

## Projekt Lista Dwukierunkowa

Generated by Doxygen 1.15.0



<b>1 redme</b>	<b>1</b>
<b>2 Hierarchical Index</b>	<b>5</b>
2.1 Class Hierarchy	5
<b>3 Class Index</b>	<b>7</b>
3.1 Class List	7
<b>4 File Index</b>	<b>9</b>
4.1 File List	9
<b>5 Class Documentation</b>	<b>11</b>
5.1 DoublyLinkedList< T > Class Template Reference	11
5.1.1 Constructor & Destructor Documentation	12
5.1.1.1 DoublyLinkedList()	12
5.1.1.2 ~DoublyLinkedList()	12
5.1.2 Member Function Documentation	12
5.1.2.1 begin_iterator()	12
5.1.2.2 clear()	12
5.1.2.3 end_iterator()	12
5.1.2.4 insert_at()	12
5.1.2.5 pop_back()	13
5.1.2.6 pop_front()	13
5.1.2.7 print_backward()	13
5.1.2.8 print_forward()	13
5.1.2.9 push_back()	13
5.1.2.10 push_front()	14
5.1.2.11 remove_at()	14
5.1.2.12 size()	14
5.2 IList< T > Class Template Reference	14
5.2.1 Constructor & Destructor Documentation	15
5.2.1.1 ~IList()	15
5.2.2 Member Function Documentation	15
5.2.2.1 clear()	15
5.2.2.2 insert_at()	15
5.2.2.3 pop_back()	15
5.2.2.4 pop_front()	15
5.2.2.5 print_backward()	16
5.2.2.6 print_forward()	16
5.2.2.7 push_back()	16
5.2.2.8 push_front()	16
5.2.2.9 remove_at()	16
5.3 IListIterator< T > Class Template Reference	17
5.3.1 Constructor & Destructor Documentation	17

5.3.1.1 ~IListIterator()	17
5.3.2 Member Function Documentation	17
5.3.2.1 has_next()	17
5.3.2.2 has_prev()	17
5.3.2.3 next()	18
5.3.2.4 prev()	18
5.3.2.5 value()	18
5.4 ListFactory Class Reference	18
5.4.1 Member Function Documentation	18
5.4.1.1 create()	18
5.5 ListIterator< T > Class Template Reference	19
5.5.1 Constructor & Destructor Documentation	19
5.5.1.1 ListIterator()	19
5.5.2 Member Function Documentation	20
5.5.2.1 get_node()	20
5.5.2.2 has_next()	20
5.5.2.3 has_prev()	20
5.5.2.4 next()	20
5.5.2.5 prev()	20
5.5.2.6 value()	21
5.6 Node< T > Struct Template Reference	21
5.6.1 Constructor & Destructor Documentation	21
5.6.1.1 Node()	21
<b>6 File Documentation</b>	<b>23</b>
6.1 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/DoublyLinkedList.h File Reference	23
6.1.1 Detailed Description	23
6.2 DoublyLinkedList.h	23
6.3 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IList.h File Reference	24
6.3.1 Detailed Description	24
6.4 IList.h	25
6.5 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IListIterator.h File Reference	25
6.5.1 Detailed Description	25
6.6 IListIterator.h	25
6.7 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListFactory.h File Reference	26
6.7.1 Detailed Description	26
6.8 ListFactory.h	26
6.9 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListIterator.h File Reference	26
6.9.1 Detailed Description	27
6.10 ListIterator.h	27
6.11 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/main.cpp File Reference	27
6.11.1 Detailed Description	28

---

6.11.2 Function Documentation . . . . .	28
6.11.2.1 main() . . . . .	28
6.12 C:/Users/Sebastian/Desktop/ListaDwukierunkowa/Node.h File Reference . . . . .	28
6.12.1 Detailed Description . . . . .	29
6.13 Node.h . . . . .	29
<b>Index</b>	<b>31</b>



