pamsi

0.5

Wygenerowano przez Doxygen 1.8.9.1

N, 3 kwi 2016 20:04:30

Spis treści

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:

Rozdział 3

Indeks klas

3.1 Lista klas

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

Rozdział 4

Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

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

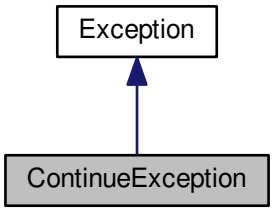
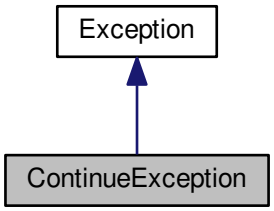


Diagram współpracy dla ContinueException:



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 destruktora

5.1.2.1 ContinueException::ContinueException (std::string *description*) [inline]

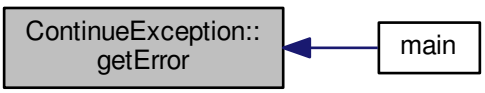
Definicja w linii 55 pliku except.hh.

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

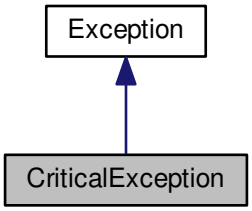
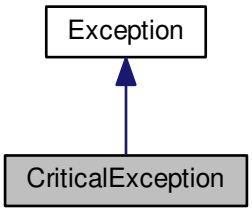


Diagram współpracy dla CriticalException:



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 destruktora

5.2.2.1 CriticalException::CriticalException (std::string *description*) [inline]

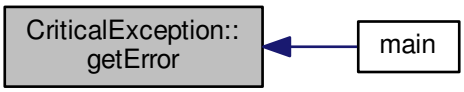
Definicja w linii 38 pliku except.hh.

5.2.3 Dokumentacja funkcji składowych

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

Definicja w linii 41 pliku except.hh.

Oto graf wywoływań 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:

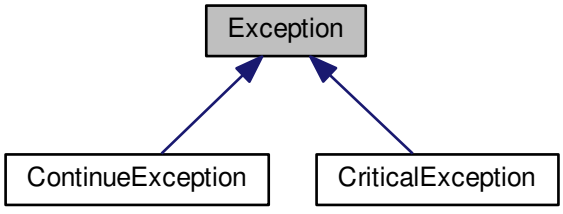
- except.hh

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:

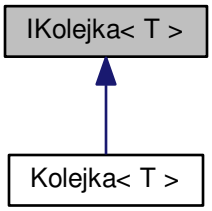
- except.hh

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

T	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

element	- 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

0	gdy niepusta
1	gdy pusta

Implementowany w Kolejka< T >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

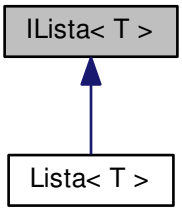
- kolejka.hh

5.5 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.

5.5.1 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 IListа< T >::add (T , int) [pure virtual]

Dodaje element do zadanego miejsca listy.

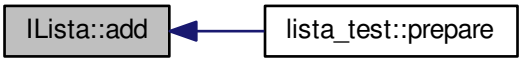
Jeśli następuje próba dodania elementu w miejscu istniejącego, następuje przesunięcie następujących po nim elementów na następne pozycje

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 IListа< T >::add (T) [pure virtual]

Dodaje element na koniec listy.

Implementowany w Lista< T >.

5.5.3.3 template<class T> virtual T IListа< T >::get (int) [pure virtual]

Zwraca element z zadanego miejsca bez usunięcia.

Zwracane wartości

T	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 IListа< 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 IList< 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 IList< T >::remove (void) [pure virtual]

Usuwa element z końca listy.

Implementowany w Lista< T >.

5.5.3.7 template<class T> virtual int IList< 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:

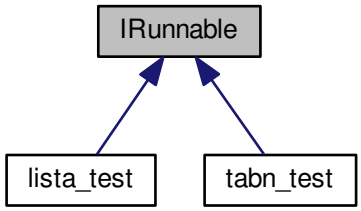
- lista.hh

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

true	zawsze

Implementowany w tabn_test i lista_test.

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

Przeprowadzanie badań

Zwracane wartości

true	zawsze

Implementowany w tabn_test i lista_test.

Dokumentacja dla tej klasy została wygenerowana z pliku:

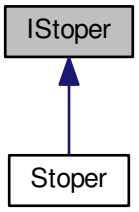
- run.hh

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 destruktora

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

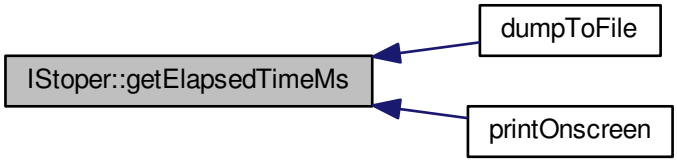
Definicja w linii 26 pliku stoper.hh.

