

pamsi  
0.5

Wygenerowano przez Doxygen 1.8.9.1

N, 3 kwi 2016 20:04:30



# Spis treści

<b>1</b>	<b>Strona główna</b>	<b>1</b>
1.1	Dokumentacja klas w repozytorium pamsi.	1
1.2	Przykład uruchomienia testu	1
1.3	Inne przykłady	1
1.3.1	Test sortowania bąbelkowego	1
<b>2</b>	<b>Indeks hierarchiczny</b>	<b>3</b>
2.1	Hierarchia klas	3
<b>3</b>	<b>Indeks klas</b>	<b>5</b>
3.1	Lista klas	5
<b>4</b>	<b>Indeks plików</b>	<b>7</b>
4.1	Lista plików	7
<b>5</b>	<b>Dokumentacja klas</b>	<b>9</b>
5.1	Dokumentacja klasy ContinueException	9
5.1.1	Opis szczegółowy	10
5.1.2	Dokumentacja konstruktora i destruktora	10
5.1.2.1	ContinueException	10
5.1.3	Dokumentacja funkcji składowych	10
5.1.3.1	getError	10
5.1.4	Dokumentacja atrybutów składowych	10
5.1.4.1	cause	10
5.2	Dokumentacja klasy CriticalException	10
5.2.1	Opis szczegółowy	11
5.2.2	Dokumentacja konstruktora i destruktora	11
5.2.2.1	CriticalException	11
5.2.3	Dokumentacja funkcji składowych	12
5.2.3.1	getError	12
5.2.4	Dokumentacja atrybutów składowych	12
5.2.4.1	cause	12
5.3	Dokumentacja klasy Exception	12

5.3.1	Opis szczegółowy	13
5.3.2	Dokumentacja konstruktora i destruktora	13
5.3.2.1	Exception	13
5.3.2.2	Exception	13
5.3.3	Dokumentacja funkcji składowych	13
5.3.3.1	getError	13
5.3.4	Dokumentacja atrybutów składowych	13
5.3.4.1	cause	13
5.4	Dokumentacja szablonu klasy IKolejka< T >	13
5.4.1	Opis szczegółowy	14
5.4.2	Dokumentacja konstruktora i destruktora	14
5.4.2.1	~IKolejka	14
5.4.3	Dokumentacja funkcji składowych	14
5.4.3.1	dequeue	14
5.4.3.2	enqueue	15
5.4.3.3	get	15
5.4.3.4	isEmpty	15
5.5	Dokumentacja szablonu klasy ILista< T >	15
5.5.1	Opis szczegółowy	16
5.5.2	Dokumentacja konstruktora i destruktora	16
5.5.2.1	~ILista	16
5.5.3	Dokumentacja funkcji składowych	17
5.5.3.1	add	17
5.5.3.2	add	17
5.5.3.3	get	17
5.5.3.4	isEmpty	17
5.5.3.5	remove	18
5.5.3.6	remove	18
5.5.3.7	size	18
5.6	Dokumentacja klasy IRunnable	18
5.6.1	Opis szczegółowy	19
5.6.2	Dokumentacja konstruktora i destruktora	19
5.6.2.1	~IRunnable	19
5.6.3	Dokumentacja funkcji składowych	19
5.6.3.1	prepare	19
5.6.3.2	run	19
5.7	Dokumentacja klasy IStoper	20
5.7.1	Opis szczegółowy	20
5.7.2	Dokumentacja konstruktora i destruktora	20
5.7.2.1	~IStoper	20

5.7.3	Dokumentacja funkcji składowych	21
5.7.3.1	getElapsedTimeMs	21
5.7.3.2	start	21
5.7.3.3	stop	21
5.8	Dokumentacja szablonu klasy IStos< T >	21
5.8.1	Opis szczegółowy	22
5.8.2	Dokumentacja konstruktora i destruktora	22
5.8.2.1	~IStos	22
5.8.3	Dokumentacja funkcji składowych	22
5.8.3.1	get	22
5.8.3.2	isEmpty	22
5.8.3.3	pull	23
5.8.3.4	push	23
5.9	Dokumentacja szablonu klasy ltabn< T >	23
5.9.1	Opis szczegółowy	24
5.9.2	Dokumentacja konstruktora i destruktora	24
5.9.2.1	~ltabn	24
5.9.3	Dokumentacja funkcji składowych	24
5.9.3.1	add	24
5.9.3.2	add	25
5.9.3.3	aSize	25
5.9.3.4	bubblesort	25
5.9.3.5	isEmpty	26
5.9.3.6	nOE	26
5.9.3.7	operator[]	26
5.9.3.8	operator[]	26
5.9.3.9	remove	26
5.9.3.10	remove	26
5.9.3.11	show	26
5.9.3.12	showElems	27
5.10	Dokumentacja szablonu klasy Kolejka< T >	27
5.10.1	Opis szczegółowy	28
5.10.2	Dokumentacja konstruktora i destruktora	28
5.10.2.1	Kolejka	28
5.10.2.2	~Kolejka	28
5.10.3	Dokumentacja funkcji składowych	29
5.10.3.1	dequeue	29
5.10.3.2	enqueue	29
5.10.3.3	get	29
5.10.3.4	isEmpty	29

5.11 Dokumentacja szablonu klasy Lista< T > . . . . .	30
5.11.1 Opis szczegółowy . . . . .	31
5.11.2 Dokumentacja konstruktora i destruktora . . . . .	31
5.11.2.1 Lista . . . . .	31
5.11.2.2 ~Lista . . . . .	31
5.11.3 Dokumentacja funkcji składowych . . . . .	31
5.11.3.1 add . . . . .	31
5.11.3.2 add . . . . .	31
5.11.3.3 get . . . . .	31
5.11.3.4 isEmpty . . . . .	32
5.11.3.5 remove . . . . .	32
5.11.3.6 remove . . . . .	33
5.11.3.7 size . . . . .	33
5.12 Dokumentacja klasy lista_test . . . . .	33
5.12.1 Opis szczegółowy . . . . .	34
5.12.2 Dokumentacja konstruktora i destruktora . . . . .	34
5.12.2.1 lista_test . . . . .	34
5.12.2.2 ~lista_test . . . . .	34
5.12.3 Dokumentacja funkcji składowych . . . . .	34
5.12.3.1 prepare . . . . .	34
5.12.3.2 run . . . . .	35
5.13 Dokumentacja klasy Stoper . . . . .	35
5.13.1 Opis szczegółowy . . . . .	36
5.13.2 Dokumentacja funkcji składowych . . . . .	36
5.13.2.1 getElapsedTimeMs . . . . .	37
5.13.2.2 start . . . . .	38
5.13.2.3 stop . . . . .	38
5.14 Dokumentacja szablonu klasy Stos< T > . . . . .	38
5.14.1 Opis szczegółowy . . . . .	39
5.14.2 Dokumentacja konstruktora i destruktora . . . . .	39
5.14.2.1 Stos . . . . .	39
5.14.2.2 ~Stos . . . . .	39
5.14.3 Dokumentacja funkcji składowych . . . . .	40
5.14.3.1 get . . . . .	40
5.14.3.2 isEmpty . . . . .	40
5.14.3.3 pull . . . . .	40
5.14.3.4 push . . . . .	41
5.15 Dokumentacja szablonu klasy tabn< T > . . . . .	41
5.15.1 Opis szczegółowy . . . . .	42
5.15.2 Dokumentacja konstruktora i destruktora . . . . .	42

5.15.2.1	tabn	42
5.15.2.2	~tabn	42
5.15.3	Dokumentacja funkcji składowych	43
5.15.3.1	add	43
5.15.3.2	add	43
5.15.3.3	aSize	43
5.15.3.4	bubblesort	43
5.15.3.5	isEmpty	43
5.15.3.6	nOE	44
5.15.3.7	operator[]	44
5.15.3.8	operator[]	44
5.15.3.9	remove	44
5.15.3.10	remove	45
5.15.3.11	show	45
5.15.3.12	showElems	45
5.16	Dokumentacja klasy tabn_test	45
5.16.1	Opis szczegółowy	46
5.16.2	Dokumentacja konstruktora i destruktora	46
5.16.2.1	tabn_test	47
5.16.2.2	~tabn_test	47
5.16.3	Dokumentacja funkcji składowych	47
5.16.3.1	prepare	47
5.16.3.2	run	47
<b>6</b>	<b>Dokumentacja plików</b>	<b>49</b>
6.1	Dokumentacja pliku except.cpp	49
6.2	Dokumentacja pliku except.hh	49
6.2.1	Opis szczegółowy	51
6.3	Dokumentacja pliku kolejka.cpp	51
6.4	Dokumentacja pliku kolejka.hh	52
6.5	Dokumentacja pliku lista.cpp	53
6.6	Dokumentacja pliku lista.hh	53
6.7	Dokumentacja pliku main.cpp	55
6.7.1	Opis szczegółowy	55
6.7.2	Dokumentacja funkcji	55
6.7.2.1	dumpToFile	55
6.7.2.2	main	56
6.7.2.3	printOnscreen	56
6.8	Dokumentacja pliku main.hh	57
6.8.1	Dokumentacja funkcji	58

6.8.1.1	dumpToFile . . . . .	58
6.8.1.2	printOnscreen . . . . .	58
6.9	Dokumentacja pliku run.cpp . . . . .	59
6.10	Dokumentacja pliku run.hh . . . . .	59
6.10.1	Opis szczegółowy . . . . .	61
6.11	Dokumentacja pliku stoper.cpp . . . . .	61
6.12	Dokumentacja pliku stoper.hh . . . . .	61
6.13	Dokumentacja pliku stos.cpp . . . . .	62
6.14	Dokumentacja pliku stos.hh . . . . .	63
6.15	Dokumentacja pliku tabl.cpp . . . . .	65
6.16	Dokumentacja pliku tabl.hh . . . . .	65
6.16.1	Opis szczegółowy . . . . .	67
6.16.2	Dokumentacja definicji . . . . .	67
6.16.2.1	SIZE . . . . .	67
<b>Indeks</b>		<b>69</b>



# Rozdział 1

## Strona główna

### 1.1 Dokumentacja klas w repozytorium pamsi.

Ten dokument zawiera dokumentację klas znajdujących się w plikach repozytorium pamsi.

### 1.2 Przykład uruchomienia testu

```
IRunnable * runner = new lista_test;
IStoper * stoper = new Stoper;
unsigned int testSize = 100;
string outputFile = "file123";
try {
    runner->prepare(testSize);
    stoper->start();
    runner->run();
    stoper->stop();
    printOnscreen(testSize, stoper);
    dumpToFile(outputFile, testSize, stoper);
}
catch (ContinueException &cex) {
    std::cout << "Exception: " << cex.getError() << std::endl;
}
catch (CriticalException & crit_ex) {
    std::cout << "Critical: " << crit_ex.getError() << std::endl;
    delete stoper;
    delete runner;
    return -1;
}
catch (...) {
    std::cerr << "Unexpected exception!" << std::endl;
    delete stoper;
    delete runner;
    return -1;
}
delete stoper;
delete runner;
```

### 1.3 Inne przykłady

#### 1.3.1 Test sortowania bąbelkowego

```
Itabn<int> * tablica = new tabn<int>;
tablica->add(7);
tablica->add(4);
tablica->add(1);
tablica->add(9);
tablica->add(10);
tablica->add(94);
tablica->add(-4);
tablica->add(5);
tablica->add(15);
tablica->add(8);
```

```
tablica->add(9);  
tablica->add(17);  
tablica->add(19);  
tablica->showElems();  
tablica->bubblesort();  
tablica->showElems();  
delete tablica;
```

## Rozdział 2

# Indeks hierarchiczny

### 2.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Exception . . . . .	12
ContinueException . . . . .	9
CriticalException . . . . .	10
IKolejka< T > . . . . .	13
Kolejka< T > . . . . .	27
ILista< T > . . . . .	15
Lista< T > . . . . .	30
ILista< std::string > . . . . .	15
IRunnable . . . . .	18
lista_test . . . . .	33
tabn_test . . . . .	45
IStoper . . . . .	20
Stoper . . . . .	35
IStos< T > . . . . .	21
Stos< T > . . . . .	38
Itabn< T > . . . . .	23
tabn< T > . . . . .	41
Itabn< int > . . . . .	23



