pamsi

0.5

Wygenerowano przez Doxygen 1.8.11

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

```
//Poniższy test wymaga, aby w folderze projektu znajdował się słownik o nazwie zadanej w metodzie virtual
       bool lista_test::prepare(int) . Należy dokonać edycji w/w metody w celu zmian. Trawją prace nad rozwiązaniem
       problemu.
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;
```

1.3.2 Test obsługi wyjątków

W poniższym teście powinien wystąpić wyjątek, związany z próbą dodania elementu o indeksie 10, gdy tablica dynamicznie rozszerzalna ma 3 elementy (czyli gdy maksymalny indeks to 2).

```
Itabn<int> * tablica = new tabn<int>;
try {
   tablica->add(1,0);
   tablica->add(2,1);
   tablica->add(6,1);
   tablica->add(10,10);
}
catch (ContinueException &cex) {
   std::cout << "Exception: " << cex.getError() << std::endl;
   delete tablica;
   exit(-1);
}
catch (CriticalException & crit_ex) {
   std::cout << "Critical: " << crit_ex.getError() << std::endl;
   delete tablica;
   exit(-2);
}
catch (...) {
   std::cerr << "Unexpected exception!" << std::endl;
   delete tablica;
   exit(-3);
}
delete tablica;
return 0;
```

1.3.3 Obsługa stosu

```
//Wykorzystanie stosu
IStos<int> * stos = new Stos<int>;
try{
   stos->push(4);
   stos->push(3);
   cout << "TOP: " << stos->pop() << endl; //Powinno być 3
   cout << "TOP: " << stos->get() << endl; //Powinno być 4
   stos->pop();
   if (stos->isEmpty()) cout << "Stos pusty!" << endl; //wykona się
   cout << "----------------" << endl;
   stos->pop(); //Wyrzuci wyjątek
}
catch (ContinueException &cex) {
   std::cout << "Exception: " << cex.getError() << std::endl;
   delete stos;
   exit(-1);
}
catch (CriticalException & crit_ex) {
   std::cout << "Critical: " << crit_ex.getError() << std::endl;
   delete stos;
   exit(-2);
}
catch (...) {
   std::cerr << "Unexpected exception!" << std::endl;
   delete stos;
   exit(-3);
}
delete stos;
return 0;
```

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 szablonu klasy Asoc< T, T2 >

`#include <asoc.hh>`

Diagram dziedziczenia dla Asoc< T, T2 >

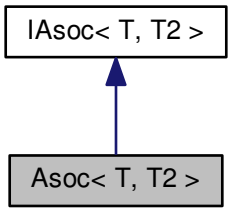
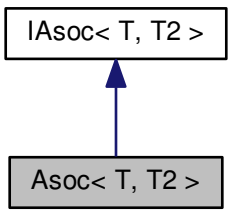


Diagram współpracy dla Asoc< T, T2 >:



Metody publiczne

- Asoc (int nOBuckets)
- ~Asoc ()
- virtual void add (T key, T2 val)
- virtual Itabn< T2 > * find (T position)
- virtual T2 findOne (T position)

5.1.1 Opis szczegółowy

template<class T, class T2>
class Asoc< T, T2 >

Definicja w linii 35 pliku asoc.hh.

5.1.2 Dokumentacja konstruktora i destruktora

5.1.2.1 template<class T, class T2> Asoc< T, T2 >::Asoc (int *nOBuckets*) [inline]

Definicja w linii 40 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:

5.1.2.2 template<class T, class T2> Asoc< T, T2 >::~Asoc ( ) [inline]

Definicja w linii 48 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:

5.1.3 Dokumentacja funkcji składowych

5.1.3.1 template<class T, class T2> virtual void Asoc< T, T2 >::add (T *key*, T2 *val*) [inline], [virtual]

Implementuje IAsoc< T, T2 >.

Definicja w linii 65 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:

5.1.3.2 template<class T, class T2> virtual Itabn<T2>* Asoc< T, T2 >::find (T *position*) [inline], [virtual]

Implementuje IAsoc< T, T2 >.

Definicja w linii 70 pliku asoc.hh.

5.1.3.3 template<class T, class T2> virtual T2 Asoc< T, T2 >::findOne (T *position*) [inline], [virtual]

Implementuje IAsoc< T, T2 >.

Definicja w linii 74 pliku asoc.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- asoc.hh

5.2 Dokumentacja klasy asoc_test

`#include <asoc.hh>`

Diagram dziedziczenia dla asoc_test

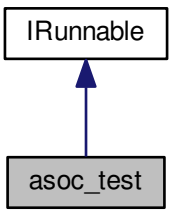
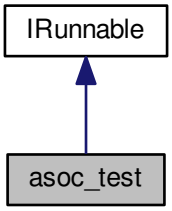


Diagram współpracy dla asoc_test:



Metody publiczne

- asoc_test ()
- asoc_test (int sizeOfTest)
- ~asoc_test ()
- virtual bool prepare (int sOT)

  Przygotowanie badań
- virtual bool run ()

  Przeprowadzanie badań

5.2.1 Opis szczegółowy

Definicja w linii 83 pliku asoc.hh.

5.2.2 Dokumentacja konstruktora i destruktora

5.2.2.1 asoc_test::asoc_test ( ) [inline]

Definicja w linii 92 pliku asoc.hh.

5.2.2.2 asoc_test::asoc_test (int *sizeOfTest*) [inline]

Definicja w linii 95 pliku asoc.hh.

5.2.2.3 asoc_test::~asoc_test ( ) [inline]

Definicja w linii 100 pliku asoc.hh.

5.2.3 Dokumentacja funkcji składowych

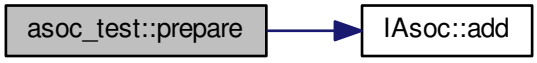
5.2.3.1 virtual bool asoc_test::prepare (int) [inline], [virtual]

Przygotowanie badań

Implementuje IRunnable.

Definicja w linii 106 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:



5.2.3.2 virtual bool asoc_test::run (void) [inline], [virtual]

Przeprowadzanie badań

Implementuje IRunnable.

Definicja w linii 136 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:

Dokumentacja dla tej klasy została wygenerowana z pliku:

- asoc.hh

5.3 Dokumentacja szablonu klasy Bucket< T, T2 >

`#include <hash.hh>`

Diagram dziedziczenia dla Bucket< T, T2 >

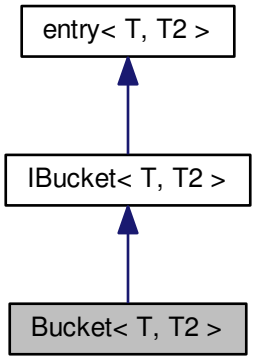
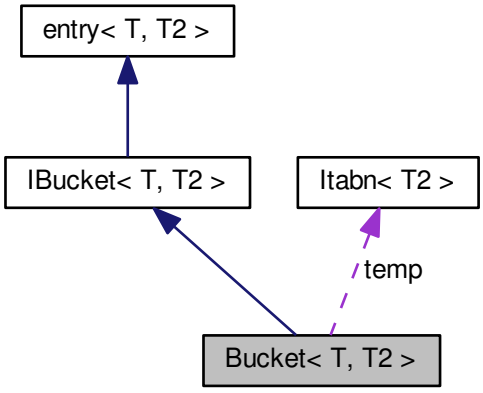


Diagram współpracy dla Bucket< T, T2 >:



Metody publiczne

- Bucket ()
- Bucket (int ID)
- ~Bucket ()
- virtual int getID (void)

- virtual void printAllElements ()
- virtual void printFoundElements (void)
- virtual void add (entry< T, T2 > Ent)
- virtual T2 remove (T position)
- virtual T2 lookup (T position)
- virtual Itabn< T2 > * lookupWhole (T position)

Atrybuty publiczne

- Itabn< T2 > * temp

5.3.1 Opis szczegółowy

template<class T, class T2>
class Bucket< T, T2 >

Definicja w linii 111 pliku hash.hh.

5.3.2 Dokumentacja konstruktora i destruktora

5.3.2.1 template<class T, class T2> Bucket< T, T2 >::Bucket ( ) [inline]

Definicja w linii 120 pliku hash.hh.

5.3.2.2 template<class T, class T2> Bucket< T, T2 >::Bucket (int *ID*) [inline]

Definicja w linii 124 pliku hash.hh.

5.3.2.3 template<class T, class T2> Bucket< T, T2 >::~Bucket ( ) [inline]

Definicja w linii 128 pliku hash.hh.

5.3.3 Dokumentacja funkcji składowych

5.3.3.1 template<class T, class T2> virtual void Bucket< T, T2 >::add (entry< T, T2 > *Ent*) [inline], [virtual]

Implementuje IBucket< T, T2 >.

Definicja w linii 145 pliku hash.hh.

Oto graf wywołań dla tej funkcji:

5.3.3.2 template<class T, class T2> virtual int Bucket< T, T2 >::getID (void) [inline], [virtual]

Implementuje IBucket< T, T2 >.

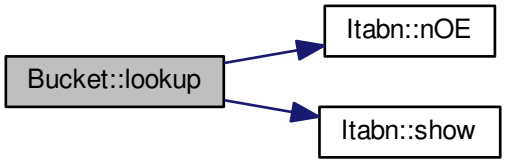
Definicja w linii 133 pliku hash.hh.

5.3.3.3 template<class T, class T2> virtual T2 Bucket< T, T2 >::lookup (T *position*) [inline], [virtual]

Implementuje IBucket< T, T2 >.

Definicja w linii 172 pliku hash.hh.

Oto graf wywołań dla tej funkcji:

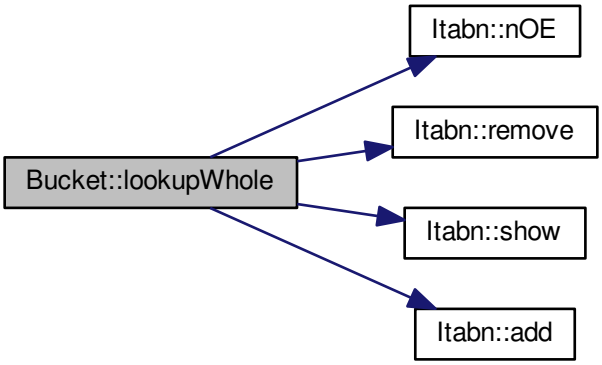


5.3.3.4 template<class T, class T2> virtual Itabn<T2>* Bucket< T, T2 >::lookupWhole (T *position*) [inline], [virtual]

Implementuje IBucket< T, T2 >.

Definicja w linii 186 pliku hash.hh.

Oto graf wywołań dla tej funkcji:

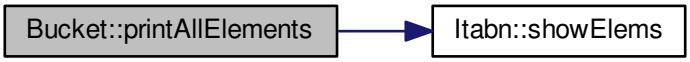


5.3.3.5 template<class T, class T2> virtual void Bucket< T, T2 >::printAllElements ( ) [inline], [virtual]

Implementuje IBucket< T, T2 >.

Definicja w linii 137 pliku hash.hh.

Oto graf wywołań dla tej funkcji:



5.3.3.6 template<class T, class T2> virtual void Bucket< T, T2 >::printFoundElements (void) [inline], [virtual]

Implementuje IBucket< T, T2 >.

Definicja w linii 141 pliku hash.hh.

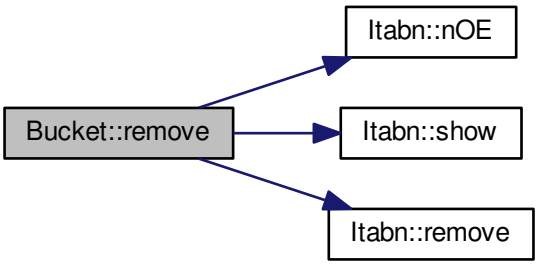
Oto graf wywołań dla tej funkcji:

5.3.3.7 template<class T, class T2> virtual T2 Bucket< T, T2 >::remove (T *position*) [inline], [virtual]

Implementuje IBucket< T, T2 >.

Definicja w linii 154 pliku hash.hh.

Oto graf wywołań dla tej funkcji:



5.3.4 Dokumentacja atrybutów składowych

5.3.4.1 template<class T, class T2> Itabn<T2>* Bucket< T, T2 >::temp

Definicja w linii 118 pliku hash.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- hash.hh

5.4 Dokumentacja klasy ContinueException

Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać.

`#include <except.hh>`

Diagram dziedziczenia dla ContinueException

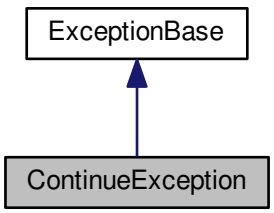
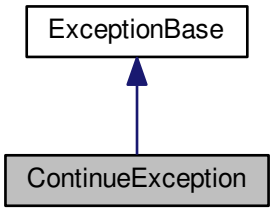


Diagram współpracy dla ContinueException:



Metody publiczne

- ContinueException ()
- ContinueException (std::string description)
- virtual void Throw ()

Dodatkowe Dziedziczone Składowe

5.4.1 Opis szczegółowy

Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać.

Definicja w linii 56 pliku except.hh.

5.4.2 Dokumentacja konstruktora i destruktora

5.4.2.1 ContinueException::ContinueException ( ) [inline]

Definicja w linii 59 pliku except.hh.

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

Definicja w linii 62 pliku except.hh.

5.4.3 Dokumentacja funkcji składowych

5.4.3.1 virtual void ContinueException::Throw ( ) [inline], [virtual]

Reimplementowana z ExceptionBase.

Definicja w linii 65 pliku except.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- except.hh

5.5 Dokumentacja klasy CriticalException

Wyjątek krytyczny, wymagający zamknięcia programu.

`#include <except.hh>`

Diagram dziedziczenia dla CriticalException

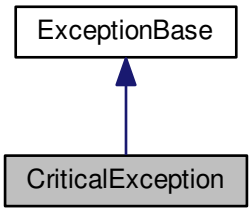
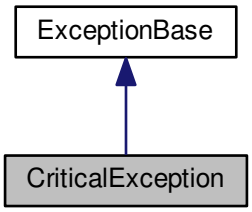


Diagram współpracy dla CriticalException:



Metody publiczne

- CriticalException ()
- CriticalException (std::string description)
- virtual void Throw ()

Dodatkowe Dziedziczone Składowe

5.5.1 Opis szczegółowy

Wyjątek krytyczny, wymagający zamknięcia programu.

Definicja w linii 38 pliku except.hh.

5.5.2 Dokumentacja konstruktora i destruktora

5.5.2.1 CriticalException::CriticalException ( ) [inline]

Definicja w linii 41 pliku except.hh.

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

Definicja w linii 44 pliku except.hh.

5.5.3 Dokumentacja funkcji składowych

5.5.3.1 virtual void CriticalException::Throw ( ) [inline], [virtual]

Reimplementowana z ExceptionBase.

Definicja w linii 47 pliku except.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- except.hh

5.6 Dokumentacja klasy edgeInfo

Klasa definiujaca typ, w ktorym zapisywana jest informacja o krawedzi grafu.

`#include <graph2.hh>`

Metody publiczne

- edgeInfo (int edg, int w=0)
- edgeInfo ()
- int conn (void)

  Zwraca informację o wierzchołku, z którym łączy ta krawędź
- int w (void)

  Zwraca informację o wadze krawędzi.
- edgeInfo & operator= (const edgeInfo &read)

Atrybuty publiczne

- int edgeTo
- int weight

Przyjaciele

- std::ostream & operator<< (std::ostream &output, const edgeInfo to)
- bool operator< (edgeInfo one, edgeInfo two)
- bool operator> (edgeInfo one, edgeInfo two)
- bool operator<= (edgeInfo one, edgeInfo two)
- bool operator>= (edgeInfo one, edgeInfo two)
- bool operator== (edgeInfo one, edgeInfo two)

5.6.1 Opis szczegółowy

Klasa definiujaca typ, w ktorym zapisywana jest informacja o krawedzi grafu.

Definicja w linii 13 pliku graph2.hh.

5.6.2 Dokumentacja konstruktora i destruktora

5.6.2.1 edgeInfo::edgeInfo (int *edg,* int *w* = 0) [inline]

Definicja w linii 21 pliku graph2.hh.

5.6.2.2 edgeInfo::edgeInfo ( ) [inline]

Definicja w linii 27 pliku graph2.hh.

5.6.3 Dokumentacja funkcji składowych

5.6.3.1 int edgeInfo::conn (void) [inline]

Zwraca informację o wierzchołku, z którym łączy ta krawędź

Definicja w linii 36 pliku graph2.hh.

5.6.3.2 edgeInfo& edgeInfo::operator= (const edgeInfo & *read*) [inline]

Definicja w linii 52 pliku graph2.hh.

5.6.3.3 int edgeInfo::w (void) [inline]

Zwraca informację o wadze krawędzi.

Definicja w linii 43 pliku graph2.hh.

5.6.4 Dokumentacja przyjaciół i funkcji związanych

5.6.4.1 bool operator< (edgeInfo *one,* edgeInfo *two*) [friend]

Definicja w linii 58 pliku graph2.hh.

5.6.4.2 std::ostream& operator<< (std::ostream & *output,* const edgeInfo *to*) [friend]

Definicja w linii 47 pliku graph2.hh.

5.6.4.3 bool operator<= (edgeInfo *one,* edgeInfo *two*) [friend]

Definicja w linii 68 pliku graph2.hh.

5.6.4.4 bool operator== (edgeInfo *one,* edgeInfo *two*) [friend]

Definicja w linii 78 pliku graph2.hh.

5.6.4.5 bool operator> (edgeInfo *one,* edgeInfo *two*) [friend]

Definicja w linii 63 pliku graph2.hh.

5.6.4.6 bool operator>= (edgeInfo *one,* edgeInfo *two*) [friend]

Definicja w linii 73 pliku graph2.hh.

5.6.5 Dokumentacja atrybutów składowych

5.6.5.1 int edgeInfo::edgeTo

Definicja w linii 17 pliku graph2.hh.

5.6.5.2 int edgeInfo::weight

Definicja w linii 18 pliku graph2.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

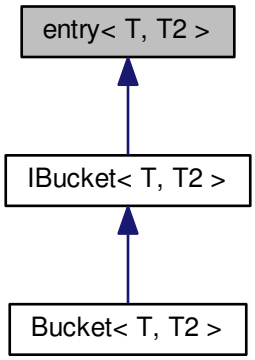
- graph2.hh

5.7 Dokumentacja szablonu klasy entry< T, T2 >

Klasa definująca obiekt typu wpis.

`#include <hash.hh>`

Diagram dziedziczenia dla entry< T, T2 >



Metody publiczne

- entry ()
- entry (T entryKey, T2 entryData)
- T2 getVal (void)
- T getKey (void)
- entry< T, T2 > & operator= (const entry< T, T2 > &read)

Przyjaciele

- bool operator< (entry< T, T2 > one, entry< T, T2 > two)
- bool operator> (entry< T, T2 > one, entry< T, T2 > two)
- bool operator<= (entry< T, T2 > one, entry< T, T2 > two)
- bool operator>= (entry< T, T2 > one, entry< T, T2 > two)
- bool operator== (entry< T, T2 > one, entry< T, T2 > two)
- std::ostream & operator<< (std::ostream &output, const entry< T, T2 > &to)
- std::istream & operator>> (std::istream &input, const entry< T, T2 > &to)

5.7.1 Opis szczegółowy

template<class T, class T2>
class entry< T, T2 >

Klasa definująca obiekt typu wpis.

Definicja w linii 13 pliku hash.hh.

5.7.2 Dokumentacja konstruktora i destruktora

5.7.2.1 template<class T, class T2> entry< T, T2 >::entry ( ) [inline]

Definicja w linii 19 pliku hash.hh.

5.7.2.2 template<class T, class T2> entry< T, T2 >::entry (T *entryKey,* T2 *entryData*) [inline]

Definicja w linii 22 pliku hash.hh.

5.7.3 Dokumentacja funkcji składowych

5.7.3.1 template<class T, class T2> T entry< T, T2 >::getKey (void) [inline]

Definicja w linii 34 pliku hash.hh.

5.7.3.2 template<class T, class T2> T2 entry< T, T2 >::getVal (void) [inline]

Definicja w linii 30 pliku hash.hh.

5.7.3.3 template<class T, class T2> entry<T,T2>& entry< T, T2 >::operator= (const entry< T, T2 > & *read*) [inline]

Definicja w linii 38 pliku hash.hh.

5.7.4 Dokumentacja przyjaciół i funkcji związanych

5.7.4.1 template<class T, class T2> bool operator< (entry< T, T2 > *one,* entry< T, T2 > *two*) [friend]

Definicja w linii 44 pliku hash.hh.

5.7.4.2 template<class T, class T2> std::ostream& operator<< (std::ostream & *output,* const entry< T, T2 > & *to*) [friend]

Definicja w linii 69 pliku hash.hh.

5.7.4.3 template<class T, class T2> bool operator<= (entry< T, T2 > *one,* entry< T, T2 > *two*) [friend]

Definicja w linii 54 pliku hash.hh.

5.7.4.4 template<class T, class T2> bool operator== (entry< T, T2 > *one,* entry< T, T2 > *two*) [friend]

Definicja w linii 64 pliku hash.hh.

5.7.4.5 template<class T, class T2> bool operator> (entry< T, T2 > *one,* entry< T, T2 > *two*) [friend]

Definicja w linii 49 pliku hash.hh.

