# PAMSI

0.1

Wygenerowano przez Doxygen 1.8.6

N, 24 maj 2015 16:42:49

# Spis treści

# Rozdział 1

# Indeks hierarchiczny

## 1.1 Hierarchia klas

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

# Rozdział 2

# Indeks klas

## 2.1 Lista klas

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

# Rozdział 3

# Indeks plików

## 3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

# Rozdział 4

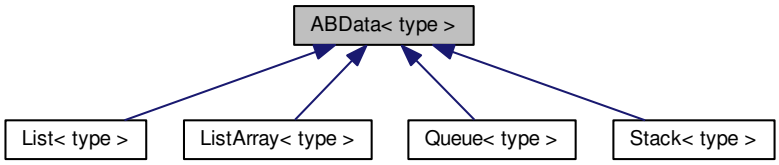
# Dokumentacja klas

## 4.1 Dokumentacja szablonu klasy ABData< type >

Modeluje klase wirtualna ABData, ktora jest interfejsem.

`#include <abdata.hh>`

Diagram dziedziczenia dla ABData< type >




### Metody publiczne

- virtual void push (const type elem)=0
- virtual void pop ()=0
- virtual unsigned int size ()=0

### 4.1.1 Opis szczegółowy

**template<class type>class ABData< type >**

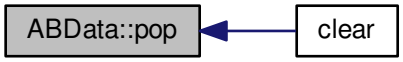
Definicja w linii 16 pliku abdata.hh.

### 4.1.2 Dokumentacja funkcji składowych

#### 4.1.2.1 template<class type> virtual void ABData< type >::pop ( ) [pure virtual]

Implementowany w ListArray< type >, List< type >, List< AssocData< typeKey, type > >, Stack< type >, Stack< Observer * > i Queue< type >.

Oto graf wywoływań tej funkcji:




#### 4.1.2.2 template<class type> virtual void ABData< type >::push ( const type *elem* ) [pure virtual]

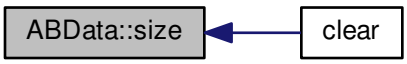
Implementowany w ListArray< type >, List< type >, List< AssocData< typeKey, type > >, Stack< type >, Stack< Observer * > i Queue< type >.

Oto graf wywoływań tej funkcji:


#### 4.1.2.3 template<class type> virtual unsigned int ABData< type >::size ( ) [pure virtual]

Implementowany w ListArray< type >, List< type >, List< AssocData< typeKey, type > >, Stack< type >, Stack< Observer * > i Queue< type >.

Oto graf wywoływań tej funkcji:




Dokumentacja dla tej klasy została wygenerowana z pliku:

- abdata.hh

## 4.2 Dokumentacja szablonu struktury AssocData< typeKey, type >

`#include <node.hh>`

### Metody publiczne

- AssocData ()
- AssocData (typeKey k)
- AssocData (typeKey k, type v)

### Atrybuty publiczne

- typeKey key
- type val

### 4.2.1 Opis szczegółowy

**template<typename typeKey, class type>struct AssocData< typeKey, type >**

Definicja w linii 6 pliku node.hh.

### 4.2.2 Dokumentacja konstruktora i destruktora

#### 4.2.2.1 template<typename typeKey, class type> AssocData< typeKey, type >::AssocData ( ) [inline]

Definicja w linii 10 pliku node.hh.

#### 4.2.2.2 template<typename typeKey, class type> AssocData< typeKey, type >::AssocData ( typeKey *k* ) [inline]

Definicja w linii 11 pliku node.hh.

#### 4.2.2.3 template<typename typeKey, class type> AssocData< typeKey, type >::AssocData ( typeKey *k*, type *v* ) [inline]

Definicja w linii 12 pliku node.hh.

### 4.2.3 Dokumentacja atrybutów składowych

#### 4.2.3.1 template<typename typeKey, class type> typeKey AssocData< typeKey, type >::key

Definicja w linii 7 pliku node.hh.

#### 4.2.3.2 template<typename typeKey, class type> type AssocData< typeKey, type >::val

Definicja w linii 8 pliku node.hh.

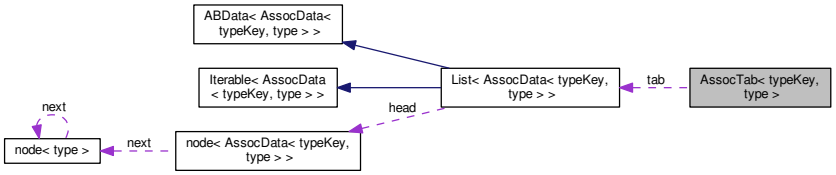
Dokumentacja dla tej struktury została wygenerowana z pliku:

- node.hh

## 4.3 Dokumentacja szablonu klasy AssocTab< typeKey, type >

```
#include <assoctab.hh>
```

Diagram współpracy dla AssocTab< typeKey, type >:




## Metody publiczne

- AssocTab ()

  *Konstruktor bezparametryczny.*
- AssocTab (unsigned int howmany)

  *Konstruktor parametryczny.*
- ~AssocTab ()

  *Destruktor.*
- void push (typeKey ikey, type toaddVal)

  *Metoda push.*
- void pop (typeKey toremoveKey)

  *Procedura pop.*
- int hash (typeKey tohashKey)

  *Metoda hash.*
- unsigned int size ()

  *Metoda size.*
- type & operator[] (const typeKey klucz)

  *Przeciazenie operatora [].*

## Atrybuty prywatne

- List< AssocData< typeKey, type > > * tab

  *Wskaznik na dynamicznie alokowana tablice z danymi.*
- int counter

  *Aktualna liczba elementow w tablicy.*

### 4.3.1 Opis szczegółowy

**template<class typeKey, class type>class AssocTab< typeKey, type >**

Definicja w linii 20 pliku assoctab.hh.

### 4.3.2 Dokumentacja konstruktora i destruktora

#### 4.3.2.1 template<class typeKey , class type > AssocTab< typeKey, type >::AssocTab ( ) [inline]

Tworzy tablice, ktora zawiera TAB list. Ustawia counter na TAB.

Definicja w linii 38 pliku assoctab.hh.

#### 4.3.2.2 template<class typeKey , class type > AssocTab< typeKey, type >::AssocTab ( unsigned int *howmany* ) [inline]

Tworzy tablice, ktora zawiera zadana ilosc list. Ustawia counter zgodnie z ta wartoscia

**Parametry**

| | | |
|---|---|---|
| in | *howmany* | Z ilu list ma skladac sie tablica asocjacyjna |

Definicja w linii 49 pliku assoctab.hh.

#### 4.3.2.3 template<class typeKey , class type > AssocTab< typeKey, type >::~AssocTab ( ) [inline]

Usuwa dynamicznie utworzona tablice danych oraz przypisuje wskaznikowi wartosc NULL.

Definicja w linii 60 pliku assoctab.hh.

### 4.3.3 Dokumentacja funkcji składowych

#### 4.3.3.1 template<class typeKey , class type > int AssocTab< typeKey, type >::hash ( typeKey *tohashKey* )

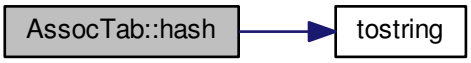
Dokonuje haszowania podanego klucza na wartosc liczbowa.

**Parametry**

| | | |
|---|---|---|
| in | *tohashKey* | Wartosc, ktora chcemy poddac haszowaniu. |

Definicja w linii 120 pliku assoctab.hh.

Oto graf wywołań dla tej funkcji:




#### 4.3.3.2 template<class typeKey , class type > type & AssocTab< typeKey, type >::operator[] ( const typeKey *klucz* )

Zwraca element odpowiadajacy podanemu kluczowi.