## Rozdział 3

# Indeks klas

### 3.1 Lista klas

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

<a href="#">ContinueException</a>	
Wyjątek, który mimo pojawienia się, pozwala na dalsze poprawne działanie programu . . . . .	9
<a href="#">CriticalException</a>	
Wyjątek krytyczny, wymagający zamknięcia programu . . . . .	10
<a href="#">Exception</a>	
Ogólny wyjątek . . . . .	12
<a href="#">IKolejka&lt; T &gt;</a>	
Interfejs klasy <a href="#">Kolejka</a> Definiuje operacje dostępne dla klasy <a href="#">Kolejka</a> . . . . .	13
<a href="#">ILista&lt; T &gt;</a>	
Interfejs listy . . . . .	15
<a href="#">IRunnable</a>	
Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm . . . . .	18
<a href="#">IStoper</a>	
Plik definiuje stoper, obliczający czas wykonywania badanych funkcji . . . . .	20
<a href="#">IStos&lt; T &gt;</a>	
Interfejs stosu . . . . .	21
<a href="#">Itabn&lt; T &gt;</a>	
Interfejs klasy tabn . . . . .	23
<a href="#">Kolejka&lt; T &gt;</a>	
Klasa modeluje kolejkę . . . . .	27
<a href="#">Lista&lt; T &gt;</a>	
Klasa lista . . . . .	30
<a href="#">lista_test</a>	
Definiuje sposób testowania wypełniania listy . . . . .	33
<a href="#">Stoper</a>	
Klasa stoper implementująca interfejs <a href="#">IStoper</a> . . . . .	35
<a href="#">Stos&lt; T &gt;</a>	
Klasa <a href="#">Stos</a> . . . . .	38
<a href="#">tabn&lt; T &gt;</a>	
Modeluje tablicę dynamicznie rozszerzalną . . . . .	41
<a href="#">tabn_test</a>	
Definiuje sposób testowania wypełniania tablicy tabn . . . . .	45



## Rozdział 4

# Indeks plików

### 4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

<a href="#">except.cpp</a>	49
<a href="#">except.hh</a>	
Plik zawiera definicje wyjątków	49
<a href="#">kolejka.cpp</a>	51
<a href="#">kolejka.hh</a>	52
<a href="#">lista.cpp</a>	53
<a href="#">lista.hh</a>	53
<a href="#">main.cpp</a>	
Główny plik programu	55
<a href="#">main.hh</a>	57
<a href="#">run.cpp</a>	59
<a href="#">run.hh</a>	
Plik definiuje interfejs <a href="#">IRunnable</a> , ujednolicający klasy umożliwiające badanie algorytmów	59
<a href="#">stoper.cpp</a>	61
<a href="#">stoper.hh</a>	61
<a href="#">stos.cpp</a>	62
<a href="#">stos.hh</a>	63
<a href="#">tabl.cpp</a>	65
<a href="#">tabl.hh</a>	
Definicja interfejsu <a href="#">Itabn</a> , klasy <a href="#">tabn</a> oraz klasy <a href="#">tabn_test</a>	65





## Rozdział 5

# Dokumentacja klas

### 5.1 Dokumentacja klasy ContinueException

Wyjątek, który mimo pojawienia się, pozwala na dalsze poprawne działanie programu.

```
#include <except.hh>
```

Diagram dziedziczenia dla ContinueException

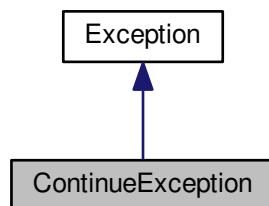
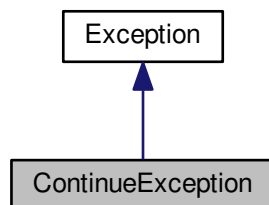


Diagram współpracy dla ContinueException:



## Metody publiczne

- [ContinueException](#) (std::string description)
- std::string [getError](#) ()

## Atrybuty publiczne

- std::string [cause](#)

### 5.1.1 Opis szczegółowy

Wyjątek, który mimo pojawienia się, pozwala na dalsze poprawne działanie programu.

Definicja w linii 49 pliku except.hh.

### 5.1.2 Dokumentacja konstruktora i destruktor

#### 5.1.2.1 `ContinueException::ContinueException ( std::string description )` `[inline]`

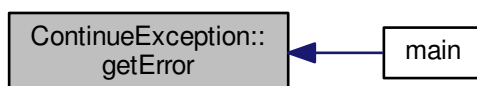
Definicja w linii 55 pliku except.hh.

### 5.1.3 Dokumentacja funkcji składowych

#### 5.1.3.1 `std::string ContinueException::getError ( )` `[inline]`

Definicja w linii 58 pliku except.hh.

Oto graf wywoływań tej funkcji:



### 5.1.4 Dokumentacja atrybutów składowych

#### 5.1.4.1 `std::string ContinueException::cause`

Definicja w linii 51 pliku except.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [except.hh](#)

## 5.2 Dokumentacja klasy `CriticalException`

Wyjątek krytyczny, wymagający zamknięcia programu.

```
#include <except.hh>
```

Diagram dziedziczenia dla `CriticalException`

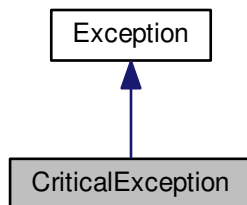
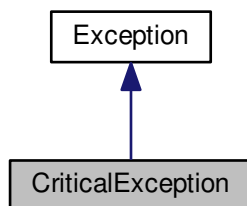


Diagram współpracy dla `CriticalException`:



### Metody publiczne

- `CriticalException` (`std::string description`)
- `std::string getError ()`

### Atrybuty publiczne

- `std::string cause`

#### 5.2.1 Opis szczegółowy

Wyjątek krytyczny, wymagający zamknięcia programu.

Definicja w linii 32 pliku `except.hh`.

#### 5.2.2 Dokumentacja konstruktora i destruktor

##### 5.2.2.1 `CriticalException::CriticalException ( std::string description ) [inline]`

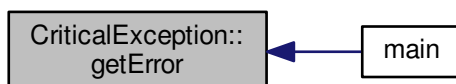
Definicja w linii 38 pliku `except.hh`.

### 5.2.3 Dokumentacja funkcji składowych

#### 5.2.3.1 `std::string CriticalException::getError ( ) [inline]`

Definicja w linii 41 pliku `except.hh`.

Oto graf wywołań tej funkcji:



### 5.2.4 Dokumentacja atrybutów składowych

#### 5.2.4.1 `std::string CriticalException::cause`

Definicja w linii 34 pliku `except.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

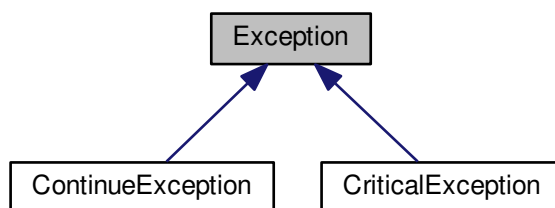
- [except.hh](#)

## 5.3 Dokumentacja klasy Exception

Ogólny wyjątek.

```
#include <except.hh>
```

Diagram dziedziczenia dla Exception



### Metody publiczne

- [Exception \( \)](#)
- [Exception \(std::string description\)](#)
- `std::string` [getError \( \)](#)

## Atrybuty publiczne

- `std::string` `cause`

### 5.3.1 Opis szczegółowy

Ogólny wyjątek.

Definicja w linii 15 pliku `except.hh`.

### 5.3.2 Dokumentacja konstruktora i destruktora

#### 5.3.2.1 `Exception::Exception ( )` `[inline]`

Definicja w linii 19 pliku `except.hh`.

#### 5.3.2.2 `Exception::Exception ( std::string description )` `[inline]`

Definicja w linii 21 pliku `except.hh`.

### 5.3.3 Dokumentacja funkcji składowych

#### 5.3.3.1 `std::string Exception::getError ( )` `[inline]`

Definicja w linii 24 pliku `except.hh`.

### 5.3.4 Dokumentacja atrybutów składowych

#### 5.3.4.1 `std::string Exception::cause`

Definicja w linii 17 pliku `except.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

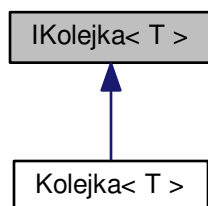
- `except.hh`

## 5.4 Dokumentacja szablonu klasy `IKolejka< T >`

Interfejs klasy `Kolejka` Definiuje operacje dostępne dla klasy `Kolejka`.

```
#include <kolejka.hh>
```

Diagram dziedziczenia dla `IKolejka< T >`



## Metody publiczne

- virtual void `enqueue` (T)=0  
*Dodaje element na koniec kolejki.*
- virtual T `dequeue` (void)=0  
*Usuwa i zwraca element z początku kolejki.*
- virtual bool `isEmpty` (void)=0  
*Sprawdza, czy kolejka nie jest pusta.*
- virtual T `get` (void)=0  
*Zwraca element z początku kolejki bez usuwania.*
- virtual `~IKolejka` ()  
*Destruktor wirtualny interfejsu.*

### 5.4.1 Opis szczegółowy

```
template<class T>class IKolejka< T >
```

Interfejs klasy `Kolejka` Definiuje operacje dostępne dla klasy `Kolejka`.

Definicja w linii 15 pliku `kolejka.hh`.

### 5.4.2 Dokumentacja konstruktora i destruktora

```
5.4.2.1 template<class T> virtual IKolejka< T >::~~IKolejka ( ) [inline],[virtual]
```

Destruktor wirtualny interfejsu.

Definicja w linii 47 pliku `kolejka.hh`.

### 5.4.3 Dokumentacja funkcji składowych

```
5.4.3.1 template<class T> virtual T IKolejka< T >::dequeue ( void ) [pure virtual]
```

Usuwa i zwraca element z początku kolejki.

## Zwracane wartości

<i>T</i>	element z początku kolejki
----------	----------------------------

Implementowany w [Kolejka< T >](#).

5.4.3.2 `template<class T> virtual void IKolejka< T >::enqueue ( T ) [pure virtual]`

Dodaje element na koniec kolejki.

## Parametry

<i>element</i>	- element do umieszczenia w kolejce
----------------	-------------------------------------

Implementowany w [Kolejka< T >](#).

5.4.3.3 `template<class T> virtual T IKolejka< T >::get ( void ) [pure virtual]`

Zwraca element z początku kolejki bez usuwania.

## Ostrzeżenie

Uwaga! Próba podglądu elementu z pustej kolejki spowoduje wyrzucenie wyjątku, zamknięcie programu i zwrócenie -1.

Sprawdź dokumentację metody [Kolejka<T>::get\(void\)](#).

Implementowany w [Kolejka< T >](#).

5.4.3.4 `template<class T> virtual bool IKolejka< T >::isEmpty ( void ) [pure virtual]`

Sprawdza, czy kolejka nie jest pusta.

## Zwracane wartości

<i>0</i>	gdy niepusta
<i>1</i>	gdy pusta

Implementowany w [Kolejka< T >](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

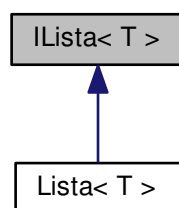
- [kolejka.hh](#)

## 5.5 Dokumentacja szablonu klasy ILista&lt; T &gt;

Interfejs listy.

```
#include <lista.hh>
```

Diagram dziedziczenia dla `ILista< T >`



## Metody publiczne

- virtual void `add` (T, int)=0  
*Dodaje element do zadanego miejsca listy.*
- virtual void `add` (T)=0  
*Dodaje element na koniec listy.*
- virtual T `remove` (int)=0  
*Usuwa element z zadanego miejsca listy.*
- virtual T `remove` (void)=0  
*Usuwa element z końca listy.*
- virtual bool `isEmpty` (void)=0  
*Sprawdza, czy lista jest pusta.*
- virtual T `get` (int)=0  
*Zwraca element z zadanego miejsca bez usunięcia.*
- virtual int `size` (void)=0  
*Zwraca ilość elementów w liście.*
- virtual `~ILista` ()  
*Destruktor wirtualny interfejsu `ILista`.*

### 5.5.1 Opis szczegółowy

```
template<class T>class ILista< T >
```

Interfejs listy.

Definiuje dostępne operacje na klasie `Lista`

Definicja w linii 18 pliku `lista.hh`.

### 5.5.2 Dokumentacja konstruktora i destruktora

5.5.2.1 `template<class T> virtual ILista< T >::~~ILista ( ) [inline],[virtual]`

Destruktor wirtualny interfejsu `ILista`.

Definicja w linii 76 pliku `lista.hh`.



### 5.5.3 Dokumentacja funkcji składowych

#### 5.5.3.1 `template<class T> virtual void ILista< T >::add ( T, int ) [pure virtual]`

Dodaje element do zadanego miejsca listy.

Jeśli następuje próba dodania elementu w miejscu istniejącego, następuje przesunięcie następujących po nim elementów na następne pozycje

#### Nota

Próba dodania elementu na miejsce dalsze niż pierwsze następujące po obecnie istniejącym spowoduje wyrzucenie wyjątku i niewykonanie akcji.

Implementowany w [Lista< T >](#).

Oto graf wywoływań tej funkcji:



#### 5.5.3.2 `template<class T> virtual void ILista< T >::add ( T ) [pure virtual]`

Dodaje element na koniec listy.

Implementowany w [Lista< T >](#).