5.7.4.6 template<class T, class T2> bool operator>= (entry< T, T2 > *one,* entry< T, T2 > *two*) [friend]

Definicja w linii 59 pliku hash.hh.

5.7.4.7 template<class T, class T2> std::istream& operator>> (std::istream & *input,* const entry< T, T2 > & *to*) [friend]

Definicja w linii 74 pliku hash.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

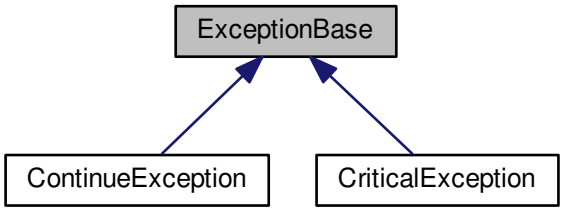
- hash.hh

5.8 Dokumentacja klasy ExceptionBase

Ogólny wyjątek.

`#include <except.hh>`

Diagram dziedziczenia dla ExceptionBase



Metody publiczne

- ExceptionBase ()
- ExceptionBase (std::string description)
- virtual void Throw ()

Atrybuty publiczne

- std::string cause

Przyjaciele

- std::ostream & operator<< (std::ostream &output, const ExceptionBase &to)

5.8.1 Opis szczegółowy

Ogólny wyjątek.

Definicja w linii 15 pliku except.hh.

5.8.2 Dokumentacja konstruktora i destruktora

5.8.2.1 ExceptionBase::ExceptionBase ( ) [inline]

Definicja w linii 19 pliku except.hh.

5.8.2.2 ExceptionBase::ExceptionBase (std::string *description*) [inline]

Definicja w linii 22 pliku except.hh.

5.8.3 Dokumentacja funkcji składowych

5.8.3.1 virtual void ExceptionBase::Throw ( ) [inline], [virtual]

Reimplementowana w ContinueException i CriticalException.

Definicja w linii 25 pliku except.hh.

5.8.4 Dokumentacja przyjaciół i funkcji związanych

5.8.4.1 std::ostream& operator<< (std::ostream & *output,* const ExceptionBase & *to*) [friend]

Definicja w linii 29 pliku except.hh.

5.8.5 Dokumentacja atrybutów składowych

5.8.5.1 std::string ExceptionBase::cause

Definicja w linii 17 pliku except.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- except.hh

5.9 Dokumentacja klasy Graph

Klasa implementująca inerfejs grafu.

`#include <graph.hh>`

Diagram dziedziczenia dla Graph

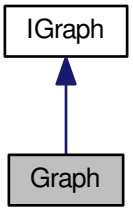
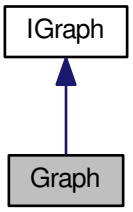


Diagram współpracy dla Graph:



Metody publiczne

- Graph ()

  Konstruktor klasy Graph.
- virtual ∼Graph ()

  Destruktor klasy Graph.
- virtual bool isEmpty (void)
- virtual void insertVertex ()

  Dodaje element do grafu.
- virtual void insertEdge (int index1, int index2)

  Dodaje powiązanie między dwoma elementami.
- virtual bool areAdjacent (int index1, int index2)

  Sprawdza, czy dwa elementy mają krawędź między sobą
- virtual Itabn< int > ∗ getNeightbours (int index)

Zwraca wskaźnik na tablicę elementów będących zbiorem sąsiadów wierzchołka index.

- virtual int numberOfEdges (void)

  Zwraca ilość krawędzi drzewa.
- virtual Itabn< int > * DFS (void)

  Przeszukuje graf wgłąb.
- virtual Itabn< int > * BFS (void)

  Przeszukuje graf wszerz.

5.9.1 Opis szczegółowy

Klasa implementująca inerfejs grafu.

Definicja w linii 36 pliku graph.hh.

5.9.2 Dokumentacja konstruktora i destruktora

5.9.2.1 Graph::Graph ( ) [inline]

Konstruktor klasy Graph.

Definicja w linii 51 pliku graph.hh.

5.9.2.2 virtual Graph::~Graph ( ) [inline], [virtual]

Destruktor klasy Graph.

Definicja w linii 61 pliku graph.hh.

5.9.3 Dokumentacja funkcji składowych

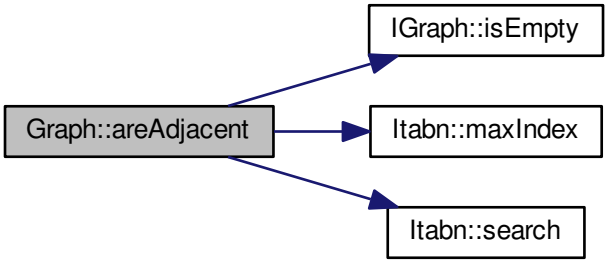
5.9.3.1 virtual bool Graph::areAdjacent (int *index1*, int *index2*) [inline], [virtual]

Sprawdza, czy dwa elementy mają krawędź między sobą

Implementuje IGraph.

Definicja w linii 103 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



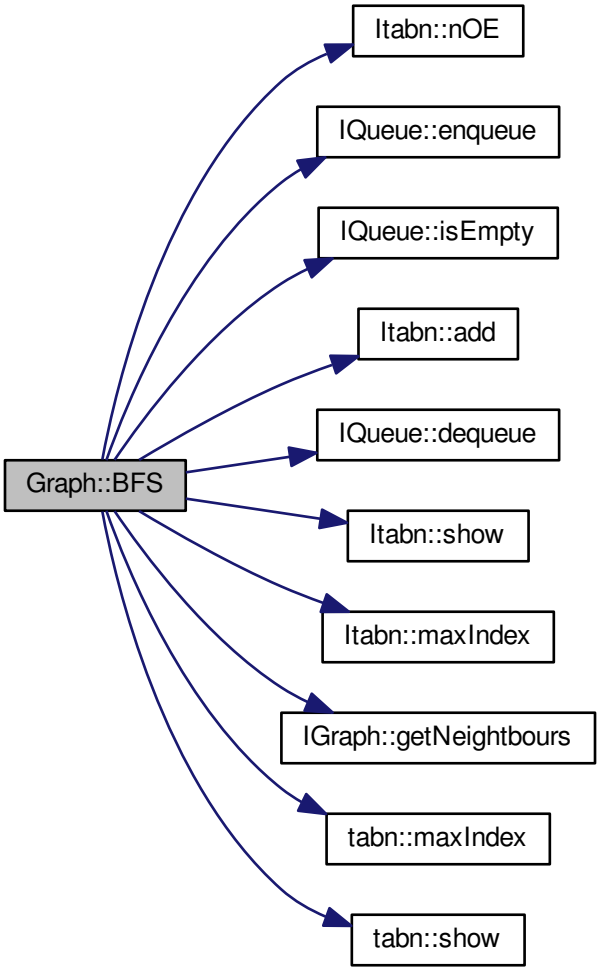
5.9.3.2 virtual Itabn<int>* Graph::BFS (void) [inline], [virtual]

Przeszukuje graf wszerz.

Implementuje IGraph.

Definicja w linii 174 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



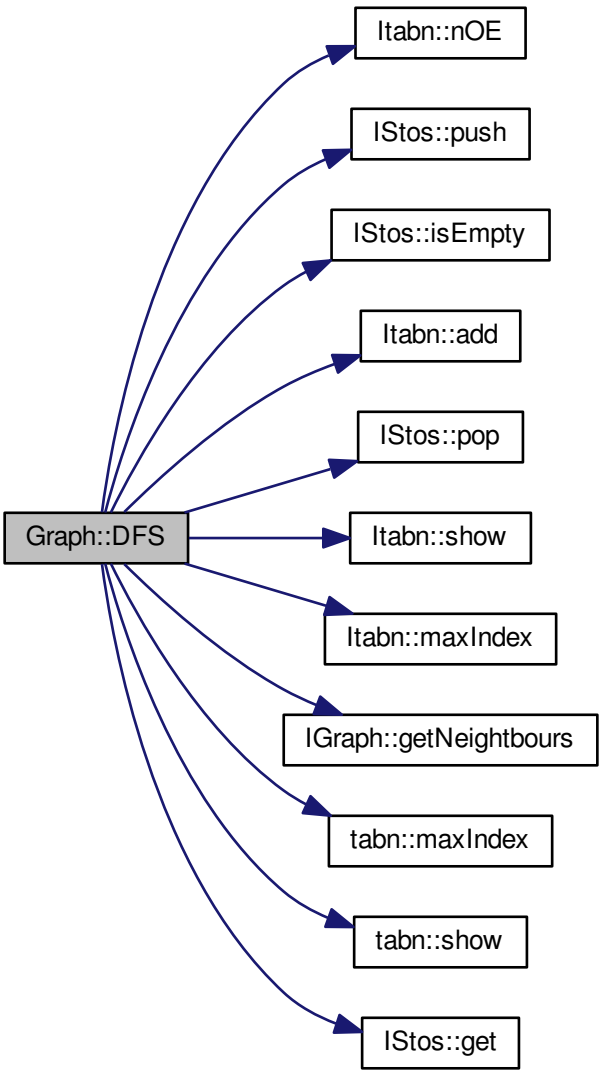
5.9.3.3 virtual Itabn<int>* Graph::DFS (void) [inline], [virtual]

Przeszukuje graf wgłąb.

Implementuje IGraph.

Definicja w linii 134 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.9.3.4 virtual Itabn<int>* Graph::getNeightbours (int *index*) [inline], [virtual]

Zwraca wskaźnik na tablicę elementów będących zbiorem sąsiadów wierzchołka index.

Implementuje IGraph.

Definicja w linii 120 pliku graph.hh.

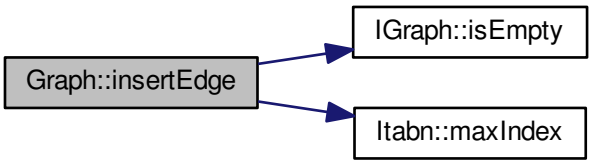
5.9.3.5 virtual void Graph::insertEdge (int *index1*, int *index2*) [inline], [virtual]

Dodaje powiązanie między dwoma elementami.

Implementuje IGraph.

Definicja w linii 87 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



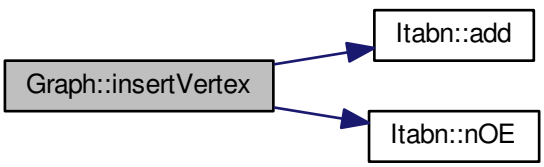
5.9.3.6 virtual void Graph::insertVertex (void) [inline], [virtual]

Dodaje element do grafu.

Implementuje IGraph.

Definicja w linii 74 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.9.3.7 virtual bool Graph::isEmpty (void) [inline], [virtual]

Implementuje IGraph.

Definicja w linii 66 pliku graph.hh.

Oto graf wywołań dla tej funkcji:

5.9.3.8 virtual int Graph::numberOfEdges (void) [inline], [virtual]

Zwraca ilość krawędzi drzewa.

Implementuje IGraph.

Definicja w linii 127 pliku graph.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- graph.hh

5.10 Dokumentacja klasy Graph2

graf2 z uwzględnieniem wag

`#include <graph2.hh>`

Diagram dziedziczenia dla Graph2

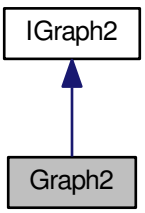
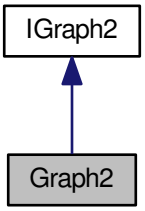


Diagram współpracy dla Graph2:



Metody publiczne

- Graph2 ()

  Konstruktor klasy Graph.
- virtual ∼Graph2 ()

  Destruktor klasy Graph.
- virtual bool isEmpty (void)

  Sprawdza czy graf jest pusty.
- virtual void insertVertex ()

  Dodaje element do grafu.
- virtual void insertEdge (int index1, int index2, int wei)

  Dodaje powiązanie między dwoma elementami.
- virtual bool areAdjacent (int index1, int index2)

  Sprawdza, czy dwa elementy mają krawędź między sobą
- virtual ltabn< edgeInfo > ∗ getNeightbours (int index)

  Zwraca wskaźnik na tablicę elementów będących zbiorem sąsiadów wierzchołka index.
- virtual int numberOfEdges (void)

  Zwraca ilość krawędzi drzewa.
- virtual ltabn< int > ∗ branchAndBound (int start, int end)

  Znajduje najkrótszą możliwą ścieżkę miedzy dwoma elementami.

5.10.1 Opis szczegółowy

graf2 z uwzględnieniem wag

Definicja w linii 110 pliku graph2.hh.

5.10.2 Dokumentacja konstruktora i destruktora

5.10.2.1 Graph2::Graph2 ( ) [inline]

Konstruktor klasy Graph.

Definicja w linii 121 pliku graph2.hh.

5.10.2.2 virtual Graph2::~Graph2 ( ) [inline], [virtual]

Destruktor klasy Graph.

Definicja w linii 131 pliku graph2.hh.

5.10.3 Dokumentacja funkcji składowych

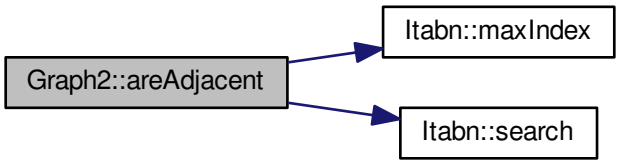
5.10.3.1 virtual bool Graph2::areAdjacent (int *index1*, int *index2*) [inline], [virtual]

Sprawdza, czy dwa elementy mają krawędź między sobą

Implementuje IGraph2.

Definicja w linii 177 pliku graph2.hh.

Oto graf wywołań dla tej funkcji:



5.10.3.2 virtual Itabn<int>* Graph2::branchAndBound (int *start*, int *end*) [inline], [virtual]

Znajduje najkrótszą możliwą ścieżkę miedzy dwoma elementami.

Implementuje IGraph2.

Definicja w linii 209 pliku graph2.hh.

5.10.3.3 virtual Itabn<edgeInfo>* Graph2::getNeightbours (int *index*) [inline], [virtual]

Zwraca wskaźnik na tablicę elementów będących zbiorem sąsiadów wierzchołka index.

Implementuje IGraph2.

Definicja w linii 195 pliku graph2.hh.

5.10.3.4 virtual void Graph2::insertEdge (int *index1,* int *index2,* int *wei*) [inline], [virtual]

Dodaje powiązanie między dwoma elementami.

Definicja w linii 160 pliku graph2.hh.

Oto graf wywołań dla tej funkcji:

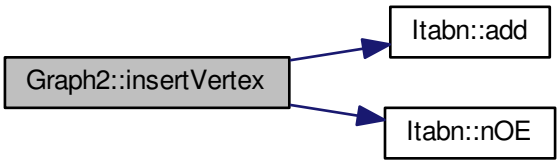
5.10.3.5 virtual void Graph2::insertVertex (void) [inline], [virtual]

Dodaje element do grafu.

Implementuje IGraph2.

Definicja w linii 147 pliku graph2.hh.

Oto graf wywołań dla tej funkcji:



5.10.3.6 virtual bool Graph2::isEmpty (void) [inline], [virtual]

Sprawdza czy graf jest pusty.

Implementuje IGraph2.

Definicja w linii 139 pliku graph2.hh.

Oto graf wywołań dla tej funkcji:

5.10.3.7 virtual int Graph2::numberOfEdges (void) [inline], [virtual]

Zwraca ilość krawędzi drzewa.

Implementuje IGraph2.

Definicja w linii 202 pliku graph2.hh.

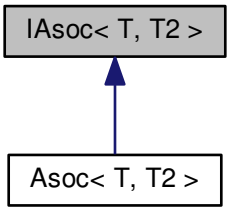
Dokumentacja dla tej klasy została wygenerowana z pliku:

- graph2.hh

5.11 Dokumentacja szablonu klasy IAsoc< T, T2 >

`#include <asoc.hh>`

Diagram dziedziczenia dla IAsoc< T, T2 >



Metody publiczne

- virtual void add (T, T2)=0
- virtual ~IAsoc ()
- virtual Itabn< T2 > * find (T)=0
- virtual T2 findOne (T)=0

Przyjaciele

- std::ostream & operator<< (std::ostream &output, IAsoc *to)

5.11.1 Opis szczegółowy

template<class T, class T2>
class IAsoc< T, T2 >

Definicja w linii 15 pliku asoc.hh.

5.11.2 Dokumentacja konstruktora i destruktora

5.11.2.1 template<class T, class T2> virtual IAsoc< T, T2 >::~IAsoc ( ) [inline], [virtual]

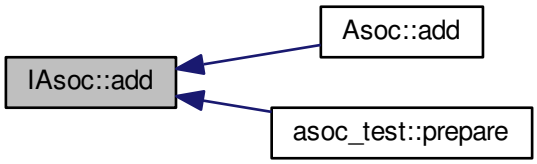
Definicja w linii 24 pliku asoc.hh.

5.11.3 Dokumentacja funkcji składowych

5.11.3.1 template<class T, class T2> virtual void IAsoc< T, T2 >::add (T , T2) [pure virtual]

Implementowany w Asoc< T, T2 >.

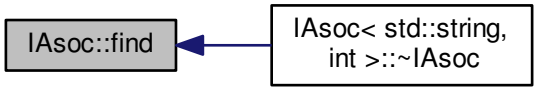
Oto graf wywoływań tej funkcji:



5.11.3.2 template<class T, class T2> virtual Itabn<T2>* IAsoc< T, T2 >::find (T) [pure virtual]

Implementowany w Asoc< T, T2 >.

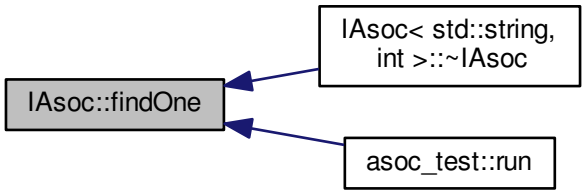
Oto graf wywoływań tej funkcji:



5.11.3.3 template<class T, class T2> virtual T2 IAsoc< T, T2 >::findOne (T) [pure virtual]

Implementowany w Asoc< T, T2 >.

Oto graf wywoływań tej funkcji:



5.11.4 Dokumentacja przyjaciół i funkcji związanych

5.11.4.1 template<class T, class T2> std::ostream& operator<< (std::ostream & *output,* IAsoc< T, T2 > * *to*) [friend]

Definicja w linii 28 pliku asoc.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- asoc.hh

5.12 Dokumentacja szablonu klasy IBucket< T, T2 >

`#include <hash.hh>`

Diagram dziedziczenia dla IBucket< T, T2 >

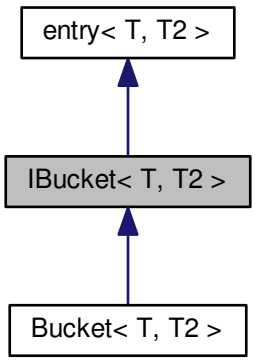
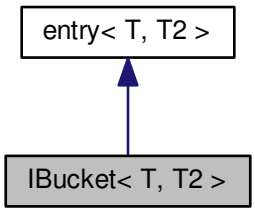


Diagram współpracy dla IBucket< T, T2 >:



Metody publiczne

- virtual void add (entry< T, T2 >)=0
- virtual T2 remove (T)=0
- virtual T2 lookup (T)=0
- virtual Itabn< T2 > * lookupWhole (T)=0
- virtual ~IBucket ()
- virtual void printAllElements ()=0
- virtual int getID (void)=0
- virtual void printFoundElements (void)=0

5.12.1 Opis szczegółowy

template<class T, class T2>
class IBucket< T, T2 >

Definicja w linii 92 pliku hash.hh.

5.12.2 Dokumentacja konstruktora i destruktora

5.12.2.1 template<class T , class T2 > virtual IBucket< T, T2 >::~IBucket ( ) [inline], [virtual]

Definicja w linii 99 pliku hash.hh.

5.12.3 Dokumentacja funkcji składowych

5.12.3.1 template<class T , class T2 > virtual void IBucket< T, T2 >::add (entry< T, T2 >) [pure virtual]

Implementowany w Bucket< T, T2 >.

5.12.3.2 template<class T , class T2 > virtual int IBucket< T, T2 >::getID (void) [pure virtual]

Implementowany w Bucket< T, T2 >.

5.12.3.3 template<class T , class T2 > virtual T2 IBucket< T, T2 >::lookup (T) [pure virtual]

Implementowany w Bucket< T, T2 >.

5.12.3.4 template<class T , class T2 > virtual Itabn<T2>* IBucket< T, T2 >::lookupWhole (T) [pure virtual]

Implementowany w Bucket< T, T2 >.

5.12.3.5 template<class T , class T2 > virtual void IBucket< T, T2 >::printAllElements ( ) [pure virtual]

Implementowany w Bucket< T, T2 >.

5.12.3.6 template<class T , class T2 > virtual void IBucket< T, T2 >::printFoundElements (void) [pure virtual]

Implementowany w Bucket< T, T2 >.

5.12.3.7 template<class T , class T2 > virtual T2 IBucket< T, T2 >::remove (T) [pure virtual]

Implementowany w Bucket< T, T2 >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

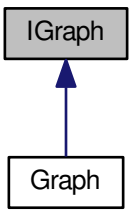
- hash.hh

5.13 Dokumentacja klasy IGraph

Interfejs grafu.

