

pamsi
0.5

Wygenerowano przez Doxygen 1.8.9.1

Pn, 21 mar 2016 10:15:18

Spis treści

1	Strona główna	1
1.1	Dokumentacja klas w repozytorium pamsi.	1
2	Indeks hierarchiczny	3
2.1	Hierarchia klas	3
3	Indeks klas	5
3.1	Lista klas	5
4	Indeks plików	7
4.1	Lista plików	7
5	Dokumentacja klas	9
5.1	Dokumentacja szablonu klasy <code>IKolejka< T ></code>	9
5.1.1	Opis szczegółowy	9
5.1.2	Dokumentacja konstruktora i destruktoru	10
5.1.2.1	<code>~IKolejka</code>	10
5.1.3	Dokumentacja funkcji składowych	10
5.1.3.1	<code>dequeue</code>	10
5.1.3.2	<code>enqueue</code>	10
5.1.3.3	<code>get</code>	10
5.1.3.4	<code>isEmpty</code>	10
5.2	Dokumentacja szablonu klasy <code>ILista< T ></code>	11
5.2.1	Opis szczegółowy	12
5.2.2	Dokumentacja konstruktora i destruktoru	12
5.2.2.1	<code>~ILista</code>	12
5.2.3	Dokumentacja funkcji składowych	12
5.2.3.1	<code>add</code>	12
5.2.3.2	<code>add</code>	12
5.2.3.3	<code>get</code>	12
5.2.3.4	<code>isEmpty</code>	13
5.2.3.5	<code>remove</code>	13
5.2.3.6	<code>remove</code>	13

5.2.3.7	size	13
5.3	Dokumentacja klasy IRunnable	14
5.3.1	Opis szczegółowy	14
5.3.2	Dokumentacja konstruktora i destruktora	14
5.3.2.1	~IRunnable	14
5.3.3	Dokumentacja funkcji składowych	15
5.3.3.1	prepare	15
5.3.3.2	run	16
5.4	Dokumentacja klasy IStoper	16
5.4.1	Opis szczegółowy	17
5.4.2	Dokumentacja konstruktora i destruktora	17
5.4.2.1	~IStoper	17
5.4.3	Dokumentacja funkcji składowych	17
5.4.3.1	getElapsedTimeMs	17
5.4.3.2	start	18
5.4.3.3	stop	18
5.5	Dokumentacja szablonu klasy IStos< T >	18
5.5.1	Opis szczegółowy	19
5.5.2	Dokumentacja konstruktora i destruktora	19
5.5.2.1	~IStos	19
5.5.3	Dokumentacja funkcji składowych	19
5.5.3.1	get	19
5.5.3.2	isEmpty	20
5.5.3.3	pull	20
5.5.3.4	push	20
5.6	Dokumentacja szablonu klasy Itabn< T >	20
5.6.1	Opis szczegółowy	21
5.6.2	Dokumentacja konstruktora i destruktora	22
5.6.2.1	~Itabn	22
5.6.3	Dokumentacja funkcji składowych	22
5.6.3.1	add	22
5.6.3.2	add	22
5.6.3.3	aSize	22
5.6.3.4	bubblesort	23
5.6.3.5	isEmpty	23
5.6.3.6	nOE	23
5.6.3.7	operator[]	24
5.6.3.8	operator[]	24
5.6.3.9	remove	24
5.6.3.10	remove	24

5.6.3.11	show	24
5.6.3.12	showElems	24
5.7	Dokumentacja szablonu klasy Kolejka< T >	25
5.7.1	Opis szczegółowy	26
5.7.2	Dokumentacja konstruktora i destruktora	26
5.7.2.1	Kolejka	26
5.7.2.2	~Kolejka	26
5.7.3	Dokumentacja funkcji składowych	26
5.7.3.1	dequeue	26
5.7.3.2	enqueue	26
5.7.3.3	get	26
5.7.3.4	isEmpty	27
5.8	Dokumentacja szablonu klasy Lista< T >	27
5.8.1	Opis szczegółowy	28
5.8.2	Dokumentacja konstruktora i destruktora	28
5.8.2.1	Lista	28
5.8.2.2	~Lista	29
5.8.3	Dokumentacja funkcji składowych	29
5.8.3.1	add	29
5.8.3.2	add	29
5.8.3.3	get	29
5.8.3.4	isEmpty	29
5.8.3.5	remove	30
5.8.3.6	remove	30
5.8.3.7	size	30
5.9	Dokumentacja klasy lista_test	31
5.9.1	Opis szczegółowy	31
5.9.2	Dokumentacja konstruktora i destruktora	32
5.9.2.1	lista_test	32
5.9.2.2	~lista_test	32
5.9.3	Dokumentacja funkcji składowych	32
5.9.3.1	prepare	32
5.9.3.2	run	32
5.10	Dokumentacja klasy Starter	33
5.10.1	Opis szczegółowy	33
5.10.2	Dokumentacja konstruktora i destruktora	33
5.10.2.1	Starter	33
5.10.2.2	~Starter	33
5.10.3	Dokumentacja funkcji składowych	33
5.10.3.1	dumpToFile	33

5.10.3.2	printResults	34
5.10.3.3	setTestSize	34
5.10.3.4	test	35
5.11	Dokumentacja klasy Stoper	35
5.11.1	Opis szczegółowy	36
5.11.2	Dokumentacja funkcji składowych	36
5.11.2.1	getElapsedTimeMs	36
5.11.2.2	start	36
5.11.2.3	stop	37
5.12	Dokumentacja szablonu klasy Stos< T >	37
5.12.1	Opis szczegółowy	38
5.12.2	Dokumentacja konstruktora i destruktor	38
5.12.2.1	Stos	38
5.12.2.2	~Stos	38
5.12.3	Dokumentacja funkcji składowych	38
5.12.3.1	get	38
5.12.3.2	isEmpty	39
5.12.3.3	pull	39
5.12.3.4	push	39
5.13	Dokumentacja szablonu klasy tabn< T >	39
5.13.1	Opis szczegółowy	41
5.13.2	Dokumentacja konstruktora i destruktor	41
5.13.2.1	tabn	41
5.13.2.2	~tabn	41
5.13.3	Dokumentacja funkcji składowych	41
5.13.3.1	add	41
5.13.3.2	add	41
5.13.3.3	aSize	42
5.13.3.4	bubblesort	42
5.13.3.5	isEmpty	42
5.13.3.6	nOE	42
5.13.3.7	operator[]	42
5.13.3.8	operator[]	43
5.13.3.9	remove	43
5.13.3.10	remove	43
5.13.3.11	show	43
5.13.3.12	showElems	43
5.14	Dokumentacja klasy tabn_test	44
5.14.1	Opis szczegółowy	45
5.14.2	Dokumentacja konstruktora i destruktor	45