#### 5.5.3.3 `template<class T> virtual T ILista< T >::get ( int ) [pure virtual]`

Zwraca element z zadanego miejsca bez usunięcia.

Zwracane wartości

<i>T</i>	element w zadanym miejscu
----------	---------------------------

#### Ostrzeżenie

Próba podglądu elementu nieistniejącego spowoduje wyrzucenie wyjątku, zakończenie programu i zwrócenie -1.

Sprawdź dokumentację metody [Lista<T>::get\(int\)](#).

Implementowany w [Lista< T >](#).

#### 5.5.3.4 `template<class T> virtual bool ILista< T >::isEmpty ( void ) [pure virtual]`

Sprawdza, czy lista jest pusta.

## Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementowany w [Lista< T >](#).

5.5.3.5 `template<class T> virtual T ILista< T >::remove ( int ) [pure virtual]`

Usuwa element z zadanego miejsca listy.

Jeśli usunięcie następuje w środku listy, następujące po usuwanym elementy są przesuwane o jedną pozycję wcześniej.

## Zwracane wartości

T	Usunięty element
---	------------------

## Ostrzeżenie

Próba usunięcia elementu nieistniejącego lub z pustej listy spowoduje wyrzucenie wyjątku, zakończenie programu i zwrócenie -1.  
Sprawdź dokumentację metody [Lista<T>::remove\(int\)](#).

Implementowany w [Lista< T >](#).

5.5.3.6 `template<class T> virtual T ILista< T >::remove ( void ) [pure virtual]`

Usuwa element z końca listy.

Implementowany w [Lista< T >](#).

5.5.3.7 `template<class T> virtual int ILista< T >::size ( void ) [pure virtual]`

Zwraca ilość elementów w liście.

## Zwracane wartości

int	ilość elementów
-----	-----------------

Implementowany w [Lista< T >](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

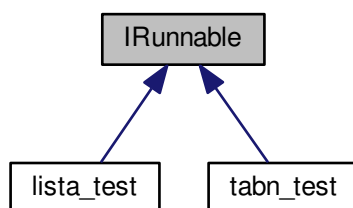
- [lista.hh](#)

## 5.6 Dokumentacja klasy IRunnable

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

```
#include <run.hh>
```

Diagram dziedziczenia dla IRunnable



### Metody publiczne

- virtual bool `prepare` (int)=0  
*Przygotowanie badań*
- virtual bool `run` ()=0  
*Przeprowadzanie badań*
- virtual `~IRunnable` ()  
*Destruktor wirtualny `IRunnable`.*

#### 5.6.1 Opis szczegółowy

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

Definicja w linii 16 pliku run.hh.

#### 5.6.2 Dokumentacja konstruktora i destruktora

5.6.2.1 virtual IRunnable::~~IRunnable ( ) [inline],[virtual]

Destruktor wirtualny `IRunnable`.

Definicja w linii 35 pliku run.hh.

#### 5.6.3 Dokumentacja funkcji składowych

5.6.3.1 virtual bool IRunnable::prepare ( int ) [pure virtual]

Przygotowanie badań

Zwracane wartości

<code>true</code>	zawsze
-------------------	--------

Implementowany w `tabn_test` i `lista_test`.

5.6.3.2 virtual bool IRunnable::run ( ) [pure virtual]

Przeprowadzanie badań

## Zwracane wartości

<i>true</i>	zawsze
-------------	--------

Implementowany w [tabn\\_test](#) i [lista\\_test](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

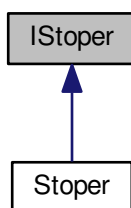
- [run.hh](#)

## 5.7 Dokumentacja klasy IStoper

Plik definiuje stoper, obliczający czas wykonywania badanych funkcji.

```
#include <stoper.hh>
```

Diagram dziedziczenia dla IStoper



### Metody publiczne

- virtual void [start](#) (void)=0
- virtual void [stop](#) (void)=0
- virtual long double [getElapsedTimeMs](#) (void)=0
- virtual [~IStoper](#) ()

#### 5.7.1 Opis szczegółowy

Plik definiuje stoper, obliczający czas wykonywania badanych funkcji.

Obliczanie czasu działania fragmentu programu na podstawie przykładu: <http://en.cppreference.com/w/cpp/chrono>

Interfejs [IStoper](#)

Definicja w linii 20 pliku stoper.hh.

#### 5.7.2 Dokumentacja konstruktora i destruktor

5.7.2.1 virtual IStoper::~IStoper ( ) [inline],[virtual]

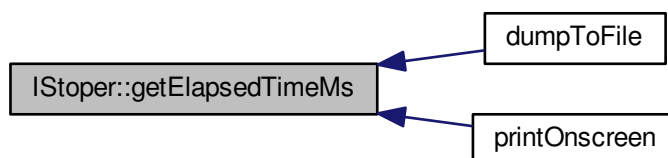
Definicja w linii 26 pliku stoper.hh.

### 5.7.3 Dokumentacja funkcji składowych

5.7.3.1 `virtual long double IStoper::getElapsedTimeMs ( void ) [pure virtual]`

Implementowany w [Stoper](#).

Oto graf wywoływań tej funkcji:



5.7.3.2 `virtual void IStoper::start ( void ) [pure virtual]`

Implementowany w [Stoper](#).

5.7.3.3 `virtual void IStoper::stop ( void ) [pure virtual]`

Implementowany w [Stoper](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

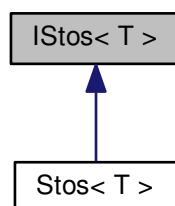
- [stoper.hh](#)

## 5.8 Dokumentacja szablonu klasy IStos< T >

Interfejs stosu.

```
#include <stos.hh>
```

Diagram dziedziczenia dla IStos< T >



## Metody publiczne

- virtual void [push](#) (T)=0  
*Umieszcza element na szczycie stosu.*
- virtual T [pull](#) (void)=0  
*Zdejmuje element ze szczytu stosu.*
- virtual bool [isEmpty](#) (void)=0  
*Sprawdza, czy stos jest pusty.*
- virtual T [get](#) (void)=0  
*Zwraca element ze szczytu stosu bez jego usuwania.*
- virtual [~IStos](#) ()  
*Destruktor wirtualny [IStos](#).*

### 5.8.1 Opis szczegółowy

```
template<class T>class IStos< T >
```

Interfejs stosu.

Definiuje dostępne operacje na klasie [Stos](#)

Definicja w linii 16 pliku stos.hh.

### 5.8.2 Dokumentacja konstruktora i destruktora

5.8.2.1 `template<class T> virtual IStos< T>::~~IStos ( ) [inline],[virtual]`

Destruktor wirtualny [IStos](#).

Definicja w linii 53 pliku stos.hh.

### 5.8.3 Dokumentacja funkcji składowych

5.8.3.1 `template<class T> virtual T IStos< T>::get ( void ) [pure virtual]`

Zwraca element ze szczytu stosu bez jego usuwania.

Zwracane wartości

<a href="#">T</a>	element ze szczytu stosu
-------------------	--------------------------

#### Ostrzeżenie

Uwaga! Próba odczytania elementu z pustego stosu spowoduje wyrzucenie wyjątku, zamknięcie programu i zwrócenie -1.

Sprawdź dokumentację metody [Stos<T>::get\(void\)](#).

Implementowany w [Stos< T >](#).

5.8.3.2 `template<class T> virtual bool IStos< T>::isEmpty ( void ) [pure virtual]`

Sprawdza, czy stos jest pusty.

Zwracane wartości

0	gdy niepusty
1	gdy pusty

Implementowany w [Stos< T >](#).

5.8.3.3 `template<class T> virtual T Istos< T >::pull ( void ) [pure virtual]`

Zdejmuje element ze szczytu stosu.

Zwracane wartości

T	element ze szczytu stosu
---	--------------------------

Ostrzeżenie

Uwaga! Próba zdjęcia elementu z pustego stosu spowoduje wyrzucenie wyjątku, zamknięcie programu i zwrócenie -1.

Sprawdź dokumentację metody [Stos<T>::pull\(void\)](#).

Implementowany w [Stos< T >](#).

5.8.3.4 `template<class T> virtual void Istos< T >::push ( T ) [pure virtual]`

Umieszcza element na szczycie stosu.

Parametry

<i>element</i>	- element do umieszczenia na stosie
----------------	-------------------------------------

Implementowany w [Stos< T >](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

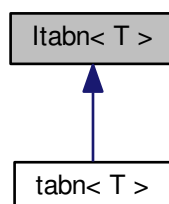
- [stos.hh](#)

## 5.9 Dokumentacja szablonu klasy Itabn< T >

Interfejs klasy tabn.

```
#include <tabl.hh>
```

Diagram dziedziczenia dla Itabn< T >



## Metody publiczne

- virtual bool `isEmpty` (void)=0  
*Sprawdza, czy tablica jest pusta.*
- virtual void `add` (T)=0  
*Dodaje element na koniec tablicy.*
- virtual void `add` (T, int)=0  
*Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.*
- virtual T `remove` ()=0  
*Usuwa i zwraca element z końca tablicy.*
- virtual T `remove` (int)=0  
*Usuwa i zwraca wybrany element z tablicy.*
- virtual T `show` (int)=0  
*Zwraca żądany element, o ile istnieje.*
- virtual void `showElems` (void)=0  
*Wyświetla elementy tablicy.*
- virtual int `nOE` (void)=0  
*Zwraca liczbę elementów w tablicy.*
- virtual int `aSize` (void)=0  
*Zwraca ilość miejsca w tablicy.*
- virtual T & `operator[]` (int)=0  
*Pozwala na dostęp do dowolnego elementu.*
- virtual T `operator[]` (int) const =0  
*Pozwala na dostęp do dowolnego elementu.*
- virtual `~Itabn` ()  
*Destruktor wirtualny interfejsu.*
- virtual void `bubblesort` ()=0  
*Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.*

### 5.9.1 Opis szczegółowy

```
template<class T>class Itabn< T >
```

Interfejs klasy tabn.

Definiuje jednolity sposób dostępu do tablicy rozszerzalnej.

Definicja w linii 20 pliku tabl.hh.

### 5.9.2 Dokumentacja konstruktora i destruktora

5.9.2.1 `template<class T> virtual Itabn< T >::~~Itabn ( ) [inline],[virtual]`

Destruktor wirtualny interfejsu.

Definicja w linii 83 pliku tabl.hh.

### 5.9.3 Dokumentacja funkcji składowych

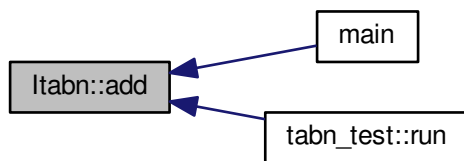
5.9.3.1 `template<class T> virtual void Itabn< T >::add ( T ) [pure virtual]`

Dodaje element na koniec tablicy.



Implementowany w [tabn< T >](#).

Oto graf wywołań tej funkcji:



**5.9.3.2** `template<class T> virtual void Itabn< T >::add ( T, int ) [pure virtual]`

Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.

Parametry

<i>element</i>	wstawiany element
<i>position</i>	indeks pola, w które ma być wstawiony element.

Implementowany w [tabn< T >](#).

**5.9.3.3** `template<class T> virtual int Itabn< T >::aSize ( void ) [pure virtual]`

Zwraca ilość miejsca w tablicy.

Implementowany w [tabn< T >](#).

Oto graf wywołań tej funkcji:



**5.9.3.4** `template<class T> virtual void Itabn< T >::bubblesort ( ) [pure virtual]`

Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

**Ostrzeżenie**

Wymaga typu danych ze zdefiniowanym operatorem porównania "większe od"

Implementowany w [tabn< T >](#).

**5.9.3.5** `template<class T> virtual bool Itabn< T >::isEmpty ( void ) [pure virtual]`

Sprawdza, czy tablica jest pusta.

Implementowany w [tabn< T >](#).

**5.9.3.6** `template<class T> virtual int Itabn< T >::nOE ( void ) [pure virtual]`

Zwraca liczbę elementów w tablicy.

Implementowany w [tabn< T >](#).

Oto graf wywoływań tej funkcji:



**5.9.3.7** `template<class T> virtual T& Itabn< T >::operator[] ( int ) [pure virtual]`

Pozwala na dostęp do dowolnego elementu.

Implementowany w [tabn< T >](#).

**5.9.3.8** `template<class T> virtual T Itabn< T >::operator[] ( int ) const [pure virtual]`

Pozwala na dostęp do dowolnego elementu.

Implementowany w [tabn< T >](#).

**5.9.3.9** `template<class T> virtual T Itabn< T >::remove ( ) [pure virtual]`

Usuwa i zwraca element z końca tablicy.

Implementowany w [tabn< T >](#).

**5.9.3.10** `template<class T> virtual T Itabn< T >::remove ( int ) [pure virtual]`

Usuwa i zwraca wybrany element z tablicy.