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:

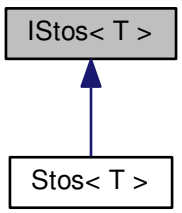
- stoper.hh

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

T	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

element	- element do umieszczenia na stosie

Implementowany w Stos< T >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

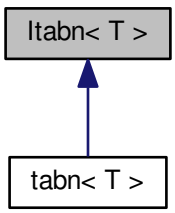
- stos.hh

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 żadany 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 witrualny 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 witrualny interfejsu.

Definicja w linii 83 pliku tabl.hh.

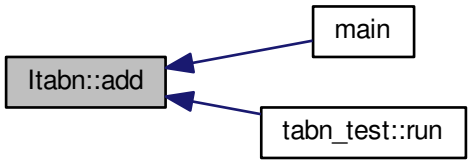
5.9.3 Dokumentacja funkcji składowych

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 wywoływań 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

element	wstawiany element
position	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 wywoływań 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

position	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 żadany element, o ile istnieje.

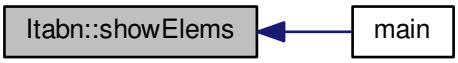
Implementowany w tabn< T >.

5.9.3.12 template<class T> virtual void Itabn< 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 >

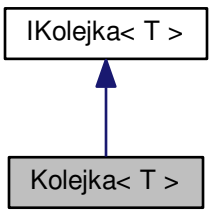
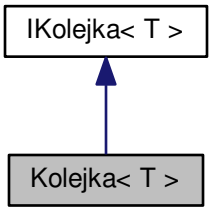


Diagram współpracy dla Kolejka< T >:



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

T	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

element	- 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 >

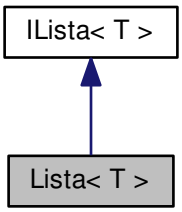
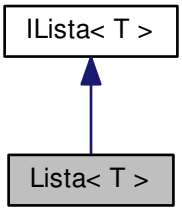


Diagram współpracy dla Lista< T >:



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

T	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

T	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

int	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

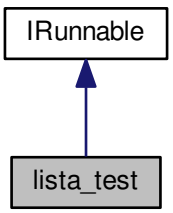
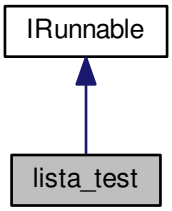


Diagram współpracy dla lista_test:



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

sizeOfTest	- rozmiar testu

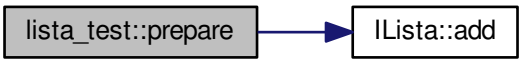
Zwracane wartości

true	gdy plik ze słownikiem został pomyślnie otwarty
false	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

true	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

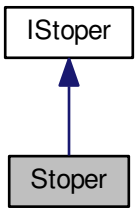
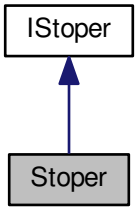


Diagram współpracy dla Stoper:



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

long_double	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 >

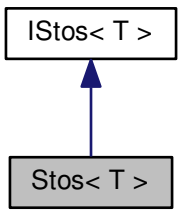
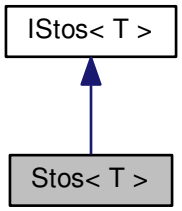


Diagram współpracy dla Stos< T >:



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

T	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

T	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

element	- 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 >

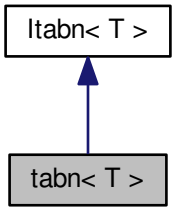
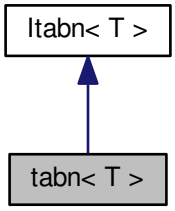


Diagram współpracy dla tabn< T >:



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 żadany 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

element	- element do dodania

Implementuje Itabn< 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

element	- wstawiany element
positionShifted	- indeks pola, w które ma być wstawiony element.

Wyjątki

ContinueException	przy próbie dodania elementu do niewłaściwego miejsca.

Implementuje Itabn< 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

int	Ilość zaalokowanych pól

Implementuje Itabn< 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

CriticalException	re-throw swap(int,int)

Implementuje Itabn< 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 Itabn< 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

int	Liczba elementów w tablicy

Implementuje Itabn< 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

index	- numer elementu tablicy

Zwracane wartości

T∗	Wskaźnik na wybrany element tablicy

Implementuje Itabn< 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

index	- numer elementu tablicy

Zwracane wartości

T	Element tablicy

Implementuje Itabn< 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

CriticalException	przy próbie usunięcia z pustej tablicy.