UWAGA! W przypadku proby odwolania sie do elementu o nieistniejacym kluczu, taki element zostanie utworzony z przypadkowa wartoscia!

**Zwraca**

Wartosc znajdujaca sie na miejscu o podanym kluczu

Definicja w linii 137 pliku assoctab.hh.

#### 4.3.3.3 template<class typeKey , class type > void AssocTab< typeKey, type >::pop ( typeKey *toremoveKey* )

Usuwa z tablicy element odpowiadajacy podanemu kluczowi.

**Parametry**

| | | |
|---|---|---|
| in | *toremoveKey* | Klucz odpowiadajacy elementowi, ktory chcemy usunac |

Definicja w linii 148 pliku assoctab.hh.

#### 4.3.3.4 template<class typeKey , class type > void AssocTab< typeKey, type >::push ( typeKey *ikey,* type *toaddVal* )

Dodaje element o podanej wartosci na miejsce odczytywane przez klucz.

**Parametry**

| | | |
|---|---|---|
| in | *ikey* | Klucz, ktorym chcemy sie posluzyc |
| in | *toaddVal* | Wartosc, ktora chcemy dodac do tablicy. |

Definicja w linii 114 pliku assoctab.hh.

#### 4.3.3.5 template<class typeKey , class type > unsigned int AssocTab< typeKey, type >::size ( ) [inline]

Daje informacje o rozmiarze tablicy (liczbie jej elementow).

**Zwraca**

Rozmiar tablicy (liczba jej elementow)

Definicja w linii 97 pliku assoctab.hh.

### 4.3.4 Dokumentacja atrybutów składowych

#### 4.3.4.1 template<class typeKey , class type > int AssocTab< typeKey, type >::counter [private]

Definicja w linii 30 pliku assoctab.hh.

#### 4.3.4.2 template<class typeKey , class type > List<AssocData<typeKey, type> >* AssocTab< typeKey, type >::tab [private]

Definicja w linii 25 pliku assoctab.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- assoctab.hh

## 4.4 Dokumentacja klasy Benchmark

Klasa Benchmark.

`#include <benchmark.hh>`

Diagram dziedziczenia dla Benchmark

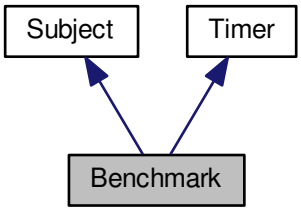
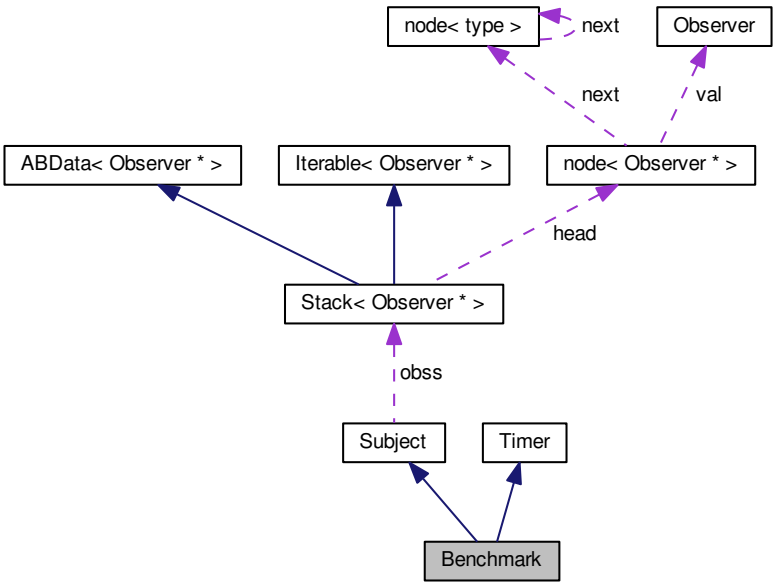



Diagram współpracy dla Benchmark:




## Metody publiczne

- Benchmark ()
- void notify ()

  *Wysyla powiadomienie do obserwatorow.*
- void stop_Ctimer ()

  *Konczy pomiar czasu.*
- void calc_mean ()

  *Oblicza srednia.*
- template<typename type >
  void runBenchmarkSort (void(∗f)(Iterable< type > &, int, int), Iterable< type > &container, int dataCount, int repeats)

*Wykonuje zadana ilosc testow zadanej funkcji sortujacej na zadanym obiekcie dla zadanej ilosc danych.*

- template<typename type >
  void runBenchmarkFillTree (void(Trees< type >::*f)(type), Trees< type > &tree, int dataCount, int repeats, char *dataFile)

  *Wykonuje zadana ilosc testow zadanej funkcji na zadanym obiekcie dla zadanej ilosc danych.*
- template<typename type >
  void runBenchmarkSearchTree (bool(Trees< type >::*f)(type), Trees< type > &tree, int dataCount, int repeats, char *dataFile)

  *Wykonuje zadana ilosc testow zadanej funkcji na zadanym obiekcie dla zadanej ilosc danych.*

### Atrybuty prywatne

- double total

  *total Zmienna przechowuje calkowity czas testow*
- double mean

  *mean Zmienna przechowuje sredni czas testow*
- int counter

  *counter Zmienna przechowuje licznik wykonanych testow*
- int amount

  *amountZmienna przechowuje ilosc danych, jaka aktualnie jest testowana*

### Dodatkowe Dziedziczone Składowe

## 4.4.1 Opis szczegółowy

Jest to klasa służąca do testowania programów.

Definicja w linii 20 pliku benchmark.hh.

## 4.4.2 Dokumentacja konstruktora i destruktora

#### 4.4.2.1 Benchmark::Benchmark ( ) [inline]

Definicja w linii 38 pliku benchmark.hh.

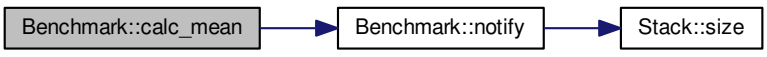
## 4.4.3 Dokumentacja funkcji składowych

#### 4.4.3.1 void Benchmark::calc_mean ( )

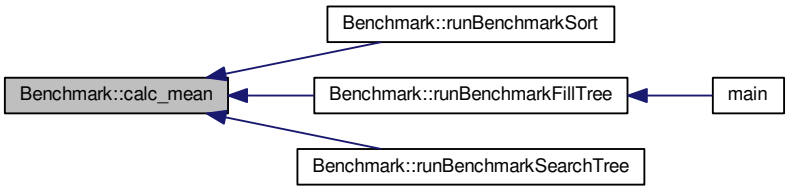
Dzieli sume pomiarow przez ich liczbe i zapisuje do zmiennej mean. Wysyla powiadomienie do obserwatorow.

Definicja w linii 19 pliku benchmark.cpp.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

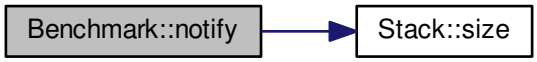



#### 4.4.3.2 void Benchmark::notify ( ) [virtual]

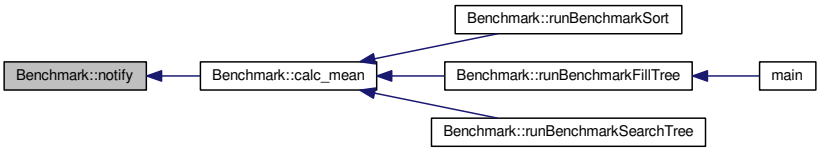
Implementuje Subject.

Definicja w linii 8 pliku benchmark.cpp.