Parametry

<i>position</i>	indeks pola, z którego ma być usunięty element.
-----------------	---

Implementowany w [tabn< T >](#).

**5.9.3.11** `template<class T> virtual T Itabn< T >::show ( int ) [pure virtual]`

Zwraca żądany element, o ile istnieje.

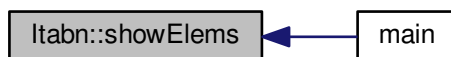
Implementowany w [tabn< T >](#).

5.9.3.12 `template<class T> virtual void ltabn< T >::showElems ( void ) [pure virtual]`

Wyświetla elementy tablicy.

Implementowany w `tabn< T >`.

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

## 5.10 Dokumentacja szablonu klasy Kolejka< T >

Klasa modeluje kolejkę

```
#include <kolejka.hh>
```

Diagram dziedziczenia dla Kolejka< T >

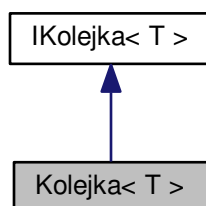
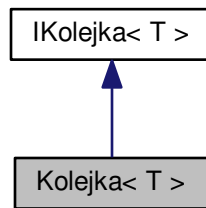


Diagram współpracy dla Kolejka< T >:



## Metody publiczne

- [Kolejka](#) ()  
*Konstruktor tablicy obsługującej kolejkę*
- virtual void [enqueue](#) (T)  
*Dodaje element na koniec kolejki.*
- virtual T [dequeue](#) (void)  
*Usuwa i zwraca element z początku kolejki.*
- virtual bool [isEmpty](#) (void)  
*Sprawdza, czy kolejka nie jest pusta.*
- virtual T [get](#) (void)  
*Zwraca element z początku kolejki bez usuwania.*
- virtual [~Kolejka](#) ()  
*Destruktor klasy [Kolejka](#).*

### 5.10.1 Opis szczegółowy

```
template<class T>class Kolejka< T >
```

Klasa modeluje kolejkę

Definicja w linii 54 pliku kolejka.hh.

### 5.10.2 Dokumentacja konstruktora i destruktora

5.10.2.1 `template<class T > Kolejka< T >::Kolejka ( ) [inline]`

Konstruktor tablicy obsługującej kolejkę

Definicja w linii 61 pliku kolejka.hh.

5.10.2.2 `template<class T > virtual Kolejka< T >::~~Kolejka ( ) [inline],[virtual]`

Destruktor klasy [Kolejka](#).

Definicja w linii 104 pliku kolejka.hh.

### 5.10.3 Dokumentacja funkcji składowych

#### 5.10.3.1 `template<class T> T Kolejka< T>::dequeue ( void ) [virtual]`

Usuwa i zwraca element z początku kolejki.

Zwracane wartości

<i>T</i>	element z początku kolejki
----------	----------------------------

Implementuje `IKolejka< T >`.

Definicja w linii 116 pliku kolejka.hh.

#### 5.10.3.2 `template<class T> void Kolejka< T>::enqueue ( T element ) [virtual]`

Dodaje element na koniec kolejki.

Parametry

<i>element</i>	- element do umieszczenia w kolejce
----------------	-------------------------------------

Implementuje `IKolejka< T >`.

Definicja w linii 110 pliku kolejka.hh.

#### 5.10.3.3 `template<class T> T Kolejka< T>::get ( void ) [virtual]`

Zwraca element z początku kolejki bez usuwania.

Ostrzeżenie

Uwaga! Próba odczytania elementu z pustej kolejki spowoduje wyrzucenie wyjątku, zamknięcie programu i zwrócenie -1.

Aby zapewnić poprawne działanie, sprawdź najpierw, czy kolejka nie jest pusta i uwarunkuj od tego wykonanie funkcji `get()`.

Przykład:

```
//Przykład korzystania z get()
IKolejka<int> * kolejka = new Kolejka<int>;
if (kolejka->isEmpty() == false) {
    cout << kolejka->get() << endl;
}
else
    cerr << "Kolejka pusta" << endl;
```

Implementuje `IKolejka< T >`.

Definicja w linii 128 pliku kolejka.hh.

#### 5.10.3.4 `template<class T> bool Kolejka< T>::isEmpty ( void ) [virtual]`

Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementuje `IKolejka< T >`.

Definicja w linii 123 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [kolejka.hh](#)

## 5.11 Dokumentacja szablonu klasy Lista< T >

Klasa lista.

```
#include <lista.hh>
```

Diagram dziedziczenia dla Lista< T >

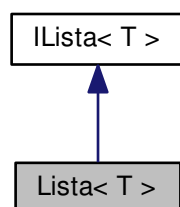
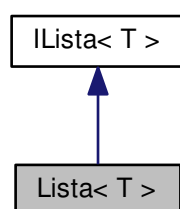


Diagram współpracy dla Lista< T >:



### Metody publiczne

- `Lista ()`  
*Konstruktor tablicy obsługującej listę*
- `virtual void add (T, int)`  
*Dodaje element do zadanego miejsca listy.*
- `virtual void add (T)`  
*Dodaje element na koniec listy.*
- `virtual T remove (int position)`  
*Usuwa element z zadanego miejsca listy.*
- `virtual T remove (void)`  
*Usuwa element z końca listy.*
- `virtual bool isEmpty (void)`  
*Sprawdza, czy lista jest pusta.*
- `virtual T get (int position)`  
*Zwraca element z zadanego miejsca bez usunięcia.*

- virtual int [size](#) (void)  
*Zwraca ilość elementów w liście.*
- virtual [~Lista](#) ()  
*Destruktor Listy.*

### 5.11.1 Opis szczegółowy

`template<class T>class Lista< T >`

Klasa lista.

Modeluje pojęcie listy

Definicja w linii 85 pliku lista.hh.

### 5.11.2 Dokumentacja konstruktora i destruktora

5.11.2.1 `template<class T > Lista< T >::Lista ( ) [inline]`

Konstruktor tablicy obsługującej listę

Definicja w linii 92 pliku lista.hh.

5.11.2.2 `template<class T > virtual Lista< T >::~Lista ( ) [inline], [virtual]`

Destruktor Listy.

Definicja w linii 172 pliku lista.hh.

### 5.11.3 Dokumentacja funkcji składowych

5.11.3.1 `template<class T > void Lista< T >::add ( T element, int position ) [virtual]`

Dodaje element do zadanego miejsca listy.

Jeśli następuje próba dodania elementu w miejscu istniejącego, następuje przesunięcie następujących po nim elementów na następne pozycje

Nota

Próba dodania elementu na miejsce dalsze niż pierwsze następujące po obecnie istniejącym spowoduje wyrzucenie wyjątku i niewykonanie akcji.

Implementuje [ILista< T >](#).

Definicja w linii 178 pliku lista.hh.

5.11.3.2 `template<class T > void Lista< T >::add ( T element ) [virtual]`

Dodaje element na koniec listy.

Implementuje [ILista< T >](#).

Definicja w linii 183 pliku lista.hh.

5.11.3.3 `template<class T > T Lista< T >::get ( int position ) [virtual]`

Zwraca element z zadanego miejsca bez usunięcia.

## Zwracane wartości

<i>T</i>	element w zadanym miejscu
----------	---------------------------

## Ostrzeżenie

Próba podglądu elementu nieistniejącego spowoduje wyrzucenie wyjątku, zakończenie programu i zwrócenie -1. Najpierw sprawdź, czy dany element istnieje.

```
//Przykład sprawdzenia poprawności podglądu
ILista<int> * list = new Lista<int>;
list->add(2); //skomentuj odpowiednio linię aby sprawdzić działanie obu przypadków
int positionToCheckAndShow = 0;
if(list->size()>positionToCheckAndShow) {
    cout << list->get(positionToCheckAndShow) << endl;
}
else
    cerr << "Element nie istnieje!" << endl;
```

Implementuje `ILista< T >`.

Definicja w linii 207 pliku lista.hh.

5.11.3.4 `template<class T> bool Lista< T >::isEmpty ( void ) [virtual]`

Sprawdza, czy lista jest pusta.

## Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementuje `ILista< T >`.

Definicja w linii 202 pliku lista.hh.

5.11.3.5 `template<class T> T Lista< T >::remove ( int position ) [virtual]`

Usuwa element z zadanego miejsca listy.

Jeśli usunięcie następuje w środku listy, następujące po usuwanym elementy są przesuwane o jedną pozycję wcześniej.

## Zwracane wartości

<i>T</i>	Usunięty element
----------	------------------

## Ostrzeżenie

Próba usunięcia elementu nieistniejącego lub z pustej listy spowoduje wyrzucenie wyjątku, zakończenie programu i zwrócenie -1. Najpierw sprawdź, czy dany element istnieje a następnie usuń element.

```
//Przykład sprawdzenia poprawności usuwania
ILista<int> * list = new Lista<int>;
list->add(2); //skomentuj odpowiednio linię aby sprawdzić działanie obu przypadków
int positionToCheckAndRemove = 0;
if(list->size()>positionToCheckAndRemove) {
    list->remove(positionToCheckAndRemove);
}
else
    cerr << "Element nie istnieje!" << endl;
```

Implementuje `ILista< T >`.

Definicja w linii 188 pliku lista.hh.



5.11.3.6 `template<class T> T Lista< T >::remove ( void ) [virtual]`

Usuwa element z końca listy.

Implementuje `ILista< T >`.

Definicja w linii 195 pliku lista.hh.

5.11.3.7 `template<class T> int Lista< T >::size ( void ) [virtual]`

Zwraca ilość elementów w liście.

Zwracane wartości

<i>int</i>	ilość elementów
------------	-----------------

Implementuje `ILista< T >`.

Definicja w linii 223 pliku lista.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

## 5.12 Dokumentacja klasy lista\_test

Definiuje sposób testowania wypełniania listy.

```
#include <lista.hh>
```

Diagram dziedziczenia dla lista\_test

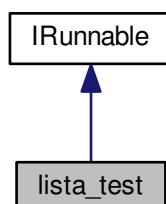
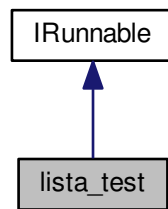


Diagram współpracy dla lista\_test:



## Metody publiczne

- `lista_test ()`  
*Konstruktor klasy testującej.*
- `~lista_test ()`  
*Destruktor klasy testującej.*
- virtual bool `prepare (int sizeOfTest)`  
*Przygotowuje rozmiar testu.*
- virtual bool `run ()`  
*Wykonuje test.*

### 5.12.1 Opis szczegółowy

Definiuje sposób testowania wypełniania listy.

Definicja w linii 235 pliku lista.hh.

### 5.12.2 Dokumentacja konstruktora i destruktora

#### 5.12.2.1 `lista_test::lista_test ( ) [inline]`

Konstruktor klasy testującej.

Definicja w linii 246 pliku lista.hh.

#### 5.12.2.2 `lista_test::~~lista_test ( ) [inline]`

Destruktor klasy testującej.

Definicja w linii 252 pliku lista.hh.

### 5.12.3 Dokumentacja funkcji składowych

#### 5.12.3.1 `virtual bool lista_test::prepare ( int sizeOfTest ) [inline],[virtual]`

Przygotowuje rozmiar testu.

## Parametry

<i>sizeofTest</i>	- rozmiar testu
-------------------	-----------------

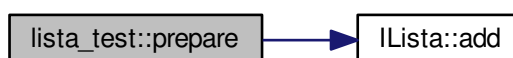
## Zwracane wartości

<i>true</i>	gdy plik ze słownikiem został pomyślnie otwarty
<i>false</i>	gdy otwieranie pliku zakończyło się błędem

Implementuje [IRunnable](#).

Definicja w linii 283 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



### 5.12.3.2 virtual bool lista\_test::run ( ) [inline],[virtual]

Wykonuje test.

Pozwala na wykonanie testu.

## Zwracane wartości

<i>true</i>	zawsze
-------------	--------

Implementuje [IRunnable](#).

Definicja w linii 309 pliku lista.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

## 5.13 Dokumentacja klasy Stoper

Klasa stoper implementująca interfejs [IStoper](#).

```
#include <stoper.hh>
```

Diagram dziedziczenia dla Stoper

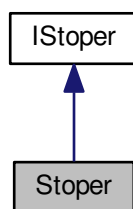
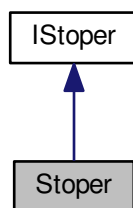


Diagram współpracy dla Stoper:



## Metody publiczne

- virtual void `start` (void)  
*Uruchamia zegar.*
- virtual void `stop` (void)  
*Zatrzymuje zegar.*
- virtual long double `getElapsedTimeMs` (void)  
*Oblicza i zwraca czas pomiędzy uruchomieniem zegara a jego zatrzymaniem.*

### 5.13.1 Opis szczegółowy

Klasa `stoper` implementująca interfejs `IStoper`.