Implementuje Itabn< 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

position	indeks pola, z którego ma być usunięty element.

Wyjątki

CriticalException	przy próbie usunięcia z pustej tablicy lub nieistniejącego elementu.
ContinueException	re-throw reduce2()

Implementuje Itabn< T >.

Definicja w linii 341 pliku tabl.hh.

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

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

Wyjątki

CriticalException	przy próbie odczytania z pustej tablicy lub dostępu do nieistniejącego elementu.

Implementuje Itabn< 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 Itabn< 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

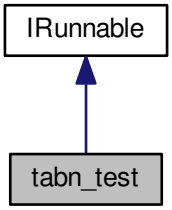
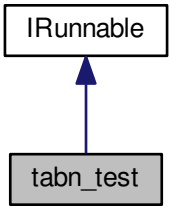


Diagram współpracy dla tabn_test:



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

sizeOfTest	- rozmiar testu

Zwracane wartości

bool	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

bool	zawsze true

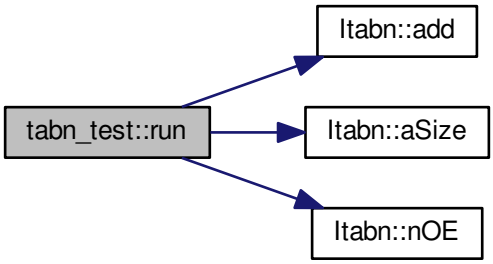
Wyjątki

ContinueException	re-throw tabn::add(int)

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

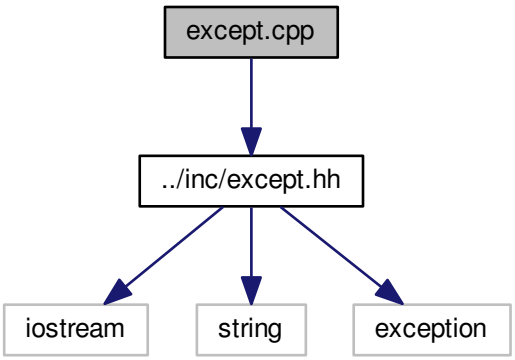
Rozdział 6

Dokumentacja plików

6.1 Dokumentacja pliku except.cpp

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

Wykres zależności załączania dla except.cpp:

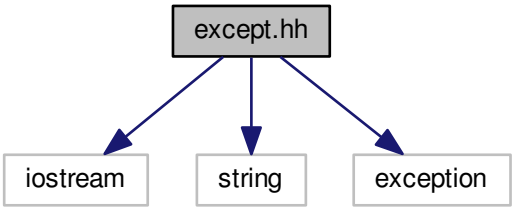


6.2 Dokumentacja pliku except.hh

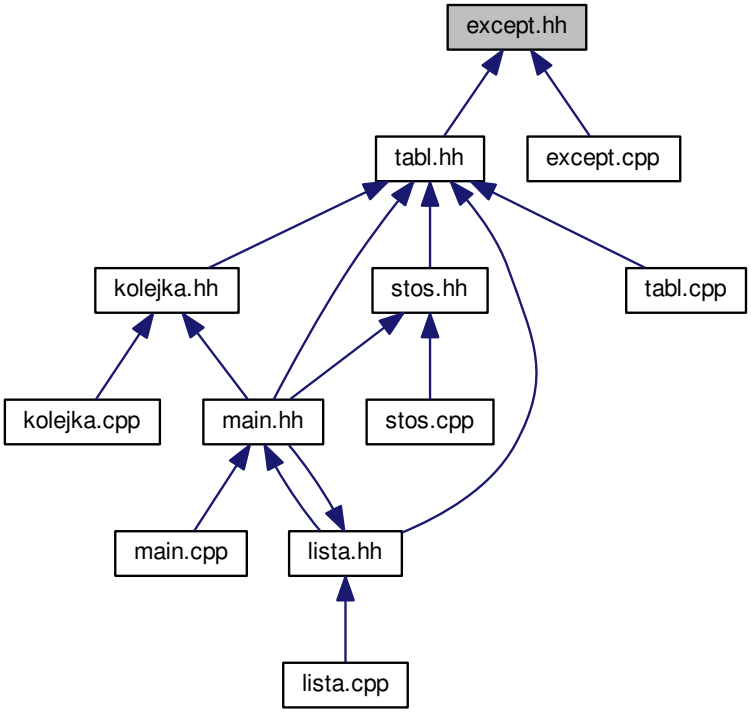
Plik zawiera definicje wyjątków.

```
#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.

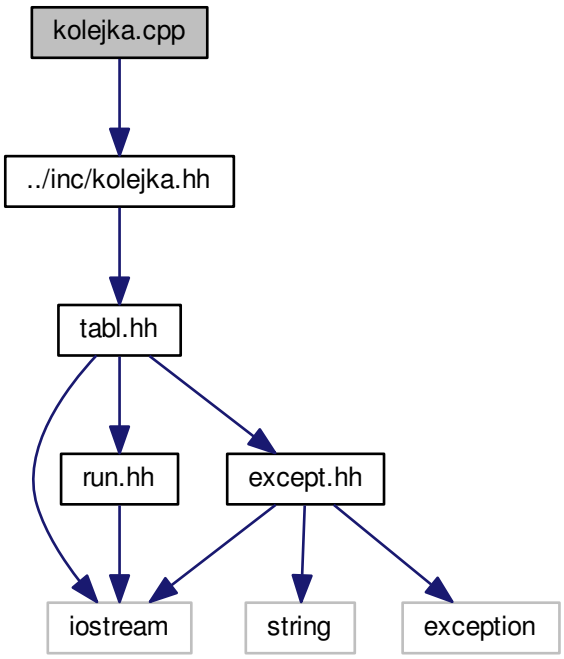
6.2.1 Opis szczegółowy

Plik zawiera definicje wyjątków.

6.3 Dokumentacja pliku kolejka.cpp

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

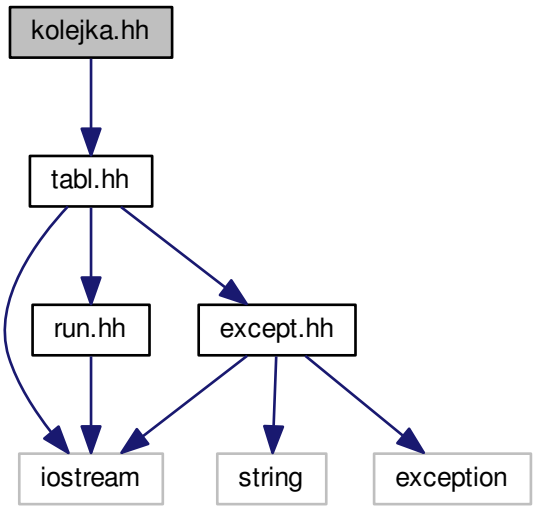
Wykres zależności załączania dla kolejka.cpp:



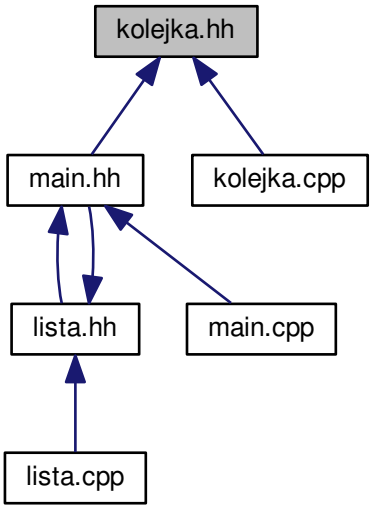
6.4 Dokumentacja pliku kolejka.hh

```
#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.