Oto graf wywołań dla tej funkcji:




Oto graf wywoływań tej funkcji:




#### 4.4.3.3 template<typename type > void Benchmark::runBenchmarkFillTree ( void(Trees< type >::*)(type) *f,* Trees< type > & *tree,* int *dataCount,* int *repeats,* char * *dataFile* )

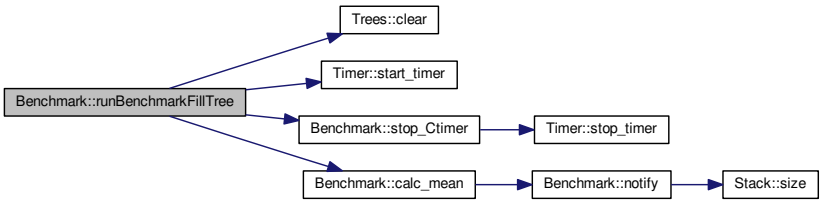
**Parametry**

| | | |
|---|---|---|
| in | *f | Zadawana funkcja wypelniajaca |

| | | |
|---|---|---|
| in | *container* | Drzewo, ktore chcemy testowac |
| in | *dataCount* | Ilosc danych |
| in | *repeats* | Ilosc testow |

Definicja w linii 98 pliku benchmark.hh.

Oto graf wywołań dla tej funkcji:




Oto graf wywoływań tej funkcji:

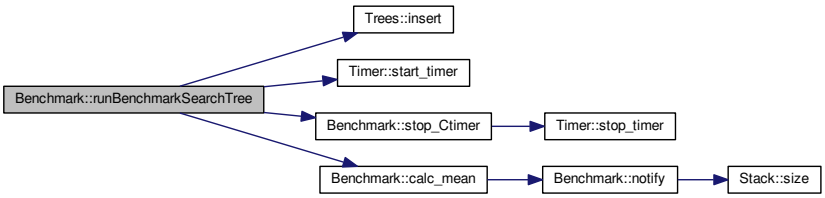

#### 4.4.3.4 template<typename type > void Benchmark::runBenchmarkSearchTree ( bool(Trees< type >::*)(type) *f*, Trees< type > & *tree*, int *dataCount*, int *repeats*, char * *dataFile* )

**Parametry**

| | | |
|---|---|---|
| in | *f | Zadawana funkcja szukajaca |
| in | *container* | Drzewo, ktore chcemy testowac |
| in | *dataCount* | Ilosc danych |
| in | *repeats* | Ilosc testow |

Definicja w linii 120 pliku benchmark.hh.

Oto graf wywołań dla tej funkcji:



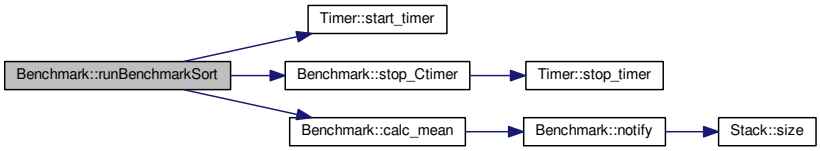
4.4.3.5 template<typename type > void Benchmark::runBenchmarkSort ( void(*)(Iterable< type > &, int, int) *f,* Iterable< type > & *container,* int *dataCount,* int *repeats* )

**Parametry**

| | | |
|---|---|---|
| in | *∗f* | Zadawana funkcja sortujaca |
| in | *container* | Stuktura, ktora chcemy posortowac |
| in | *dataCount* | Ilosc danych |
| in | *repeats* | Ilosc testow |

Definicja w linii 26 pliku benchmark.cpp.

Oto graf wywołań dla tej funkcji:

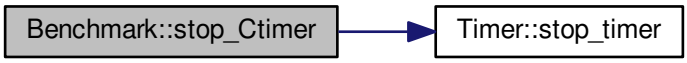



#### 4.4.3.6 void Benchmark::stop_Ctimer ( )

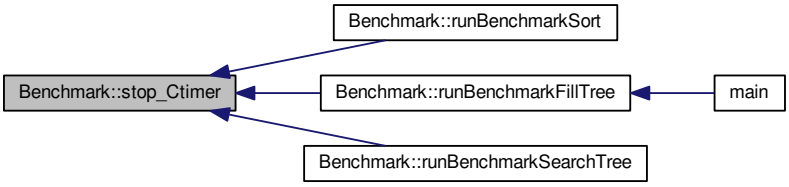
Zapisuje moment zakonczenia pomiaru do zmiennej end, oblicza zmierzony czas i zapisuje do zmiennej time, zwieksza counter o 1.

Definicja w linii 13 pliku benchmark.cpp.

Oto graf wywołań dla tej funkcji:




Oto graf wywoływań tej funkcji:




### 4.4.4 Dokumentacja atrybutów składowych

#### 4.4.4.1 int Benchmark::amount [private]

Definicja w linii 36 pliku benchmark.hh.

#### 4.4.4.2 int Benchmark::counter [private]

Definicja w linii 32 pliku benchmark.hh.

#### 4.4.4.3 double Benchmark::mean [private]

Definicja w linii 28 pliku benchmark.hh.

#### 4.4.4.4 double Benchmark::total [private]

Definicja w linii 24 pliku benchmark.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

- benchmark.hh
- benchmark.cpp

## 4.5 Dokumentacja szablonu klasy BinaryTree< type >

Klasa BinaryTree - drzewo binarne.

`#include <binarytree.hh>`

Diagram dziedziczenia dla BinaryTree< type >

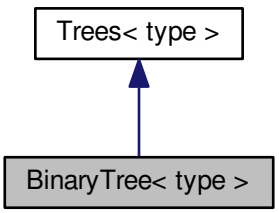
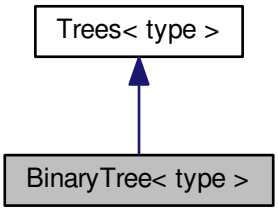


Diagram współpracy dla BinaryTree< type >:




### Metody publiczne

- BinaryTree ()

  *Konstruktor bezparametryczny.*
- ~BinaryTree ()

  *Destruktor.*
- void insert (const type elem)

  *Metoda insert.*
- bool remove (const type elem)

  *Metoda remove.*
- bool search (const type elem)

  *Metoda search.*
- void print ()

  *Metoda print.*
- int height ()

  *Procedura height.*
- void clear ()

  *Procedura clear.*

### Metody prywatne

- void insert (const type elem, treenode< type > *leaf)

  *Metoda prywatna insert.*
- void print (treenode< type > *root)

  *Metoda prywatna print.*
- treenode< type > * remove (treenode< type > *node, const type elem)

  *Metoda prywatna insert.*
- treenode< type > * findMin (treenode< type > *node)

  *Metoda findMin.*
- bool rotateLeft (treenode< type > *node)

  *Metoda rotateLeft.*
- bool rotateRight (treenode< type > *node)

  *Metoda rotateRight.*
- void balance (treenode< type > *root)

  *Procedura balance.*

- void deleteTree (treenode< type > *node)

  *Metoda deleteTree.*
- int height (treenode< type > *node)

  *Metoda prywatna height.*

### Atrybuty prywatne

- int numberOfNodes

  *numberOfNodes - ilosc wezlow w drzewie*
- treenode< type > * root

  *Root - korzen drzewa - wskaznik na pierwszy element drzewa.*

## 4.5.1 Opis szczegółowy

**template<class type>class BinaryTree< type >**

Definicja w linii 50 pliku binarytree.hh.

## 4.5.2 Dokumentacja konstruktora i destruktora

### 4.5.2.1 template<class type > BinaryTree< type >::BinaryTree ( ) [inline]

Definicja w linii 136 pliku binarytree.hh.