5.14.2.1	tabn_test	45
5.14.2.2	~tabn_test	45
5.14.3	Dokumentacja funkcji składowych	45
5.14.3.1	prepare	45
5.14.3.2	run	45
6	Dokumentacja plików	47
6.1	Dokumentacja pliku kolejka.cpp	47
6.2	Dokumentacja pliku kolejka.hh	47
6.3	Dokumentacja pliku lista.cpp	48
6.4	Dokumentacja pliku lista.hh	49
6.5	Dokumentacja pliku main.cpp	51
6.5.1	Opis szczegółowy	51
6.5.2	Dokumentacja funkcji	51
6.5.2.1	main	51
6.6	Dokumentacja pliku main.hh	52
6.7	Dokumentacja pliku run.cpp	53
6.8	Dokumentacja pliku run.hh	53
6.8.1	Opis szczegółowy	54
6.9	Dokumentacja pliku starter.cpp	55
6.10	Dokumentacja pliku starter.hh	55
6.10.1	Opis szczegółowy	56
6.11	Dokumentacja pliku stoper.cpp	57
6.12	Dokumentacja pliku stoper.hh	57
6.13	Dokumentacja pliku stos.cpp	58
6.14	Dokumentacja pliku stos.hh	59
6.15	Dokumentacja pliku tabl.cpp	60
6.16	Dokumentacja pliku tabl.hh	61
6.16.1	Opis szczegółowy	62
6.16.2	Dokumentacja definicji	62
6.16.2.1	SIZE	62
Indeks		63

Rozdział 1

Strona główna

1.1 Dokumentacja klas w repozytorium pamsi.

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

Rozdział 2

Indeks hierarchiczny

2.1 Hierarchia klas

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

IKolejka< T >	9
Kolejka< T >	25
ILista< T >	11
Lista< T >	27
ILista< string >	11
IRunnable	14
lista_test	31
tabn_test	44
IStoper	16
Stoper	35
IStos< T >	18
Stos< T >	37
Itabn< T >	20
tabn< T >	39
Itabn< int >	20
Starter	33

Rozdział 3

Indeks klas

3.1 Lista klas

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

IKolejka< T >	Interfejs klasy Kolejka Definiuje operacje dostępne dla klasy Kolejka	9
ILista< T >	Interfejs listy	11
IRunnable	Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm	14
IStoper	Interfejs IStoper	16
IStos< T >	Interfejs stosu	18
Itabn< T >	Interfejs klasy tabn	20
Kolejka< T >	Klasa modeluje kolejkę	25
Lista< T >	Klasa lista	27
lista_test	Definiuje sposób testowania wypełniania listy	31
Starter	Klasa pozwala na przeprowadzenie testów	33
Stoper	Klasa stoper implementująca interfejs IStoper	35
Stos< T >	Klasa Stos	37
tabn< T >	Modeluje tablicę dynamicznie rozszerzalną	39
tabn_test	Definiuje sposób testowania wypełniania tablicy tabn	44

Rozdział 4

Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

kolejka.cpp	47
kolejka.hh	47
lista.cpp	48
lista.hh	49
main.cpp	
Główny plik programu	51
main.hh	52
run.cpp	53
run.hh	
Plik definiuje interfejs IRunnable , ujednolicający klasy umożliwiające badanie algorytmów	53
starter.cpp	55
starter.hh	
Plik definiuje klasę Starter	55
stoper.cpp	57
stoper.hh	57
stos.cpp	58
stos.hh	59
tabl.cpp	60
tabl.hh	
Definicja interfejsu Itabn , klasy tabn oraz klasy tabn_test	61

Rozdział 5

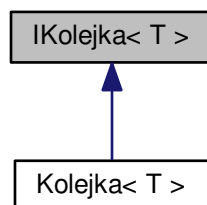
Dokumentacja klas

5.1 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.

5.1.1 Opis szczegółowy

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

Interfejs klasy [Kolejka](#) Definiuje operacje dostępne dla klasy [Kolejka](#).

Definicja w linii 15 pliku kolejka.hh.

5.1.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny interfejsu.

Definicja w linii 47 pliku kolejka.hh.

5.1.3 Dokumentacja funkcji składowych

5.1.3.1 `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 >](#).

5.1.3.2 `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 >](#).

5.1.3.3 `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 >](#).

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

Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

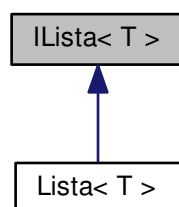
- [kolejka.hh](#)

5.2 Dokumentacja szablonu klasy ILista< T >

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`.

5.2.1 Opis szczegółowy

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

Interfejs listy.

Definiuje dostępne operacje na klasie [Lista](#)

Definicja w linii 19 pliku lista.hh.

5.2.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny interfejsu [ILista](#).

Definicja w linii 77 pliku lista.hh.

5.2.3 Dokumentacja funkcji składowych

5.2.3.1 `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:



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

Dodaje element na koniec listy.

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

5.2.3.3 `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 >](#).

5.2.3.4 `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 >](#).

5.2.3.5 `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

<i>T</i>	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 >](#).

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

Usuwa element z końca listy.

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

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

Zwraca ilość elementów w liście.

Zwracane wartości

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

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

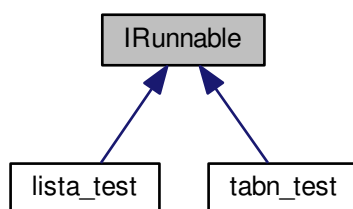
- [lista.hh](#)

5.3 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](#).

5.3.1 Opis szczegółowy

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

Definicja w linii 18 pliku run.hh.

5.3.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny [IRunnable](#).

Definicja w linii 37 pliku run.hh.

5.3.3 Dokumentacja funkcji składowych

5.3.3.1 `virtual bool IRunnable::prepare (int) [pure virtual]`

Przygotowanie badań

Zwracane wartości

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

Implementowany w [tabn_test](#) i [lista_test](#).

Oto graf wywoływań tej funkcji:



5.3.3.2 `virtual bool IRunnable::run () [pure virtual]`

Przeprowadzanie badań

Zwracane wartości

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

Implementowany w [tabn_test](#) i [lista_test](#).

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

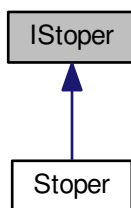
- [run.hh](#)