# Chapter 1

## redme

```
// PoVRay 3.7 Scene File " ... .pov" // author: Rafa3 Curzyd3o // 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 = cam-
era {/*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( AxisLen↵
X, 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 difference

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 difference

} //end difference

} // end of cone -----

## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

IList< T > . . . . .	14
DoublyLinkedList< T > . . . . .	11
IListIterator< T > . . . . .	17
ListIterator< T > . . . . .	19
ListFactory . . . . .	18
Node< T > . . . . .	21



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">DoublyLinkedList&lt; T &gt;</a>	11
<a href="#">IList&lt; T &gt;</a>	14
<a href="#">IListIterator&lt; T &gt;</a>	17
<a href="#">ListFactory</a>	18
<a href="#">ListIterator&lt; T &gt;</a>	19
<a href="#">Node&lt; T &gt;</a>	21



## Chapter 4

# File Index

### 4.1 File List

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

C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ <a href="#">DoublyLinkedList.h</a>	
Lista dwukierunkowa przechowywana na sterzie . . . . .	23
C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ <a href="#">IList.h</a>	
Interfejs dla listy dwukierunkowej (szablon) . . . . .	24
C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ <a href="#">IListIterator.h</a>	
Interfejs iteratora dla listy . . . . .	25
C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ <a href="#">ListFactory.h</a>	
Prosty wzorzec Factory do tworzenia listy dwukierunkowej . . . . .	26
C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ <a href="#">ListIterator.h</a>	
Iterator dla listy dwukierunkowej . . . . .	26
C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ <a href="#">main.cpp</a>	
Testowanie funkcjonalności listy dwukierunkowej . . . . .	27
C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ <a href="#">Node.h</a>	
Definicja węzła listy dwukierunkowej . . . . .	28



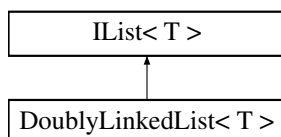


## Chapter 5

# 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 [IList< 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]
```

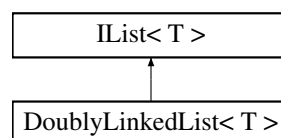
Zwraca liczbe elementow

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

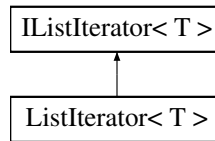
Implemented in [DoublyLinkedList< T >](#).

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

- C:/Users/Sebastian/Desktop/ListuDwukierunkowa/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

---



|          |                     |
|----------|---------------------|
| <i>T</i> | Typ elementów listy |
|----------|---------------------|

**Returns**

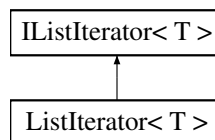
Wskaźnik do nowo utworzonej listy

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**

|              |                                                  |
|--------------|--------------------------------------------------|
| <i>start</i> | Wskaźnik na element, od którego zaczyna iterację |
|--------------|--------------------------------------------------|

## 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 wskaźnik na aktualny węzeł (używane przez listę)

### 5.5.2.2 has\_next()

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

Sprawdza, czy istnieje następny 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 następny 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**  
*Wskaźnik na poprzedni element.*
- [Node](#)< T > \* **next**  
*Wskaźnik na następny 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

|              |                               |
|--------------|-------------------------------|
| <i>value</i> | 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/DoublyLinkedList.h File Reference

Lista dwukierunkowa przechowywana na stercie.

```
#include "IList.h"
#include "Node.h"
#include "IListIterator.h"
#include "ListIterator.h"
#include <cstdlib>
#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.](#)

```
00001 #ifndef DOUBLYLINKEDLIST_H
00002 #define DOUBLYLINKEDLIST_H
00003
00004 #include "IList.h"
00005 #include "Node.h"
00006 #include "IListIterator.h"
00007 #include "ListIterator.h"
00008 #include <cstdlib>
00009
00014
00015 template<typename T>
00016 class DoublyLinkedList : public IList<T> {
00017 private:
```