### 4.5.2.2 template<class type > BinaryTree< type >::~BinaryTree ( ) [inline]

Definicja w linii 143 pliku binarytree.hh.

Oto graf wywołań dla tej funkcji:


## 4.5.3 Dokumentacja funkcji składowych

### 4.5.3.1 template<class type > void BinaryTree< type >::balance ( treenode< type > * *root* ) [private]

Balansuje drzewo

**Parametry**

| | | |
|---|---|---|
| in | *root* | Korzen drzewa, ktore chcemy balansowac |

Definicja w linii 355 pliku binarytree.hh.

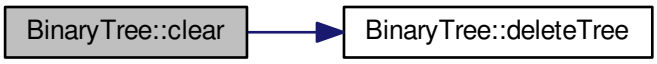
#### 4.5.3.2 template<class type > void BinaryTree< type >::clear ( ) [inline], [virtual]

Czysci drzewo, usuwa wszystkie jego wezly

Implementuje Trees< type >.

Definicja w linii 196 pliku binarytree.hh.

Oto graf wywołań dla tej funkcji:




#### 4.5.3.3 template<class type > void BinaryTree< type >::deleteTree ( treenode< type > * *node* ) [private]

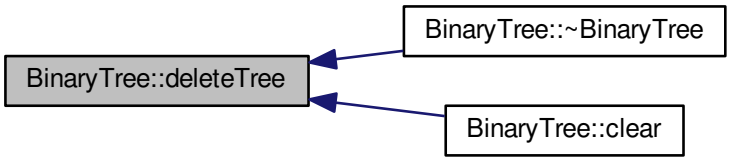
Usuwa drzewa zaczynajace sie w podanym wezle

**Parametry**

| | | |
|---|---|---|
| in | *node* | Korzen drzewa, ktore chcemy usunac |

Definicja w linii 369 pliku binarytree.hh.

Oto graf wywoływań tej funkcji:




#### 4.5.3.4 template<class type > treenode< type > * BinaryTree< type >::findMin ( treenode< type > * *node* ) [private]

Metoda pomocniczna znajdujaca najmniejsza wartosc w podanym poddrzewie

Definicja w linii 318 pliku binarytree.hh.

#### 4.5.3.5 template<class type > int BinaryTree< type >::height ( treenode< type > * *node* ) [private]

Znajduje wysokosc poddrzewa rozpoczynajacego sie w zadanym korzeniu

**Parametry**

| | | |
|---|---|---|
| in | *node* | Korzen drzewa, ktorego wysokosc chcemy znalezc |

**Zwraca**

Wysokosc drzewa

Definicja w linii 378 pliku binarytree.hh.

#### 4.5.3.6 template<class type > int BinaryTree< type >::height ( )

**Zwraca**

Zwraca wysokosc calego drzewa

Definicja w linii 385 pliku binarytree.hh.

#### 4.5.3.7 template<class type > void BinaryTree< type >::insert ( const type *elem*, treenode< type > * *leaf* ) [private]

Metoda pomocniczna do wywolan rekurencyjnych

Definicja w linii 200 pliku binarytree.hh.

#### 4.5.3.8 template<class type > void BinaryTree< type >::insert ( const type *elem* ) [virtual]

Sluzy do wstawienia zadanego elementu w odpowniednie miejsce drzewa

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc do wstawienia |

Implementuje Trees< type >.

Definicja w linii 218 pliku binarytree.hh.

#### 4.5.3.9 template<class type > void BinaryTree< type >::print ( treenode< type > * *root* ) [private]

Metoda pomocniczna do wywolan rekurencyjnych

Definicja w linii 309 pliku binarytree.hh.

#### 4.5.3.10 template<class type > void BinaryTree< type >::print ( )

Sluzy do wyswietlenia elementow drzewa na standardowym strumieniu wyjsciowym. Elementy wyswietlane sa w nastepujacej kolejności: korzen, lewe poddrzewo, prawe poddrzewo

Definicja w linii 300 pliku binarytree.hh.

#### 4.5.3.11 template<class type > treenode< type > * BinaryTree< type >::remove ( treenode< type > * *node*, const type *elem* ) [private]

Metoda pomocniczna do wywolan rekurencyjnych

Definicja w linii 257 pliku binarytree.hh.

#### 4.5.3.12 template<class type > bool BinaryTree< type >::remove ( const type *elem* ) [virtual]

Sluzy do usuniecia z drzewa elementu o zadanej wartosci

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc elementu do usuniecia |

**Zwracane wartości**

| | |
|---|---|
| *TRUE* | Usunieto elemeny]t |
| *FALSE* | Brak elemetu o zadanej wartosci |

Implementuje Trees< type >.

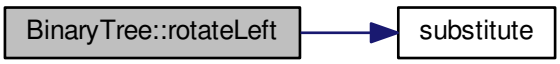
Definicja w linii 229 pliku binarytree.hh.

#### 4.5.3.13 template<class type > bool BinaryTree< type >::rotateLeft ( treenode< type > * *node* ) [private]

Metoda pomocnicza pomocna przy balansowaniu drzewa

Definicja w linii 326 pliku binarytree.hh.

Oto graf wywołań dla tej funkcji:




#### 4.5.3.14 template<class type > bool BinaryTree< type >::rotateRight ( treenode< type > * *node* ) [private]

Metoda pomocnicza pomocna przy balansowaniu drzewa

**Parametry**

| | | |
|---|---|---|
| in | *node* | |

Definicja w linii 340 pliku binarytree.hh.

Oto graf wywołań dla tej funkcji:


#### 4.5.3.15 template<class type > bool BinaryTree< type >::search ( const type *elem* ) [virtual]

Sluzy do sprawdzenia, czy w drzewie znajduje sie element o zadanej wartosci

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc elementu do znalezienia |

**Zwracane wartości**

| | |
|---|---|
| *TRUE* | Element znajduje sie w drzewie |
| *FALSE* | Brak elementu o zadanej wartosci |

Implementuje Trees< type >.

Definicja w linii 285 pliku binarytree.hh.

### 4.5.4 Dokumentacja atrybutów składowych

#### 4.5.4.1 template<class type > int BinaryTree< type >::numberOfNodes [private]

Definicja w linii 55 pliku binarytree.hh.

#### 4.5.4.2 template<class type > treenode<type>* BinaryTree< type >::root [private]

Definicja w linii 59 pliku binarytree.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

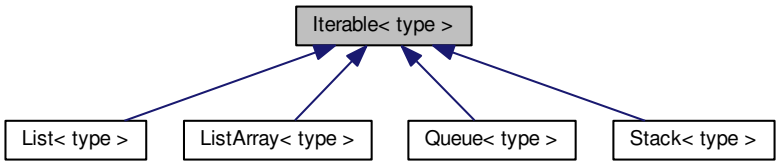
- binarytree.hh

## 4.6 Dokumentacja szablonu klasy Iterable< type >

Modeluje klase wirtualna Iterable.

`#include <iterable.hh>`

Diagram dziedziczenia dla Iterable< type >




### Metody publiczne

- virtual type & operator[] (const unsigned int index)=0

### 4.6.1 Opis szczegółowy

template<class type>class Iterable< type >

Jest to interfejs dla klas z przeciazonym operatorem indeksowania [].

Definicja w linii 15 pliku iterable.hh.

