

    Wartosc przechowywana w wezle.
- Node< T > ∗ **prev**

    Wskaznik na poprzedni element.
- Node< T > ∗ **next**

    Wskaznik na nastepny element.

5.6.1 Constructor & Destructor Documentation

5.6.1.1 Node()

```
template<typename T>
Node< T >::Node (
                const T & value)  [inline]
```

Konstruktor wezla.

Parameters

value	Wartosc przechowywana w wezle

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

- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/Node.h

Chapter 6

File Documentation

6.1 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/DoublyLinked↩ List.h File Reference

Lista dwukierunkowa przechowywana na stercie.

```
#include "IList.h"
#include "Node.h"
#include "IListIterator.h"
#include "ListIterator.h"
#include <cstddef>
#include "DoublyLinkedList.tpp"
```

Classes

- class DoublyLinkedList< T >

6.1.1 Detailed Description

Lista dwukierunkowa przechowywana na stercie.

6.2 DoublyLinkedList.h

Go to the documentation of this file.

```
#ifndef DOUBLYLINKEDLIST_H
#define DOUBLYLINKEDLIST_H

#include "IList.h"
#include "Node.h"
#include "IListIterator.h"
#include "ListIterator.h"
#include <cstddef>


template<typename T>
class DoublyLinkedList : public IList<T> {
private:
    Node<T>* head;
    Node<T>* tail;
    size_t size_;

public:
    DoublyLinkedList();

    ~DoublyLinkedList();

    void push_front(const T& value) override;

    void push_back(const T& value) override;

    void insert_at(size_t index, const T& value) override;

    void pop_front() override;

    void pop_back() override;

    void remove_at(size_t index) override;

    void clear() override;

    void print_forward() const override;

    void print_backward() const override;

    size_t size() const { return size_; }

    ListIterator<T> begin_iterator() const { return ListIterator<T>(head); }

    ListIterator<T> end_iterator() const { return ListIterator<T>(tail); }
};

#include "DoublyLinkedList.tpp"

#endif // DOUBLYLINKEDLIST_H
```

6.3 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IList.h File Reference

Interfejs dla listy dwukierunkowej (szablon).

`#include <cstddef>`

Classes

- class IList< T >

6.3.1 Detailed Description

Interfejs dla listy dwukierunkowej (szablon).

Definiuje podstawowe operacje na liście, takie jak dodawanie, usuwanie, czyszczenie i wyświetlanie elementów.

6.4 IList.h

Go to the documentation of this file.

```
#ifndef ILIST_H
#define ILIST_H


#include <cstddef>

template<typename T>
class IList {
public:
    virtual ~IList() = default;

    virtual void push_front(const T& value) = 0;

    virtual void push_back(const T& value) = 0;

    virtual void insert_at(size_t index, const T& value) = 0;

    virtual void pop_front() = 0;

    virtual void pop_back() = 0;

    virtual void remove_at(size_t index) = 0;

    virtual void clear() = 0;

    virtual void print_forward() const = 0;

    virtual void print_backward() const = 0;
};

#endif // ILIST_H
```

6.5 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IListIterator.h File Reference

Interfejs iteratora dla listy.

Classes

- class IListIterator< T >

6.5.1 Detailed Description

Interfejs iteratora dla listy.

Definiuje podstawowe operacje iteratora: poruszanie się po liście oraz pobieranie wartości bieżącego elementu.

6.6 IListIterator.h

Go to the documentation of this file.

```
#ifndef ILISTITERATOR_H
#define ILISTITERATOR_H


template<typename T>
class IListIterator {
public:
    virtual ~IListIterator() = default;
    virtual bool has_next() const = 0;

    virtual bool has_prev() const = 0;

    virtual void next() = 0;

    virtual void prev() = 0;

    virtual T& value() = 0;
};

#endif // ILISTITERATOR_H
```

6.7 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListFactory.h File Reference

Prosty wzorzec Factory do tworzenia listy dwukierunkowej.

```
#include "DoublyLinkedList.h"
```

Classes

- class ListFactory

6.7.1 Detailed Description

Prosty wzorzec Factory do tworzenia listy dwukierunkowej.

6.8 ListFactory.h

Go to the documentation of this file.

```
#ifndef LISTFACTORY_H
#define LISTFACTORY_H

#include "DoublyLinkedList.h"

class ListFactory {
public:
    template<typename T>
    static DoublyLinkedList<T>* create() {
        return new DoublyLinkedList<T>();
    }
};

#endif // LISTFACTORY_H
```

6.9 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListIterator.h File Reference

Iterator dla listy dwukierunkowej.

```
#include "IListIterator.h"
#include "Node.h"
#include <stdexcept>
```

Classes

- class ListIterator< T >

6.9.1 Detailed Description

Iterator dla listy dwukierunkowej.

Umozliwia poruszanie sie po elementach listy dwukierunkowej w obu kierunkach.

6.10 ListIterator.h

Go to the documentation of this file.

```
#ifndef LISTITERATOR_H
#define LISTITERATOR_H

#include "IListIterator.h"
#include "Node.h"
#include <stdexcept>

template<typename T>
class ListIterator : public IListIterator<T> {
private:
    Node<T>* current;

public:
    explicit ListIterator(Node<T>* start) : current(start) {}

    bool has_next() const override { return current && current->next; }

    bool has_prev() const override { return current && current->prev; }

    void next() override {
        if (!current) throw std::out_of_range("Iterator: brak elementu");
        if (!current->next) throw std::out_of_range("Iterator: brak kolejnego elementu");
        current = current->next;
    }

    void prev() override {
        if (!current) throw std::out_of_range("Iterator: brak elementu");
        if (!current->prev) throw std::out_of_range("Iterator: brak poprzedniego elementu");
        current = current->prev;
    }

    T& value() override {
        if (!current) throw std::out_of_range("Iterator: brak elementu");
        return current->data;
    }

    Node<T>* get_node() const { return current; }
};

#endif // LISTITERATOR_H
```

6.11 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/main.cpp File Reference

Testowanie funkcjonalnosci listy dwukierunkowej.

```
#include <iostream>
#include "ListFactory.h"
```

Functions

- int main ()

6.11.1 Detailed Description

Testowanie funkcjonalnosci listy dwukierunkowej.

Program tworzy liste przy uzyciu wzorca Factory, dodaje i usuwa elementy, wyswietla liste w obu kierunkach oraz testuje iterator.

6.11.2 Function Documentation

6.11.2.1 main()

```
int main ()
```

< Dodaje element na koniec

< Dodaje element na poczatek

< Dodaje element na koniec

< Wstawia element na indeks 1 (lista: 5, 7, 10, 20)

< Wyswietlenie listy od poczatku

< Wyswietlenie listy od konca

< Usuwa pierwszy element

< Usuwa ostatni element

< Wyswietlenie listy po usunieciach

< Usuwa element pod indeksem 1

< Wyswietlenie listy po remove_at

< Zwolnienie pamieci

6.12 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/Node.h File Reference

Definicja wezla listy dwukierunkowej.

Classes

- struct Node< T >

6.12.1 Detailed Description

Definicja wezla listy dwukierunkowej.

Kazdy wezel przechowuje wartosc typu T oraz wskazniki na poprzedni i nastepny element listy.

6.13 Node.h

Go to the documentation of this file.

```
#ifndef NODE_H
#define NODE_H


template<typename T>
struct Node {
    T data;
    Node<T>* prev;
    Node<T>* next;

    Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}
};

#endif // NODE_H
```

Index