`#include <graph.hh>`

Diagram dziedziczenia dla IGraph



Metody publiczne

- virtual ~IGraph ()
- virtual bool isEmpty (void)=0
- virtual void insertVertex (void)=0
- virtual void insertEdge (int, int)=0
- virtual ltabn< int > * getNeightbours (int)=0
- virtual bool areAdjacent (int, int)=0
- virtual int numberOfEdges (void)=0
- virtual ltabn< int > * DFS (void)=0
- virtual ltabn< int > * BFS (void)=0

5.13.1 Opis szczegółowy

Interfejs grafu.

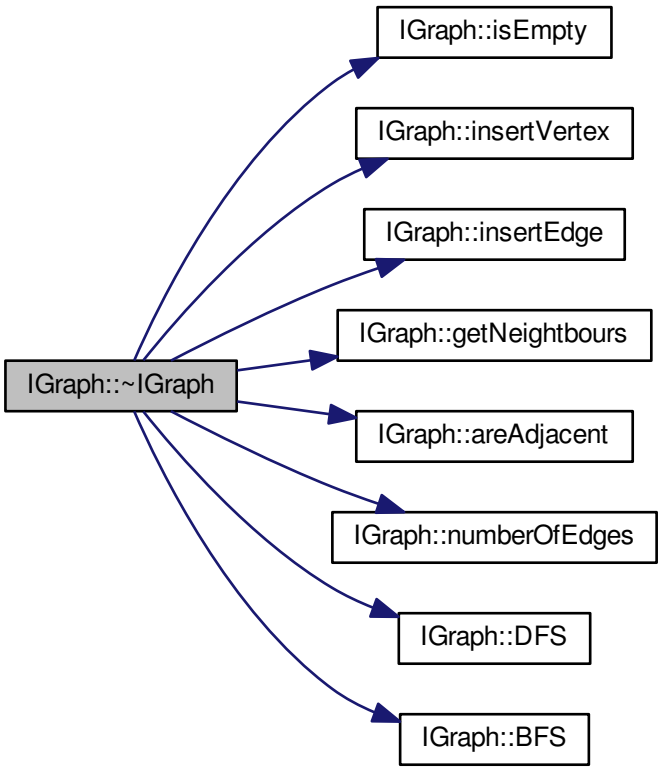
Definicja w linii 13 pliku graph.hh.

5.13.2 Dokumentacja konstruktora i destruktora

5.13.2.1 virtual IGraph::~IGraph ( ) [inline], [virtual]

Definicja w linii 16 pliku graph.hh.

Oto graf wywołań dla tej funkcji:

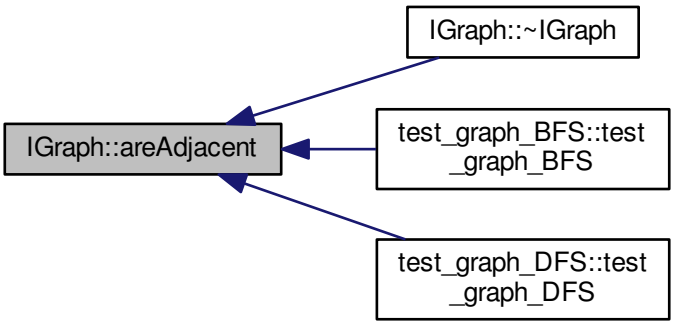


5.13.3 Dokumentacja funkcji składowych

5.13.3.1 virtual bool IGraph::areAdjacent (int , int) [pure virtual]

Implementowany w Graph.

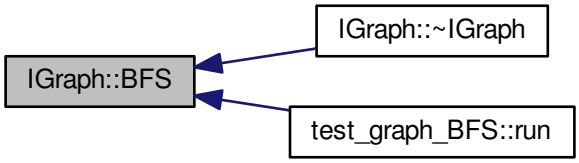
Oto graf wywoływań tej funkcji:



5.13.3.2 virtual Itabn<int>* IGraph::BFS (void) [pure virtual]

Implementowany w Graph.

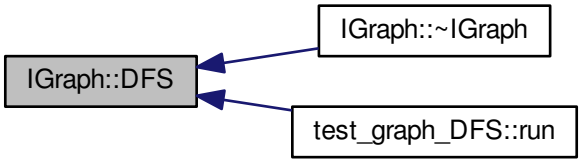
Oto graf wywoływań tej funkcji:



5.13.3.3 virtual Itabn<int>* IGraph::DFS (void) [pure virtual]

Implementowany w Graph.

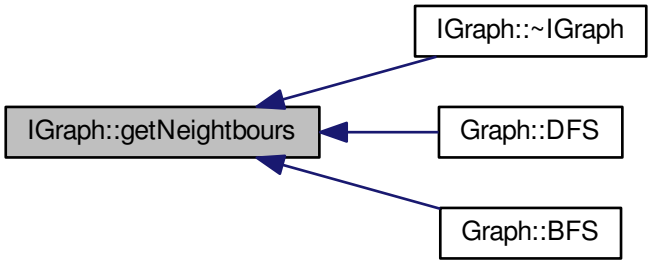
Oto graf wywoływań tej funkcji:



5.13.3.4 virtual Itabn<int>* IGraph::getNeightbours (int) [pure virtual]

Implementowany w Graph.

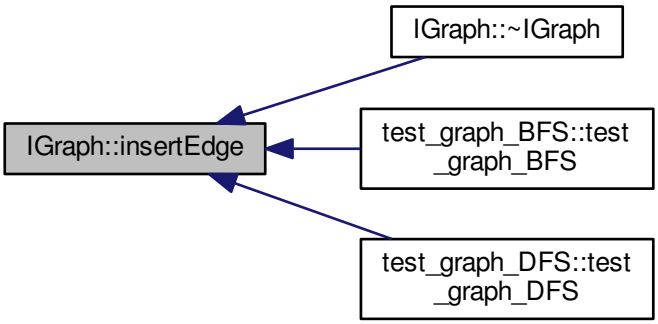
Oto graf wywoływań tej funkcji:



5.13.3.5 virtual void IGraph::insertEdge (int , int) [pure virtual]

Implementowany w Graph.

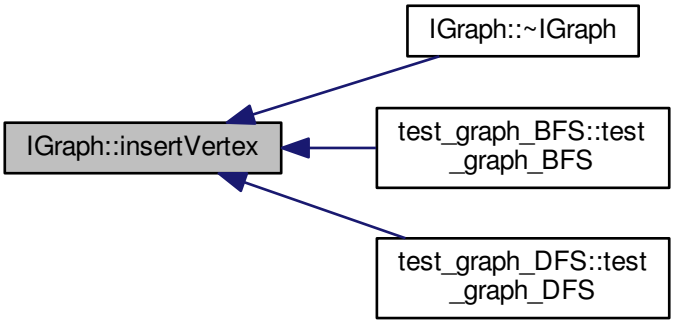
Oto graf wywoływań tej funkcji:



5.13.3.6 virtual void IGraph::insertVertex (void) [pure virtual]

Implementowany w Graph.

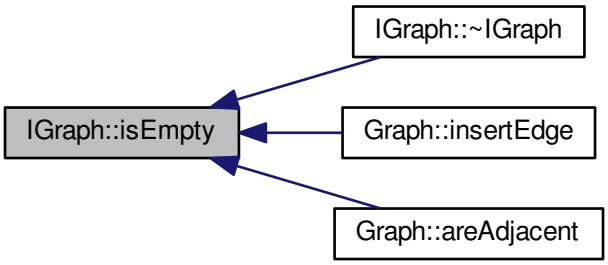
Oto graf wywoływań tej funkcji:



5.13.3.7 virtual bool IGraph::isEmpty (void) [pure virtual]

Implementowany w Graph.

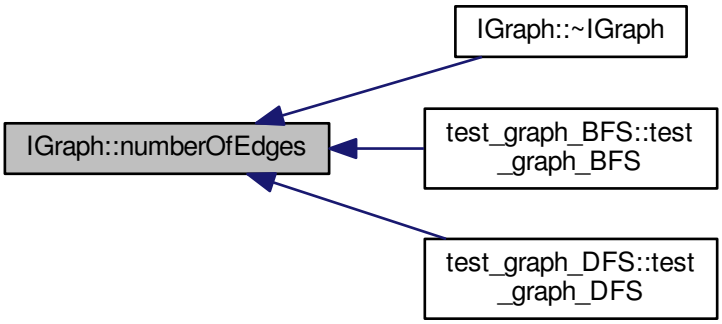
Oto graf wywoływań tej funkcji:



5.13.3.8 virtual int IGraph::numberOfEdges (void) [pure virtual]

Implementowany w Graph.

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

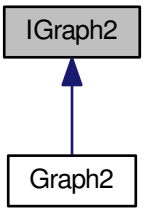
- graph.hh

5.14 Dokumentacja klasy IGraph2

Interfejs grafu2 z uwzględnieniem wag.

`#include <graph2.hh>`

Diagram dziedziczenia dla IGraph2



Metody publiczne

- virtual ∼IGraph2 ()
- virtual bool isEmpty (void)=0
- virtual void insertVertex (void)=0
- virtual void insertEdge (int, int)=0
- virtual ltabn< edgeInfo > ∗ getNeightbours (int)=0
- virtual bool areAdjacent (int, int)=0
- virtual int numberOfEdges (void)=0
- virtual ltabn< int > ∗ branchAndBound (int, int)=0

5.14.1 Opis szczegółowy

Interfejs grafu2 z uwzględnieniem wag.

Definicja w linii 87 pliku graph2.hh.

5.14.2 Dokumentacja konstruktora i destruktora

5.14.2.1 virtual IGraph2::∼IGraph2 ( ) [inline], [virtual]

Definicja w linii 90 pliku graph2.hh.

5.14.3 Dokumentacja funkcji składowych

5.14.3.1 virtual bool IGraph2::areAdjacent (int , int) [pure virtual]

Implementowany w Graph2.

5.14.3.2 virtual Itabn<int>* IGraph2::branchAndBound (int , int) [pure virtual]

Implementowany w Graph2.

5.14.3.3 virtual Itabn<edgeInfo>* IGraph2::getNeightbours (int) [pure virtual]

Implementowany w Graph2.

5.14.3.4 virtual void IGraph2::insertEdge (int , int) [pure virtual]

5.14.3.5 virtual void IGraph2::insertVertex (void) [pure virtual]

Implementowany w Graph2.

5.14.3.6 virtual bool IGraph2::isEmpty (void) [pure virtual]

Implementowany w Graph2.

5.14.3.7 virtual int IGraph2::numberOfEdges (void) [pure virtual]

Implementowany w Graph2.

Dokumentacja dla tej klasy została wygenerowana z pliku:

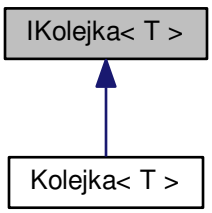
- graph2.hh

5.15 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.

Przyjaciele

- std::ostream & operator<< (std::ostream &output, IKolejka *to)

  Przeciążenie operatora <<.

5.15.1 Opis szczegółowy

template<class T>
class IKolejka< T >

Interfejs klasy Kolejka Definiuje operacje dostępne dla klasy Kolejka.

Definicja w linii 15 pliku kolejka.hh.

5.15.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny interfejsu.

Definicja w linii 47 pliku kolejka.hh.

5.15.3 Dokumentacja funkcji składowych

5.15.3.1 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 >.

Oto graf wywoływań tej funkcji:

5.15.3.2 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 >.

Oto graf wywoływań tej funkcji:

5.15.3.3 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
Sprawdź dokumentację metody Kolejka<T>::get(void).

Implementowany w Kolejka< T >.

5.15.3.4 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 KolejkaT >.

Oto graf wywoływań tej funkcji:

5.15.4 Dokumentacja przyjaciół i funkcji związanych

5.15.4.1 templateclass T > std::ostream& operator<< (std::ostream & *output,* IKolejkaT > * *to*) [friend]

Przeciążenie operatora <<.

Definicja w linii 60 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

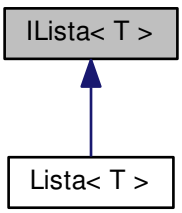
- kolejka.hh

5.16 Dokumentacja szablonu klasy IListaT >

Interfejs listy.

`#include <lista.hh>`

Diagram dziedziczenia dla IListaT >



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 void qs (int, int)=0
- virtual ~ILista ()

  Destruktor wirtualny interfejsu ILista.

5.16.1 Opis szczegółowy

template<class T>
class ILista< T >

Interfejs listy.

Definiuje dostępne operacje na klasie Lista

Definicja w linii 17 pliku lista.hh.

5.16.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny interfejsu ILista.

Definicja w linii 75 pliku lista.hh.

5.16.3 Dokumentacja funkcji składowych

5.16.3.1 template<class T> virtual void IListа< T >::add (T , int) [pure virtual]

Dodaje element do zadanego miejsca listy.

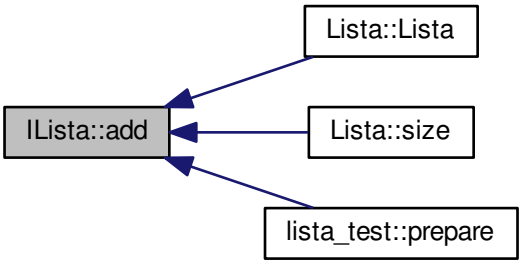
Jeśli następuje próba dodania elementu w miejscu istniejącego, następuje przesunięcie następujących po nim elementów na następne pozycje

Nota

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

Implementowany w Lista< T >.

Oto graf wywoływań tej funkcji:



5.16.3.2 template<class T> virtual void IListа< T >::add (T) [pure virtual]

Dodaje element na koniec listy.

Implementowany w Lista< T >.

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

Zwraca element z zadanego miejsca bez usunięcia.

Zwracane wartości

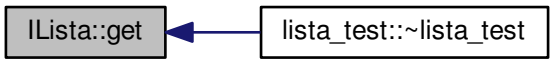
T	element w zadanym miejscu

Ostrzeżenie

Próba podglądu elementu nieistniejącego spowoduje wyrzucenie wyjątku.
Sprawdź dokumentację metody Lista<T>::get(int).

Implementowany w Lista< T >.

Oto graf wywoływań tej funkcji:



5.16.3.4 template<class T> virtual bool IListaT >::isEmpty (void) [pure virtual]

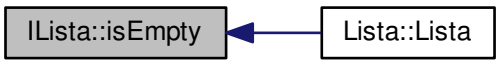
Sprawdza, czy lista jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementowany w Lista< T >.

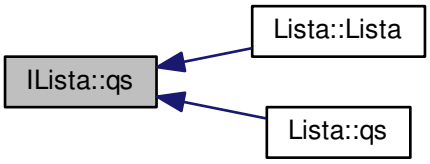
Oto graf wywoływań tej funkcji:



5.16.3.5 template<class T> virtual void IListaT >::qs (int , int) [pure virtual]

Implementowany w Lista< T >.

Oto graf wywoływań tej funkcji:



5.16.3.6 template<class T> virtual T IListа< 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.
Sprawdź dokumentację metody Lista<T>::remove(int).

Implementowany w Lista< T >.

5.16.3.7 template<class T> virtual T IListа< T >::remove (void) [pure virtual]

Usuwa element z końca listy.

Implementowany w Lista< T >.

5.16.3.8 template<class T> virtual int IListа< T >::size (void) [pure virtual]

Zwraca ilość elementów w liście.

Zwracane wartości

int	ilość elementów

Implementowany w Lista< T >.

Oto graf wywoływań tej funkcji:

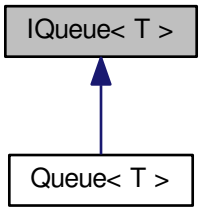
Dokumentacja dla tej klasy została wygenerowana z pliku:

- lista.hh

5.17 Dokumentacja szablonu klasy IQueue< T >

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

Diagram dziedziczenia dla IQueue< 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 ~IQueue ()

  Destruktor wirtualny interfejsu.

Przyjaciele

- std::ostream & operator<< (std::ostream &output, IQueue *to)

  Przeciążenie operatora <<.

5.17.1 Opis szczegółowy

template<class T>
class IQueue< T >

Definicja w linii 227 pliku kolejka.hh.

5.17.2 Dokumentacja konstruktora i destruktora

5.17.2.1 template<class T> virtual IQueue< T >::~IQueue ( ) [inline], [virtual]

Destruktor wirtualny interfejsu.

Definicja w linii 259 pliku kolejka.hh.

5.17.3 Dokumentacja funkcji składowych

5.17.3.1 template<class T> virtual T IQueue< T >::dequeue (void) [pure virtual]

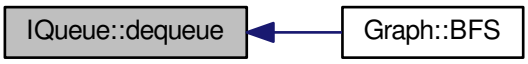
Usuwa i zwraca element z początku kolejki.

Zwracane wartości

T	element z początku kolejki

Implementowany w Queue< T >.

Oto graf wywoływań tej funkcji:



5.17.3.2 template<class T> virtual void IQueue< T >::enqueue (T) [pure virtual]

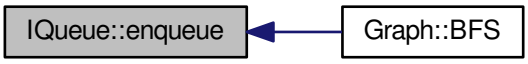
Dodaje element na koniec kolejki.

Parametry

element	- element do umieszczenia w kolejce

Implementowany w Queue< T >.

Oto graf wywoływań tej funkcji:



5.17.3.3 template<class T> virtual T IQueue< T >::get (void) [pure virtual]

Zwraca element z początku kolejki bez usuwania.

Ostrzeżenie

Uwaga! Próba podglądu elementu z pustej kolejki spowoduje wyrzucenie wyjątku
Sprawdź dokumentację metody Kolejka<T>::get(void).

Implementowany w Queue< T >.

5.17.3.4 template<class T> virtual bool IQueue< T >::isEmpty (void) [pure virtual]

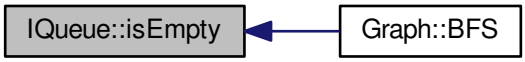
Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementowany w Queue< T >.

Oto graf wywoływań tej funkcji:



5.17.4 Dokumentacja przyjaciół i funkcji związanych

5.17.4.1 template<class T> std::ostream& operator<< (std::ostream & *output,* IQueue< T > * *to*) [friend]

Przeciążenie operatora <<.

Definicja w linii 272 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

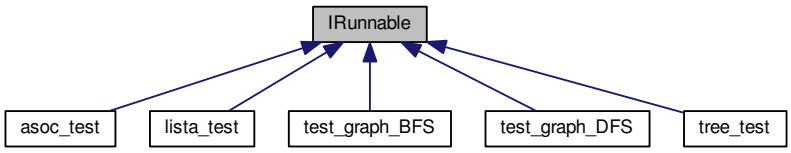
- kolejka.hh

5.18 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.

5.18.1 Opis szczegółowy

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

Definicja w linii 16 pliku run.hh.

5.18.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny IRunnable.

Definicja w linii 31 pliku run.hh.

5.18.3 Dokumentacja funkcji składowych

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

Przygotowanie badań

Implementowany w tree_test, test_graph_DFS, lista_test, test_graph_BFS i asoc_test.

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

Przeprowadzanie badań

Implementowany w tree_test, test_graph_DFS, lista_test, test_graph_BFS i asoc_test.

Dokumentacja dla tej klasy została wygenerowana z pliku:

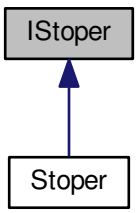
- run.hh

5.19 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 ()

5.19.1 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.

5.19.2 Dokumentacja konstruktora i destruktora

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

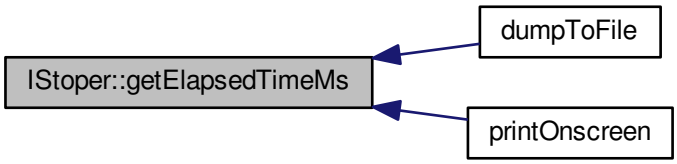
Definicja w linii 25 pliku stoper.hh.

5.19.3 Dokumentacja funkcji składowych

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

Implementowany w Stoper.

Oto graf wywoływań tej funkcji:



5.19.3.2 virtual void IStoper::start (void) [pure virtual]

Implementowany w Stoper.

5.19.3.3 virtual void IStoper::stop (void) [pure virtual]

Implementowany w Stoper.

Dokumentacja dla tej klasy została wygenerowana z pliku:

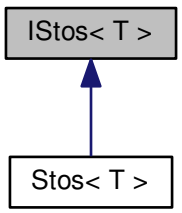
- stoper.hh

5.20 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 pop (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.

Przyjaciele

- std::ostream & operator<< (std::ostream &output, IStos *to)

  Przeciążenie operatora <<.

5.20.1 Opis szczegółowy

template<class T>
class IStos< T >

Interfejs stosu.

Definiuje dostępne operacje na klasie Stos

Definicja w linii 16 pliku stos.hh.

5.20.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny IStos.

Definicja w linii 53 pliku stos.hh.

5.20.3 Dokumentacja funkcji składowych

5.20.3.1 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
Sprawdź dokumentację metody Stos<T>::get(void).

Implementowany w Stos< T >.

Oto graf wywoływań tej funkcji:

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

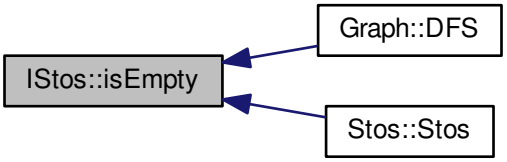
Sprawdza, czy stos jest pusty.