### 4.6.2 Dokumentacja funkcji składowych

#### 4.6.2.1 template<class type> virtual type& Iterable< type >::operator[] ( const unsigned int *index* ) [pure virtual]

Implementowany w ListArray< type >, List< type >, List< AssocData< typeKey, type > >, Stack< type >, Stack< Observer * > i Queue< type >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- iterable.hh

## 4.7 Dokumentacja szablonu klasy List< type >

`#include <list.hh>`

Diagram dziedziczenia dla List< type >

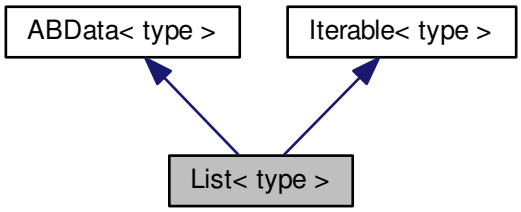
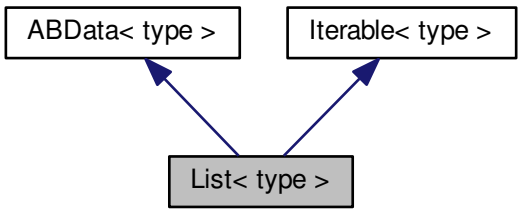



Diagram współpracy dla List< type >:




### Metody publiczne

- List ()

*Konstruktor bezparametryczny.*

- void push (const type elem)

  *Metoda push.*
- void pop ()

  *Procedura pop.*
- void pop (unsigned int index)

  *Procedura pop.*
- unsigned int size ()

  *Metoda size.*
- type & operator[] (const unsigned int index)

  *Przeciazenie operatora [].*

### Atrybuty prywatne

- node< type > * head

  *Wskaznik head.*
- int iterator

  *Iterator.*

## 4.7.1 Opis szczegółowy

**template<class type>class List< type >**

Definicja w linii 15 pliku list.hh.

## 4.7.2 Dokumentacja konstruktora i destruktora

### 4.7.2.1 template<class type> List< type >::List ( ) [inline]

Ustawia poczatek listy na NULL

Definicja w linii 34 pliku list.hh.

## 4.7.3 Dokumentacja funkcji składowych

### 4.7.3.1 template<class type > type & List< type >::operator[] ( const unsigned int *index* ) [virtual]

Zwraca element o podanym indeksie (indeksowanie zaczyna się od 0) W przypadku odwolania sie poza zakres, program przerywany jest z bledem 1.

**Zwraca**

Wartosc znajdujaca sie na miejscu o podanym indeksie

Implementuje Iterable< type >.

Definicja w linii 144 pliku list.hh.

### 4.7.3.2 template<class type > void List< type >::pop ( ) [virtual]

Usuwa pierwszy element listy.

Implementuje ABData< type >.

Definicja w linii 97 pliku list.hh.

#### 4.7.3.3 template<class type > void List< type >::pop ( unsigned int *index* )

Usuwa element o wybranym indeksie z listy.

Definicja w linii 109 pliku list.hh.

#### 4.7.3.4 template<class type> void List< type >::push ( const type *elem* ) [virtual]

Dodaje podana wartosc na poczatek listy.

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc, ktora chcemy dodac na poczatek listy. |

Implementuje ABData< type >.

Definicja w linii 87 pliku list.hh.

#### 4.7.3.5 template<class type > unsigned int List< type >::size ( ) [virtual]

Daje informacje o rozmiarze listy (liczbie jej elementow).

**Zwraca**

Rozmiar listy (liczba jej elementow)

Implementuje ABData< type >.

Definicja w linii 139 pliku list.hh.

### 4.7.4 Dokumentacja atrybutów składowych

#### 4.7.4.1 template<class type> node<type>* List< type >::head [private]

Wskaznik na pierwszy element listy

Definicja w linii 21 pliku list.hh.

#### 4.7.4.2 template<class type> int List< type >::iterator [private]

Przechowuje informacje o liczbie elementow znajdujacych sie na liscie

Definicja w linii 27 pliku list.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- list.hh

## 4.8 Dokumentacja szablonu klasy ListArray< type >

```
#include <listarray.hh>
```

Diagram dziedziczenia dla ListArray< type >

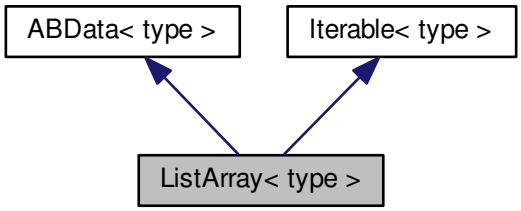
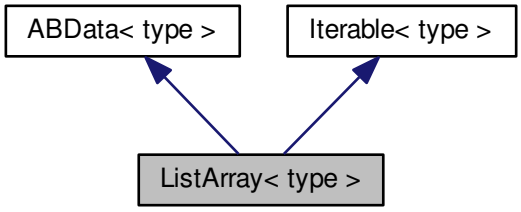



Diagram współpracy dla ListArray< type >:




## Metody publiczne

- ListArray ()

  *Konstruktor bezparametryczny.*
- ~ListArray ()

  *Destruktor.*
- void push (const type elem)

  *Metoda push.*
- void pop ()

  *Procedura pop.*
- unsigned int size ()

  *Metoda size.*
- type & operator[] (const unsigned int index)

  *Przeciazenie operatora [].*

## Atrybuty prywatne

- int counter

  *Counter.*
- int iterator

*Iterator.*

- type * tab

  *Wskaznik na dynamicznie alokowana tablice z danymi.*

### 4.8.1 Opis szczegółowy

**template<class type>class ListArray< type >**

Definicja w linii 13 pliku listarray.hh.

### 4.8.2 Dokumentacja konstruktora i destruktora

#### 4.8.2.1 template<class type > ListArray< type >::ListArray ( ) [inline]

Ustawia wskaznik na tablice na NULL, iterator na 0

Definicja w linii 36 pliku listarray.hh.

#### 4.8.2.2 template<class type > ListArray< type >::~ListArray ( ) [inline]

Usuwa dynamicznie utworzona tablice danych

Definicja w linii 47 pliku listarray.hh.

### 4.8.3 Dokumentacja funkcji składowych

#### 4.8.3.1 template<class type > type & ListArray< type >::operator[] ( const unsigned int *index* ) [virtual]

Zwraca element o podanym indeksie (indeksowanie zaczyna się od 0) W przypadku odwolania sie poza zakres, program przerywany jest z bledem 1.

**Zwraca**

Wartosc znajdujaca sie na miejscu o podanym indeksie

Implementuje Iterable< type >.

Definicja w linii 132 pliku listarray.hh.

#### 4.8.3.2 template<class type > void ListArray< type >::pop ( ) [virtual]

Usuwa ostatni element listy.

Implementuje ABData< type >.

Definicja w linii 110 pliku listarray.hh.

#### 4.8.3.3 template<class type > void ListArray< type >::push ( const type *elem* ) [virtual]

Dodaje podana wartosc na koniec listy.

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc, ktora chcemy dodac na koniec listy. |

Implementuje ABData< type >.

Definicja w linii 86 pliku listarray.hh.

#### 4.8.3.4 template<class type > unsigned int ListArray< type >::size ( ) [virtual]

Daje informacje o rozmiarze listy (liczbie jej elementow).

**Zwraca**

Rozmiar listy (liczba jej elementow)

Implementuje ABData< type >.

Definicja w linii 127 pliku listarray.hh.

### 4.8.4 Dokumentacja atrybutów składowych

#### 4.8.4.1 template<class type > int ListArray< type >::counter [private]

Przechowuje informacje o liczbie elementow znajdujacych sie na liscie

Definicja w linii 19 pliku listarray.hh.

#### 4.8.4.2 template<class type > int ListArray< type >::iterator [private]

Przechowuje informacje o aktualnej pozycji ostatniego elementu w tablicy

Definicja w linii 25 pliku listarray.hh.