5.4 Dokumentacja klasy IStoper

Interfejs [IStoper](#).

```
#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](#) ()

5.4.1 Opis szczegółowy

Interfejs [IStoper](#).

Definicja w linii 21 pliku stoper.hh.

5.4.2 Dokumentacja konstruktora i destruktor

5.4.2.1 `virtual IStoper::~~IStoper () [inline],[virtual]`

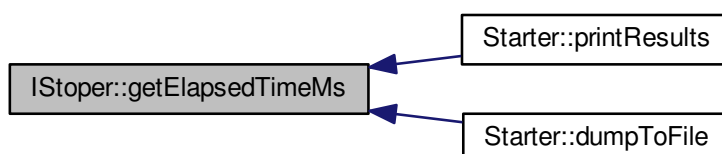
Definicja w linii 27 pliku stoper.hh.

5.4.3 Dokumentacja funkcji składowych

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

Implementowany w [Stoper](#).

Oto graf wywoływań tej funkcji:



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

Implementowany w [Stoper](#).

Oto graf wywoływań tej funkcji:



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

Implementowany w [Stoper](#).

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

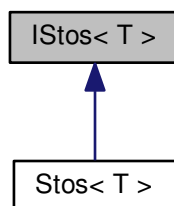
- [stoper.hh](#)

5.5 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.

5.5.1 Opis szczegółowy

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

Interfejs stosu.

Definiuje dostępne operacje na klasie **Stos**

Definicja w linii 17 pliku stos.hh.

5.5.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny **IStos**.

Definicja w linii 54 pliku stos.hh.

5.5.3 Dokumentacja funkcji składowych

5.5.3.1 `template<class T> virtual T IStos< T >::get (void) [pure 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.

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

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

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

Sprawdza, czy stos jest pusty.

Zwracane wartości

<i>0</i>	gdy niepusty
<i>1</i>	gdy pusty

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

5.5.3.3 `template<class T> virtual T IStos< T>::pull (void) [pure 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.

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

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

5.5.3.4 `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:

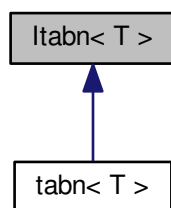
- [stos.hh](#)

5.6 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 void **remove** ()=0
Usuwa element z końca tablicy.
- virtual void **remove** (int)=0
Usuwa wybrany element z listy.
- virtual T **show** (int)=0
Zwraca żądany element, o ile istnieje.
- virtual void **showElems** (void)=0
Wyświetla elementy listy.
- 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.

5.6.1 Opis szczegółowy

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

Interfejs klasy tabn.

Definiuje jednolity sposób dostępu do tablicy rozszerzalnej.

Definicja w linii 22 pliku tabl.hh.

5.6.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny interfejsu.

Definicja w linii 85 pliku `tabl.hh`.

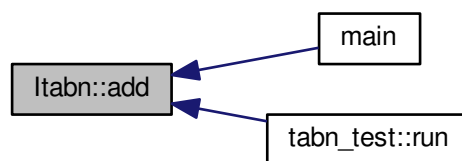
5.6.3 Dokumentacja funkcji składowych

5.6.3.1 `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:



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

Implementowany w `tabn< T >`.

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

Zwraca ilość miejsca w tablicy.

Implementowany w `tabn< T >`.

Oto graf wywoływań tej funkcji:



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

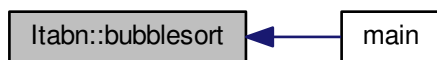
Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

Ostrzeżenie

Wymaga typu danych ze zdefiniowanym operatorem porównania "większe od"

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

Oto graf wywoływań tej funkcji:



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

Sprawdza, czy tablica jest pusta.

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

5.6.3.6 `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:



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

Pozwala na dostęp do dowolnego elementu.

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

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

Pozwala na dostęp do dowolnego elementu.

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

5.6.3.9 `template<class T> virtual void Itabn< T >::remove () [pure virtual]`

Usuwa element z końca tablicy.

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

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

Usuwa wybrany element z listy.

Parametry

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

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

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

Zwraca żądany element, o ile istnieje.

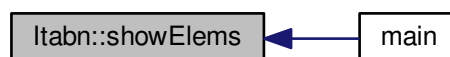
Implementowany w [tabn< T >](#).

5.6.3.12 `template<class T> virtual void Itabn< T >::showElems (void) [pure virtual]`

Wyświetla elementy listy.

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

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

5.7 Dokumentacja szablonu klasy Kolejka< T >

Klasa modeluje kolejkę

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

Diagram dziedziczenia dla Kolejka< T >

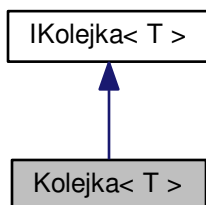
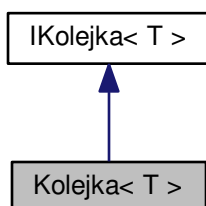


Diagram współpracy dla Kolejka< T >:



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.

5.7.1 Opis szczegółowy

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

Klasa modeluje kolejkę

Definicja w linii 54 pliku kolejka.hh.

5.7.2 Dokumentacja konstruktora i destruktora

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

Konstruktor tablicy obsługującej kolejkę

Definicja w linii 61 pliku kolejka.hh.

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

Destruktor klasy [Kolejka](#).

Definicja w linii 104 pliku kolejka.hh.

5.7.3 Dokumentacja funkcji składowych

5.7.3.1 `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.

5.7.3.2 `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.

5.7.3.3 `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.

5.7.3.4 `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](#)

5.8 Dokumentacja szablonu klasy Lista< T >

Klasa lista.

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

Diagram dziedziczenia dla Lista< T >

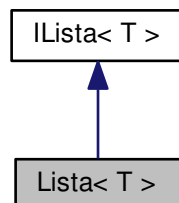
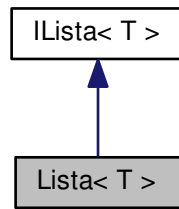


Diagram współpracy dla Lista< T >:



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.

5.8.1 Opis szczegółowy

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

Klasa lista.

Modeluje pojęcie listy

Definicja w linii 86 pliku lista.hh.

5.8.2 Dokumentacja konstruktora i destruktora

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

Konstruktor tablicy obsługującej listę

Definicja w linii 93 pliku lista.hh.

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

Destruktor Listy.

Definicja w linii 173 pliku lista.hh.

5.8.3 Dokumentacja funkcji składowych

5.8.3.1 `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 179 pliku lista.hh.

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

Dodaje element na koniec listy.

Implementuje `ILista< T >`.

Definicja w linii 184 pliku lista.hh.

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

Zwraca element z zadanego miejsca bez usunięcia.

Zwracane wartości

<code>T</code>	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 208 pliku lista.hh.

5.8.3.4 `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 203 pliku lista.hh.

5.8.3.5 `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 189 pliku lista.hh.

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

Usuwa element z końca listy.

Implementuje [ILista< T >](#).

Definicja w linii 196 pliku lista.hh.

5.8.3.7 `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 224 pliku lista.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

5.9 Dokumentacja klasy lista_test

Definiuje sposób testowania wypełniania listy.

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

Diagram dziedziczenia dla lista_test

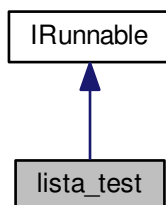
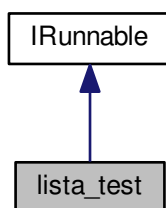


Diagram współpracy dla lista_test:



Metody publiczne

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

5.9.1 Opis szczegółowy

Definiuje sposób testowania wypełniania listy.

Definicja w linii 243 pliku lista.hh.

5.9.2 Dokumentacja konstruktora i destruktora

5.9.2.1 lista_test::lista_test () [inline]

Konstruktor klasy testującej.

Definicja w linii 254 pliku lista.hh.

5.9.2.2 lista_test::~~lista_test () [inline]

Destruktor klasy testującej.

Definicja w linii 260 pliku lista.hh.

5.9.3 Dokumentacja funkcji składowych

5.9.3.1 virtual bool lista_test::prepare (int *sizeOfTest*) [inline],[virtual]

Przygotowuje rozmiar testu.

Parametry

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

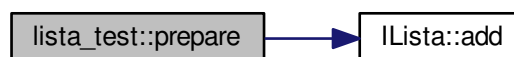
Zwracane wartości

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

Implementuje [IRunnable](#).

Definicja w linii 291 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



5.9.3.2 virtual bool lista_test::run () [inline],[virtual]

Wykonuje test.

Pozwala na wykonanie testu.

Zwracane wartości

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

Implementuje [IRunnable](#).

Definicja w linii 317 pliku lista.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

5.10 Dokumentacja klasy Starter

Klasa pozwala na przeprowadzenie testów.