Zwracane wartości

0	gdy niepusty
1	gdy pusty

Implementowany w Stos< T >.

Oto graf wywoływań tej funkcji:



5.20.3.3 template<class T> virtual T IStos< T >::pop (void) [pure virtual]

Zdejmuje element ze szczytu stosu.

Zwracane wartości

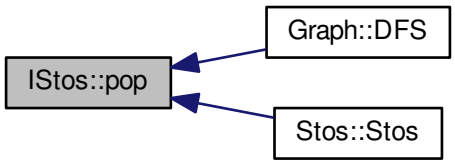
T	element ze szczytu stosu

Ostrzeżenie

Uwaga! Próba zdjęcia elementu z pustego stosu spowoduje wyrzucenie wyjątku
Sprawdź dokumentację metody Stos<T>::pop(void).

Implementowany w Stos< T >.

Oto graf wywoływań tej funkcji:



5.20.3.4 template<class T> virtual void IStos< T >::push (T) [pure virtual]

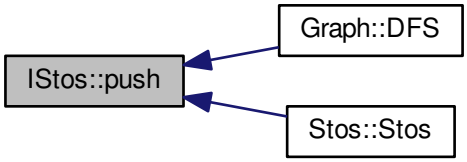
Umieszcza element na szczycie stosu.

Parametry

element	- element do umieszczenia na stosie

Implementowany w Stos< T >.

Oto graf wywoływań tej funkcji:



5.20.4 Dokumentacja przyjaciół i funkcji związanych

5.20.4.1 template<class T> std::ostream& operator<< (std::ostream & *output,* IStos< T > * *to*) [friend]

Przeciążenie operatora <<.

Definicja w linii 66 pliku stos.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

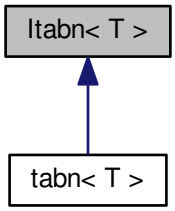
- stos.hh

5.21 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) const =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 maxIndex (void)=0

  Zwraca index ostatniego elementu.
- 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.
- virtual bool search (T)=0

  znajduje element w tablicy
- virtual int searchIndex (T)=0

  znajduje element w tablicy

Przyjaciele

- std::ostream & operator<< (std::ostream &output, Itabn< T > ∗to)

5.21.1 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.

5.21.2 Dokumentacja konstruktora i destruktora

5.21.2.1 template<class T> virtual Itabn< T >::∼Itabn ( ) [inline], [virtual]

Destruktor witrualny interfejsu.

Definicja w linii 88 pliku tabl.hh.

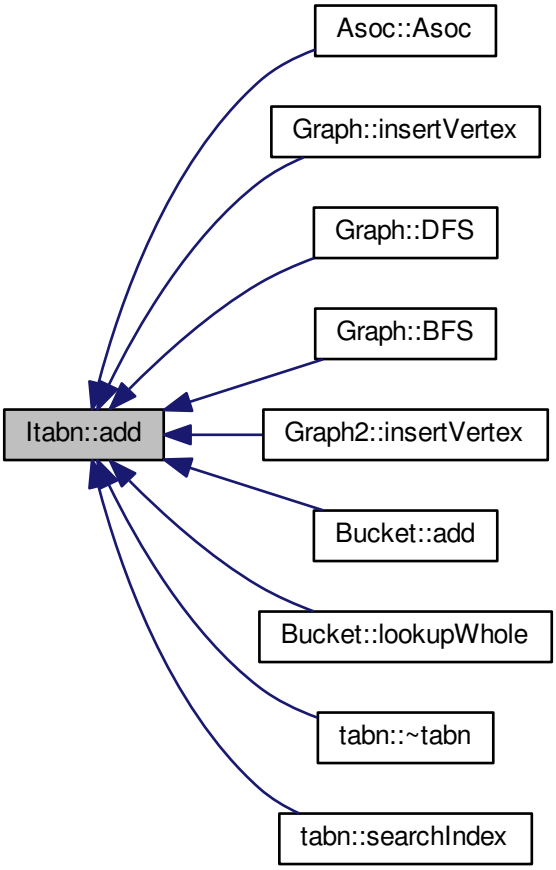
5.21.3 Dokumentacja funkcji składowych

5.21.3.1 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:



5.21.3.2 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.

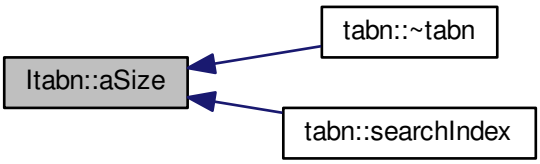
Implementowany w tabn< T >.

5.21.3.3 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:



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

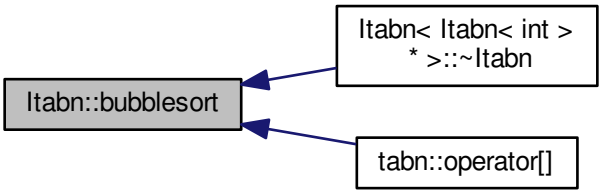
Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

Ostrzeżenie

Wymaga typu danych ze zdefiniowanym operatorem porównania "większe od"

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:

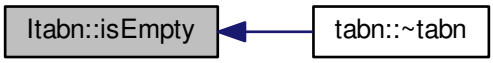


5.21.3.5 template<class T> virtual bool Itabn< T >::isEmpty (void) [pure virtual]

Sprawdza, czy tablica jest pusta.

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:

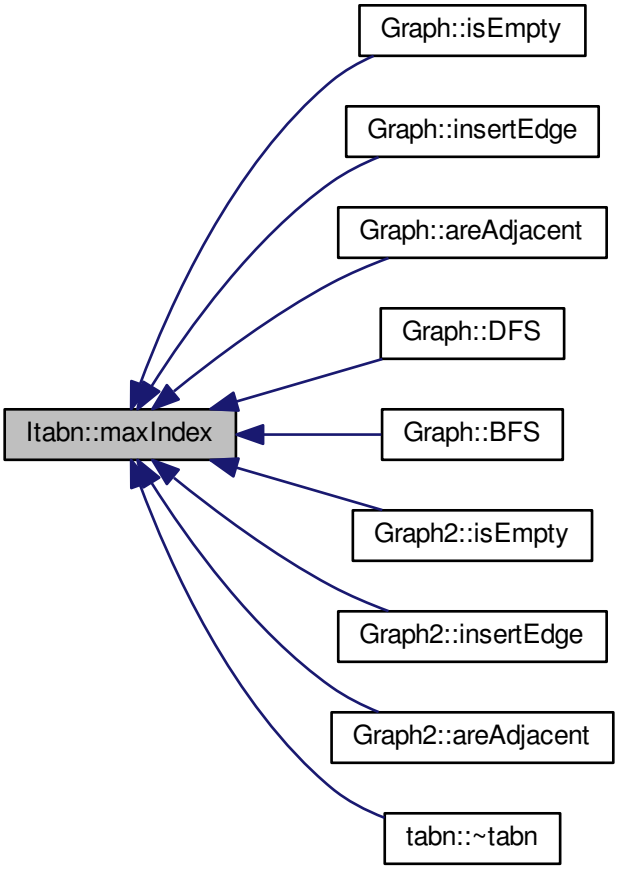


5.21.3.6 template<class T> virtual int Itabn< T >::maxIndex (void) [pure virtual]

Zwraca index ostatniego elementu.

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:

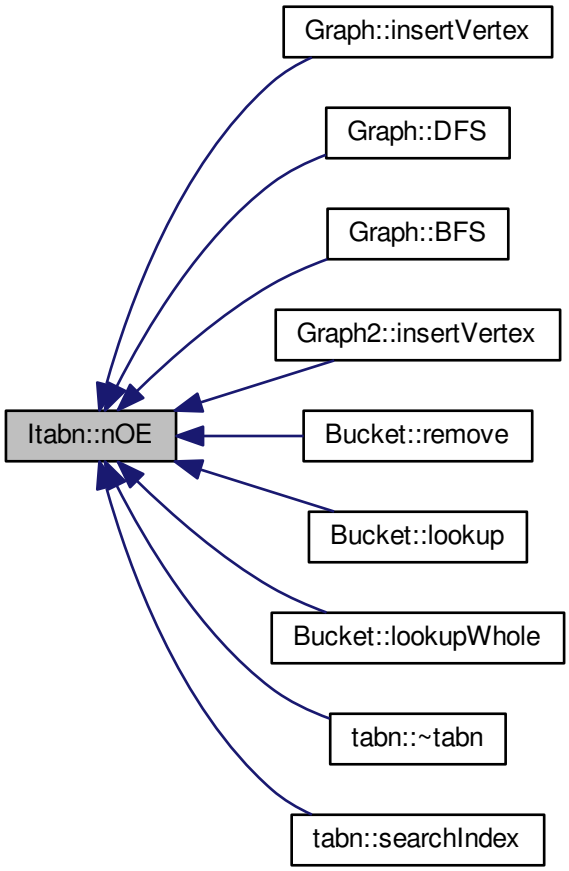


5.21.3.7 template<class T> virtual int Itabn< T >::nOE (void) [pure virtual]

Zwraca liczbę elementów w tablicy.

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:



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

Pozwala na dostęp do dowolnego elementu.

Implementowany w tabn< T >.

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

Pozwala na dostęp do dowolnego elementu.

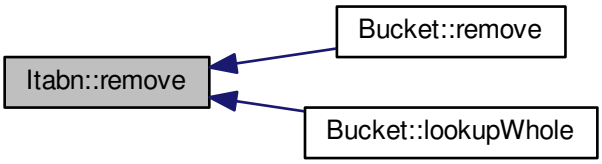
Implementowany w tabn< T >.

5.21.3.10 template<class T> virtual T Itabn< T >::remove ( ) [pure virtual]

Usuwa i zwraca element z końca tablicy.

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:



5.21.3.11 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 >.

5.21.3.12 template<class T> virtual bool Itabn< T >::search (T) [pure virtual]

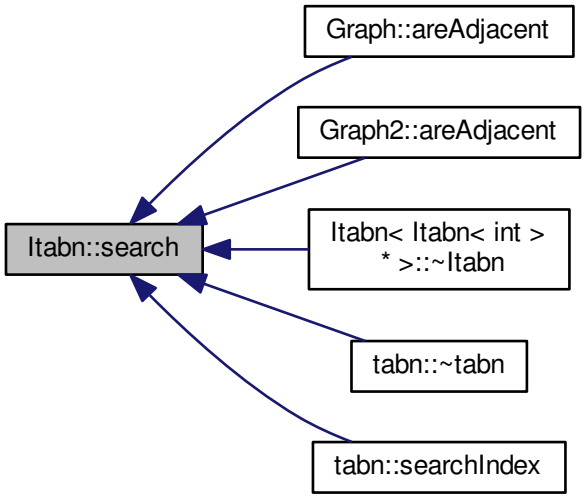
znajduje element w tablicy

Zwracane wartości

true	gdy element został znaleziony

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:



5.21.3.13 template<class T> virtual int ltabn< T >::searchIndex (T) [pure virtual]

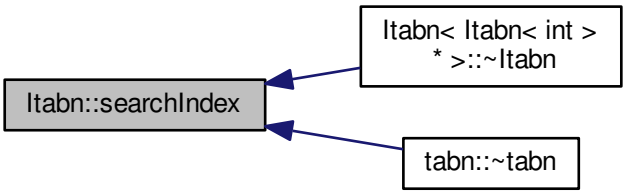
znajduje element w tablicy

Zwracane wartości

indeks	znalezionego elementu

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:

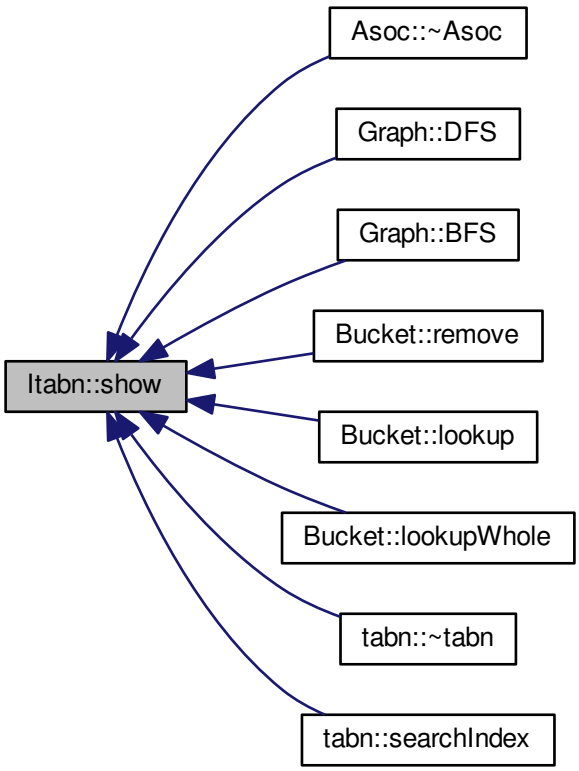


5.21.3.14 template<class T> virtual T Itabn< T >::show (int) const [pure virtual]

Zwraca żadany element, o ile istnieje.

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:

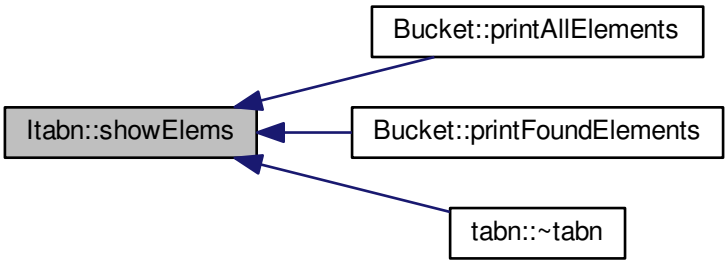


5.21.3.15 template<class T> virtual void Itabn< T >::showElems (void) [pure virtual]

Wyświetla elementy tablicy.

Implementowany w tabn< T >.

Oto graf wywoływań tej funkcji:



5.21.4 Dokumentacja przyjaciół i funkcji związanych

5.21.4.1 template<class T> std::ostream& operator<< (std::ostream & *output,* Itabn< T > * *to*) [friend]

Definicja w linii 111 pliku tabl.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

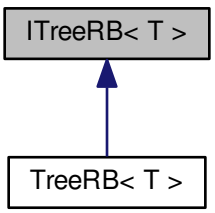
- tabl.hh

5.22 Dokumentacja szablonu klasy ITreeRB< T >

Interfejs klasy drzewa czerwono-czarnego.

`#include <tree.hh>`

Diagram dziedziczenia dla ITreeRB< T >



Metody publiczne

- virtual void insert (T)=0
- virtual void insert (T, nodeRB< T > *)=0
- virtual bool search (T)=0
- virtual ~ITreeRB ()
- virtual void leftRot (nodeRB< T > *)=0
- virtual void rightRot (nodeRB< T > *)=0
- virtual nodeRB< T > * retRoot (void)=0

Przyjaciele

- std::ostream & operator<< (std::ostream &output, ITreeRB *to)

5.22.1 Opis szczegółowy

template<class T>
class ITreeRB< T >

Interfejs klasy drzewa czerwono-czarnego.

Definicja w linii 177 pliku tree.hh.

5.22.2 Dokumentacja konstruktora i destruktora

5.22.2.1 template<class T> virtual ITreeRB< T >::~ITreeRB ( ) [inline], [virtual]

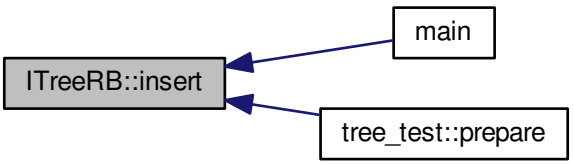
Definicja w linii 184 pliku tree.hh.

5.22.3 Dokumentacja funkcji składowych

5.22.3.1 template<class T> virtual void ITreeRB< T >::insert (T) [pure virtual]

Implementowany w TreeRB< T >.

Oto graf wywoływań tej funkcji:



5.22.3.2 template<class T> virtual void ITreeRB< T >::insert (T , nodeRB< T > *) [pure virtual]

Implementowany w TreeRB< T >.

5.22.3.3 template<class T> virtual void ITreeRB< T >::leftRot (nodeRB< T > *) [pure virtual]

Implementowany w TreeRB< T >.

5.22.3.4 template<class T> virtual nodeRB<T>* ITreeRB< T >::retRoot (void) [pure virtual]

Implementowany w TreeRB< T >.

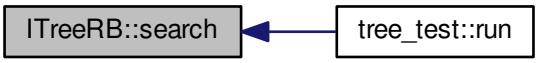
5.22.3.5 template<class T> virtual void ITreeRB< T >::rightRot (nodeRB< T > *) [pure virtual]

Implementowany w TreeRB< T >.

5.22.3.6 template<class T> virtual bool ITreeRB< T >::search (T) [pure virtual]

Implementowany w TreeRB< T >.

Oto graf wywoływań tej funkcji:



5.22.4 Dokumentacja przyjaciół i funkcji związanych

5.22.4.1 template<class T> std::ostream& operator<< (std::ostream & *output*, ITreeRB< T > * *to*) [friend]

Definicja w linii 189 pliku tree.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- tree.hh

5.23 Dokumentacja szablonu klasy Kolejka< T >

Klasa modeluje kolejkę

`#include <kolejka.hh>`

Diagram dziedziczenia dla Kolejka< T >

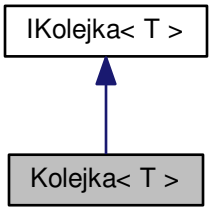
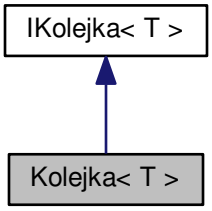


Diagram współpracy dla Kolejka< T >:



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.

5.23.1 Opis szczegółowy

template<class T>
class Kolejka< T >

Klasa modeluje kolejkę

Definicja w linii 70 pliku kolejka.hh.

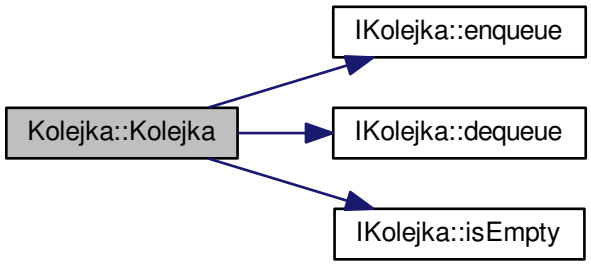
5.23.2 Dokumentacja konstruktora i destruktora

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

Konstruktor tablicy obsługującej kolejkę

Definicja w linii 77 pliku kolejka.hh.

Oto graf wywołań dla tej funkcji:



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

Destruktor klasy Kolejka.

Definicja w linii 123 pliku kolejka.hh.

5.23.3 Dokumentacja funkcji składowych

5.23.3.1 template<class T > T Kolejka< T >::dequeue (void) [virtual]

Usuwa i zwraca element z początku kolejki.

Zwracane wartości

T	element z początku kolejki

Wyjątki

CriticalException	re-throw z tabn<T>::remove()
ContinueException	re-throw z tabn<T>::remove()

Implementuje IKolejka< T >.

Definicja w linii 140 pliku kolejka.hh.

5.23.3.2 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 134 pliku kolejka.hh.

5.23.3.3 template<class T > T Kolejka< T >::get (void) [virtual]

Zwraca element z początku kolejki bez usuwania.

Zwracane wartości

T	element z początku kolejki

Wyjątki

CriticalException	re-throw z tab::show(int)

Ostrzeżenie

Uwaga! Próba odczytania elementu z pustej kolejki spowoduje wyrzucenie wyjątku.
Przykład sprawdzenia:

```
//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 157 pliku kolejka.hh.

5.23.3.4 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 152 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- kolejka.hh

5.24 Dokumentacja szablonu klasy Lista< T >

Klasa lista.

`#include <lista.hh>`

Diagram dziedziczenia dla Lista< T >

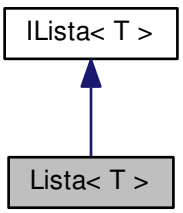
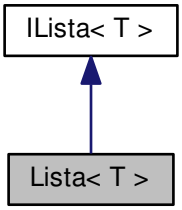


Diagram współpracy dla Lista< T >:



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 void qs (int, int)
- virtual ~Lista ()

  Destruktor Listy.

5.24.1 Opis szczegółowy

template<class T>
class Lista< T >

Klasa lista.

Modeluje pojęcie listy

Definicja w linii 84 pliku lista.hh.

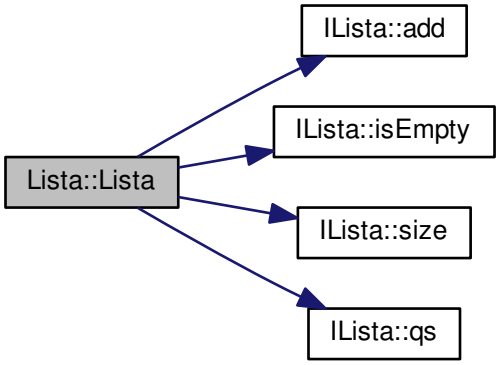
5.24.2 Dokumentacja konstruktora i destruktora

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

Konstruktor tablicy obsługującej listę

Definicja w linii 91 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



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

Destruktor Listy.

Definicja w linii 183 pliku lista.hh.

5.24.3 Dokumentacja funkcji składowych

5.24.3.1 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

Wyjątki

ContinueException	re-throw z tabn<T>::add(T,int)

Nota

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

Implementuje ILista< T >.

Definicja w linii 189 pliku lista.hh.

5.24.3.2 template<class T > void Lista< T >::add (T *element*) [virtual]

Dodaje element na koniec listy.

Implementuje ILista< T >.

Definicja w linii 199 pliku lista.hh.

5.24.3.3 template<class T > T Lista< T >::get (int *position*) [virtual]

Zwraca element z zadanego miejsca bez usunięcia.

Zwracane wartości

T	element w zadanym miejscu

Wyjątki

CriticalException	re-throw z tab<T>::show(int)

Ostrzeżenie

Próba podglądu elementu nieistniejącego spowoduje wyrzucenie wyjątku.
Przykład sprawdzenia:

```
//Przykład sprawdzenia poprawności podglądu
IListaa<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 226 pliku lista.hh.