#### 4.8.4.3 template<class type > type* ListArray< type >::tab [private]

Definicja w linii 29 pliku listarray.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- listarray.hh

## 4.9 Dokumentacja szablonu struktury node< type >

`#include <node.hh>`

Diagram współpracy dla node< type >:

### Metody publiczne

- node ()

  *Konstruktor bezparametryczny.*
- node (type elem)

  *Konstruktor parametryczny.*

### Atrybuty publiczne

- type val

  *Przechowywane dane.*
- node ∗ next

  *Wskaznik na nastepny node.*

### 4.9.1 Opis szczegółowy

**template<typename type>struct node< type >**

Definicja w linii 24 pliku node.hh.

### 4.9.2 Dokumentacja konstruktora i destruktora

#### 4.9.2.1 template<typename type> node< type >::node ( ) [inline]

Definicja w linii 36 pliku node.hh.

#### 4.9.2.2 template<typename type> node< type >::node ( type *elem* ) [inline]

Definicja w linii 42 pliku node.hh.

### 4.9.3 Dokumentacja atrybutów składowych

#### 4.9.3.1 template<typename type> node∗ node< type >::next

Definicja w linii 32 pliku node.hh.

#### 4.9.3.2 template<typename type> type node< type >::val

Definicja w linii 28 pliku node.hh.

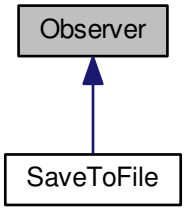
Dokumentacja dla tej struktury została wygenerowana z pliku:

- node.hh

## 4.10 Dokumentacja klasy Observer

`#include <observer.hh>`

Diagram dziedziczenia dla Observer




### Metody publiczne

- virtual void update (int dataNumber, double mean)=0

### 4.10.1 Opis szczegółowy

Definicja w linii 9 pliku observer.hh.

### 4.10.2 Dokumentacja funkcji składowych

#### 4.10.2.1 virtual void Observer::update ( int *dataNumber,* double *mean* ) [pure virtual]

Implementowany w SaveToFile.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- observer.hh

## 4.11 Dokumentacja szablonu klasy Queue< type >

```
#include <queue.hh>
```

Diagram dziedziczenia dla Queue< type >

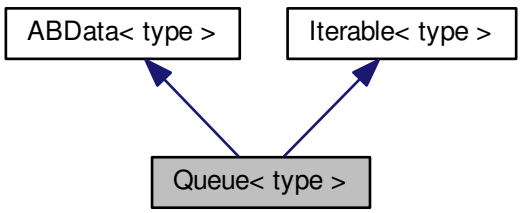
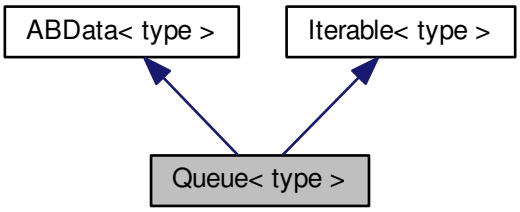


Diagram współpracy dla Queue< type >:




### Metody publiczne

- Queue ()

  *Konstruktor bezparametryczny.*
- void push (const type elem)

  *Metoda push.*
- void pop ()

  *Procedura pop.*
- unsigned int size ()

  *Metoda size.*
- type & operator[] (const unsigned int index)

  *Przeciazenie operatora [].*
- void display ()

### Atrybuty prywatne

- node< type > * head

  *Wskaznik head.*
- int iterator

  *Iterator.*

## 4.11.1 Opis szczegółowy

**template<class type>class Queue< type >**

Definicja w linii 11 pliku queue.hh.

## 4.11.2 Dokumentacja konstruktora i destruktora

### 4.11.2.1 template<class type > Queue< type >::Queue ( ) [inline]

Ustawia poczatek listy na NULL

Definicja w linii 31 pliku queue.hh.

### 4.11.3 Dokumentacja funkcji składowych

#### 4.11.3.1 template<class type > void Queue< type >::display ( ) [inline]

Definicja w linii 71 pliku queue.hh.

#### 4.11.3.2 template<class type > type & Queue< type >::operator[] ( const unsigned int *index* ) [virtual]

Zwraca element o podanym indeksie (indeksowanie zaczyna się od 0) W przypadku odwolania sie poza zakres, program przerywany jest z bledem 1.

**Zwraca**

Wartosc znajdujaca sie na miejscu o podanym indeksie

Implementuje Iterable< type >.

Definicja w linii 115 pliku queue.hh.

#### 4.11.3.3 template<class type > void Queue< type >::pop ( ) [virtual]

Usuwa pierwszy element stosu.

Implementuje ABData< type >.

Definicja w linii 98 pliku queue.hh.

#### 4.11.3.4 template<class type > void Queue< type >::push ( const type *elem* ) [virtual]

Dodaje podana wartosc na poczatek listy.

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc, ktora chcemy dodac na poczatek listy. |

Implementuje ABData< type >.

Definicja w linii 82 pliku queue.hh.

#### 4.11.3.5 template<class type > unsigned int Queue< type >::size ( ) [virtual]

Daje informacje o rozmiarze stosu (liczbie jego elementow).

**Zwraca**

Rozmiar stosu (liczba jego elementow)

Implementuje ABData< type >.

Definicja w linii 110 pliku queue.hh.

### 4.11.4 Dokumentacja atrybutów składowych

#### 4.11.4.1 template<class type > node<type>* Queue< type >::head [private]

Wskaznik na pierwszy element kolejki

Definicja w linii 17 pliku queue.hh.

#### 4.11.4.2 template<class type > int Queue< type >::iterator [private]

Przechowuje informacje o liczbie elementow znajdujacych sie w kolejce

Definicja w linii 23 pliku queue.hh.

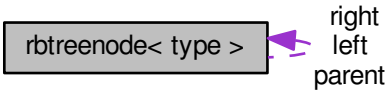
Dokumentacja dla tej klasy została wygenerowana z pliku:

- queue.hh

## 4.12 Dokumentacja szablonu struktury rbtreenode< type >

`#include <redblacktree.hh>`

Diagram współpracy dla rbtreenode< type >:




### Metody publiczne

- rbtreenode (type elem)

  *Konstruktor parametryczny.*

### Atrybuty publiczne

- type val

  *Przechowywana wartosc.*
- rbtreenode * left

  *Wskaznik na lewy wezel.*
- rbtreenode * right

  *Wskaznik na prawy wezel.*
- rbtreenode * parent

  *Wskaznik na rodzica.*
- char color

  *Kolor wezla.*

### 4.12.1 Opis szczegółowy

template<typename type>struct rbtreenode< type >

Definicja w linii 8 pliku redblacktree.hh.

### 4.12.2 Dokumentacja konstruktora i destruktora

#### 4.12.2.1 template<typename type> rbtreenode< type >::rbtreenode ( type *elem* ) [inline]

Tworzy wezel o podanej wartosci ze wskaznikami ustawionymi na NULL

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Zadana wartosc |

Definicja w linii 39 pliku redblacktree.hh.

### 4.12.3 Dokumentacja atrybutów składowych

#### 4.12.3.1 template<typename type> char rbtreenode< type >::color

true - wezel ma kolor czerwony false - wezel ma kolor czarny

Definicja w linii 31 pliku redblacktree.hh.

#### 4.12.3.2 template<typename type> rbtreenode* rbtreenode< type >::left

Definicja w linii 16 pliku redblacktree.hh.

#### 4.12.3.3 template<typename type> rbtreenode* rbtreenode< type >::parent

Definicja w linii 24 pliku redblacktree.hh.

#### 4.12.3.4 template<typename type> rbtreenode* rbtreenode< type >::right

Definicja w linii 20 pliku redblacktree.hh.