```
#include <starter.hh>
```

Metody publiczne

- [Starter](#) ()
Konstruktor klasy tabn.
- virtual [~Starter](#) ()
Destruktor klasy tabn.
- void [setTestSize](#) (unsigned int)
Metoda ustawia wielkość testu.
- void [printResults](#) (void)
Metoda wyświetla czas trwania testu na standardowym wyjściu.
- void [test](#) (void)
Metoda przeprowadza test.
- void [dumpToFile](#) (string)
Metoda dopisuje dane do pliku.

5.10.1 Opis szczegółowy

Klasa pozwala na przeprowadzenie testów.

Definicja w linii 18 pliku starter.hh.

5.10.2 Dokumentacja konstruktora i destruktora

5.10.2.1 [Starter::Starter](#) () [inline]

Konstruktor klasy tabn.

Definicja w linii 28 pliku starter.hh.

5.10.2.2 virtual [Starter::~~Starter](#) () [inline],[virtual]

Destruktor klasy tabn.

Definicja w linii 34 pliku starter.hh.

5.10.3 Dokumentacja funkcji składowych

5.10.3.1 void [Starter::dumpToFile](#) (string *nameOfFile*)

Metoda dopisuje dane do pliku.

Format zapisu: wielkość_testu czas_trwania_ms

Parametry

<i>nameOfFile</i>	- nazwa pliku wyjściowego
-------------------	---------------------------

Definicja w linii 20 pliku starter.cpp.

Oto graf wywołań dla tej funkcji:



5.10.3.2 void Starter::printResults (void)

Metoda wyświetla czas trwania testu na standardowym wyjściu.

Definicja w linii 8 pliku starter.cpp.

Oto graf wywołań dla tej funkcji:

5.10.3.3 void Starter::setTestSize (unsigned int *testsize*)

Metoda ustawia wielkość testu.

Parametry

<i>testsize</i>	- wielkość testu
-----------------	------------------

Definicja w linii 3 pliku starter.cpp.

Oto graf wywołań dla tej funkcji:

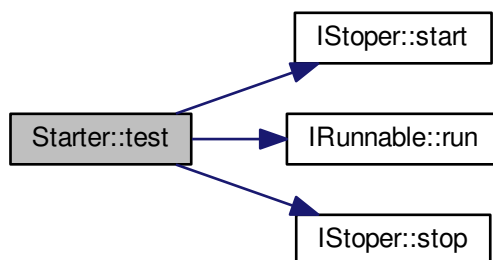


5.10.3.4 void Starter::test (void)

Metoda przeprowadza test.

Definicja w linii 14 pliku starter.cpp.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z plików:

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

5.11 Dokumentacja klasy Stoper

Klasa `stoper` implementująca interfejs `IStoper`.

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

Diagram dziedziczenia dla `Stoper`

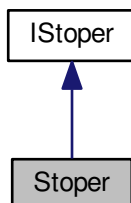
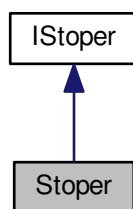


Diagram współpracy dla Stoper:



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.

5.11.1 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 37 pliku `stoper.hh`.

5.11.2 Dokumentacja funkcji składowych

5.11.2.1 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`.

5.11.2.2 void Stoper::start (void) [virtual]

Uruchamia zegar.

Implementuje `IStoper`.

Definicja w linii 4 pliku `stoper.cpp`.

5.11.2.3 `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](#)

5.12 Dokumentacja szablonu klasy Stos< T >

Klasa [Stos](#).

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

Diagram dziedziczenia dla Stos< T >

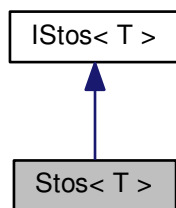
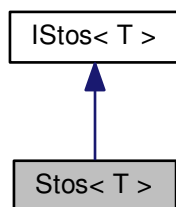


Diagram współpracy dla Stos< T >:



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.

5.12.1 Opis szczegółowy

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

Klasa `Stos`.

Modeluje pojęcie stosu

Definicja w linii 64 pliku `stos.hh`.

5.12.2 Dokumentacja konstruktora i destruktora

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

Konstruktor tablicy obsługującej stos.

Definicja w linii 71 pliku `stos.hh`.

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

Destruktor stosu.

Definicja w linii 128 pliku `stos.hh`.

5.12.3 Dokumentacja funkcji składowych

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

Zwraca element ze szczytu stosu bez jego usuwania.

Zwracane wartości

<code>T</code>	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 151 pliku `stos.hh`.

5.12.3.2 `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 146 pliku `stos.hh`.

5.12.3.3 `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 139 pliku `stos.hh`.

5.12.3.4 `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 134 pliku `stos.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [stos.hh](#)

5.13 Dokumentacja szablonu klasy `tabn< T >`

Modeluje tablicę dynamicznie rozszerzalną

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

Diagram dziedziczenia dla `tabn< T >`

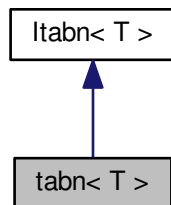
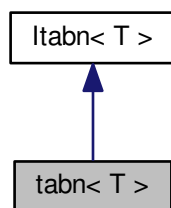


Diagram współpracy dla `tabn< T >`:



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 void remove ()`
Usuwa ostatni element z listy.
- `virtual void remove (int)`
Usuwa wybrany element z listy, przesuwając wszystkie następne elementy o miejsce w lewo.
- `virtual T show (int)`
Zwraca żądany element, o ile istnieje.