Klasa symuluje działanie stopera - zapisuje początkowy i końcowy moment działania (użycie `start` i `stop`), oraz odejmuje obie te wartości od siebie, by uzyskać czas działania.

Definicja w linii 36 pliku `stoper.hh`.

### 5.13.2 Dokumentacja funkcji składowych

5.13.2.1 `long double Stoper::getElapsedTimeMs ( void ) [virtual]`

Oblicza i zwraca czas pomiędzy uruchomieniem zegara a jego zatrzymaniem.

## Zwracane wartości

<i>long_double</i>	Czas pomiędzy startem a zatrzymaniem zegara
--------------------	---

Implementuje [IStoper](#).

Definicja w linii 12 pliku stoper.cpp.

#### 5.13.2.2 void Stoper::start ( void ) [virtual]

Uruchamia zegar.

Implementuje [IStoper](#).

Definicja w linii 4 pliku stoper.cpp.

#### 5.13.2.3 void Stoper::stop ( void ) [virtual]

Zatrzymuje zegar.

Implementuje [IStoper](#).

Definicja w linii 8 pliku stoper.cpp.

Dokumentacja dla tej klasy została wygenerowana z plików:

- [stoper.hh](#)
- [stoper.cpp](#)

## 5.14 Dokumentacja szablonu klasy Stos< T >

Klasa [Stos](#).

```
#include <stos.hh>
```

Diagram dziedziczenia dla Stos< T >

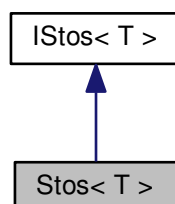
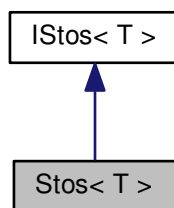


Diagram współpracy dla Stos< T >:



## Metody publiczne

- `Stos()`  
*Konstruktor tablicy obsługującej stos.*
- `virtual void push(T)`  
*Umieszcza element na szczycie stosu.*
- `virtual T pull(void)`  
*Zdejmuje element ze szczytu stosu.*
- `virtual bool isEmpty(void)`  
*Sprawdza, czy stos jest pusty.*
- `virtual T get(void)`  
*Zwraca element ze szczytu stosu bez jego usuwania.*
- `virtual ~Stos()`  
*Destruktor stosu.*

### 5.14.1 Opis szczegółowy

```
template<class T>class Stos< T >
```

Klasa `Stos`.

Modeluje pojęcie stosu

Definicja w linii 63 pliku `stos.hh`.

### 5.14.2 Dokumentacja konstruktora i destruktora

5.14.2.1 `template<class T> Stos< T >::Stos( ) [inline]`

Konstruktor tablicy obsługującej stos.

Definicja w linii 70 pliku `stos.hh`.

5.14.2.2 `template<class T> virtual Stos< T >::~~Stos( ) [inline],[virtual]`

Destruktor stosu.

Definicja w linii 127 pliku `stos.hh`.

### 5.14.3 Dokumentacja funkcji składowych

#### 5.14.3.1 `template<class T> T Stos<T>::get ( void ) [virtual]`

Zwraca element ze szczytu stosu bez jego usuwania.

Zwracane wartości

<i>T</i>	element ze szczytu stosu
----------	--------------------------

#### Ostrzeżenie

Uwaga! Próba odczytania elementu z pustego stosu spowoduje wyrzucenie wyjątku, zamknięcie programu i zwrócenie -1.

Aby zapewnić poprawne działanie, sprawdź najpierw, czy stos nie jest pusty i uwarunkuj od tego wykonanie funkcji `get()`.

Przykład:

```
//Przykład korzystania z get()
IStos<int> * stos = new Stos<int>;
if (stos->isEmpty() == false) {
    cout << stos->get() << endl;
}
else
    cerr << "Stos pusty" << endl;
```

Implementuje `IStos<T>`.

Definicja w linii 150 pliku `stos.hh`.

#### 5.14.3.2 `template<class T> bool Stos<T>::isEmpty ( void ) [virtual]`

Sprawdza, czy stos jest pusty.

Zwracane wartości

0	gdy niepusty
1	gdy pusty

Implementuje `IStos<T>`.

Definicja w linii 145 pliku `stos.hh`.

#### 5.14.3.3 `template<class T> T Stos<T>::pull ( void ) [virtual]`

Zdejmuje element ze szczytu stosu.

Zwracane wartości

<i>T</i>	element ze szczytu stosu
----------	--------------------------

#### Ostrzeżenie

Uwaga! Próba zdjęcia elementu z pustego stosu spowoduje wyrzucenie wyjątku, zamknięcie programu i zwrócenie -1.

Aby zapewnić poprawne działanie, sprawdź najpierw, czy stos nie jest pusty i uwarunkuj od tego wykonanie funkcji `pull()`.

Przykład:

```
//Przykład korzystania z get()
IStos<int> * stos = new Stos<int>;
if (stos->isEmpty() == false) {
    cout << stos->pull() << endl;
}
else
    cerr << "Stos pusty" << endl;
```



Implementuje `IStos< T >`.

Definicja w linii 138 pliku `stos.hh`.

5.14.3.4 `template<class T> void Stos< T >::push ( T element ) [virtual]`

Umieszcza element na szczycie stosu.

Parametry

<i>element</i>	- element do umieszczenia na stosie
----------------	-------------------------------------

Implementuje `IStos< T >`.

Definicja w linii 133 pliku `stos.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [stos.hh](#)

## 5.15 Dokumentacja szablonu klasy `tabn< T >`

Modeluje tablicę dynamicznie rozszerzalną

```
#include <tabl.hh>
```

Diagram dziedziczenia dla `tabn< T >`

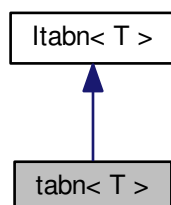
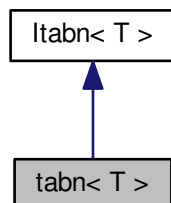


Diagram współpracy dla `tabn< T >`:



## Metody publiczne

- `tabn()`  
*Konstruktor klasy tabn.*
- `virtual ~tabn()`  
*Destruktor klasy tabn.*
- `virtual bool isEmpty()` (void)  
*Sprawdza, czy tablica jest pusta.*
- `virtual void add(T)`  
*Dodaje element* Dodaje element do tablicy dynamicznej, odpowiednio ją rozszerzając.
- `virtual void add(T, int)`  
*Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.*
- `virtual T remove()`  
*Usuwa i zwraca ostatni element z tablicy.*
- `virtual T remove(int)`  
*Usuwa i zwraca wybrany element z tablicy, przesuwając wszystkie następne elementy o miejsce w lewo.*
- `virtual T show(int)`  
*Zwraca żądany element, o ile istnieje, bez jego usuwania.*
- `virtual void showElems()` (void)  
*Wyświetla listę elementów.*
- `virtual int nOE()` (void)  
*zwraca liczbę elementów w tablicy*
- `virtual int aSize()` (void)  
*zwraca wielkość zaalokowanej przestrzeni dla tablicy*
- `virtual T & operator[] (int)`  
*Umożliwia dostęp do dowolnego elementu tablicy bez sprawdzania zakresu (debug)*
- `virtual T operator[] (int) const`  
*Umożliwia odczyt dowolnego elementu tablicy bez sprawdzania zakresu (debug)*
- `virtual void bubblesort()` (void)  
*Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.*

### 5.15.1 Opis szczegółowy

```
template<class T>class tabn< T >
```

Modeluje tablicę dynamicznie rozszerzalną

Przechowuje elementy w rozszerzalnej tablicy o rozmiarze początkowym SIZE

Definicja w linii 99 pliku tabl.hh.

### 5.15.2 Dokumentacja konstruktora i destruktora

#### 5.15.2.1 `template<class T> tabn< T>::tabn()` [inline]

Konstruktor klasy tabn.

Definicja w linii 110 pliku tabl.hh.

#### 5.15.2.2 `template<class T> virtual tabn< T>::~~tabn()` [inline], [virtual]

Destruktor klasy tabn.

Definicja w linii 119 pliku tabl.hh.

### 5.15.3 Dokumentacja funkcji składowych

#### 5.15.3.1 `template<class T> void tabn< T >::add ( T element ) [virtual]`

Dodaje element Dodaje element do tablicy dynamicznej, odpowiednio ją rozszerzając.

Parametry

<i>element</i>	- element do dodania
----------------	----------------------

Implementuje `ltabn< T >`.

Definicja w linii 279 pliku `tabl.hh`.

#### 5.15.3.2 `template<class T> void tabn< T >::add ( T element, int position ) [virtual]`

Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.

Parametry

<i>element</i>	- wstawiany element
<i>positionShifted</i>	- indeks pola, w które ma być wstawiony element.

Wyjątki

<a href="#"><i>ContinueException</i></a>	przy próbie dodania elementu do niewłaściwego miejsca.
--	--

Implementuje `ltabn< T >`.

Definicja w linii 287 pliku `tabl.hh`.

#### 5.15.3.3 `template<class T> int tabn< T >::aSize ( void ) [virtual]`

zwraca wielkość zaalokowanej przestrzeni dla tablicy

Zwracane wartości

<i>int</i>	Ilość zaalokowanych pól
------------	-------------------------

Implementuje `ltabn< T >`.

Definicja w linii 464 pliku `tabl.hh`.

#### 5.15.3.4 `template<class T> void tabn< T >::bubblesort ( void ) [virtual]`

Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

Ostrzeżenie

Wymaga typu danych ze zdefiniowanym operatorem porównania "większe od"

Wyjątki

<a href="#"><i>CriticalException</i></a>	re-throw swap(int,int)
--	------------------------

Implementuje `ltabn< T >`.

Definicja w linii 485 pliku `tabl.hh`.

#### 5.15.3.5 `template<class T> bool tabn< T >::isEmpty ( void ) [virtual]`

Sprawdza, czy tablica jest pusta.

## Zwracane wartości

0	gdy tablica nie jest pusta
1	gdy tablica jest pusta

Implementuje `ltabn< T >`.

Definicja w linii 413 pliku `tabl.hh`.

#### 5.15.3.6 `template<class T> int tabn< T >::nOE ( void ) [virtual]`

zwraca liczbę elementów w tablicy

## Zwracane wartości

<i>int</i>	Liczba elementów w tablicy
------------	----------------------------

Implementuje `ltabn< T >`.

Definicja w linii 459 pliku `tabl.hh`.

#### 5.15.3.7 `template<class T> T & tabn< T >::operator[] ( int index ) [virtual]`

Umożliwia dostęp do dowolnego elementu tablicy bez sprawdzania zakresu (debug)

## Parametry

<i>index</i>	- numer elementu tablicy
--------------	--------------------------

## Zwracane wartości

<i>T*</i>	Wskaźnik na wybrany element tablicy
-----------	-------------------------------------

Implementuje `ltabn< T >`.

Definicja w linii 429 pliku `tabl.hh`.

#### 5.15.3.8 `template<class T> T tabn< T >::operator[] ( int index ) const [virtual]`

Umożliwia odczyt dowolnego elementu tablicy bez sprawdzania zakresu (debug)

## Parametry

<i>index</i>	- numer elementu tablicy
--------------	--------------------------

## Zwracane wartości

<i>T</i>	Element tablicy
----------	-----------------

Implementuje `ltabn< T >`.

Definicja w linii 434 pliku `tabl.hh`.

#### 5.15.3.9 `template<class T> T tabn< T >::remove ( void ) [virtual]`

Usuwa i zwraca ostatni element z tablicy.

## Wyjątki

<a href="#"><code>CriticalException</code></a>	przy próbie usunięcia z pustej tablicy.
--	---

Implementuje `ltabn< T >`.

Definicja w linii 318 pliku `tabl.hh`.

**5.15.3.10** `template<class T> T tabn< T >::remove ( int position ) [virtual]`

Usuwa i zwraca wybrany element z tablicy, przesuwając wszystkie następne elementy o miejsce w lewo.

Parametry

<i>position</i>	indeks pola, z którego ma być usunięty element.
-----------------	---

Wyjątki

<a href="#"><code>CriticalException</code></a>	przy próbie usunięcia z pustej tablicy lub nieistniejącego elementu.
<a href="#"><code>ContinueException</code></a>	re-throw <code>reduce2()</code>

Implementuje `ltabn< T >`.

Definicja w linii 341 pliku `tabl.hh`.

**5.15.3.11** `template<class T> T tabn< T >::show ( int position ) [virtual]`

Zwraca żądany element, o ile istnieje, bez jego usuwania.

Wyjątki

<a href="#"><code>CriticalException</code></a>	przy próbie odczytania z pustej tablicy lub dostępu do nieistniejącego elementu.
--	--

Implementuje `ltabn< T >`.

Definicja w linii 439 pliku `tabl.hh`.

**5.15.3.12** `template<class T> void tabn< T >::showElems ( void ) [virtual]`

Wyświetla listę elementów.