5.24.3.4 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 221 pliku lista.hh.

5.24.3.5 template<class T > void Lista< T >::qs (int *indexFront,* int *indexBack*) [virtual]

Implementuje ILista< T >.

Definicja w linii 261 pliku lista.hh.

Oto graf wywołań dla tej funkcji:

5.24.3.6 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

Wyjątki

CriticalException	re-throw z tabn<T>::remove()
ContinueException	re-throw z tabn<T>::remove()

Ostrzeżenie

Próba usunięcia elementu nieistniejącego lub z pustej listy spowoduje wyrzucenie wyjątku.
Przykład sprawdzenia:

```
//Przykład sprawdzenia poprawności usuwania
```

```
IList<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 IList< T >.

Definicja w linii 204 pliku lista.hh.

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

Usuwa element z końca listy.

Implementuje IList< T >.

Definicja w linii 216 pliku lista.hh.

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

Zwraca ilość elementów w liście.

Zwracane wartości

int	ilość elementów

Implementuje IList< T >.

Definicja w linii 238 pliku lista.hh.

Oto graf wywołań dla tej funkcji:

Dokumentacja dla tej klasy została wygenerowana z pliku:

- lista.hh

5.25 Dokumentacja klasy lista_test

Definiuje sposób testowania wypełniania listy.

`#include <lista.hh>`

Diagram dziedziczenia dla lista_test

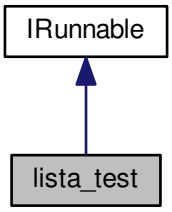
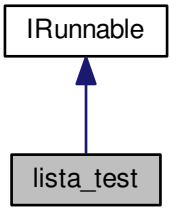


Diagram współpracy dla lista_test:



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.

5.25.1 Opis szczegółowy

Definiuje sposób testowania wypełniania listy.

Definicja w linii 303 pliku lista.hh.

5.25.2 Dokumentacja konstruktora i destruktora

5.25.2.1 lista_test::lista_test ( ) [inline]

Konstruktor klasy testującej.

Definicja w linii 315 pliku lista.hh.

5.25.2.2 lista_test::~lista_test ( ) [inline]

Destruktor klasy testującej.

Definicja w linii 321 pliku lista.hh.

Oto graf wywołań dla tej funkcji:

5.25.3 Dokumentacja funkcji składowych

5.25.3.1 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

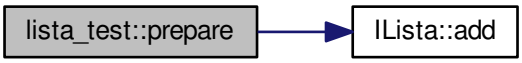
Wyjątki

CriticalException	gdy wystąpił błąd przy otwarciu pliku

Implementuje IRunnable.

Definicja w linii 359 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



5.25.3.2 virtual bool lista_test::run (void) [inline], [virtual]

Wykonuje test.

Pozwala na wykonanie testu.

Zwracane wartości

true	gdy test zakończył się sukcesem
false	gdy test zakończył się niepomyślnie

Wyjątki

CriticalException	re-throw z lista_test::wordSearch(std::string)

Implementuje IRunnable.

Definicja w linii 390 pliku lista.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- lista.hh

5.26 Dokumentacja szablonu klasy node< T >

Węzeł kolejki.

`#include <kolejka.hh>`

Metody publiczne

- node (T o)
- node (void)
- node< T > & operator= (const node< T > &read)

Atrybuty publiczne

- T value
- class node< T > * next
- class node< T > * previous

Przyjaciele

- bool operator< (node< T > one, node< T > two)
- bool operator> (node< T > one, node< T > two)
- bool operator<= (node< T > one, node< T > two)
- bool operator>= (node< T > one, node< T > two)
- bool operator== (node< T > one, node< T > two)
- std::ostream & operator<< (std::ostream &output, const node< T > &to)

5.26.1 Opis szczegółowy

template<class T>
class node< T >

Węzeł kolejki.

Definicja w linii 175 pliku kolejka.hh.

5.26.2 Dokumentacja konstruktora i destruktora

5.26.2.1 template<class T> node< T >::node (T *o*) [inline]

Definicja w linii 181 pliku kolejka.hh.

5.26.2.2 template<class T> node< T >::node (void) [inline]

Definicja w linii 184 pliku kolejka.hh.

5.26.3 Dokumentacja funkcji składowych

5.26.3.1 template<class T> node<T>& node< T >::operator= (const node< T > & *read*) [inline]

Definicja w linii 188 pliku kolejka.hh.

5.26.4 Dokumentacja przyjaciół i funkcji związanych

5.26.4.1 template<class T> bool operator< (node< T > *one,* node< T > *two*) [friend]

Definicja w linii 195 pliku kolejka.hh.

5.26.4.2 template<class T> std::ostream& operator<< (std::ostream & *output,* const node< T > & *to*) [friend]

Definicja w linii 220 pliku kolejka.hh.

5.26.4.3 template<class T> bool operator<= (node< T > *one,* node< T > *two*) [friend]

Definicja w linii 205 pliku kolejka.hh.

5.26.4.4 template<class T> bool operator== (node< T > *one,* node< T > *two*) [friend]

Definicja w linii 215 pliku kolejka.hh.

5.26.4.5 template<class T> bool operator> (node< T > *one,* node< T > *two*) [friend]

Definicja w linii 200 pliku kolejka.hh.

5.26.4.6 template<class T> bool operator>= (node< T > *one,* node< T > *two*) [friend]

Definicja w linii 210 pliku kolejka.hh.

5.26.5 Dokumentacja atrybutów składowych

5.26.5.1 template<class T> class node< T >* node< T >::next

Definicja w linii 178 pliku kolejka.hh.

5.26.5.2 template<class T> class node< T >* node< T >::previous

Definicja w linii 179 pliku kolejka.hh.

5.26.5.3 template<class T> T node< T >::value

Definicja w linii 177 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- kolejka.hh

5.27 Dokumentacja szablonu klasy nodeRB< T >

`#include <tree.hh>`

Metody publiczne

- nodeRB (T addKey, Colour col=red, nodeRB< T > *addUp=NULL, nodeRB< T > *addLeft=NULL, nodeRB< T > *addRight=NULL)

  brief_desc
- T getKey (void)
- Colour getColour (void)
- nodeRB< T > * getLeft (void)
- nodeRB< T > * getRight (void)
- nodeRB< T > * getParent (void)
- T getLeftKey (void)
- T getRightKey (void)
- T getParentKey (void)
- void setKey (T keyToSet)
- void setColour (Colour colourToSet)
- void setLeft (nodeRB< T > *leftDescendant)
- void setRight (nodeRB< T > *rightDescendant)
- void setParent (nodeRB< T > *parent)
- nodeRB< T > & operator= (const nodeRB< T > &read)

Atrybuty publiczne

- T key
- Colour colour
- int balanceFactor
- class nodeRB< T > * left
- class nodeRB< T > * right
- class nodeRB< T > * up

Przyjaciele

- bool operator< (nodeRB< T > one, nodeRB< T > two)
- bool operator> (nodeRB< T > one, nodeRB< T > two)
- bool operator<= (nodeRB< T > one, nodeRB< T > two)
- bool operator>= (nodeRB< T > one, nodeRB< T > two)
- bool operator== (nodeRB< T > one, nodeRB< T > two)
- std::ostream & operator<< (std::ostream &output, const nodeRB< T > *to)
- std::istream & operator>> (std::istream &input, const nodeRB< T > *to)

5.27.1 Opis szczegółowy

template<class T>
class nodeRB< T >

Definicja w linii 23 pliku tree.hh.

5.27.2 Dokumentacja konstruktora i destruktora

5.27.2.1 template<class T> nodeRB< T >::nodeRB (T *addKey*, Colour *col* = red, nodeRB< T > * *addUp* = NULL, nodeRB< T > * *addLeft* = NULL, nodeRB< T > * *addRight* = NULL) [inline]

brief_desc

Definicja w linii 37 pliku tree.hh.

5.27.3 Dokumentacja funkcji składowych

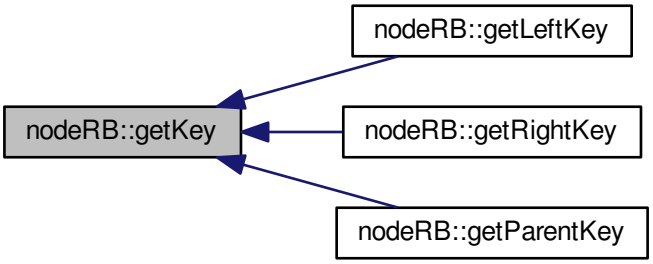
5.27.3.1 template<class T> Colour nodeRB< T >::getColour (void) [inline]

Definicja w linii 50 pliku tree.hh.

5.27.3.2 template<class T> T nodeRB< T >::getKey (void) [inline]

Definicja w linii 46 pliku tree.hh.

Oto graf wywoływań tej funkcji:



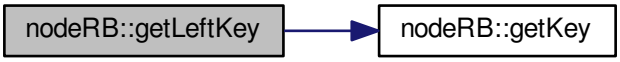
5.27.3.3 template<class T> nodeRB<T>* nodeRB< T >::getLeft (void) [inline]

Definicja w linii 55 pliku tree.hh.

5.27.3.4 template<class T> T nodeRB< T >::getLeftKey (void) [inline]

Definicja w linii 67 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



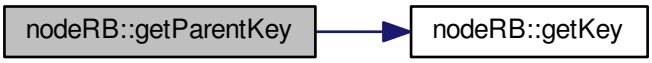
5.27.3.5 template<class T> nodeRB<T>* nodeRB< T >::getParent (void) [inline]

Definicja w linii 63 pliku tree.hh.

5.27.3.6 template<class T> T nodeRB< T >::getParentKey (void) [inline]

Definicja w linii 81 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



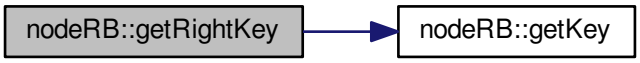
5.27.3.7 template<class T> nodeRB<T>* nodeRB< T >::getRight (void) [inline]

Definicja w linii 59 pliku tree.hh.

5.27.3.8 template<class T> T nodeRB< T >::getRightKey (void) [inline]

Definicja w linii 74 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



5.27.3.9 template<class T> nodeRB<T>& nodeRB< T >::operator= (const nodeRB< T > & *read*) [inline]

Definicja w linii 108 pliku tree.hh.

5.27.3.10 template<class T> void nodeRB< T >::setColour (Colour *colourToSet*) [inline]

Definicja w linii 92 pliku tree.hh.

5.27.3.11 template<class T> void nodeRB< T >::setKey (T *keyToSet*) [inline]

Definicja w linii 88 pliku tree.hh.

5.27.3.12 template<class T> void nodeRB< T >::setLeft (nodeRB< T > * *leftDescendant*) [inline]

Definicja w linii 96 pliku tree.hh.

Oto graf wywoływań tej funkcji:

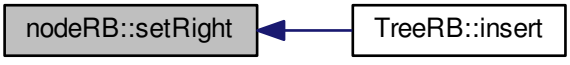
5.27.3.13 template<class T> void nodeRB< T >::setParent (nodeRB< T > * *parent*) [inline]

Definicja w linii 104 pliku tree.hh.

5.27.3.14 template<class T> void nodeRB< T >::setRight (nodeRB< T > * *rightDescendant*) [inline]

Definicja w linii 100 pliku tree.hh.

Oto graf wywoływań tej funkcji:



5.27.4 Dokumentacja przyjaciół i funkcji związanych

5.27.4.1 template<class T> bool operator< (nodeRB< T > *one,* nodeRB< T > *two*) [friend]

Definicja w linii 117 pliku tree.hh.

5.27.4.2 template<class T> std::ostream& operator<< (std::ostream & *output,* const nodeRB< T > * *to*) [friend]

Definicja w linii 143 pliku tree.hh.

5.27.4.3 template<class T> bool operator<= (nodeRB< T > *one,* nodeRB< T > *two*) [friend]

Definicja w linii 127 pliku tree.hh.

5.27.4.4 template<class T> bool operator== (nodeRB< T > *one,* nodeRB< T > *two*) [friend]

Definicja w linii 138 pliku tree.hh.

5.27.4.5 template<class T> bool operator> (nodeRB< T > *one,* nodeRB< T > *two*) [friend]

Definicja w linii 122 pliku tree.hh.

5.27.4.6 template<class T> bool operator>= (nodeRB< T > *one,* nodeRB< T > *two*) [friend]

Definicja w linii 133 pliku tree.hh.

5.27.4.7 template<class T> std::istream& operator>> (std::istream & *input,* const nodeRB< T > * *to*) [friend]

Definicja w linii 159 pliku tree.hh.

5.27.5 Dokumentacja atrybutów składowych

5.27.5.1 template<class T> int nodeRB< T >::balanceFactor

Definicja w linii 27 pliku tree.hh.

5.27.5.2 template<class T> Colour nodeRB< T >::colour

Definicja w linii 26 pliku tree.hh.

5.27.5.3 template<class T> T nodeRB< T >::key

Definicja w linii 25 pliku tree.hh.

5.27.5.4 template<class T> class nodeRB< T >* nodeRB< T >::left

Definicja w linii 28 pliku tree.hh.

5.27.5.5 template<class T> class nodeRB< T >* nodeRB< T >::right

Definicja w linii 29 pliku tree.hh.

5.27.5.6 template<class T> class nodeRB< T >* nodeRB< T >::up

Definicja w linii 30 pliku tree.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- tree.hh

5.28 Dokumentacja szablonu klasy Queue< T >

Kolejka oparta na węzłach.

`#include <kolejka.hh>`

Diagram dziedziczenia dla Queue< T >

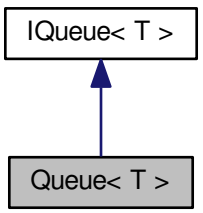
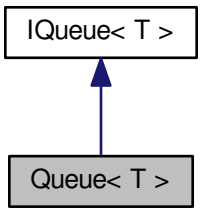


Diagram współpracy dla Queue< T >:



Metody publiczne

- Queue (void)
- virtual void enqueue (T element)

  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 ~Queue ()

  Destruktor.

5.28.1 Opis szczegółowy

template< class T >
class Queue< T >

Kolejka oparta na węzłach.

Definicja w linii 283 pliku kolejka.hh.

5.28.2 Dokumentacja konstruktora i destruktora

5.28.2.1 template< class T > Queue< T >::Queue (void) [inline]

Definicja w linii 289 pliku kolejka.hh.

5.28.2.2 template< class T > virtual Queue< T >::~Queue ( ) [inline], [virtual]

Destruktor.

Definicja w linii 363 pliku kolejka.hh.

5.28.3 Dokumentacja funkcji składowych

5.28.3.1 template<class T > virtual T Queue< T >::dequeue (void) [inline], [virtual]

Usuwa i zwraca element z początku kolejki.

Zwracane wartości

T	element z początku kolejki

Implementuje IQueue< T >.

Definicja w linii 327 pliku kolejka.hh.

5.28.3.2 template<class T > virtual void Queue< T >::enqueue (T *element*) [inline], [virtual]

Dodaje element na koniec kolejki.

Parametry

element	- element do umieszczenia w kolejce

Implementuje IQueue< T >.

Definicja w linii 310 pliku kolejka.hh.

5.28.3.3 template<class T > virtual T Queue< T >::get (void) [inline], [virtual]

Zwraca element z początku kolejki bez usuwania.

Implementuje IQueue< T >.

Definicja w linii 355 pliku kolejka.hh.

5.28.3.4 template<class T > virtual bool Queue< T >::isEmpty (void) [inline], [virtual]

Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementuje IQueue< T >.

Definicja w linii 347 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- kolejka.hh

5.29 Dokumentacja klasy Stoper

Klasa stoper implementująca interfejs IStoper.

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

Diagram dziedziczenia dla Stoper

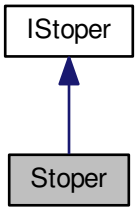
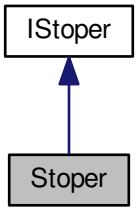


Diagram współpracy dla Stoper:



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.

5.29.1 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 35 pliku stoper.hh.

5.29.2 Dokumentacja funkcji składowych

5.29.2.1 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.

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

Uruchamia zegar.

Implementuje IStoper.

Definicja w linii 4 pliku stoper.cpp.

5.29.2.3 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

5.30 Dokumentacja szablonu klasy Stos< T >

Klasa Stos.

`#include <stos.hh>`

Diagram dziedziczenia dla Stos< T >

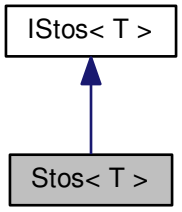
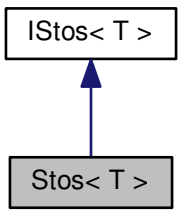


Diagram współpracy dla Stos< T >:



Metody publiczne

- Stos ()
  Konstruktor tablicy obsługującej stos.
- virtual void push (T)
  Umieszcza element na szczycie stosu.
- virtual T pop (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.

5.30.1 Opis szczegółowy

template<class T>
class Stos< T >

Klasa Stos.

Modeluje pojęcie stosu

Definicja w linii 79 pliku stos.hh.

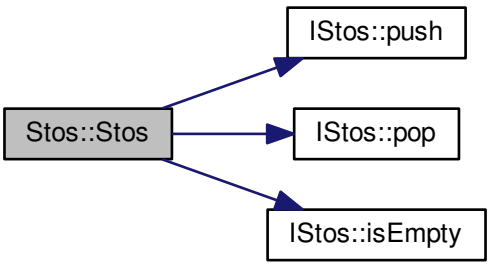
5.30.2 Dokumentacja konstruktora i destruktora

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

Konstruktor tablicy obsługującej stos.

Definicja w linii 86 pliku stos.hh.

Oto graf wywołań dla tej funkcji:



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

Destruktor stosu.

Definicja w linii 145 pliku stos.hh.

5.30.3 Dokumentacja funkcji składowych

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

Zwraca element ze szczytu stosu bez jego usuwania.

Zwracane wartości

T	element ze szczytu stosu

Wyjątki

CriticalException	re-throw z tabn<T>::show(int)

Ostrzeżenie

Uwaga! Próba odczytania elementu z pustego stosu spowoduje wyrzucenie wyjątku
Przykład sprawdzenia:

```
//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 179 pliku stos.hh.

5.30.3.2 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 174 pliku stos.hh.

5.30.3.3 template<class T > T Stos< T >::pop (void) [virtual]

Zdejmuje element ze szczytu stosu.

Zwracane wartości

T	element ze szczytu stosu

Wyjątki

CriticalException	re-throw z tabn<T>::remove()
ContinueException	re-throw z tabn<T>::remove()

Ostrzeżenie

Uwaga! Próba zdjęcia elementu z pustego stosu spowoduje wyrzucenie wyjątku
Przykład sprawdzenia:

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

Implementuje IStos< T >.

Definicja w linii 162 pliku stos.hh.

5.30.3.4 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 157 pliku stos.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- stos.hh

5.31 Dokumentacja szablonu klasy tabn< T >

Modeluje tablicę dynamicznie rozszerzalną

`#include <tabl.hh>`

Diagram dziedziczenia dla tabn< T >

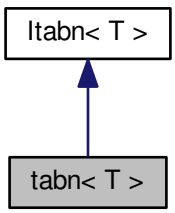
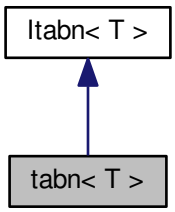


Diagram współpracy dla tabn< T >:



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) const

  Zwraca żądany element, o ile istnieje, bez jego usuwania.
- virtual void showElems (void)

  Wyświetla listę elementów.
- virtual int nOE (void)

  zwraca liczbę elementów w tablicy
- virtual int maxIndex (void)

  Zwraca index ostatniego elementu.
- virtual int aSize (void)

  zwraca wielkość zaalokowanej przestrzeni dla tablicy
- virtual bool search (T)

  znajduje element w tablicy
- virtual int searchIndex (T)

  znajduje element w tablicy
- virtual T & operator[ ] (int index)

  Umożliwia dostęp do dowolnego elementu tablicy bez sprawdzania zakresu (debug)
- virtual T operator[ ] (int index) const

  Umożliwia odczyt dowolnego elementu tablicy bez sprawdzania zakresu (debug)
- virtual void bubblesort (void)

  Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

5.31.1 Opis szczegółowy

template<class T>
class tabn< T >

Modeluje tablicę dynamicznie rozszerzalną

Przechowuje elementy w rozszerzalnej tablicy o rozmiarze początkowym SIZE

Definicja w linii 128 pliku tabl.hh.