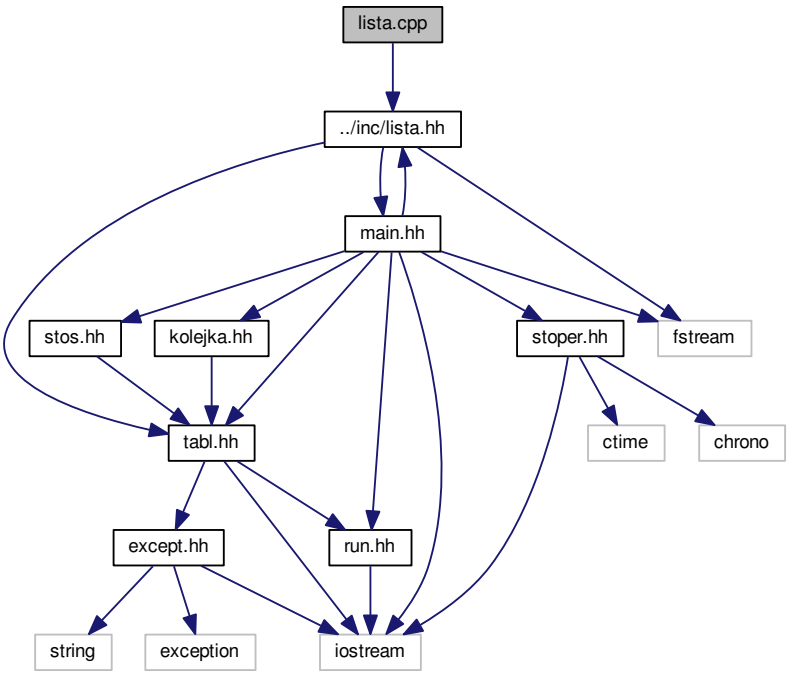
- class Kolejka< T >

  Klasa modeluje kolejkę

6.5 Dokumentacja pliku lista.cpp

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

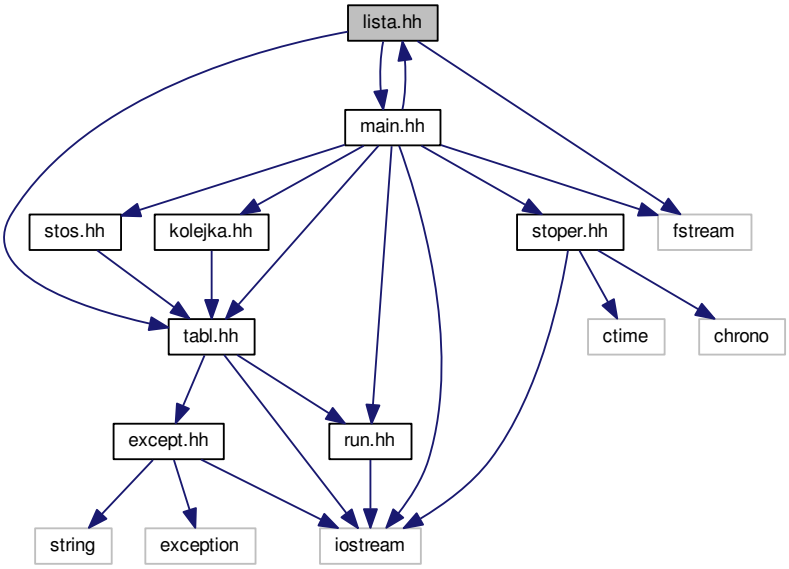
Wykres zależności załączania dla lista.cpp:



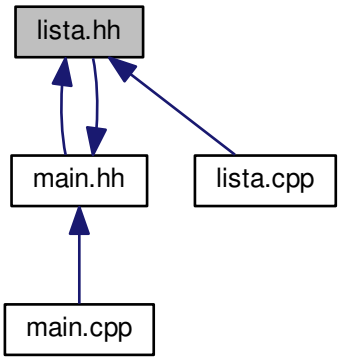
6.6 Dokumentacja pliku lista.hh

```
#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

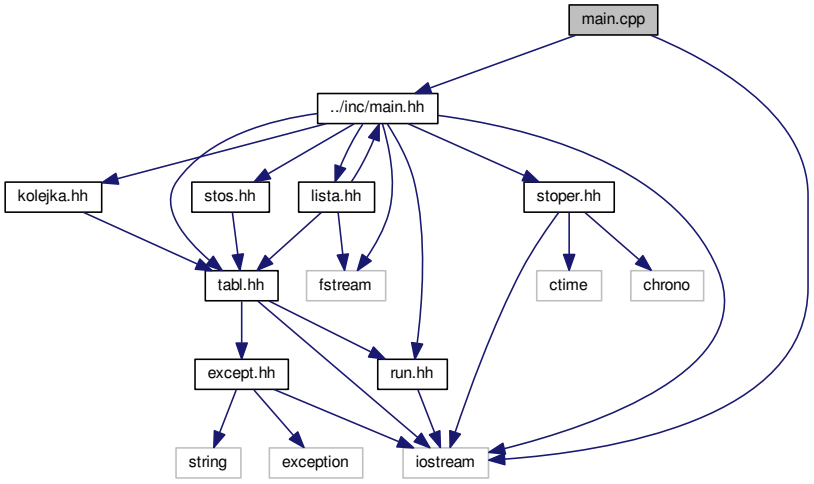
  Definiuje sposób testowania wypełniania listy.

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

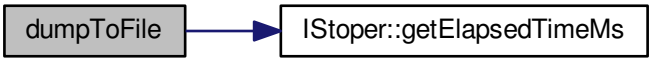
Główny plik programu.

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.

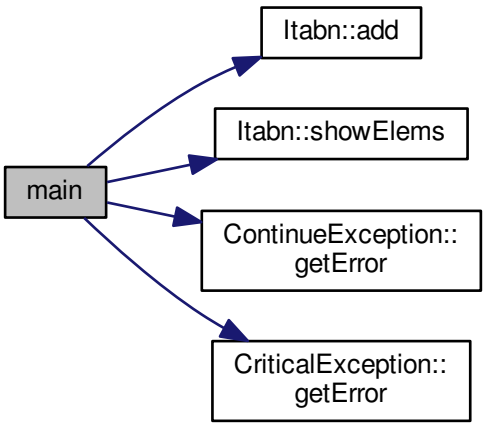
Oto graf wywołań dla tej funkcji:



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

testSize	rozmiar testu
stoper	klasa stopera

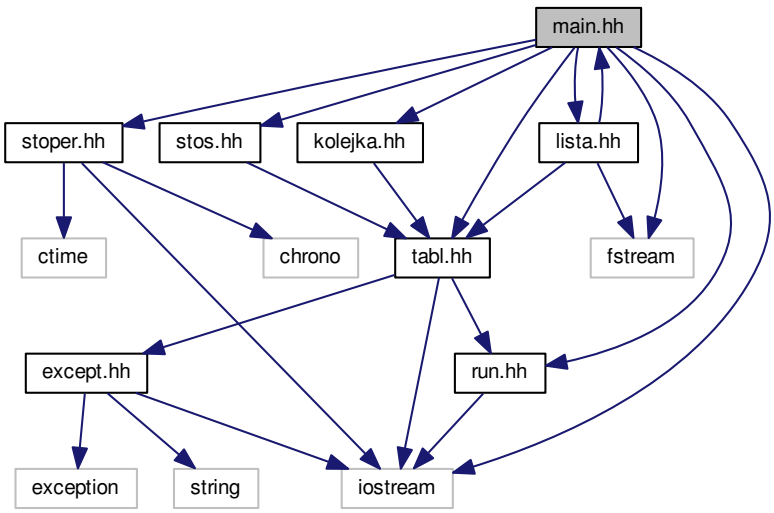
Definicja w linii 100 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

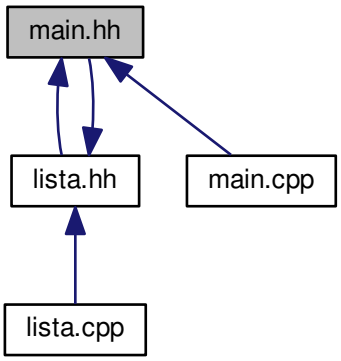
6.8 Dokumentacja pliku main.hh

```
#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

nameOfFile	nazwa pliku wynikowego
testsize	rozmiar testu
stoper	klasa stopera

6.8.1.2 void printOnscreen (unsigned *int,* IStoper ∗)

Wyświetla wynik na standardowym wyjściu.

Parametry

testSize	rozmiar testu
stoper	klasa stopera

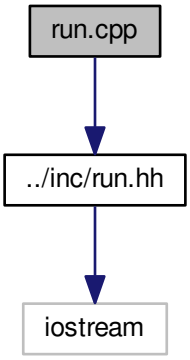
Definicja w linii 100 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

6.9 Dokumentacja pliku run.cpp

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

Wykres zależności załączania dla run.cpp:

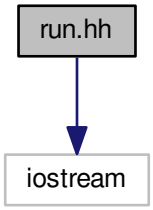


6.10 Dokumentacja pliku run.hh

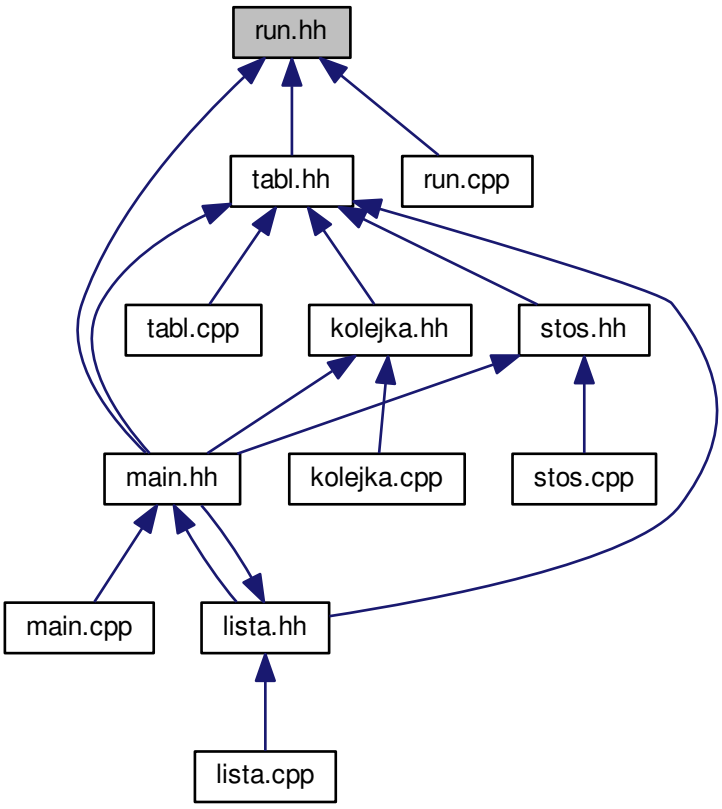
Plik definiuje interfejs IRunnable, ujednolicający klasy umożliwiające badanie algorytmów.

```
#include <iostream>
```