- `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.

5.13.1 Opis szczegółowy

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

Modeluje tablicę dynamicznie rozszerzalną

Przechowuje elementy w rozszerzalnej tablicy o rozmiarze początkowym SIZE

Definicja w linii 101 pliku `tabl.hh`.

5.13.2 Dokumentacja konstruktora i destruktora

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

Konstruktor klasy `tabn`.

Definicja w linii 112 pliku `tabl.hh`.

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

Destruktor klasy `tabn`.

Definicja w linii 121 pliku `tabl.hh`.

5.13.3 Dokumentacja funkcji składowych

5.13.3.1 `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 260 pliku `tabl.hh`.

5.13.3.2 `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.

Implementuje `ltabn< T >`.

Definicja w linii 268 pliku `tabl.hh`.

5.13.3.3 `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 447 pliku `tabl.hh`.

5.13.3.4 `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"

Implementuje `ltabn< T >`.

Definicja w linii 462 pliku `tabl.hh`.

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

Sprawdza, czy tablica jest pusta.

Zwracane wartości

<i>0</i>	gdy tablica nie jest pusta
<i>1</i>	gdy tablica jest pusta

Implementuje `ltabn< T >`.

Definicja w linii 394 pliku `tabl.hh`.

5.13.3.6 `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 442 pliku `tabl.hh`.

5.13.3.7 `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 411 pliku `tabl.hh`.

5.13.3.8 `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 416 pliku `tabl.hh`.

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

Usuwa ostatni element z listy.

Implementuje `ltabn< T >`.

Definicja w linii 300 pliku `tabl.hh`.

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

Usuwa wybrany element z listy, przesuując wszystkie następne elementy o miejsce w lewo.

Parametry

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

Implementuje `ltabn< T >`.

Definicja w linii 323 pliku `tabl.hh`.

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

Zwraca żądany element, o ile istnieje.

Implementuje `ltabn< T >`.

Definicja w linii 421 pliku `tabl.hh`.

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

Wyświetla listę elementów.

Implementuje `ltabn< T >`.

Definicja w linii 432 pliku `tabl.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

5.14 Dokumentacja klasy `tabn_test`

Definiuje sposób testowania wypełniania tablicy `tabn`.

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

Diagram dziedziczenia dla `tabn_test`

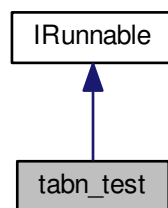
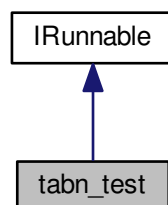


Diagram współpracy dla `tabn_test`:



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.

5.14.1 Opis szczegółowy

Definiuje sposób testowania wypełniania tablicy `tabn`.

Definicja w linii 491 pliku `tabl.hh`.

5.14.2 Dokumentacja konstruktora i destruktora

5.14.2.1 `tabn_test::tabn_test ()` `[inline]`

Konstruktor klasy `tabn_test`.

Definicja w linii 499 pliku `tabl.hh`.

5.14.2.2 `virtual tabn_test::~~tabn_test ()` `[inline]`, `[virtual]`

Destruktor klasy `tabn_test`.

Definicja w linii 505 pliku `tabl.hh`.

5.14.3 Dokumentacja funkcji składowych

5.14.3.1 `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 536 pliku `tabl.hh`.

5.14.3.2 `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()`

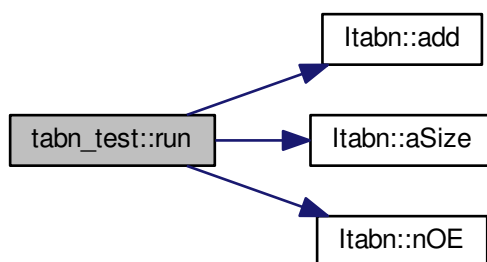
Zwracane wartości

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

Implementuje `IRunnable`.

Definicja w linii 551 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

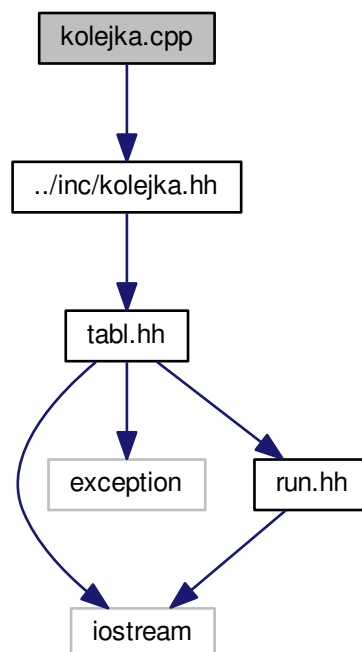
Rozdział 6

Dokumentacja plików

6.1 Dokumentacja pliku kolejka.cpp

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

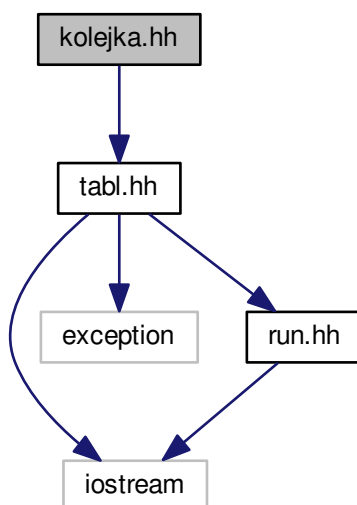
Wykres zależności załączania dla kolejka.cpp:



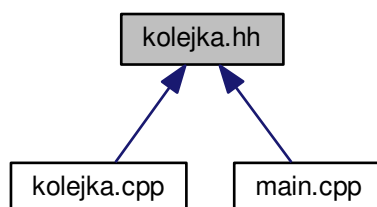
6.2 Dokumentacja pliku kolejka.hh

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

Wykres zależności załączania dla kolejka.hh:



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



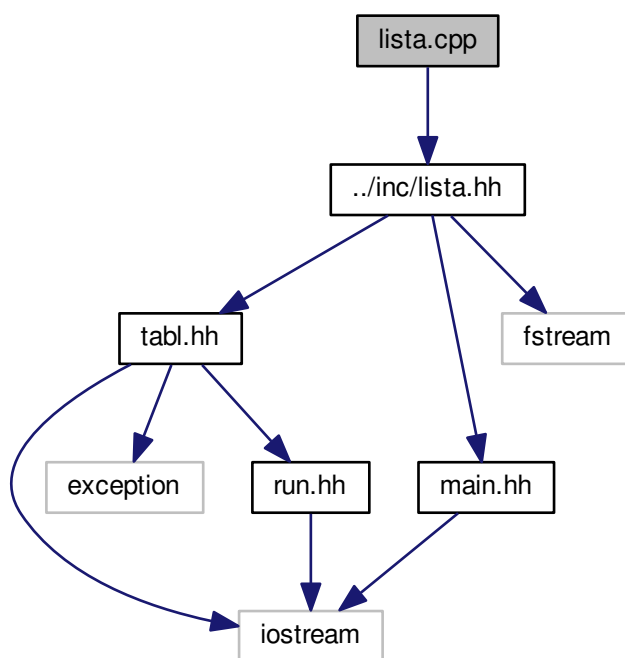
Komponenty

- class `IKolejka< T >`
Interfejs klasy `Kolejka` Definiuje operacje dostępne dla klasy `Kolejka`.
- class `Kolejka< T >`
Klasa modeluje kolejkę

6.3 Dokumentacja pliku lista.cpp

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

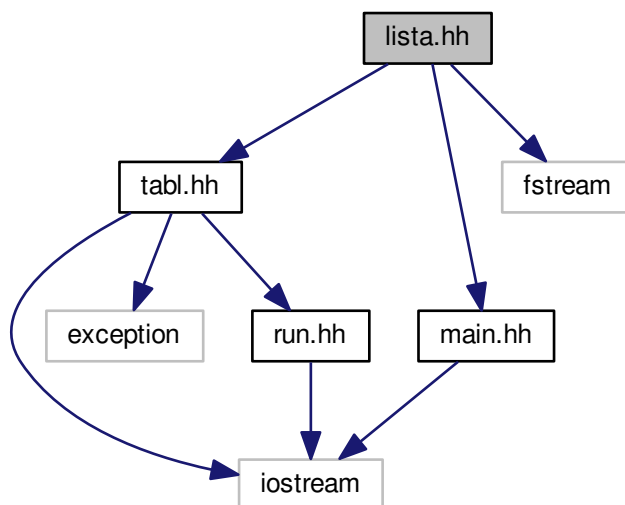

Wykres zależności załączania dla lista.cpp:



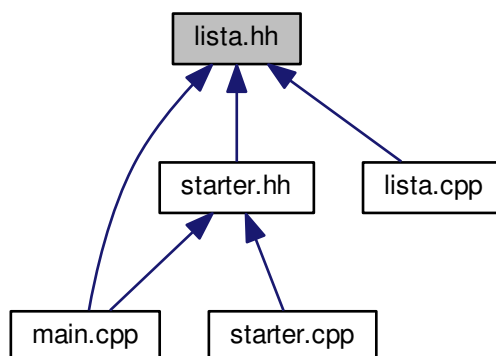
6.4 Dokumentacja pliku lista.hh