```

00018     Node<T>* head;
00019     Node<T>* tail;
00020     size_t size_;
00021
00022 public:
00023     DoublyLinkedList();
00024     ~DoublyLinkedList();
00025
00026     void push_front(const T& value) override;
00027     void push_back(const T& value) override;
00028     void insert_at(size_t index, const T& value) override;
00029     void pop_front() override;
00030     void pop_back() override;
00031     void remove_at(size_t index) override;
00032     void clear() override;
00033     void print_forward() const override;
00034     void print_backward() const override;
00035     size_t size() const { return size_; }
00036     ListIterator<T> begin_iterator() const { return ListIterator<T>(head); }
00037     ListIterator<T> end_iterator() const { return ListIterator<T>(tail); }
00038 };
00039
00040 #include "DoublyLinkedList.tpp"
00041
00042 #endif // DOUBLYLINKEDLIST_H

```

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

Interfejs dla listy dwukierunkowej (szablon).

```
#include <cstdint>
```

### 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.](#)

```

00001 #ifndef ILIST_H
00002 #define ILIST_H
00003
00011
00012 #include <cstddef>
00013
00014 template<typename T>
00015 class IList {
00016 public:
00018     virtual ~IList() = default;
00019
00021     virtual void push_front(const T& value) = 0;
00022
00024     virtual void push_back(const T& value) = 0;
00025
00027     virtual void insert_at(size_t index, const T& value) = 0;
00028
00030     virtual void pop_front() = 0;
00031
00033     virtual void pop_back() = 0;
00034
00036     virtual void remove_at(size_t index) = 0;
00037
00039     virtual void clear() = 0;
00040
00042     virtual void print_forward() const = 0;
00043
00045     virtual void print_backward() const = 0;
00046 };
00047
00048 #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.](#)

```

00001 #ifndef ILISTITERATOR_H
00002 #define ILISTITERATOR_H
00003
00011
00012 template<typename T>
00013 class IListIterator {
00014 public:
00016     virtual ~IListIterator() = default;

```

```

00017
00019     virtual bool has_next() const = 0;
00020
00022     virtual bool has_prev() const = 0;
00023
00025     virtual void next() = 0;
00026
00028     virtual void prev() = 0;
00029
00031     virtual T& value() = 0;
00032 };
00033
00034 #endif // ILLISTITERATOR_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.](#)

```

00001 #ifndef LISTFACTORY_H
00002 #define LISTFACTORY_H
00003
00004 #include "DoublyLinkedList.h"
00005
00010 class ListFactory {
00011 public:
00017     template<typename T>
00018     static DoublyLinkedList<T>* create() {
00019         return new DoublyLinkedList<T>();
00020     }
00021 };
00022
00023 #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.

Umożliwia poruszanie się po elementach listy dwukierunkowej w obu kierunkach.

**6.10 ListIterator.h**

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

```

00001 #ifndef LISTITERATOR_H
00002 #define LISTITERATOR_H
00003
00004 #include "IListIterator.h"
00005 #include "Node.h"
00006 #include <stdexcept>
00007
00014 template<typename T>
00015 class ListIterator : public IListIterator<T> {
00016 private:
00017     Node<T>* current;
00018
00019 public:
00024     explicit ListIterator(Node<T>* start) : current(start) {}
00025
00027     bool has_next() const override { return current && current->next; }
00028
00030     bool has_prev() const override { return current && current->prev; }
00031
00033     void next() override {
00034         if (!current) throw std::out_of_range("Iterator: brak elementu");
00035         if (!current->next) throw std::out_of_range("Iterator: brak kolejnego elementu");
00036         current = current->next;
00037     }
00038
00040     void prev() override {
00041         if (!current) throw std::out_of_range("Iterator: brak elementu");
00042         if (!current->prev) throw std::out_of_range("Iterator: brak poprzedniego elementu");
00043         current = current->prev;
00044     }
00045
00047     T& value() override {
00048         if (!current) throw std::out_of_range("Iterator: brak elementu");
00049         return current->data;
00050     }
00051
00053     Node<T>* get_node() const { return current; }
00054 };
00055
00056 #endif // LISTITERATOR_H

```

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

Testowanie funkcjonalności 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 usunieciech