#### 4.12.3.5 template<typename type> type rbtreenode< type >::val

Definicja w linii 12 pliku redblacktree.hh.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- redblacktree.hh

## 4.13 Dokumentacja szablonu klasy RedBlackTree< type >

Klasa RedBlackTree - drzewo czerwono-czarne.

`#include <redblacktree.hh>`

Diagram dziedziczenia dla RedBlackTree< type >

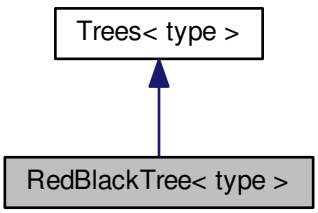
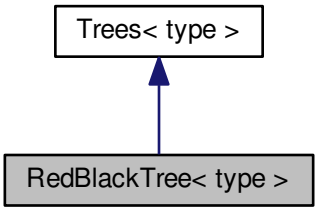


Diagram współpracy dla RedBlackTree< type >:




## Metody publiczne

- RedBlackTree ()

  *Konstruktor bezparametryczny.*
- ~RedBlackTree ()

  *Destruktor.*
- void insert (const type data)

  *Metoda insert.*
- bool remove (const type elem)

  *Metoda usuwajaca wezel o zadanej wartosci z drzewa.*
- bool search (const type elem)

  *Metoda search.*
- void print ()

  *Metoda print.*
- void clear ()

  *Procedura clear.*

## Metody prywatne

- void print (rbtreenode< type > *root)

  *Metoda pomocnicza przy wypisaniu drzewa.*
- rbtreenode< type > * remove (rbtreenode< type > *root, const type elem)

  *Metoda pomocnicza przy usuwaniu zadanej wartosci z drzewa.*
- void init (rbtreenode< type > *toInit, type data)

  *Inicjuje wezel zadana wartoscia, wskazniki left i right ustawia na sentinel.*
- rbtreenode< type > * findSuitableParent (type data)

  *Znajduje odpowiedniego rodzica dla zadanej wartosci.*
- void createBindings (rbtreenode< type > *parent, rbtreenode< type > *child)

  *Tworzy odpowiednie relacje ojca-syna pomiedzy podanymi wezlami.*
- rbtreenode< type > * findMin (rbtreenode< type > *node)

  *Znajduje najmniejsza wartosc w poddrzewie rozpoczynajacym sie w podanym wezle.*
- void setNewRoot (rbtreenode< type > *elem)

  *Ustawia podany element jako nowy korzen.*
- void rotateRight (rbtreenode< type > *elem)

  *Dokonuje rotacji w prawo.*

- void rotateLeft (rbtreenode< type > *elem)

  *Dokonuje rotacji w lewo.*
- void correct (rbtreenode< type > *elem)
- void correctLeft (rbtreenode< type > *elem)
- void correctRight (rbtreenode< type > *elem)
- void deleteTree (rbtreenode< type > *node)

  *Metoda deleteTree.*

### Atrybuty prywatne

- rbtreenode< type > * root

  *Root - korzen drzewa - wskaznik na pierwszy element drzewa.*
- rbtreenode< type > * sentinel

  *Sentinel - straznik.*

## 4.13.1 Opis szczegółowy

**template<typename type>class RedBlackTree< type >**

Definicja w linii 46 pliku redblacktree.hh.

## 4.13.2 Dokumentacja konstruktora i destruktora

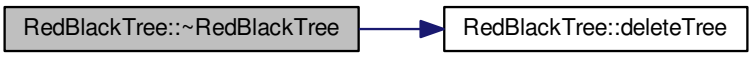
### 4.13.2.1 template<typename type> RedBlackTree< type >::RedBlackTree ( ) [inline]

Definicja w linii 122 pliku redblacktree.hh.

### 4.13.2.2 template<typename type> RedBlackTree< type >::~RedBlackTree ( ) [inline]

Definicja w linii 132 pliku redblacktree.hh.

Oto graf wywołań dla tej funkcji:




## 4.13.3 Dokumentacja funkcji składowych

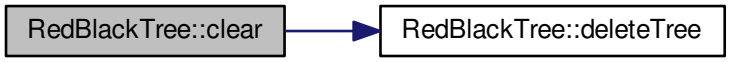
### 4.13.3.1 template<typename type> void RedBlackTree< type >::clear ( ) [inline], [virtual]

Czysci drzewo, usuwa wszystkie jego wezly

Implementuje Trees< type >.

Definicja w linii 170 pliku redblacktree.hh.

Oto graf wywołań dla tej funkcji:




#### 4.13.3.2 template<class type > void RedBlackTree< type >::correct ( rbtreenode< type > * *elem* ) [private]

Definicja w linii 392 pliku redblacktree.hh.

#### 4.13.3.3 template<class type > void RedBlackTree< type >::correctLeft ( rbtreenode< type > * *elem* ) [private]

Definicja w linii 404 pliku redblacktree.hh.

#### 4.13.3.4 template<class type > void RedBlackTree< type >::correctRight ( rbtreenode< type > * *elem* ) [private]

Definicja w linii 434 pliku redblacktree.hh.

#### 4.13.3.5 template<class type > void RedBlackTree< type >::createBindings ( rbtreenode< type > * *parent*, rbtreenode< type > * *child* ) [private]

**Parametry**

| | | |
|---|---|---|
| in | *parent* | Wezel ojciec |
| in | *child* | Wezel syn |

Definicja w linii 374 pliku redblacktree.hh.

#### 4.13.3.6 template<class type > void RedBlackTree< type >::deleteTree ( rbtreenode< type > * *node* ) [private]

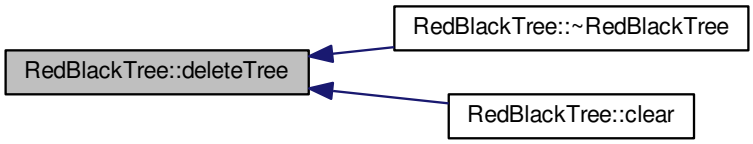
Usuwa drzewa zaczynajace sie w podanym wezle

**Parametry**

| | | |
|---|---|---|
| in | *node* | Korzen drzewa, ktore chcemy usunac |

Definicja w linii 467 pliku redblacktree.hh.

Oto graf wywoływań tej funkcji:




#### 4.13.3.7 template<class type > rbtreenode< type > * RedBlackTree< type >::findMin ( rbtreenode< type > * *node* ) [private]

**Parametry**

| | | |
|---|---|---|
| in | *node* | Korzen poddrzewa, w ktorym znaleziona zostanie najmniejsza wartosc |

Definicja w linii 457 pliku redblacktree.hh.

#### 4.13.3.8 template<class type > rbtreenode< type > * RedBlackTree< type >::findSuitableParent ( type *data* ) [private]

**Parametry**

| | | |
|---|---|---|
| in | *data* | Wartosc, dla ktorej poszukujemy wezla rodzica |

**Zwracane wartości**

| | |
|---|---|
| *Znaleziony* | wezel rodzic |

Definicja w linii 297 pliku redblacktree.hh.

#### 4.13.3.9 template<class type > void RedBlackTree< type >::init ( rbtreenode< type > * *tolnit,* type *data* ) [private]

Definicja w linii 287 pliku redblacktree.hh.

#### 4.13.3.10 template<class type > void RedBlackTree< type >::insert ( const type *data* ) [virtual]

Sluzy do wstawienia zadanego elementu w odpowniednie miejsce drzewa

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc do wstawienia |

Implementuje Trees< type >.

Definicja w linii 177 pliku redblacktree.hh.

#### 4.13.3.11 template<class type > void RedBlackTree< type >::print ( rbtreenode< type > * *root* ) [private]

Definicja w linii 276 pliku redblacktree.hh.