Implementuje `ltabn< T >`.

Definicja w linii 449 pliku `tabl.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

## 5.16 Dokumentacja klasy `tabn_test`

Definiuje sposób testowania wypełniania tablicy `tabn`.

```
#include <tabl.hh>
```

Diagram dziedziczenia dla `tabn_test`

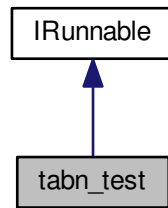
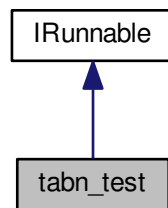


Diagram współpracy dla `tabn_test`:



## Metody publiczne

- `tabn_test ()`  
*Konstruktor klasy `tabn_test`.*
- `virtual ~tabn_test ()`  
*Destruktor klasy `tabn_test`.*
- `virtual bool prepare (int sizeOfTest)`  
*Przygotowuje rozmiar testu.*
- `virtual bool run ()`  
*Wykonuje test.*

### 5.16.1 Opis szczegółowy

Definiuje sposób testowania wypełniania tablicy `tabn`.

Definicja w linii 519 pliku `tabl.hh`.

### 5.16.2 Dokumentacja konstruktora i destruktora

#### 5.16.2.1 `tabn_test::tabn_test ( )` `[inline]`

Konstruktor klasy `tabn_test`.

Definicja w linii 527 pliku `tabl.hh`.

#### 5.16.2.2 `virtual tabn_test::~~tabn_test ( )` `[inline]`, `[virtual]`

Destruktor klasy `tabn_test`.

Definicja w linii 533 pliku `tabl.hh`.

### 5.16.3 Dokumentacja funkcji składowych

#### 5.16.3.1 `virtual bool tabn_test::prepare ( int sizeOfTest )` `[inline]`, `[virtual]`

Przygotowuje rozmiar testu.

Parametry

<i>sizeOfTest</i>	- rozmiar testu
-------------------	-----------------

Zwracane wartości

<i>bool</i>	zawsze true
-------------	-------------

Implementuje `IRunnable`.

Definicja w linii 564 pliku `tabl.hh`.

#### 5.16.3.2 `virtual bool tabn_test::run ( )` `[inline]`, `[virtual]`

Wykonuje test.

Pozwala na wykonanie testu w pętli `for` iterującej `counter` razy. Zasila funkcję dodawania generując losowe cyfry w funkcji `generateRandomDgt()`

Zwracane wartości

<i>bool</i>	zawsze true
-------------	-------------

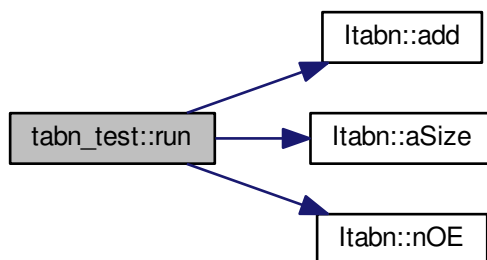
Wyjątki

<code>ContinueException</code>	re-throw <code>tabn::add(int)</code>
--------------------------------	--------------------------------------

Implementuje `IRunnable`.

Definicja w linii 581 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)



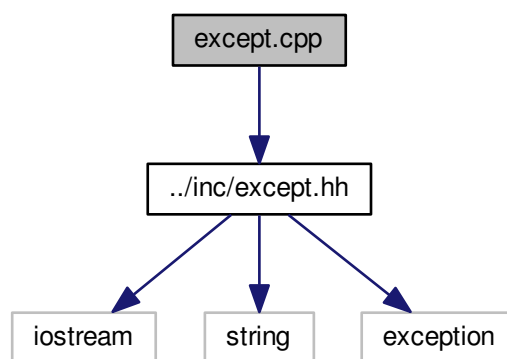
## Rozdział 6

# Dokumentacja plików

### 6.1 Dokumentacja pliku except.cpp

```
#include "../inc/except.hh"
```

Wykres zależności załączania dla except.cpp:

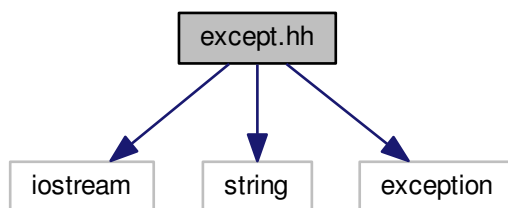


### 6.2 Dokumentacja pliku except.hh

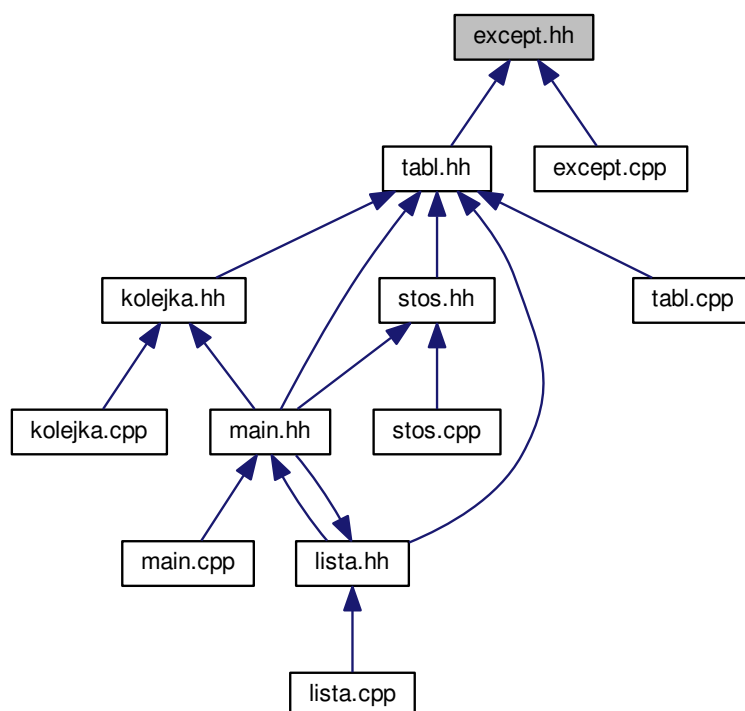
Plik zawiera definicje wyjątków.

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

Wykres zależności załączania dla except.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Komponenty

- class [Exception](#)  
*Ogólny wyjątek.*
- class [CriticalException](#)  
*Wyjątek krytyczny, wymagający zamknięcia programu.*
- class [ContinueException](#)  
*Wyjątek, który mimo pojawienia się, pozwala na dalsze poprawne działanie programu.*

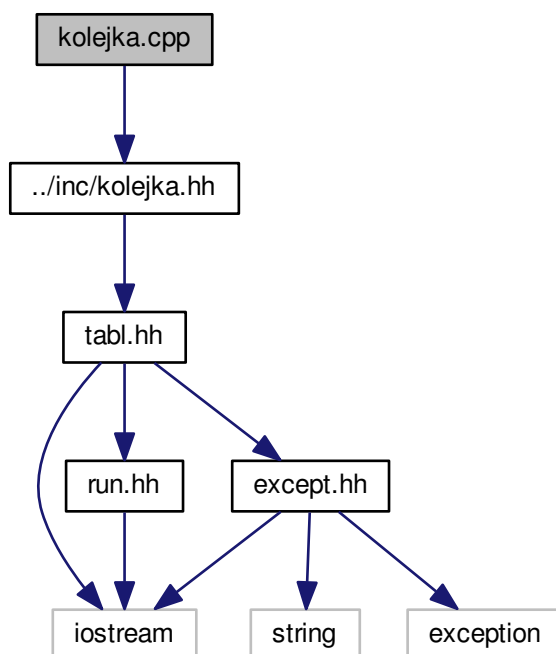
### 6.2.1 Opis szczegółowy

Plik zawiera definicje wyjątków.

## 6.3 Dokumentacja pliku kolejka.cpp

```
#include "../inc/kolejka.hh"
```

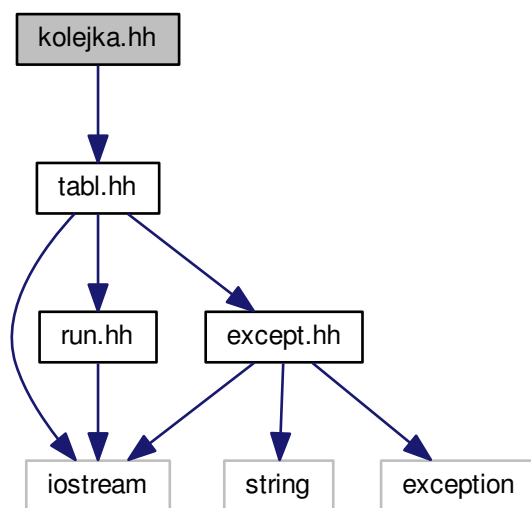
Wykres zależności załączania dla kolejka.cpp:



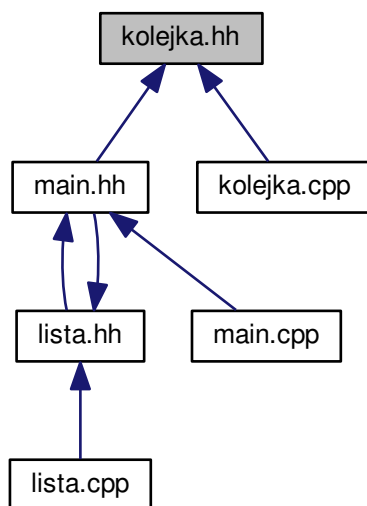
## 6.4 Dokumentacja pliku kolejka.hh

```
#include "tabl.hh"
```

Wykres zależności załączania dla kolejka.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Komponenty

- class `IKolejka< T >`

Interfejs klasy `Kolejka` Definiuje operacje dostępne dla klasy `Kolejka`.

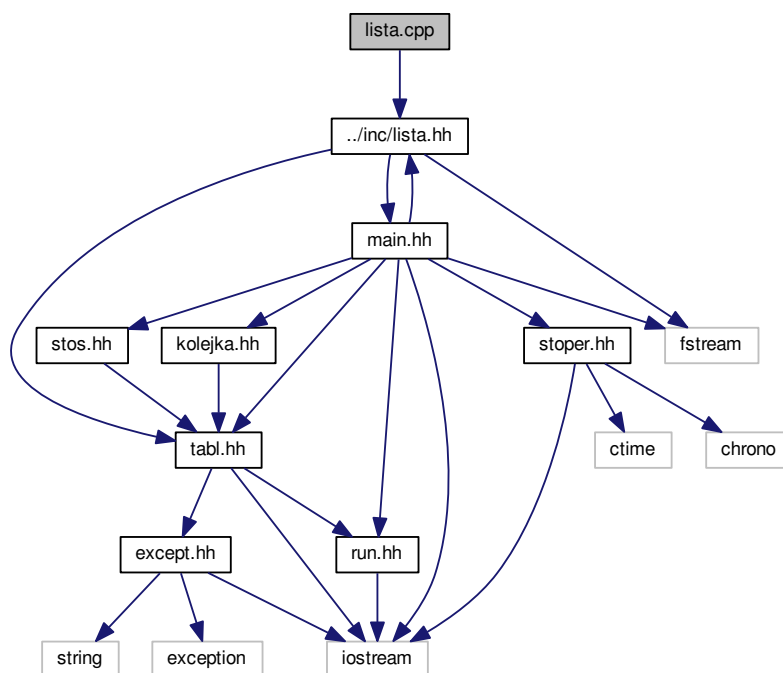
- class `Kolejka< T >`

Klasa modeluje kolejkę

## 6.5 Dokumentacja pliku lista.cpp

```
#include "../inc/lista.hh"
```

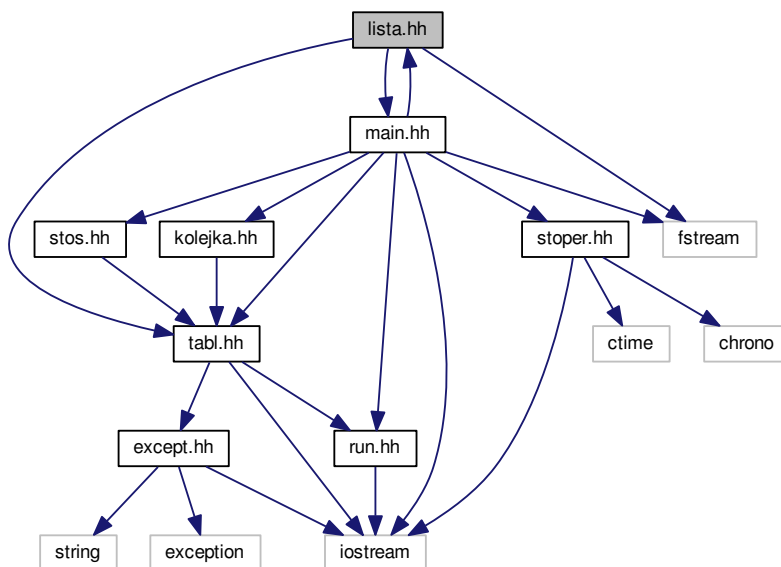
Wykres zależności załączania dla lista.cpp:



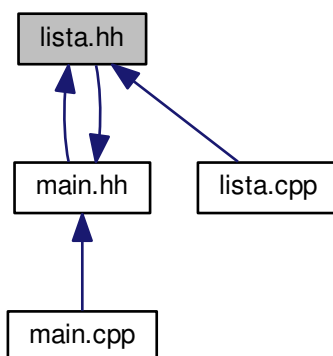
## 6.6 Dokumentacja pliku lista.hh

```
#include "tabl.hh"
#include "main.hh"
#include <fstream>
```

Wykres zależności załączania dla lista.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Komponenty

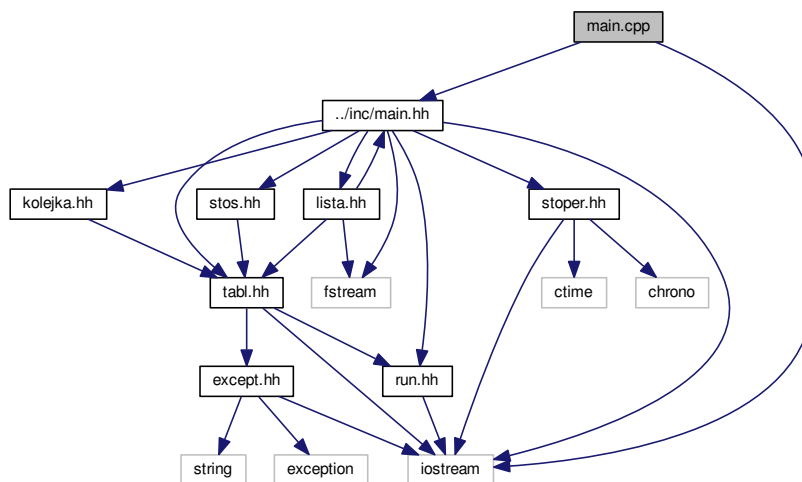
- class `ILista< T >`  
*Interfejs listy.*
- class `Lista< T >`  
*Klasa lista.*
- class `lista_test`  
*Definiuje sposób testowania wypełniania listy.*

## 6.7 Dokumentacja pliku main.cpp

Główny plik programu.

```
#include <iostream>
#include "../inc/main.hh"
```

Wykres zależności załączania dla main.cpp:



### Funkcje

- int [main](#) (void)
- void [dumpToFile](#) (string nameOfFile, unsigned int testsize, [IStoper](#) \*stoper)
- void [printOnscreen](#) (unsigned int testsize, [IStoper](#) \*stoper)

*Wyświetla wynik na standardowym wyjściu.*

### 6.7.1 Opis szczegółowy

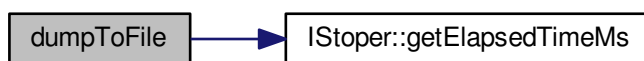
Główny plik programu.

### 6.7.2 Dokumentacja funkcji

6.7.2.1 void [dumpToFile](#) ( string *nameOfFile*, unsigned int *testsize*, [IStoper](#) \* *stoper* )

Definicja w linii 90 pliku main.cpp.

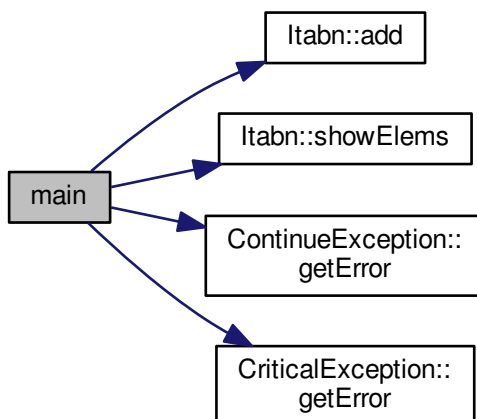
Oto graf wywołań dla tej funkcji:



#### 6.7.2.2 `int main ( void )`

Definicja w linii 12 pliku `main.cpp`.

Oto graf wywołań dla tej funkcji:



#### 6.7.2.3 `void printOnscreen ( unsigned int, IStoper * )`

Wyświetla wynik na standardowym wyjściu.

Parametry

<i>testSize</i>	rozmiar testu
<i>stoper</i>	klasa stopera

Definicja w linii 100 pliku `main.cpp`.



Oto graf wywołań dla tej funkcji:

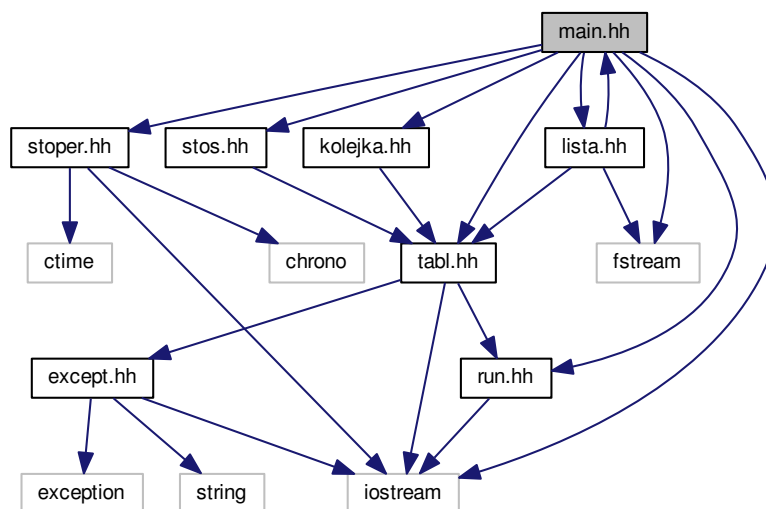


## 6.8 Dokumentacja pliku main.hh

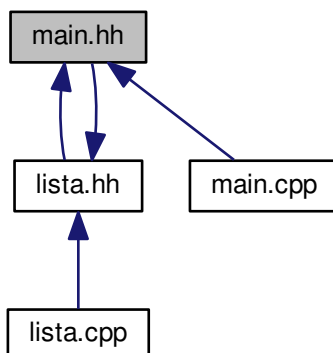
```

#include <iostream>
#include <fstream>
#include "stoper.hh"
#include "tabl.hh"
#include "run.hh"
#include "lista.hh"
#include "stos.hh"
#include "kolejka.hh"
  
```

Wykres zależności załączania dla main.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Funkcje

- void `dumpToFile` (std::string, unsigned int, `IStoper *`)

*Zrzuca dane wynikowe do pliku.*

- void `printOnscreen` (unsigned int, `IStoper *`)

*Wyświetla wynik na standardowym wyjściu.*

### 6.8.1 Dokumentacja funkcji

#### 6.8.1.1 void dumpToFile ( std::string , unsigned int, IStoper \* )

Zrzuca dane wynikowe do pliku.

Parametry

<i>nameOfFile</i>	nazwa pliku wynikowego
<i>testsize</i>	rozmiar testu
<i>stoper</i>	klasa stopera

#### 6.8.1.2 void printOnscreen ( unsigned int, IStoper \* )

Wyświetla wynik na standardowym wyjściu.

Parametry

<i>testSize</i>	rozmiar testu
<i>stoper</i>	klasa stopera

Definicja w linii 100 pliku main.cpp.

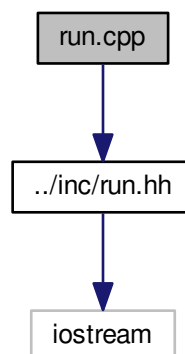
Oto graf wywołań dla tej funkcji:



## 6.9 Dokumentacja pliku run.cpp

```
#include "../inc/run.hh"
```

Wykres zależności załączania dla run.cpp:

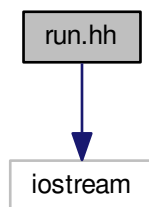


## 6.10 Dokumentacja pliku run.hh

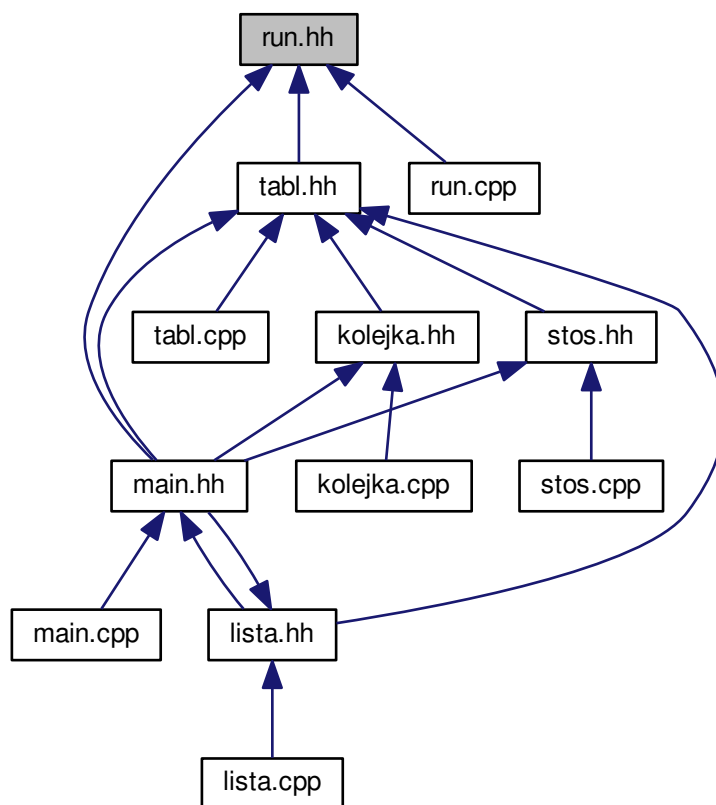
Plik definiuje interfejs [IRunnable](#), ujednolicający klasy umożliwiające badanie algorytmów.

```
#include <iostream>
```

Wykres zależności załączania dla run.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Komponenty

- class [IRunnable](#)

*Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.*

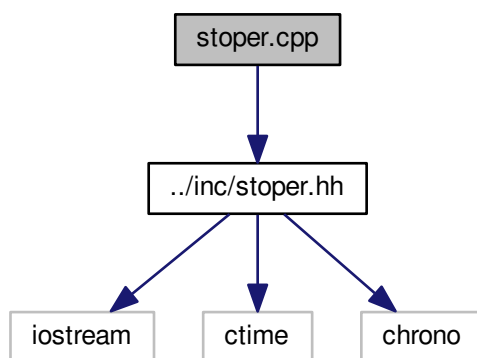
### 6.10.1 Opis szczegółowy

Plik definiuje interfejs `IRunnable`, ujednolicający klasy umożliwiające badanie algorytmów.

## 6.11 Dokumentacja pliku stoper.cpp

```
#include "../inc/stoper.hh"
```

Wykres zależności załączania dla stoper.cpp:



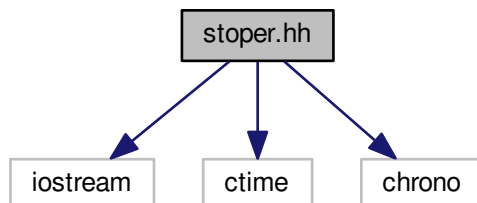
## 6.12 Dokumentacja pliku stoper.hh

```
#include <iostream>
```

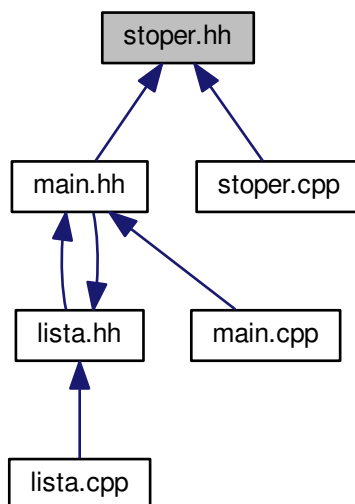
```
#include <ctime>
```

```
#include <chrono>
```

Wykres zależności załączania dla stoper.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Komponenty

- class [IStoper](#)

*Plik definiuje stoper, obliczający czas wykonywania badanych funkcji.*

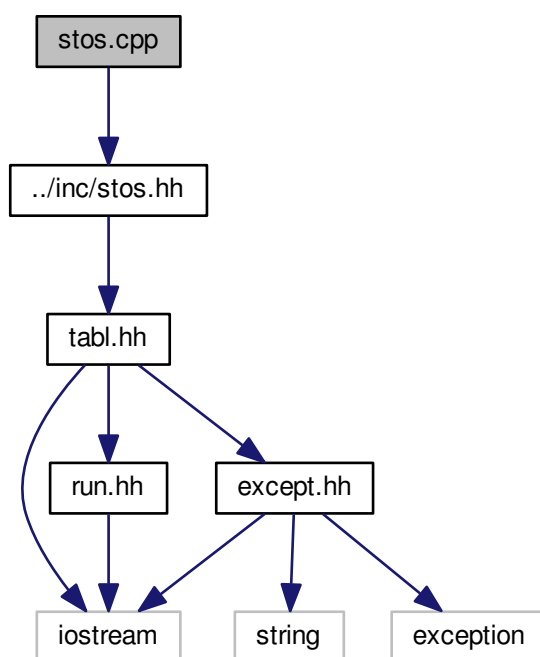
- class [Stoper](#)