< Usuwa element pod indeksem 1

< Wyswietlenie listy po `remove_at`

< Zwolnienie pamieci

## 6.12 C:/Users/Sebastian/Desktop/ListaNwukierunkowa/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.](#)

```
00001 #ifndef NODE_H
00002 #define NODE_H
00003
00010
00011 template<typename T>
00012 struct Node {
00013     T data;
00014     Node<T>* prev;
00015     Node<T>* next;
00016
00021     Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}
00022 };
00023
00024 #endif // NODE_H
```



# Index

- ~DoublyLinkedList
  - DoublyLinkedList< T >, 12
- ~IList
  - IList< T >, 15
- ~IListIterator
  - IListIterator< T >, 17
- begin\_iterator
  - DoublyLinkedList< T >, 12
- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/DoublyLinkedList.h, 14
- 23
- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IList.h, 24, 25
- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/IListIterator.h, 25
- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListFactory.h, 26
- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/ListIterator.h, 26, 27
- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/main.cpp, 27
- C:/Users/Sebastian/Desktop/ListaDwukierunkowa/Node.h, 28, 29
- clear
  - DoublyLinkedList< T >, 12
  - IList< T >, 15
- create
  - ListFactory, 18
- DoublyLinkedList
  - DoublyLinkedList< T >, 12
- DoublyLinkedList< T >, 11
  - ~DoublyLinkedList, 12
  - begin\_iterator, 12
  - clear, 12
  - DoublyLinkedList, 12
  - end\_iterator, 12
  - insert\_at, 12
  - pop\_back, 12
  - pop\_front, 13
  - print\_backward, 13
  - print\_forward, 13
  - push\_back, 13
  - push\_front, 13
  - remove\_at, 14
  - size, 14
- end\_iterator
  - DoublyLinkedList< T >, 12
- get\_node
  - ListIterator< T >, 20
- has\_next
  - IListIterator< T >, 17
  - ListIterator< T >, 20
- has\_prev
  - IListIterator< T >, 17
  - ListIterator< T >, 20
- IList, 15
- clear, 15
- insert\_at, 15
- pop\_back, 15
- pop\_front, 15
- print\_backward, 15
- print\_forward, 16
- push\_back, 16
- push\_front, 16
- remove\_at, 16
- IListIterator< T >, 17
- ~IListIterator, 17
- has\_next, 17
- has\_prev, 17
- next, 17
- prev, 18
- value, 18
- insert\_at
  - DoublyLinkedList< T >, 12
  - IList< T >, 15
- ListFactory, 18
  - create, 18
- ListIterator
  - ListIterator< T >, 19
- ListIterator< T >, 19
  - get\_node, 20
  - has\_next, 20
  - has\_prev, 20
  - ListIterator, 19
  - next, 20
  - prev, 20
  - value, 20
- main
  - main.cpp, 28
- main.cpp
  - main, 28
- next

- [IListIterator< T >](#), [17](#)
  - [ListIterator< T >](#), [20](#)
- Node
  - [Node< T >](#), [21](#)
- [Node< T >](#), [21](#)
  - [Node](#), [21](#)
- pop\_back
  - [DoublyLinkedList< T >](#), [12](#)
  - [IList< T >](#), [15](#)
- pop\_front
  - [DoublyLinkedList< T >](#), [13](#)
  - [IList< T >](#), [15](#)
- prev
  - [IListIterator< T >](#), [18](#)
  - [ListIterator< T >](#), [20](#)
- print\_backward
  - [DoublyLinkedList< T >](#), [13](#)
  - [IList< T >](#), [15](#)
- print\_forward
  - [DoublyLinkedList< T >](#), [13](#)
  - [IList< T >](#), [16](#)
- push\_back
  - [DoublyLinkedList< T >](#), [13](#)
  - [IList< T >](#), [16](#)
- push\_front
  - [DoublyLinkedList< T >](#), [13](#)
  - [IList< T >](#), [16](#)
- redme, [1](#)
- remove\_at
  - [DoublyLinkedList< T >](#), [14](#)
  - [IList< T >](#), [16](#)
- size
  - [DoublyLinkedList< T >](#), [14](#)
- value
  - [IListIterator< T >](#), [18](#)
  - [ListIterator< T >](#), [20](#)