Wykres zależności załączania dla run.hh:



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



Komponenty

- class IRunnable

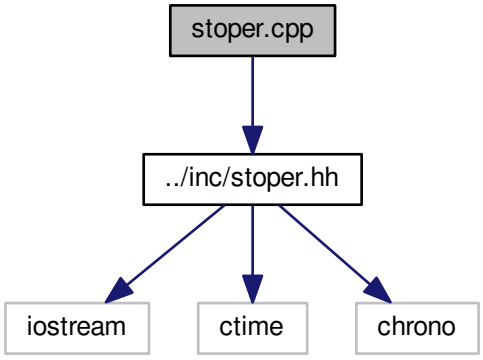
Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

6.10.1 Opis szczegółowy

Plik definiuje interfejs IRunnable, ujednolicający klasy umożliwiające badanie algorytmów.

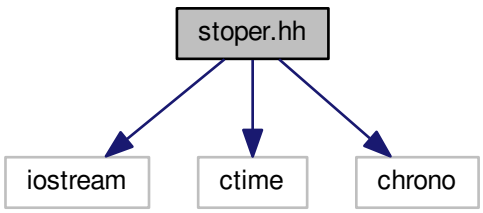
6.11 Dokumentacja pliku stoper.cpp

```
#include "../inc/stoper.hh"
```
Wykres zależności załączania dla stoper.cpp:

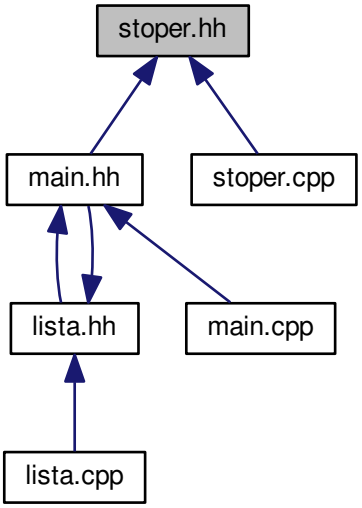


6.12 Dokumentacja pliku stoper.hh

```
#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.

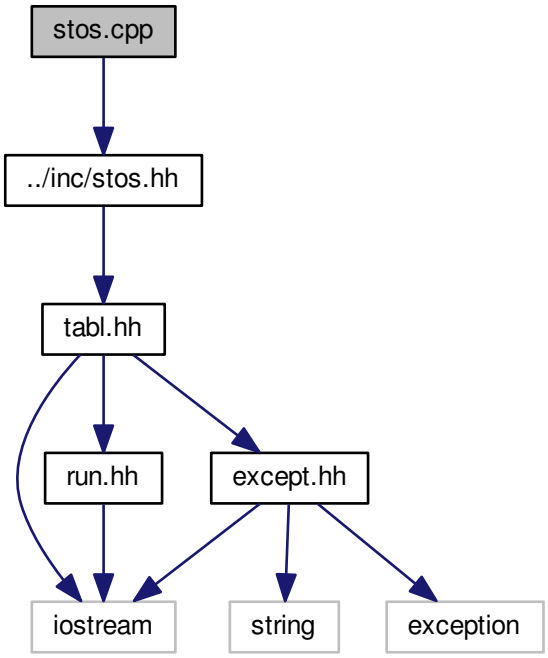
- class Stoper

  Klasa stoper implementująca interfejs IStoper.

6.13 Dokumentacja pliku stos.cpp

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

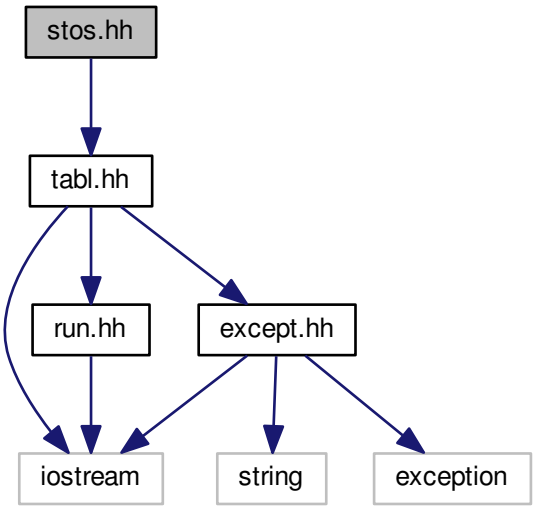
Wykres zależności załączania dla stos.cpp:



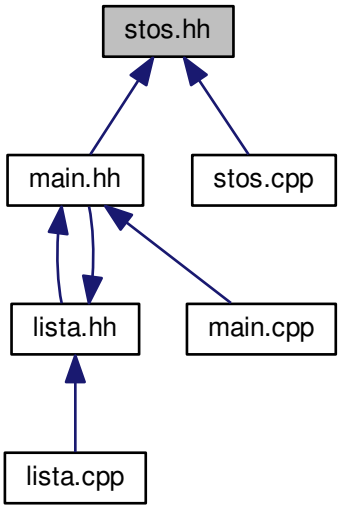
6.14 Dokumentacja pliku stos.hh