*Klasa stoper implementująca interfejs [IStoper](#).*

## 6.13 Dokumentacja pliku stos.cpp

```
#include "../inc/stos.hh"
```

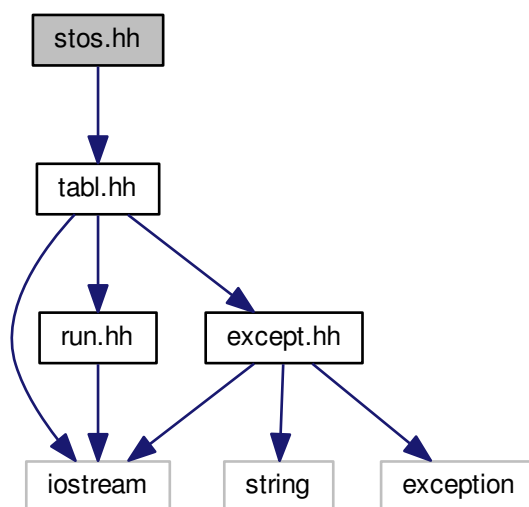
Wykres zależności załączania dla stos.cpp:



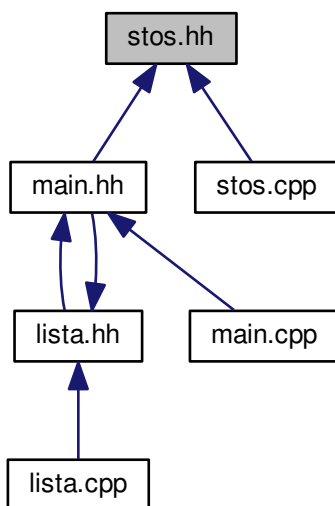
## 6.14 Dokumentacja pliku stos.hh

```
#include "tabl.hh"
```

Wykres zależności załączania dla stos.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Komponenty

- class `IStos< T >`  
*Interfejs stosu.*



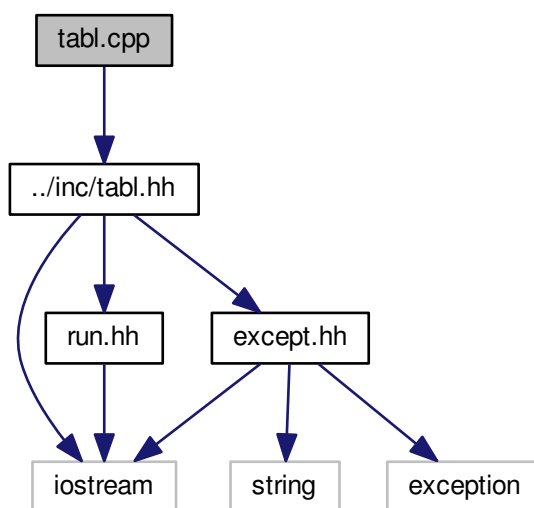
- class `Stos< T >`

Klasa `Stos`.

## 6.15 Dokumentacja pliku tabl.cpp

```
#include "../inc/tabl.hh"
```

Wykres zależności załączania dla tabl.cpp:

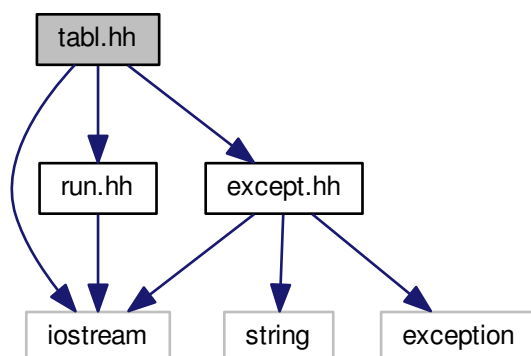


## 6.16 Dokumentacja pliku tabl.hh

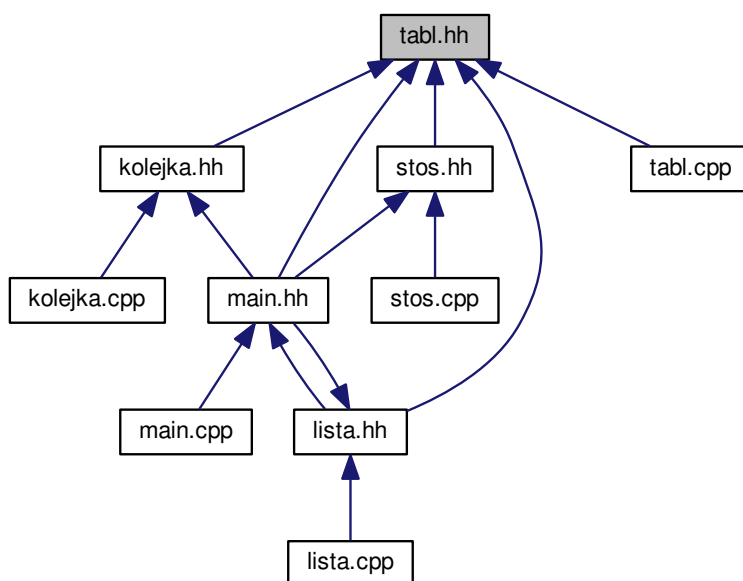
Definicja interfejsu `Itabn`, klasy `tabn` oraz klasy `tabn_test`.

```
#include <iostream>
#include "run.hh"
#include "except.hh"
```

Wykres zależności załączania dla `tabl.hh`:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



## Komponenty

- class `ltabn< T >`  
*Interfejs klasy `tabn`.*
- class `tabn< T >`  
*Modeluje tablicę dynamicznie rozszerzalną*
- class `tabn_test`

*Definiuje sposób testowania wypełniania tablicy tabn.*

## Definicje

- #define `SIZE` 10

### 6.16.1 Opis szczegółowy

Definicja interfejsu `ltabn`, klasy `tabn` oraz klasy `tabn_test`.

### 6.16.2 Dokumentacja definicji

#### 6.16.2.1 #define `SIZE` 10

Definicja w linii 12 pliku tabl.hh.



# Skorowidz

- ~IKolejka
  - IKolejka, [14](#)
- ~ILista
  - ILista, [16](#)
- ~IRunnable
  - IRunnable, [19](#)
- ~IStoper
  - IStoper, [20](#)
- ~IStos
  - IStos, [22](#)
- ~Itabn
  - Itabn, [24](#)
- ~Kolejka
  - Kolejka, [28](#)
- ~Lista
  - Lista, [31](#)
- ~Stos
  - Stos, [39](#)
- ~lista\_test
  - lista\_test, [34](#)
- ~tabn
  - tabn, [42](#)
- ~tabn\_test
  - tabn\_test, [47](#)

- aSize
  - Itabn, [25](#)
  - tabn, [43](#)
- add
  - ILista, [17](#)
  - Itabn, [24](#), [25](#)
  - Lista, [31](#)
  - tabn, [43](#)

- bubblesort
  - Itabn, [25](#)
  - tabn, [43](#)

- cause
  - ContinueException, [10](#)
  - CriticalException, [12](#)
  - Exception, [13](#)
- ContinueException, [9](#)
  - cause, [10](#)
  - ContinueException, [10](#)
  - getError, [10](#)
- CriticalException, [10](#)
  - cause, [12](#)
  - CriticalException, [11](#)
  - getError, [12](#)

- dequeue
  - IKolejka, [14](#)
  - Kolejka, [29](#)
- dumpToFile
  - main.cpp, [55](#)
  - main.hh, [58](#)

- enqueue
  - IKolejka, [15](#)
  - Kolejka, [29](#)
- except.cpp, [49](#)
- except.hh, [49](#)
- Exception, [12](#)
  - cause, [13](#)
  - Exception, [13](#)
  - getError, [13](#)

- get
  - IKolejka, [15](#)
  - ILista, [17](#)
  - IStos, [22](#)
  - Kolejka, [29](#)
  - Lista, [31](#)
  - Stos, [40](#)
- getElapsedTimeMs
  - IStoper, [21](#)
  - Stoper, [36](#)

- getError
  - ContinueException, [10](#)
  - CriticalException, [12](#)
  - Exception, [13](#)

- IKolejka
  - ~IKolejka, [14](#)
  - dequeue, [14](#)
  - enqueue, [15](#)
  - get, [15](#)
  - isEmpty, [15](#)
- IKolejka< T >, [13](#)
- ILista
  - ~ILista, [16](#)
  - add, [17](#)
  - get, [17](#)
  - isEmpty, [17](#)
  - remove, [18](#)
  - size, [18](#)
- ILista< T >, [15](#)
- IRunnable, [18](#)
  - ~IRunnable, [19](#)
- prepare, [19](#)

- run, 19
- IStoper, 20
  - ~IStoper, 20
  - getElapsedTimeMs, 21
  - start, 21
  - stop, 21
- IStos
  - ~IStos, 22
  - get, 22
  - isEmpty, 22
  - pull, 23
  - push, 23
- IStos< T >, 21
- isEmpty
  - IKolejka, 15
  - ILista, 17
  - IStos, 22
  - Itabn, 25
  - Kolejka, 29
  - Lista, 32
  - Stos, 40
  - tabn, 43
- Itabn
  - ~Itabn, 24
  - aSize, 25
  - add, 24, 25
  - bubblesort, 25
  - isEmpty, 25
  - nOE, 26
  - operator[], 26
  - remove, 26
  - show, 26
  - showElems, 26
- Itabn< T >, 23
- Kolejka
  - ~Kolejka, 28
  - dequeue, 29
  - enqueue, 29
  - get, 29
  - isEmpty, 29
  - Kolejka, 28
- Kolejka< T >, 27
- kolejka.cpp, 51
- kolejka.hh, 52
- Lista
  - ~Lista, 31
  - add, 31
  - get, 31
  - isEmpty, 32
  - Lista, 31
  - remove, 32
  - size, 33
- Lista< T >, 30
- lista.cpp, 53
- lista.hh, 53
- lista\_test, 33
  - ~lista\_test, 34
- lista\_test, 34
- prepare, 34
- run, 35
- main
  - main.cpp, 56
- main.cpp, 55
  - dumpToFile, 55
  - main, 56
  - printOnscreen, 56
- main.hh, 57
  - dumpToFile, 58
  - printOnscreen, 58
- nOE
  - Itabn, 26
  - tabn, 44
- operator[]
  - Itabn, 26
  - tabn, 44
- prepare
  - IRunnable, 19
  - lista\_test, 34
  - tabn\_test, 47
- printOnscreen
  - main.cpp, 56
  - main.hh, 58
- pull
  - IStos, 23
  - Stos, 40
- push
  - IStos, 23
  - Stos, 41
- remove
  - ILista, 18
  - Itabn, 26
  - Lista, 32
  - tabn, 44, 45
- run
  - IRunnable, 19
  - lista\_test, 35
  - tabn\_test, 47
- run.cpp, 59
- run.hh, 59
- SIZE
  - tabl.hh, 67
- show
  - Itabn, 26
  - tabn, 45
- showElems
  - Itabn, 26
  - tabn, 45
- size
  - ILista, 18
  - Lista, 33
- start

- IStoper, [21](#)
- Stoper, [38](#)
- stop
  - IStoper, [21](#)
  - Stoper, [38](#)
- Stoper, [35](#)
  - getElapsedTimeMs, [36](#)
  - start, [38](#)
  - stop, [38](#)
- stoper.cpp, [61](#)
- stoper.hh, [61](#)
- Stos
  - ~Stos, [39](#)
  - get, [40](#)
  - isEmpty, [40](#)
  - pull, [40](#)
  - push, [41](#)
  - Stos, [39](#)
- Stos< T >, [38](#)
- stos.cpp, [62](#)
- stos.hh, [63](#)
- tabl.cpp, [65](#)
- tabl.hh, [65](#)
  - SIZE, [67](#)
- tabn
  - ~tabn, [42](#)
  - aSize, [43](#)
  - add, [43](#)
  - bubblesort, [43](#)
  - isEmpty, [43](#)
  - nOE, [44](#)
  - operator[], [44](#)
  - remove, [44](#), [45](#)
  - show, [45](#)
  - showElems, [45](#)
  - tabn, [42](#)
- tabn< T >, [41](#)
- tabn\_test, [45](#)
  - ~tabn\_test, [47](#)
  - prepare, [47](#)
  - run, [47](#)
  - tabn\_test, [46](#)