5.31.2 Dokumentacja konstruktora i destruktora

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

Konstruktor klasy tabn.

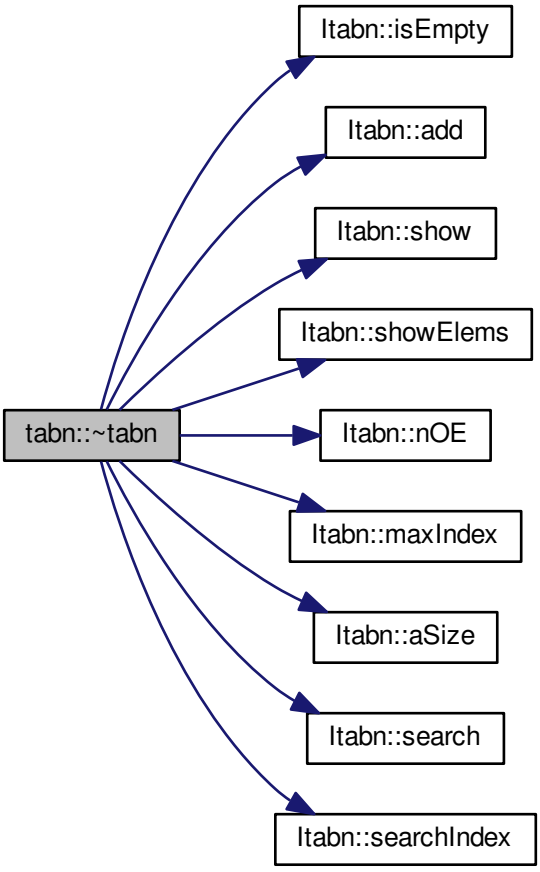
Definicja w linii 139 pliku tabl.hh.

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

Destruktor klasy tabn.

Definicja w linii 148 pliku tabl.hh.

Oto graf wywołań dla tej funkcji:



5.31.3 Dokumentacja funkcji składowych

5.31.3.1 template<class T> virtual void tabn< T >::add (T) [virtual]

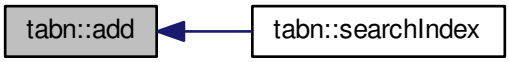
Dodaje element Dodaje element do tablicy dynamicznej, odpowiednio ją rozszerzając.

Parametry

element	- element do dodania

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:



5.31.3.2 template<class T> virtual void tabn< T >::add (T , int) [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	WrongPositionToShiftFromRightException przy próbie dodania elementu do niewłaściwego miejsca (re-throw z tabn<T>::shiftRight(T,int)).

Implementuje Itabn< T >.

5.31.3.3 template<class T> virtual int tabn< T >::aSize (void) [virtual]

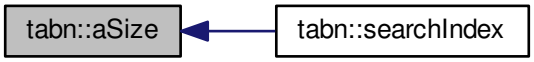
zwraca wielkość zaalokowanej przestrzeni dla tablicy

Zwracane wartości

int	Ilość zaalokowanych pól

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:



5.31.3.4 template<class T> virtual 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 z tabn<T>::swap(int,int)

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:

5.31.3.5 template<class T> virtual 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 >.

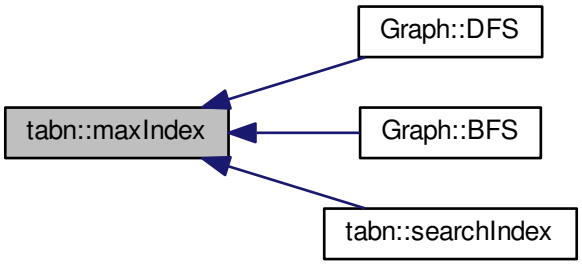
Oto graf wywoływań tej funkcji:

5.31.3.6 template<class T> virtual int tabn< T >::maxIndex (void) [virtual]

Zwraca index ostatniego elementu.

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:



5.31.3.7 template<class T> virtual int tabn< T >::nOE (void) [virtual]

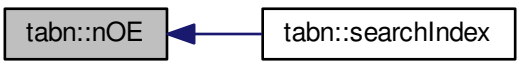
zwraca liczbę elementów w tablicy

Zwracane wartości

int	Liczba elementów w tablicy

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:



5.31.3.8 template<class T> virtual T& tabn< T >::operator[] (int *index*) [inline], [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 297 pliku tabl.hh.

5.31.3.9 template<class T> virtual T tabn< T >::operator[] (int *index*) const [inline], [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 309 pliku tabl.hh.

Oto graf wywołań dla tej funkcji:

5.31.3.10 template<class T> virtual T tabn< T >::remove ( ) [virtual]

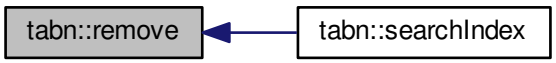
Usuwa i zwraca ostatni element z tablicy.

Wyjątki

CriticalException	EmptyTableException przy próbie usunięcia z pustej tablicy (re-throw z tabn<T>::isEmptyException()).
CriticalException	WrongIndexException przy próbie usunięcia z pustej tablicy (re-throw z tabn<T>::show(int)).
ContinueException	re-throw z tabn<T>::reduce2().

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:



5.31.3.11 template<class T> virtual T tabn< T >::remove (int) [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	EmptyTableException przy próbie usunięcia z pustej tablicy lub nieistniejącego elementu (re-throw z tabn<T>::isEmptyException()).
CriticalException	WrongIndexException przy próbie usunięcia z pustej tablicy lub nieistniejącego elementu (re-throw z tabn<T>::show(int)).
ContinueException	re-throw z tabn<T>::reduce2().

Implementuje Itabn< T >.

5.31.3.12 template<class T > bool tabn< T >::search (T *elem*) [virtual]

znajduje element w tablicy

Zwracane wartości

true	gdy element został znaleziony

Implementuje Itabn< T >.

Definicja w linii 343 pliku tabl.hh.

5.31.3.13 template<class T > int tabn< T >::searchIndex (T *elem*) [virtual]

znajduje element w tablicy

Zwracane wartości

index	indeks znalezionego elementu

Definiuje sposób testowania wypełniania tablicy tabn

Konstruktor klasy tabn_test.

Destruktor klasy tabn_test.

Metoda ustawia punkt startowy generatora pseudolosowego.

Metoda generuje liczbę pseudolosową z zakresu 0..9

Zwracane wartości

Liczba	pseudolosowa z zakresu 0..9

Przygotowuje rozmiar testu

Parametry

sizeOfTest	- rozmiar testu

Zwracane wartości

bool	zawsze true

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

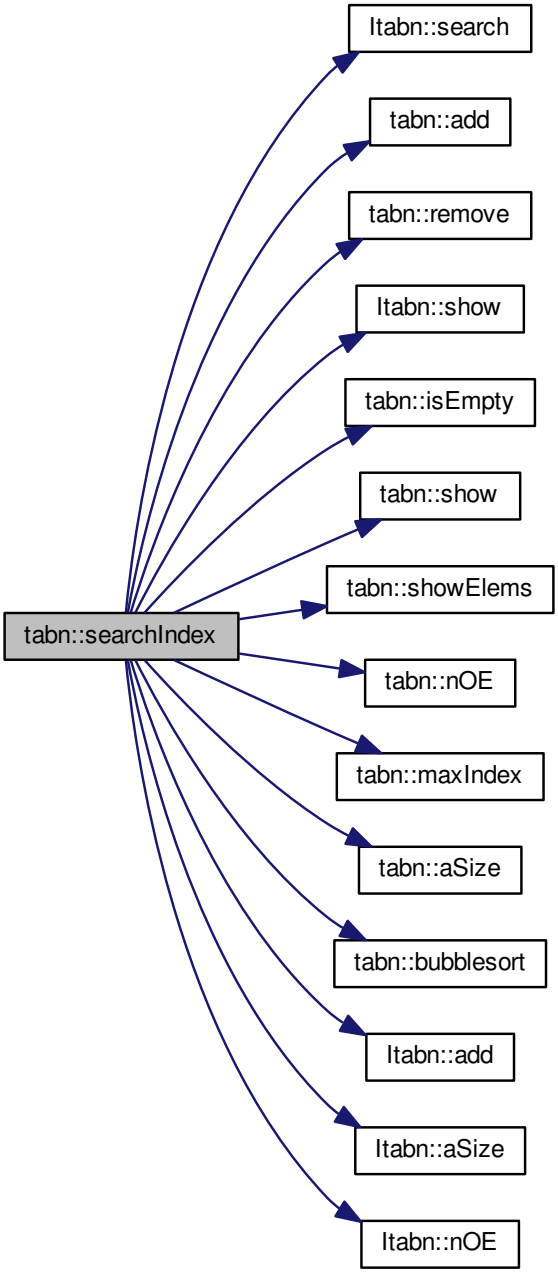
Wyjątki

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

Implementuje Itabn< T >.

Definicja w linii 351 pliku tabl.hh.

Oto graf wywołań dla tej funkcji:



5.31.3.14 template<class T> virtual T tabn< T >::show (int) const [virtual]

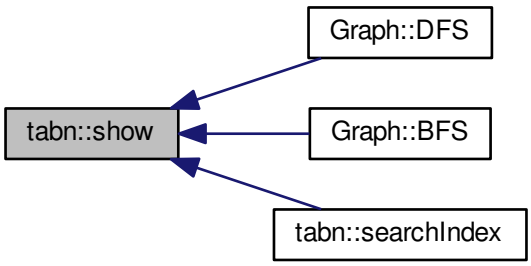
Zwraca żądany element, o ile istnieje, bez jego usuwania.

Wyjątki

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

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:



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

Wyświetla listę elementów.

Implementuje Itabn< T >.

Oto graf wywoływań tej funkcji:

Dokumentacja dla tej klasy została wygenerowana z pliku:

- tabl.hh

5.32 Dokumentacja klasy test_graph_BFS

`#include <graph.hh>`

Diagram dziedziczenia dla test_graph_BFS

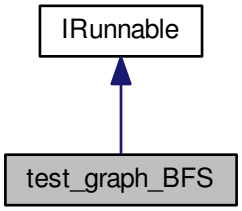
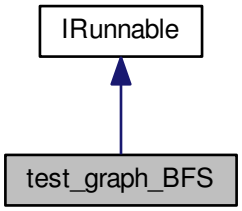


Diagram współpracy dla test_graph_BFS:



Metody publiczne

- test_graph_BFS ()
- bool prepare (int testSize)

  Przygotowanie badań
- bool run (void)

  Przeprowadzanie badań

5.32.1 Opis szczegółowy

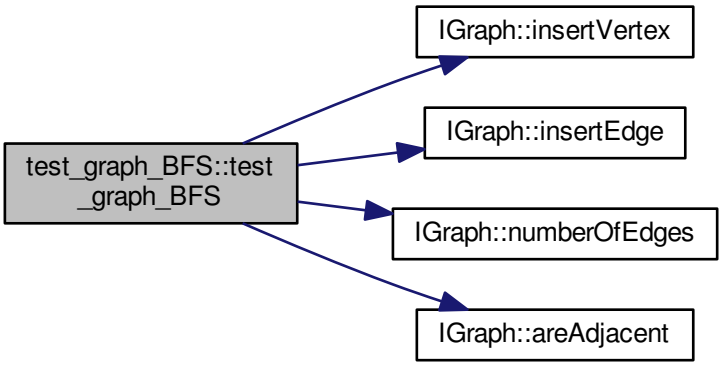
Definicja w linii 215 pliku graph.hh.

5.32.2 Dokumentacja konstruktora i destruktora

5.32.2.1 test_graph_BFS::test_graph_BFS ( ) [inline]

Definicja w linii 232 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.32.3 Dokumentacja funkcji składowych

5.32.3.1 bool test_graph_BFS::prepare (int) [inline], [virtual]

Przygotowanie badań

Implementuje IRunnable.

Definicja w linii 307 pliku graph.hh.

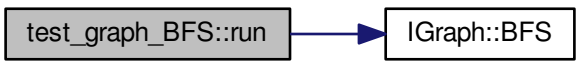
5.32.3.2 bool test_graph_BFS::run (void) [inline], [virtual]

Przeprowadzanie badań

Implementuje IRunnable.

Definicja w linii 318 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- graph.hh

5.33 Dokumentacja klasy test_graph_DFS

`#include <graph.hh>`

Diagram dziedziczenia dla test_graph_DFS

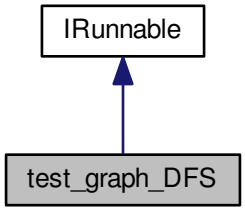
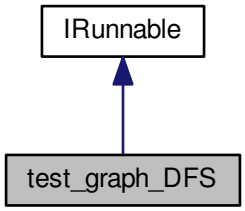


Diagram współpracy dla test_graph_DFS:



Metody publiczne

- test_graph_DFS ()
- bool prepare (int testSize)

  Przygotowanie badań
- bool run (void)

  Przeprowadzanie badań

5.33.1 Opis szczegółowy

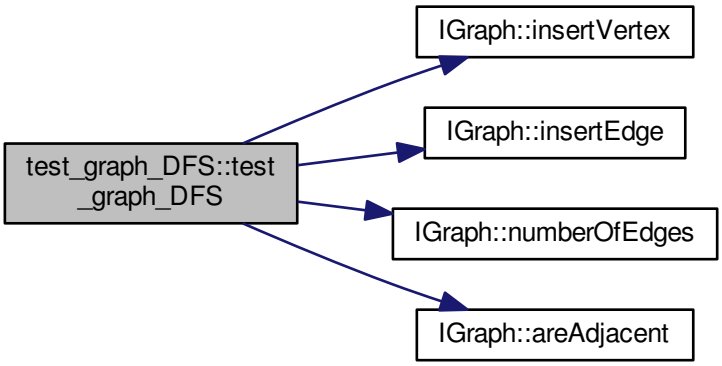
Definicja w linii 327 pliku graph.hh.

5.33.2 Dokumentacja konstruktora i destruktora

5.33.2.1 test_graph_DFS::test_graph_DFS ( ) [inline]

Definicja w linii 344 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.33.3 Dokumentacja funkcji składowych

5.33.3.1 bool test_graph_DFS::prepare (int) [inline], [virtual]

Przygotowanie badań

Implementuje IRunnable.

Definicja w linii 419 pliku graph.hh.

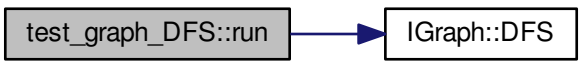
5.33.3.2 bool test_graph_DFS::run (void) [inline], [virtual]

Przeprowadzanie badań

Implementuje IRunnable.

Definicja w linii 430 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- graph.hh

5.34 Dokumentacja klasy tree_test

`#include <tree.hh>`

Diagram dziedziczenia dla tree_test

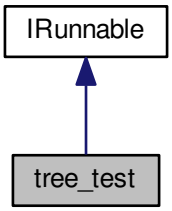
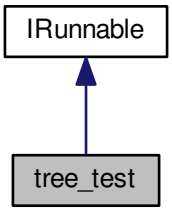


Diagram współpracy dla tree_test:



Metody publiczne

- tree_test ()
- virtual ~tree_test ()
- virtual bool prepare (int sizeOfTest)

  Przygotowanie badań
- virtual bool run ()

  Przeprowadzanie badań

5.34.1 Opis szczegółowy

Definicja w linii 507 pliku tree.hh.

5.34.2 Dokumentacja konstruktora i destruktora

5.34.2.1 tree_test::tree_test ( ) [inline]

Definicja w linii 530 pliku tree.hh.

5.34.2.2 virtual tree_test::~tree_test ( ) [inline], [virtual]

Definicja w linii 534 pliku tree.hh.

5.34.3 Dokumentacja funkcji składowych

5.34.3.1 virtual bool tree_test::prepare (int) [inline], [virtual]

Przygotowanie badań

Implementuje IRunnable.

Definicja w linii 538 pliku tree.hh.

Oto graf wywołań dla tej funkcji:

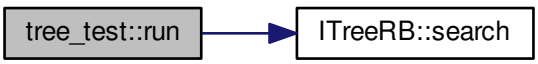
5.34.3.2 virtual bool tree_test::run ( ) [inline], [virtual]

Przeprowadzanie badań

Implementuje IRunnable.

Definicja w linii 552 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- tree.hh

5.35 Dokumentacja szablonu klasy TreeRB< T >

Klasa implementująca interfejs drzewa czerwono-czarnego.

`#include <tree.hh>`

Diagram dziedziczenia dla TreeRB< T >

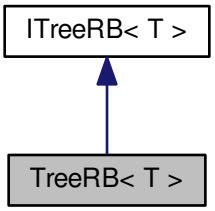
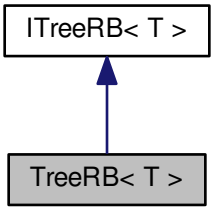


Diagram współpracy dla TreeRB< T >:



Metody publiczne

- virtual nodeRB< T > * retRoot (void)
- virtual void leftRot (nodeRB< T > *nd)

  Obraca wybrane poddrzewo w lewo.
- virtual void rightRot (nodeRB< T > *nd)
- TreeRB ()

  Konstruktor.
- virtual ~TreeRB ()

  Destruktor.
- virtual void insert (T element)

  Wstawia element do drzewa.
- virtual void insert (T element, nodeRB< T > *node)
- virtual bool search (T k)

5.35.1 Opis szczegółowy

template<class T>
class TreeRB< T >

Klasa implementująca interfejs drzewa czerwono-czarnego.

Definicja w linii 200 pliku tree.hh.

5.35.2 Dokumentacja konstruktora i destruktora

5.35.2.1 template<class T > TreeRB< T >::TreeRB ( ) [inline]

Konstruktor.

Definicja w linii 329 pliku tree.hh.

5.35.2.2 template<class T > virtual TreeRB< T >::~TreeRB ( ) [inline], [virtual]

Destruktor.

Definicja w linii 335 pliku tree.hh.

5.35.3 Dokumentacja funkcji składowych

5.35.3.1 template<class T > virtual void TreeRB< T >::insert (T *element*) [inline], [virtual]

Wstawia element do drzewa.

Ostrzeżenie

Nowy element jest domyślnie zakolorowany na czerwono (patrz konstruktor nodeRB())

Implementuje ITreeRB< T >.

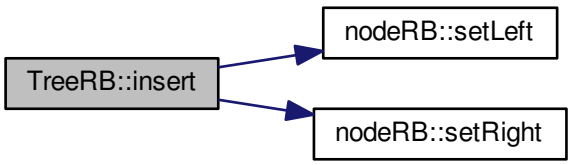
Definicja w linii 343 pliku tree.hh.

5.35.3.2 template<class T > virtual void TreeRB< T >::insert (T *element*, nodeRB< T > * *node*) [inline], [virtual]

Implementuje ITreeRB< T >.

Definicja w linii 348 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



5.35.3.3 template<class T > virtual void TreeRB< T >::leftRot (nodeRB< T > * *nd*) [inline], [virtual]

Obraca wybrane poddrzewo w lewo.

Implementuje ITreeRB< T >.

Definicja w linii 253 pliku tree.hh.

5.35.3.4 template<class T > virtual nodeRB<T>* TreeRB< T >::retRoot (void) [inline], [virtual]

Implementuje ITreeRB< T >.

Definicja w linii 246 pliku tree.hh.

5.35.3.5 template<class T > virtual void TreeRB< T >::rightRot (nodeRB< T > * *nd*) [inline], [virtual]

Implementuje ITreeRB< T >.

Definicja w linii 286 pliku tree.hh.

5.35.3.6 template<class T > virtual bool TreeRB< T >::search (T *k*) [inline], [virtual]

Implementuje ITreeRB< T >.