```
#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

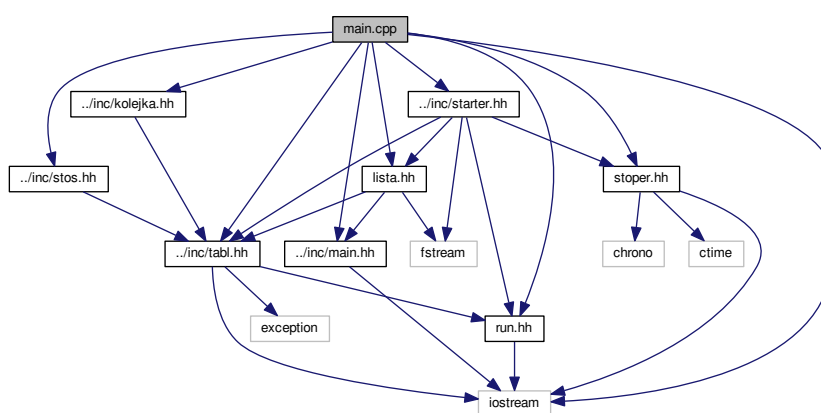
- class `ILista< T >`
Interfejs listy.
- class `Lista< T >`
Klasa lista.
- class `lista_test`
Definiuje sposób testowania wypełniania listy.

6.5 Dokumentacja pliku main.cpp

Główny plik programu.

```
#include <iostream>
#include "../inc/main.hh"
#include "../inc/tabl.hh"
#include "../inc/run.hh"
#include "../inc/starter.hh"
#include "../inc/stoper.hh"
#include "../inc/lista.hh"
#include "../inc/stos.hh"
#include "../inc/kolejka.hh"
```

Wykres zależności załączania dla main.cpp:



Funkcje

- int `main` (void)

6.5.1 Opis szczegółowy

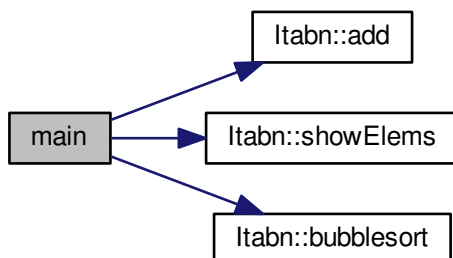
Główny plik programu.

6.5.2 Dokumentacja funkcji

6.5.2.1 int main (void)

Definicja w linii 21 pliku main.cpp.

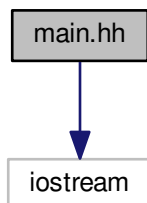
Oto graf wywołań dla tej funkcji:



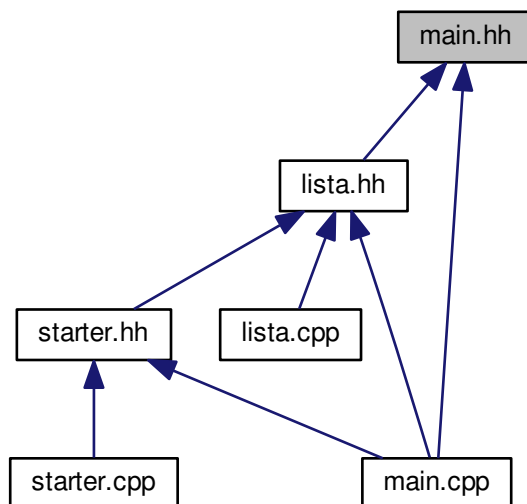
6.6 Dokumentacja pliku main.hh

```
#include <iostream>
```

Wykres zależności załączania dla main.hh:



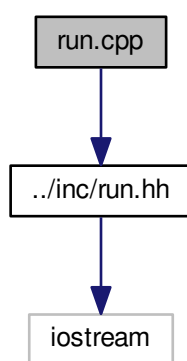
Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



6.7 Dokumentacja pliku run.cpp

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

Wykres zależności załączania dla run.cpp:

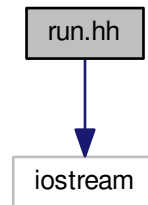


6.8 Dokumentacja pliku run.hh

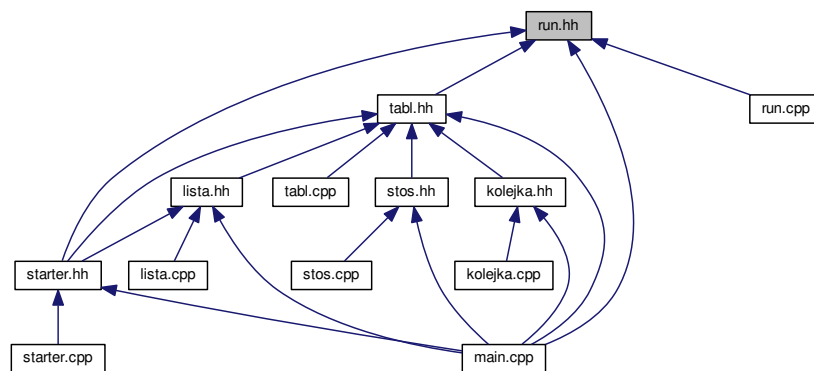
Plik definiuje interfejs [IRunnable](#), ujednolicający klasy umożliwiające badanie algorytmów.

```
#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.

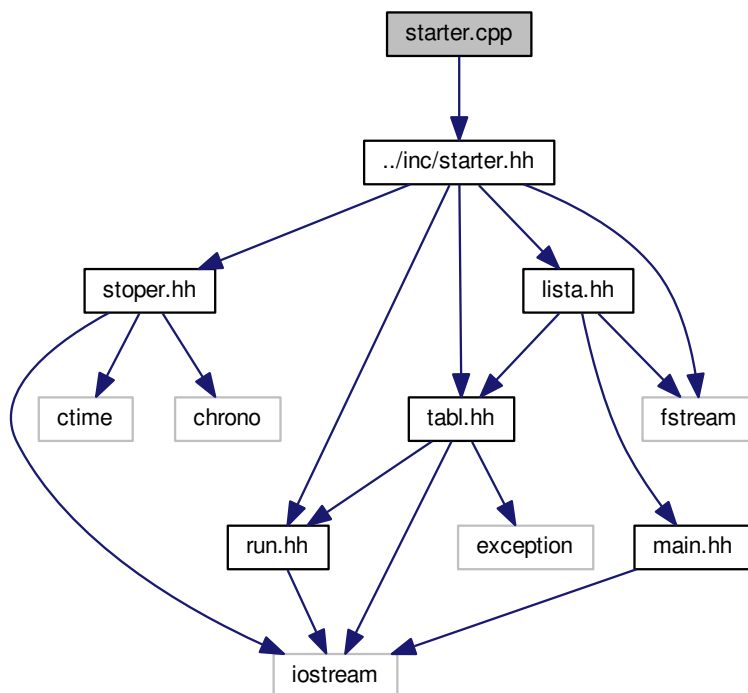
6.8.1 Opis szczegółowy

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

6.9 Dokumentacja pliku starter.cpp

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

Wykres zależności załączania dla starter.cpp:

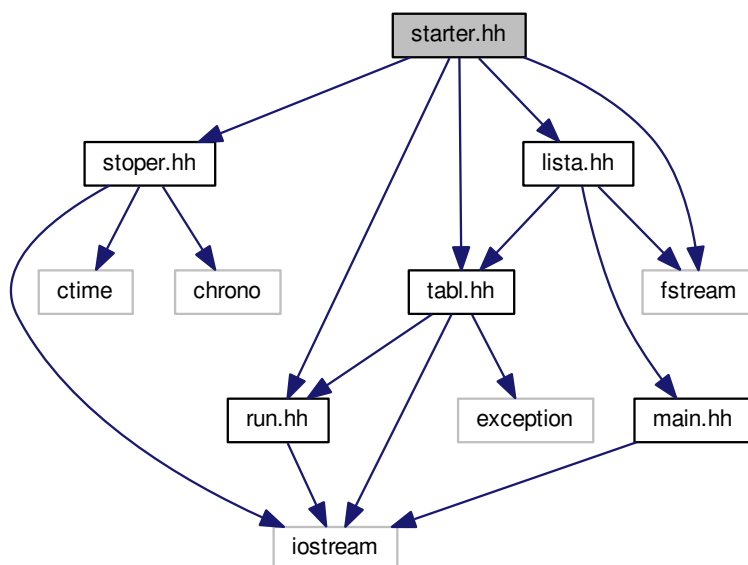


6.10 Dokumentacja pliku starter.hh

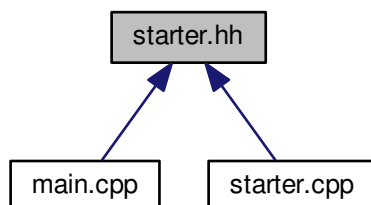
Plik definiuje klasę [Starter](#).

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

Wykres zależności załączania dla starter.hh:



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



Komponenty

- class [Starter](#)

Klasa pozwala na przeprowadzenie testów.

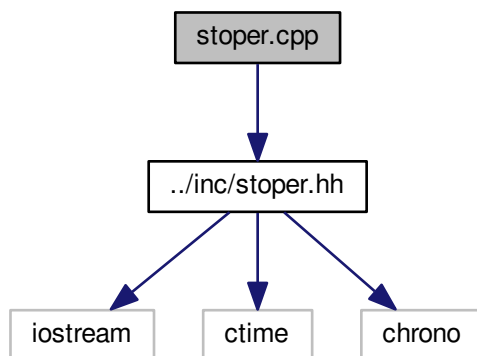
6.10.1 Opis szczegółowy

Plik definiuje klasę [Starter](#).

6.11 Dokumentacja pliku stoper.cpp

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

Wykres zależności załączania dla stoper.cpp:



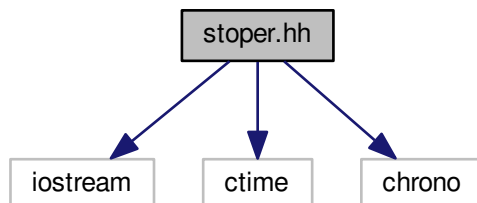
6.12 Dokumentacja pliku stoper.hh