```
#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.

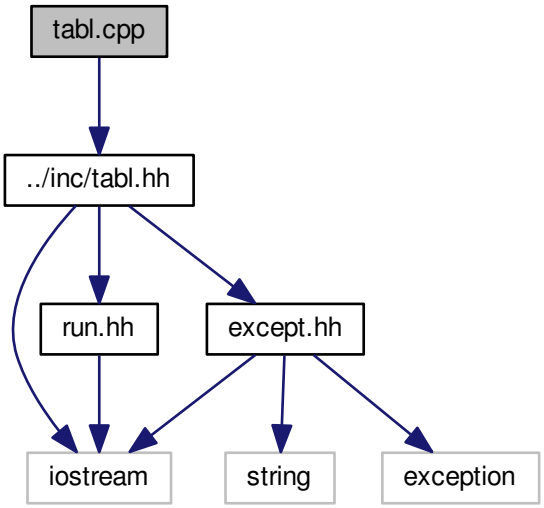
- class Stos< T >

Klasa Stos.

6.15 Dokumentacja pliku tabl.cpp

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

Wykres zależności załączania dla tabl.cpp:

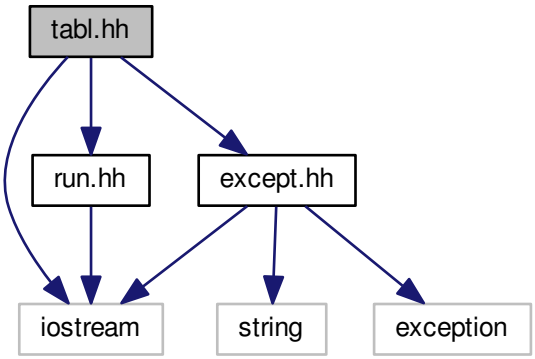


6.16 Dokumentacja pliku tabl.hh

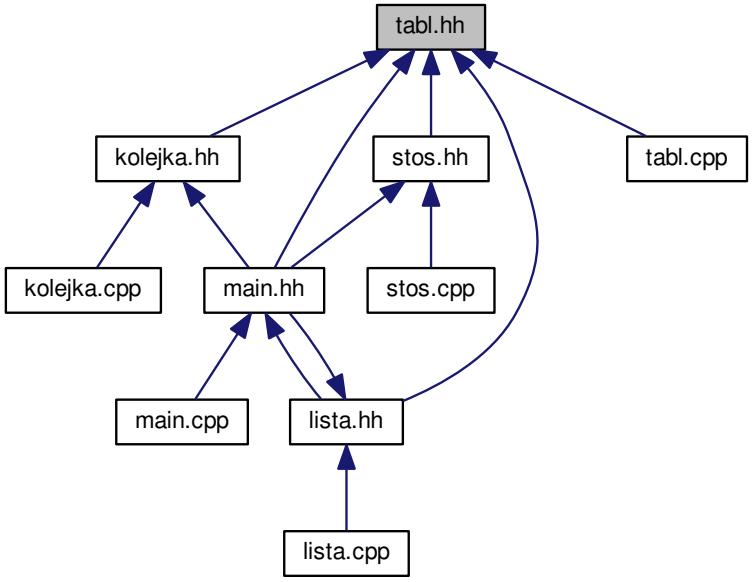
Definicja interfejsu Itabn, klasy tabn oraz klasy tabn_test.

```
#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 Itabn< 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 Itabn, 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