Definicja w linii 484 pliku tree.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- tree.hh

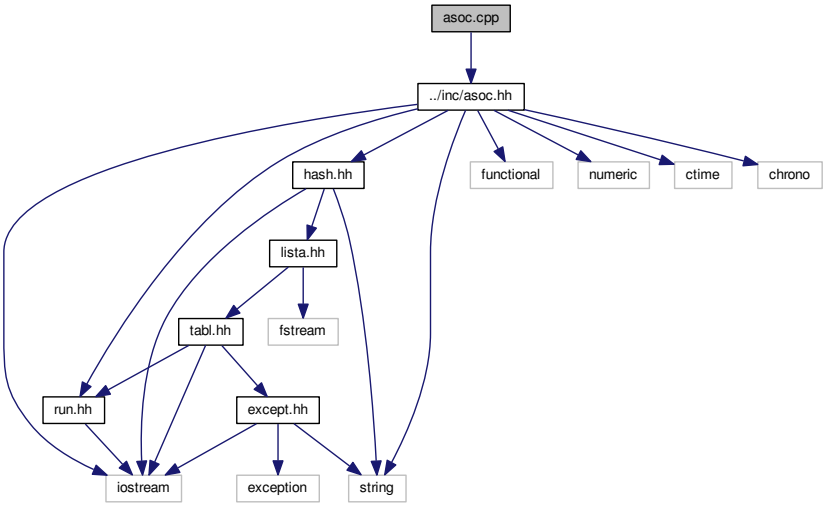
Rozdział 6

Dokumentacja plików

6.1 Dokumentacja pliku asoc.cpp

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

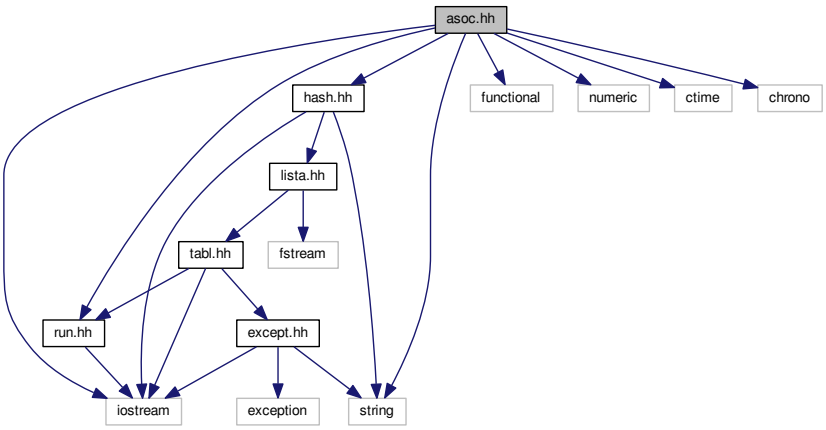
Wykres zależności załączania dla asoc.cpp:



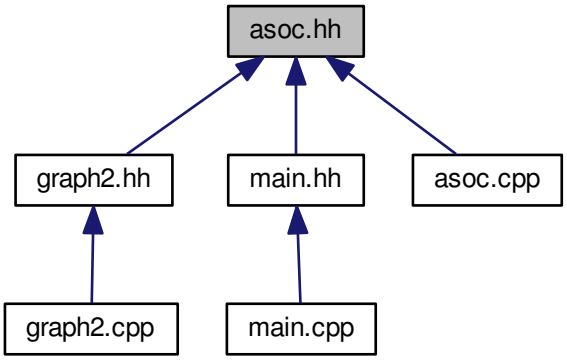
6.2 Dokumentacja pliku asoc.hh

```
#include <iostream>
#include <string>
#include <functional>
#include <numeric>
#include "hash.hh"
#include "run.hh"
#include <ctime>
#include <chrono>
```

Wykres zależności załączania dla asoc.hh:



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



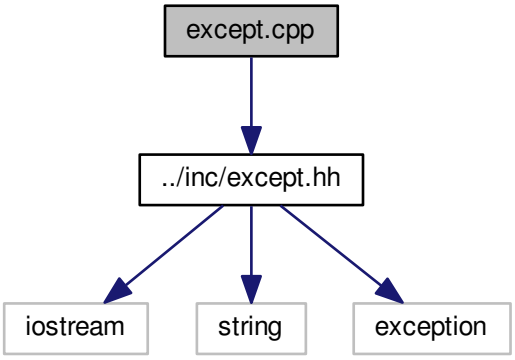
Komponenty

- class IAsoc< T, T2 >
- class Asoc< T, T2 >
- class asoc_test

6.3 Dokumentacja pliku except.cpp

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

Wykres zależności załączania dla except.cpp:

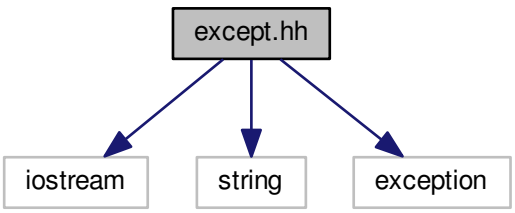


6.4 Dokumentacja pliku except.hh

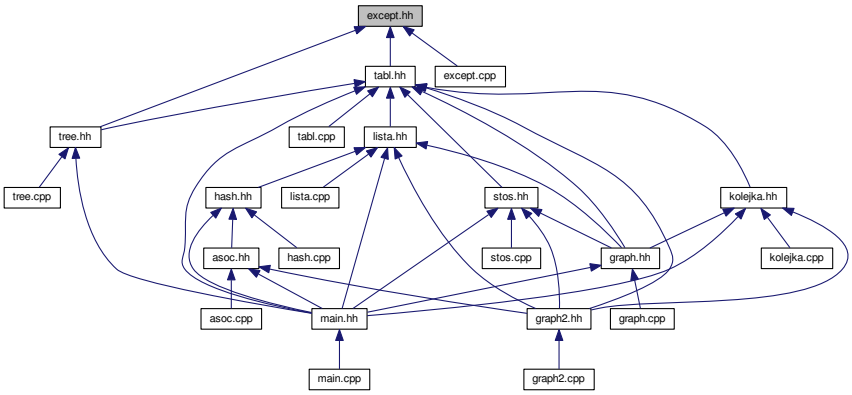
Plik zawiera definicje wyjątków.

```
#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 ExceptionBase

  Ogólny wyjątek.
- class CriticalException

  Wyjątek krytyczny, wymagający zamknięcia programu.
- class ContinueException

  Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać.

Funkcje

- template<class ExceptT >
  void what (ExceptT &except)

6.4.1 Opis szczegółowy

Plik zawiera definicje wyjątków.

6.4.2 Dokumentacja funkcji

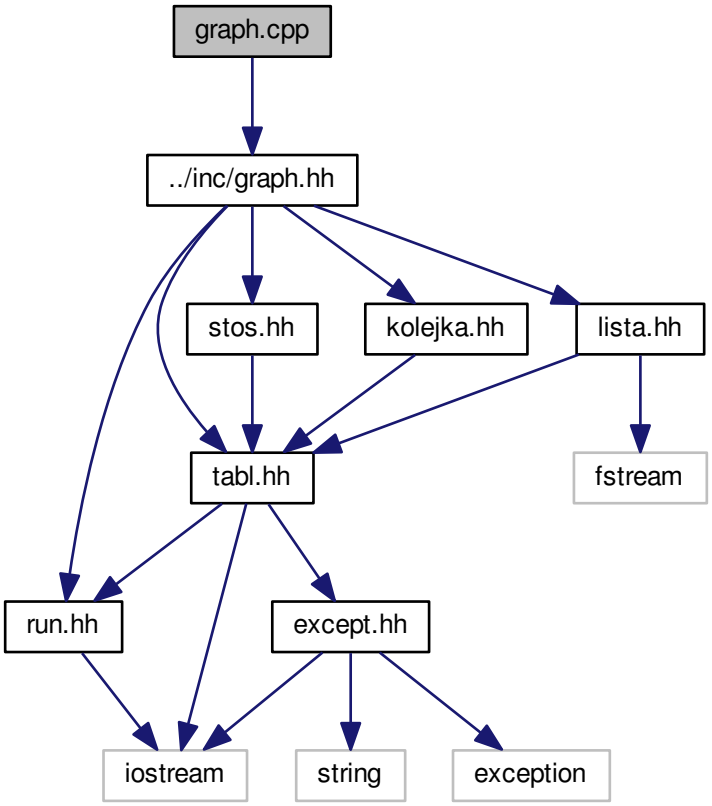
6.4.2.1 template<class ExceptT > void what (ExceptT & *except*)

Definicja w linii 72 pliku except.hh.

6.5 Dokumentacja pliku graph.cpp

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

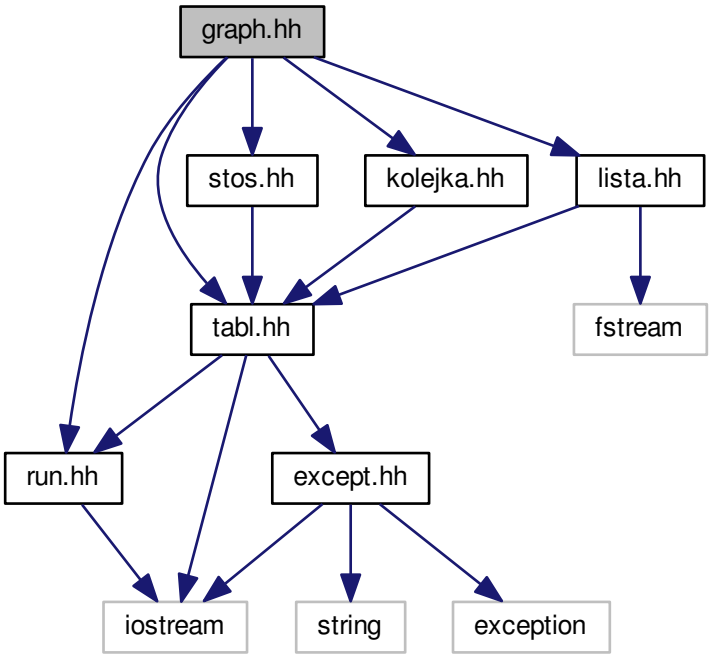
Wykres zależności załączania dla graph.cpp:



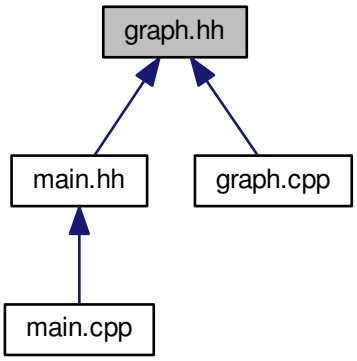
6.6 Dokumentacja pliku graph.hh

```
#include "tabl.hh"
#include "lista.hh"
#include "stos.hh"
#include "kolejka.hh"
#include "run.hh"
```

Wykres zależności załączania dla graph.hh:



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



Komponenty

- class IGraph

Interfejs grafu.

- class Graph

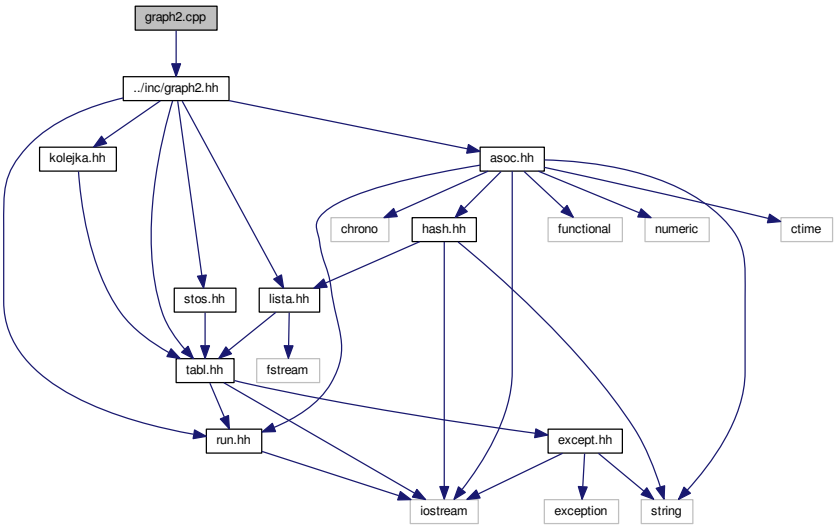
  Klasa implementująca inerfejs grafu.

- class test_graph_BFS
- class test_graph_DFS

6.7 Dokumentacja pliku graph2.cpp

```
#include "../inc/graph2.hh"
```

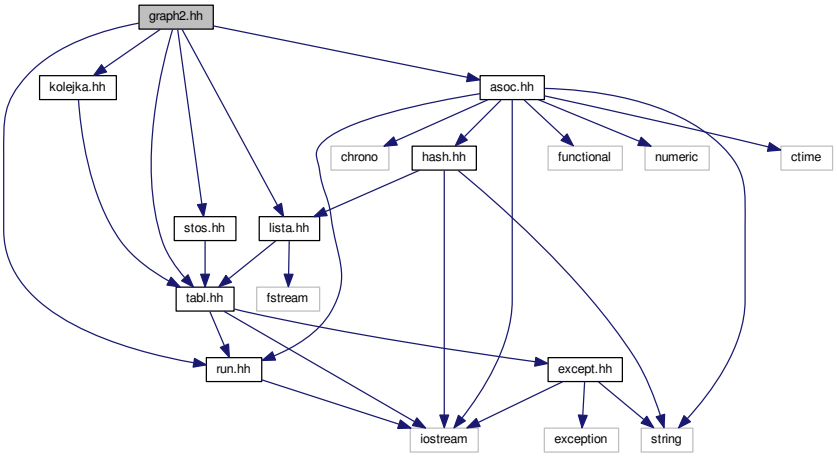
Wykres zależności załączania dla graph2.cpp:



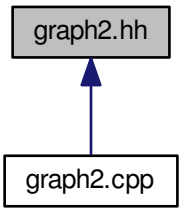
6.8 Dokumentacja pliku graph2.hh

```
#include "tabl.hh"
#include "lista.hh"
#include "stos.hh"
#include "kolejka.hh"
#include "run.hh"
#include "asoc.hh"
```

Wykres zależności załączania dla graph2.hh:



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



Komponenty

- class edgeInfo

  Klasa definiujaca typ, w ktorym zapisywana jest informacja o krawedzi grafu.
- class IGraph2

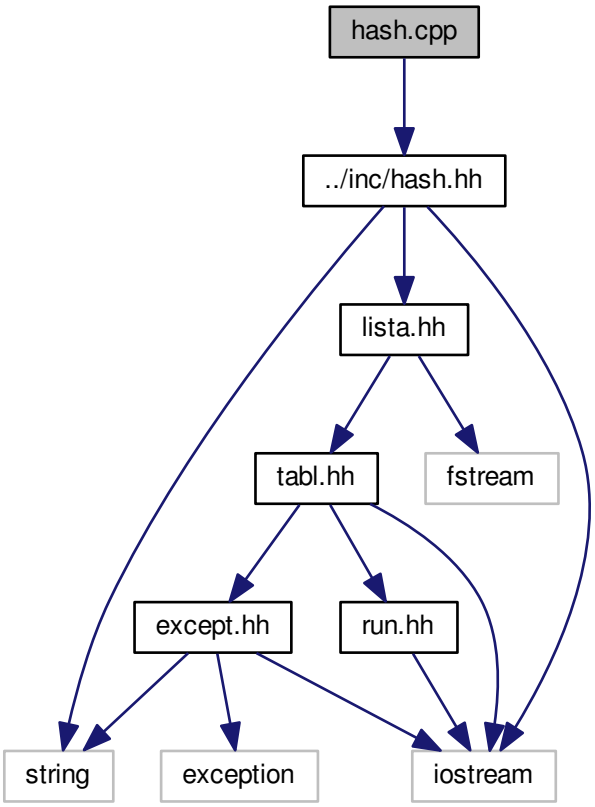
  Interfejs grafu2 z uwzględnieniem wag.
- class Graph2

  graf2 z uwzględnieniem wag

6.9 Dokumentacja pliku hash.cpp

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

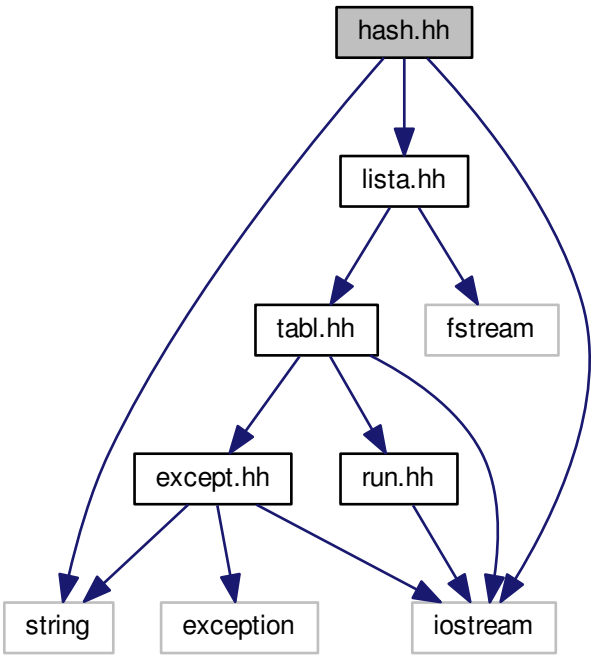
Wykres zależności załączania dla hash.cpp:



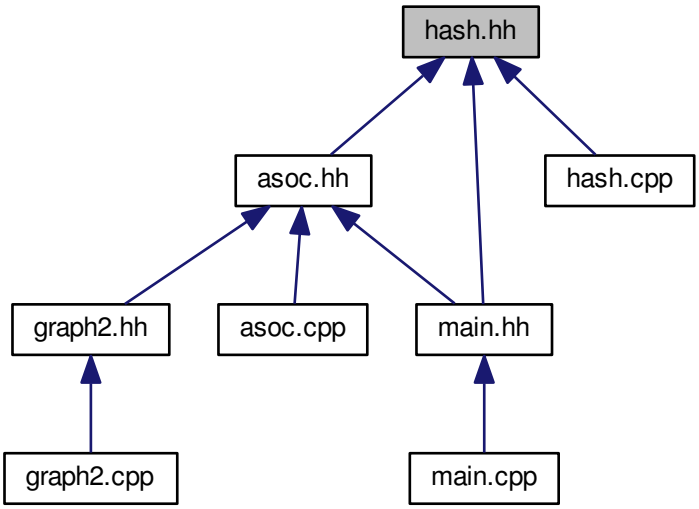
6.10 Dokumentacja pliku hash.hh

```
#include <iostream>
#include <string>
#include "lista.hh"
```

Wykres zależności załączania dla hash.hh:



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



Komponenty

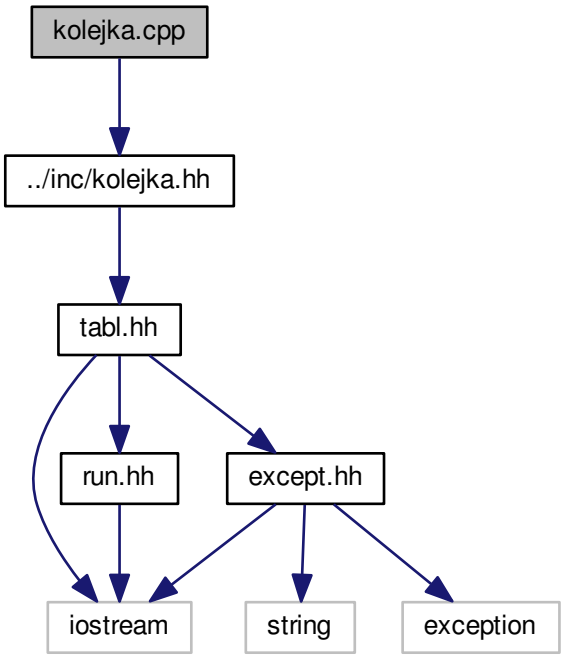
- class entry< T, T2 >

  Klasa definująca obiekt typu wpis.
- class IBucket< T, T2 >
- class Bucket< T, T2 >

6.11 Dokumentacja pliku kolejka.cpp

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

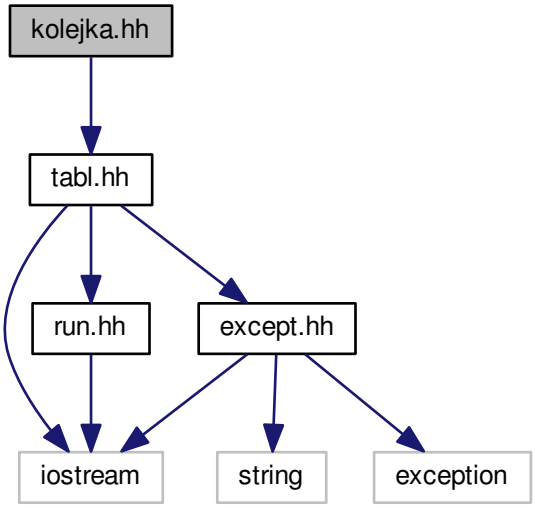
Wykres zależności załączania dla kolejka.cpp:



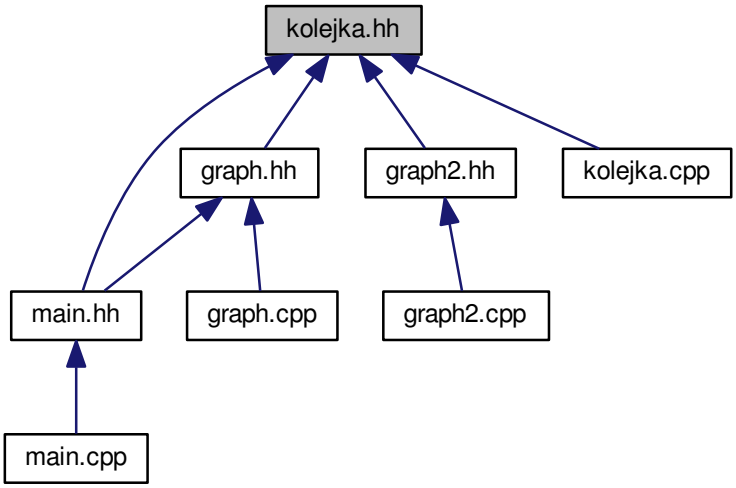
6.12 Dokumentacja pliku kolejka.hh

```
#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
- class node< T >

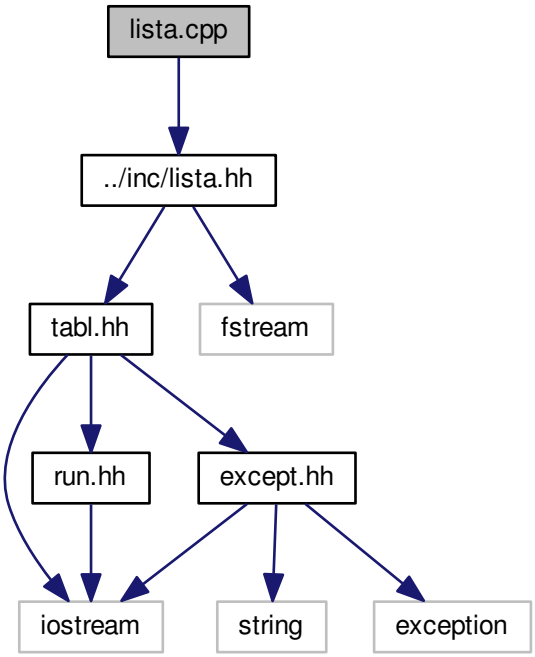
  Węzeł kolejki.
- class IQueue< T >
- class Queue< T >

  Kolejka oparta na węzłach.