```
#include <iostream>
```

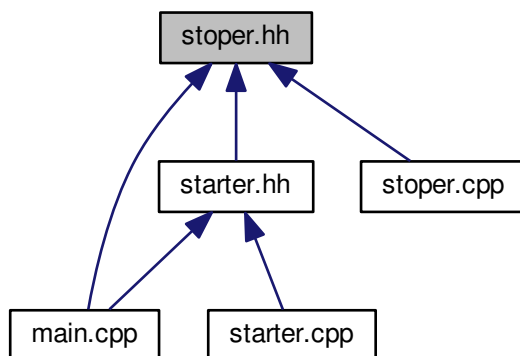
```
#include <ctime>
```

```
#include <chrono>
```

Wykres zależności załączania dla stoper.hh:



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



Komponenty

- class [IStoper](#)

Interfejs [IStoper](#).

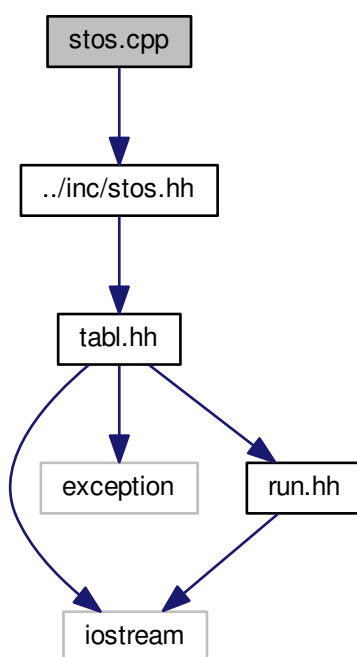
- class [Stoper](#)

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

6.13 Dokumentacja pliku stos.cpp

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

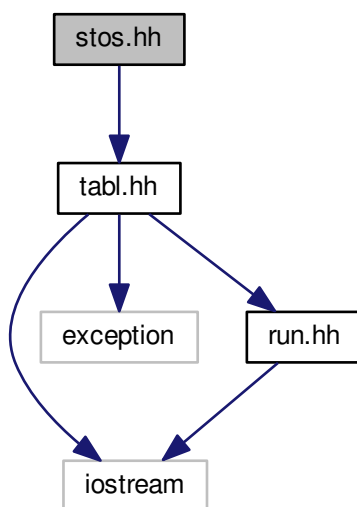
Wykres zależności załączania dla stos.cpp:



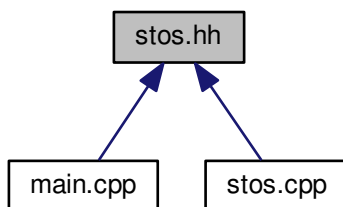
6.14 Dokumentacja pliku stos.hh

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

Wykres zależności załączania dla stos.hh:



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



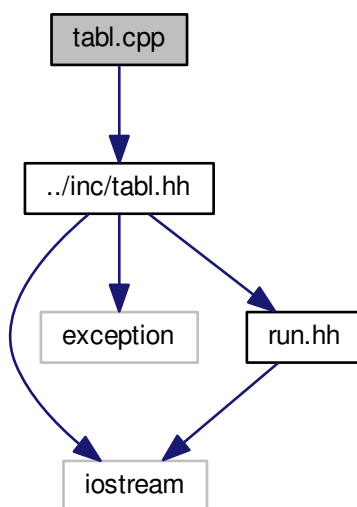
Komponenty

- 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:

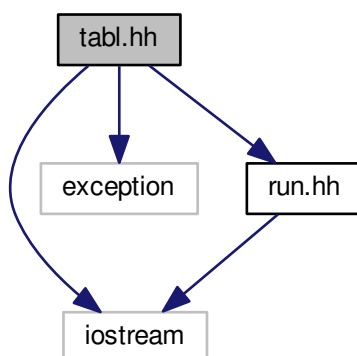


6.16 Dokumentacja pliku tabl.hh

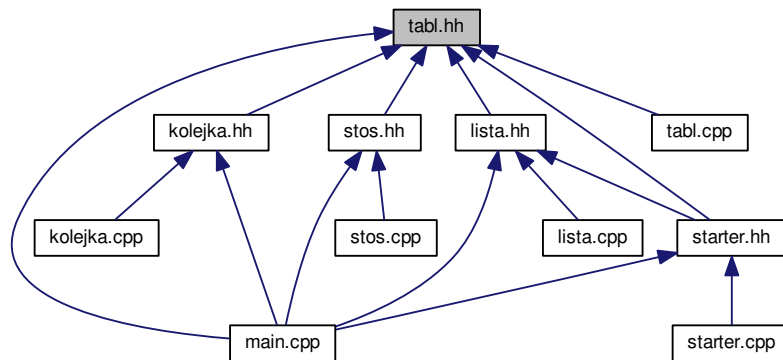
Definicja interfejsu `ltabn`, klasy `tabn` oraz klasy `tabn_test`.

```
#include <iostream>
#include <exception>
#include "run.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, [10](#)
- ~ILista
 - ILista, [12](#)
- ~IRunnable
 - IRunnable, [14](#)
- ~IStoper
 - IStoper, [17](#)
- ~IStos
 - IStos, [19](#)
- ~Itabn
 - Itabn, [22](#)
- ~Kolejka
 - Kolejka, [26](#)
- ~Lista
 - Lista, [28](#)
- ~Starter
 - Starter, [33](#)
- ~Stos
 - Stos, [38](#)
- ~lista_test
 - lista_test, [32](#)
- ~tabn
 - tabn, [41](#)
- ~tabn_test
 - tabn_test, [45](#)

- aSize
 - Itabn, [22](#)
 - tabn, [42](#)

- add
 - ILista, [12](#)
 - Itabn, [22](#)
 - Lista, [29](#)
 - tabn, [41](#)

- bubblesort
 - Itabn, [23](#)
 - tabn, [42](#)

- dequeue
 - IKolejka, [10](#)
 - Kolejka, [26](#)

- dumpToFile
 - Starter, [33](#)

- enqueue
 - IKolejka, [10](#)
 - Kolejka, [26](#)

- get

- IKolejka, [10](#)
- ILista, [12](#)
- IStos, [19](#)
- Kolejka, [26](#)
- Lista, [29](#)
- Stos, [38](#)
- getElapsedTimeMs
 - IStoper, [17](#)
 - Stoper, [36](#)

- IKolejka
 - ~IKolejka, [10](#)
 - dequeue, [10](#)
 - enqueue, [10](#)
 - get, [10](#)
 - isEmpty, [10](#)
- IKolejka< T >, [9](#)
- ILista
 - ~ILista, [12](#)
 - add, [12](#)
 - get, [12](#)
 - isEmpty, [13](#)
 - remove, [13](#)
 - size, [13](#)
- ILista< T >, [11](#)
- IRunnable, [14](#)
 - ~IRunnable, [14](#)
 - prepare, [15](#)
 - run, [16](#)
- IStoper, [16](#)
 - ~IStoper, [17](#)
 - getElapsedTimeMs, [17](#)
 - start, [18](#)
 - stop, [18](#)
- IStos
 - ~IStos, [19](#)
 - get, [19](#)
 - isEmpty, [20](#)
 - pull, [20](#)
 - push, [20](#)
- IStos< T >, [18](#)
- isEmpty
 - IKolejka, [10](#)
 - ILista, [13](#)
 - IStos, [20](#)
 - Itabn, [23](#)
 - Kolejka, [27](#)
 - Lista, [29](#)
 - Stos, [39](#)
 - tabn, [42](#)

- ltabn
 - ~ltabn, [22](#)
 - aSize, [22](#)
 - add, [22](#)
 - bubblesort, [23](#)
 - isEmpty, [23](#)
 - nOE, [23](#)
 - operator[], [24](#)
 - remove, [24](#)
 - show, [24](#)
 - showElems, [24](#)
- ltabn< T >, [20](#)
- Kolejka
 - ~Kolejka, [26](#)
 - dequeue, [26](#)
 - enqueue, [26](#)
 - get, [26](#)
 - isEmpty, [27](#)
 - Kolejka, [26](#)
- Kolejka< T >, [25](#)
- kolejka.cpp, [47](#)
- kolejka.hh, [47](#)
- Lista
 - ~Lista, [28](#)
 - add, [29](#)
 - get, [29](#)
 - isEmpty, [29](#)
 - Lista, [28](#)
 - remove, [30](#)
 - size, [30](#)
- Lista< T >, [27](#)
- lista.cpp, [48](#)
- lista.hh, [49](#)
- lista_test, [31](#)
 - ~lista_test, [32](#)
 - lista_test, [32](#)
 - prepare, [32](#)
 - run, [32](#)
- main
 - main.cpp, [51](#)
- main.cpp, [51](#)
 - main, [51](#)
- main.hh, [52](#)
- nOE
 - ltabn, [23](#)
 - tabn, [42](#)
- operator[]
 - ltabn, [24](#)
 - tabn, [42, 43](#)
- prepare
 - IRunnable, [15](#)
 - lista_test, [32](#)
 - tabn_test, [45](#)
- printResults
 - Starter, [34](#)
- pull
 - IStos, [20](#)
 - Stos, [39](#)
- push
 - IStos, [20](#)
 - Stos, [39](#)
- remove
 - ILista, [13](#)
 - ltabn, [24](#)
 - Lista, [30](#)
 - tabn, [43](#)
- run
 - IRunnable, [16](#)
 - lista_test, [32](#)
 - tabn_test, [45](#)
- run.cpp, [53](#)
- run.hh, [53](#)
- SIZE
 - tabl.hh, [62](#)
- setTestSize
 - Starter, [34](#)
- show
 - ltabn, [24](#)
 - tabn, [43](#)
- showElems
 - ltabn, [24](#)
 - tabn, [43](#)
- size
 - ILista, [13](#)
 - Lista, [30](#)
- start
 - IStoper, [18](#)
 - Stoper, [36](#)
- Starter, [33](#)
 - ~Starter, [33](#)
 - dumpToFile, [33](#)
 - printResults, [34](#)
 - setTestSize, [34](#)
 - Starter, [33](#)
 - test, [34](#)
- starter.cpp, [55](#)
- starter.hh, [55](#)
- stop
 - IStoper, [18](#)
 - Stoper, [36](#)
- Stoper, [35](#)
 - getElapsedTimeMs, [36](#)
 - start, [36](#)
 - stop, [36](#)
- stoper.cpp, [57](#)
- stoper.hh, [57](#)
- Stos
 - ~Stos, [38](#)
 - get, [38](#)
 - isEmpty, [39](#)
 - pull, [39](#)

- push, [39](#)
 - Stos, [38](#)
- Stos< T >, [37](#)
- stos.cpp, [58](#)
- stos.hh, [59](#)

- tabl.cpp, [60](#)
- tabl.hh, [61](#)
 - SIZE, [62](#)
- tabn
 - ~tabn, [41](#)
 - aSize, [42](#)
 - add, [41](#)
 - bubblesort, [42](#)
 - isEmpty, [42](#)
 - nOE, [42](#)
 - operator[], [42](#), [43](#)
 - remove, [43](#)
 - show, [43](#)
 - showElems, [43](#)
 - tabn, [41](#)
- tabn< T >, [39](#)
- tabn_test, [44](#)
 - ~tabn_test, [45](#)
 - prepare, [45](#)
 - run, [45](#)
 - tabn_test, [45](#)
- test
 - Starter, [34](#)