6.13 Dokumentacja pliku lista.cpp

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

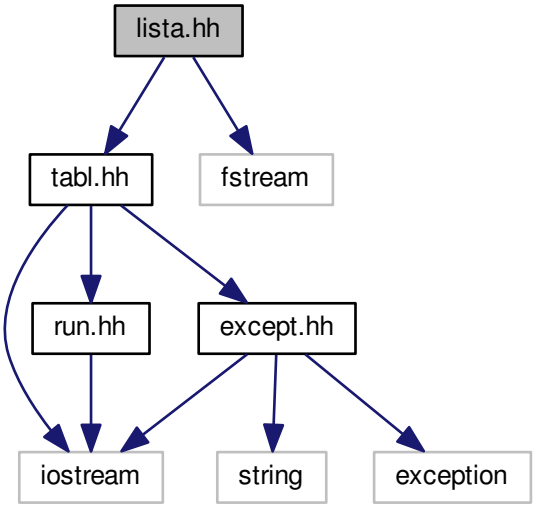
Wykres zależności załączania dla lista.cpp:



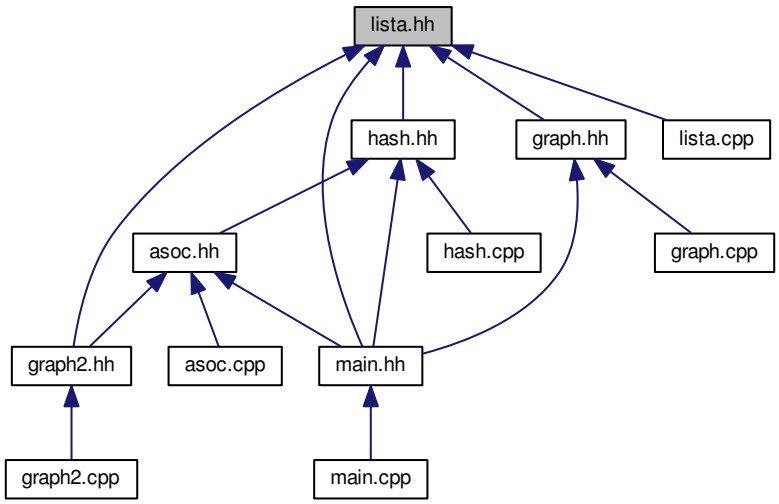
6.14 Dokumentacja pliku lista.hh

```
#include "tabl.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

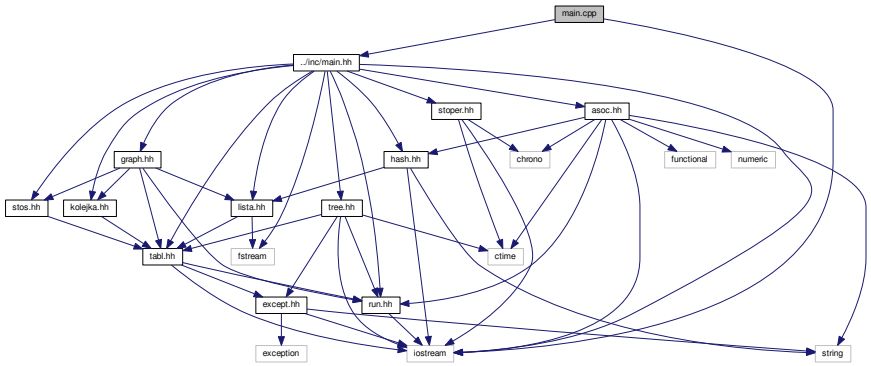
  Definiuje sposób testowania wypełniania listy.

6.15 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.

6.15.1 Opis szczegółowy

Główny plik programu.

6.15.2 Dokumentacja funkcji

6.15.2.1 void dumpToFile (string *nameOfFile*, unsigned int *testsize*, IStoper ∗ *stoper*)

Definicja w linii 423 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

6.15.2.2 int main (void)

Definicja w linii 163 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

6.15.2.3 void printOnscreen (unsigned *int*, IStoper *)

Wyświetla wynik na standardowym wyjściu.

Parametry

testSize	rozmiar testu
stoper	klasa stopera

Definicja w linii 437 pliku main.cpp.

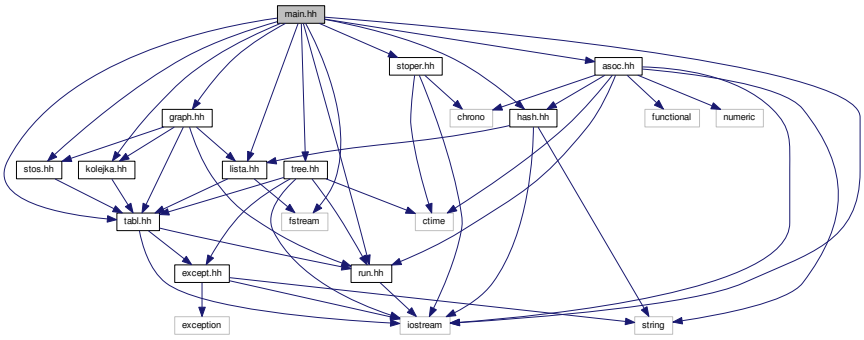
Oto graf wywołań dla tej funkcji:

6.16 Dokumentacja pliku main.hh

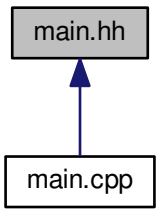
```
#include <iostream>
```

```
#include <fstream>
#include "stoper.hh"
#include "tabl.hh"
#include "run.hh"
#include "lista.hh"
#include "stos.hh"
#include "kolejka.hh"
#include "asoc.hh"
#include "hash.hh"
#include "tree.hh"
#include "graph.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.

6.16.1 Dokumentacja funkcji

6.16.1.1 void dumpToFile (std::string , unsigned *int,* IStoper ∗)

Zrzuca dane wynikowe do pliku.

Parametry

nameOfFile	nazwa pliku wynikowego
testsize	rozmiar testu
stoper	klasa stopera

6.16.1.2 void printOnscreen (unsigned *int,* IStoper *)

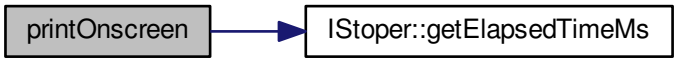
Wyświetla wynik na standardowym wyjściu.

Parametry

testSize	rozmiar testu
stoper	klasa stopera

Definicja w linii 437 pliku main.cpp.

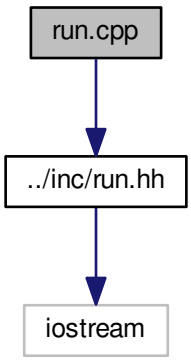
Oto graf wywołań dla tej funkcji:



6.17 Dokumentacja pliku run.cpp

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

Wykres zależności załączania dla run.cpp:

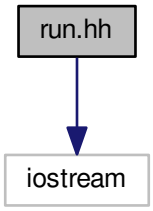


6.18 Dokumentacja pliku run.hh

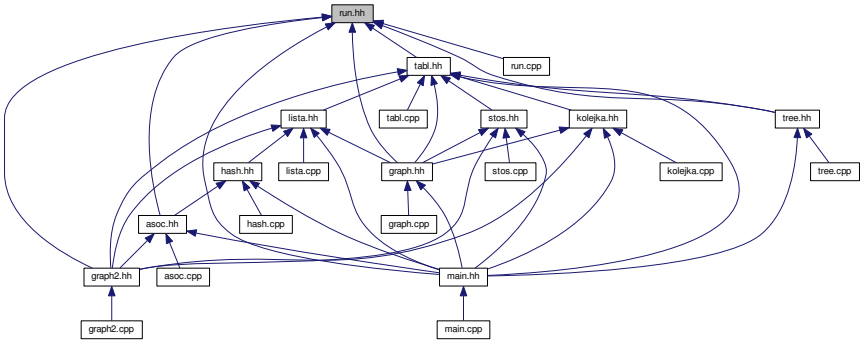
Plik definiuje interfejs IRunnable, ujednolicający klasy umożliwiające badanie algorytmów.

```
#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.

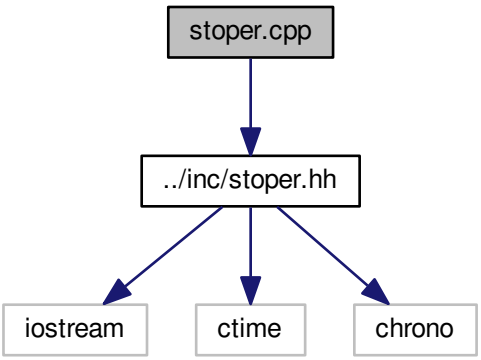
6.18.1 Opis szczegółowy

Plik definiuje interfejs IRunnable, ujednolicający klasy umożliwiające badanie algorytmów.

6.19 Dokumentacja pliku stoper.cpp

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

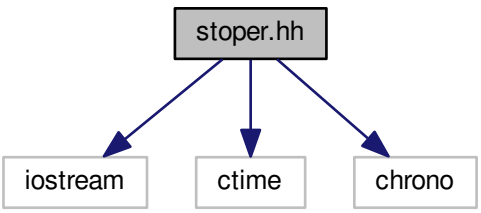
Wykres zależności załączania dla stoper.cpp:



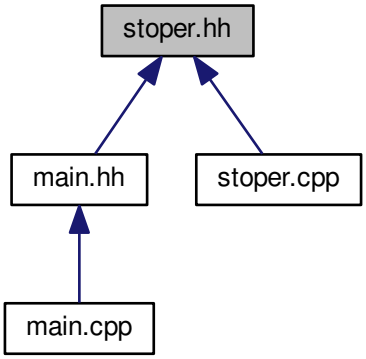
6.20 Dokumentacja pliku stoper.hh

```
#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.

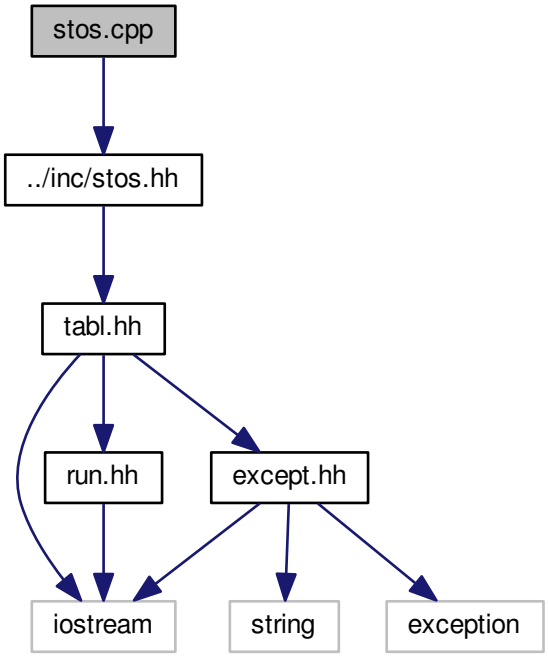
- class Stoper

  Klasa stoper implementująca interfejs IStoper.

6.21 Dokumentacja pliku stos.cpp

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

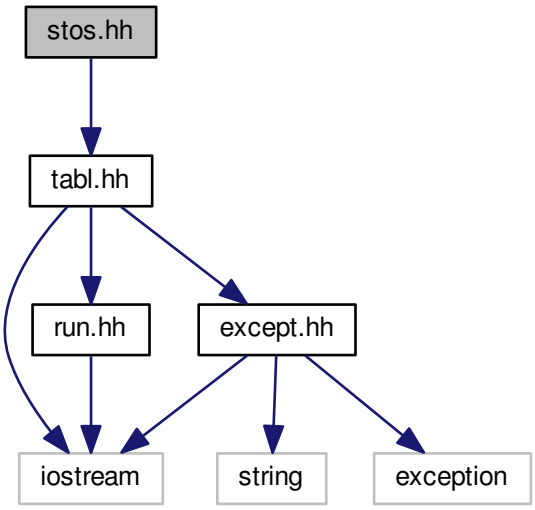
Wykres zależności załączania dla stos.cpp:



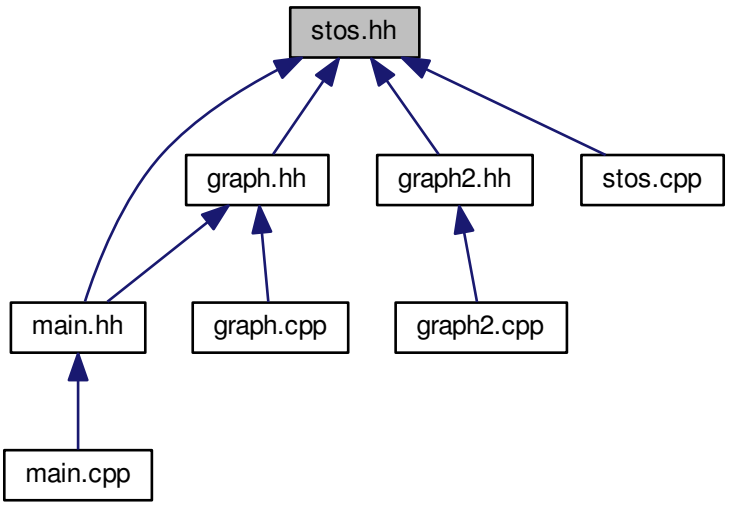
6.22 Dokumentacja pliku stos.hh

```
#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.

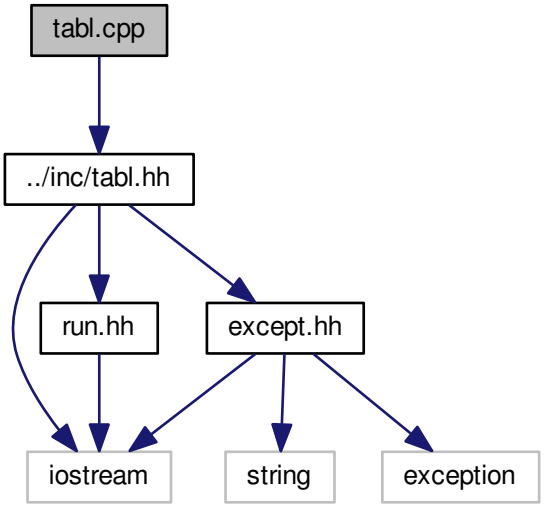
- class Stos< T >

  Klasa Stos.

6.23 Dokumentacja pliku tabl.cpp

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

Wykres zależności załączania dla tabl.cpp:

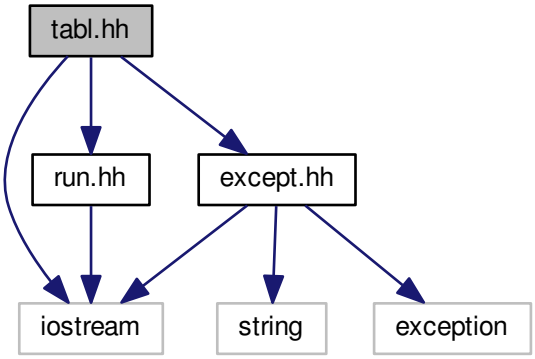


6.24 Dokumentacja pliku tabl.hh

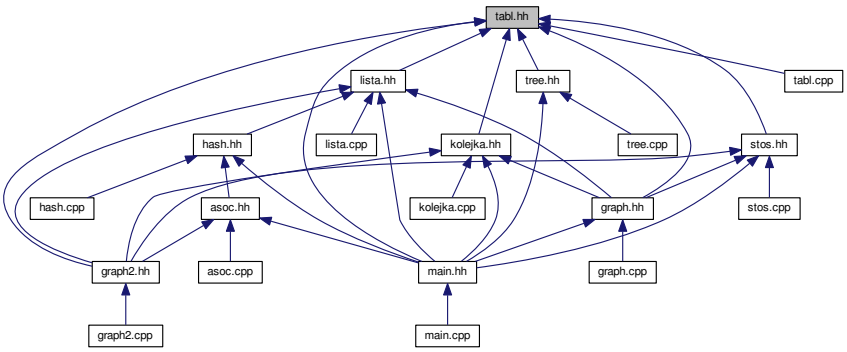
Definicja interfejsu Itabn, klasy tabn oraz klasy tabn_test.

```
#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

Definicje

- #define SIZE 10

6.24.1 Opis szczegółowy

Definicja interfejsu Itabn, klasy tabn oraz klasy tabn_test.

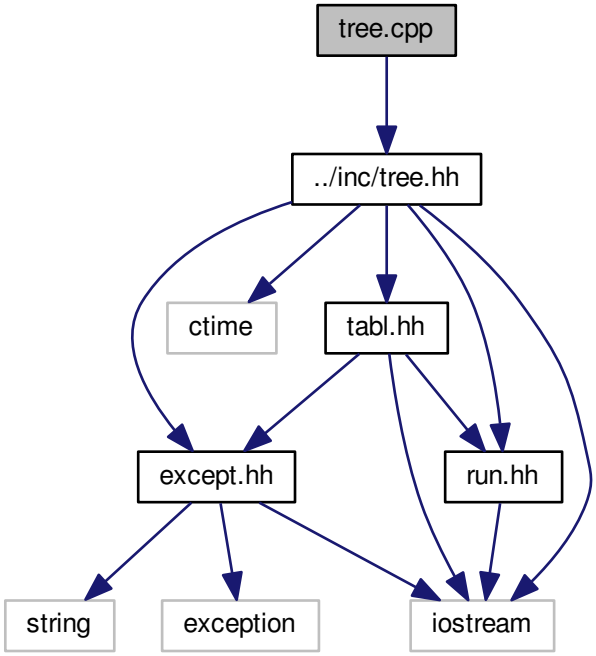
6.24.2 Dokumentacja definicji

6.24.2.1 #define SIZE 10

Definicja w linii 12 pliku tabl.hh.

6.25 Dokumentacja pliku tree.cpp

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



Funkcje

- std::ostream & operator<< (std::ostream &output, Colour col)
  Wyświetlanie koloru node'a.

6.25.1 Dokumentacja funkcji

6.25.1.1 std::ostream& operator<< (std::ostream & *output,* Colour *col*)

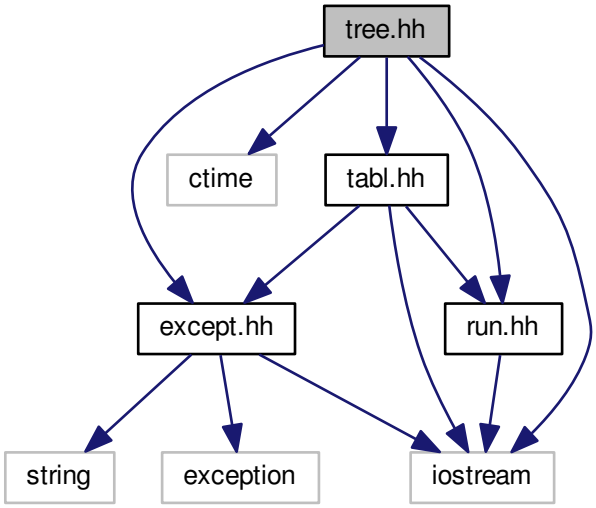
Wyświetlanie koloru node'a.

Definicja w linii 3 pliku tree.cpp.

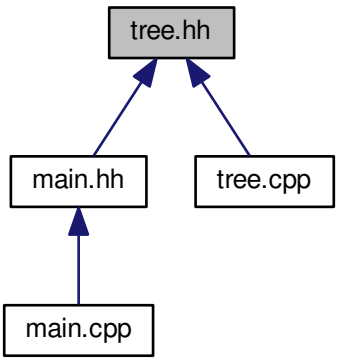
6.26 Dokumentacja pliku tree.hh

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

Wykres zależności załączania dla tree.hh:



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



Komponenty

- class nodeRB< T >
- class ITreeRB< T >

  Interfejs klasy drzewa czerwono-czarnego.
- class TreeRB< T >

  Klasa implementująca interfejs drzewa czerwono-czarnego.
- class tree_test

Wyliczenia

- enum Colour { red, black }

  Kolory node'a drzewa czerwono-czarnego.

Funkcje

- std::ostream & operator<< (std::ostream &, Colour)

  Wyświetlanie koloru node'a.

6.26.1 Dokumentacja typów wyliczanych

6.26.1.1 enum Colour

Kolory node'a drzewa czerwono-czarnego.

Wartości wyliczeń

red

black

Definicja w linii 12 pliku tree.hh.

6.26.2 Dokumentacja funkcji

6.26.2.1 std::ostream& operator<< (std::ostream & , Colour)

Wyświetlanie koloru node'a.

Definicja w linii 3 pliku tree.cpp.

Skorowidz