#### 4.13.3.12 template<class type > void RedBlackTree< type >::print ( )

Sluzy do wyswietlenia elementow drzewa na standardowym strumieniu wyjsciowym. Elementy wyswietlane sa w nastepujacej kolejności: korzen, lewe poddrzewo, prawe poddrzewo

Definicja w linii 265 pliku redblacktree.hh.

#### 4.13.3.13 template<class type > rbtreenode< type > * RedBlackTree< type >::remove ( rbtreenode< type > * *root,* const type *elem* ) [private]

Definicja w linii 235 pliku redblacktree.hh.

#### 4.13.3.14 template<class type > bool RedBlackTree< type >::remove ( const type *elem* ) [virtual]

Implementuje Trees< type >.

Definicja w linii 205 pliku redblacktree.hh.

#### 4.13.3.15 template<class type > void RedBlackTree< type >::rotateLeft ( rbtreenode< type > * *elem* ) [private]

Definicja w linii 341 pliku redblacktree.hh.

#### 4.13.3.16 template<class type > void RedBlackTree< type >::rotateRight ( rbtreenode< type > * *elem* ) [private]

Definicja w linii 317 pliku redblacktree.hh.

#### 4.13.3.17 template<class type > bool RedBlackTree< type >::search ( const type *elem* ) [virtual]

Sluzy do sprawdzenia, czy w drzewie znajduje sie element o zadanej wartosci

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc elementu do znalezienia |

**Zwracane wartości**

| | |
|---|---|
| *TRUE* | Element znajduje sie w drzewie |
| *FALSE* | Brak elementu o zadanej wartosci |

Implementuje Trees< type >.

Definicja w linii 188 pliku redblacktree.hh.

#### 4.13.3.18 template<class type > void RedBlackTree< type >::setNewRoot ( rbtreenode< type > * *elem* ) [private]

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Nowy korzen |

Definicja w linii 365 pliku redblacktree.hh.

### 4.13.4 Dokumentacja atrybutów składowych

#### 4.13.4.1 template<typename type> rbtreenode<type>* RedBlackTree< type >::root [private]

Definicja w linii 51 pliku redblacktree.hh.

#### 4.13.4.2 template<typename type> rbtreenode<type>* RedBlackTree< type >::sentinel [private]

Definicja w linii 55 pliku redblacktree.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- redblacktree.hh

## 4.14 Dokumentacja klasy SaveToFile

`#include <observer.hh>`

Diagram dziedziczenia dla SaveToFile

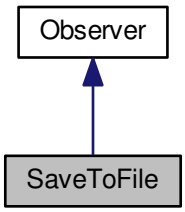
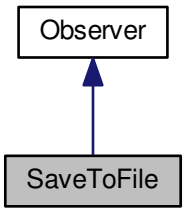



Diagram współpracy dla SaveToFile:




### Metody publiczne

- void update (int dataNumber, double mean)

### 4.14.1 Opis szczegółowy

Definicja w linii 27 pliku observer.hh.

### 4.14.2 Dokumentacja funkcji składowych

#### 4.14.2.1 void SaveToFile::update ( int *dataNumber,* double *mean* ) [inline],[virtual]

Implementuje Observer.

Definicja w linii 32 pliku observer.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- observer.hh

## 4.15 Dokumentacja szablonu klasy Stack< type >

`#include <stack.hh>`

Diagram dziedziczenia dla Stack< type >

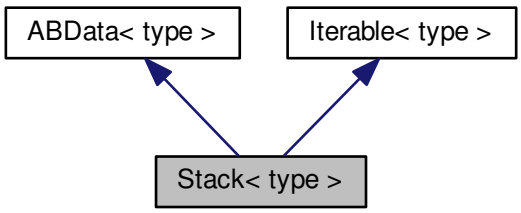
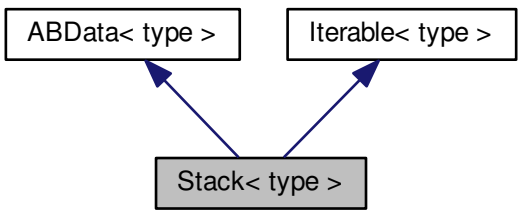



Diagram współpracy dla Stack< type >:




### Metody publiczne

- Stack ()

*Konstruktor bezparametryczny.*

- void push (const type elem)

  *Metoda push.*

- void pop ()

  *Procedura pop.*

- unsigned int size ()

  *Metoda size.*

- type & operator[] (const unsigned int index)

  *Przeciazenie operatora [].*

- void display ()

### Atrybuty prywatne

- node< type > * head

  *Wskaznik head.*

- int iterator

  *Iterator.*

## 4.15.1 Opis szczegółowy

**template<class type>class Stack< type >**

Definicja w linii 12 pliku stack.hh.

## 4.15.2 Dokumentacja konstruktora i destruktora

### 4.15.2.1 template<class type> Stack< type >::Stack ( ) [inline]

Ustawia poczatek listy na NULL

Definicja w linii 33 pliku stack.hh.

## 4.15.3 Dokumentacja funkcji składowych

### 4.15.3.1 template<class type> void Stack< type >::display ( ) [inline]

Definicja w linii 74 pliku stack.hh.

### 4.15.3.2 template<class type > type & Stack< type >::operator[] ( const unsigned int *index* ) [virtual]

Zwraca element o podanym indeksie (indeksowanie zaczyna się od 0) W przypadku odwolania sie poza zakres, program przerywany jest z bledem 1.

**Zwraca**

Wartosc znajdujaca sie na miejscu o podanym indeksie

Implementuje Iterable< type >.

Definicja w linii 111 pliku stack.hh.

#### 4.15.3.3 template<class type > void Stack< type >::pop ( ) [virtual]

Usuwa pierwszy element stosu.

Implementuje ABData< type >.

Definicja w linii 94 pliku stack.hh.

#### 4.15.3.4 template<class type> void Stack< type >::push ( const type *elem* ) [virtual]

Dodaje podana wartosc na poczatek listy.

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Wartosc, ktora chcemy dodac na poczatek listy. |

Implementuje ABData< type >.

Definicja w linii 84 pliku stack.hh.

#### 4.15.3.5 template<class type > unsigned int Stack< type >::size ( ) [virtual]

Daje informacje o rozmiarze stosu (liczbie jego elementow).

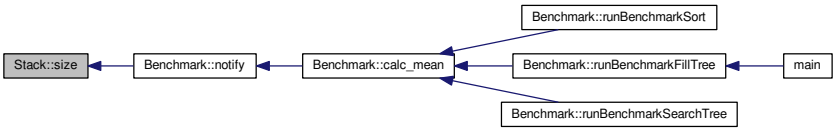
**Zwraca**

Rozmiar stosu (liczba jego elementow)

Implementuje ABData< type >.

Definicja w linii 106 pliku stack.hh.

Oto graf wywoływań tej funkcji:




### 4.15.4 Dokumentacja atrybutów składowych

#### 4.15.4.1 template<class type> node<type>* Stack< type >::head [private]

Wskaznik na pierwszy element stosu

Definicja w linii 19 pliku stack.hh.

#### 4.15.4.2 template<class type> int Stack< type >::iterator [private]

Przechowuje informacje o liczbie elementow znajdujacych sie na stosie

Definicja w linii 25 pliku stack.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- stack.hh

## 4.16 Dokumentacja klasy Subject

`#include <observer.hh>`

Diagram dziedziczenia dla Subject

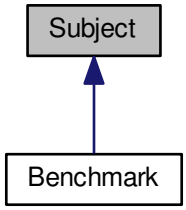
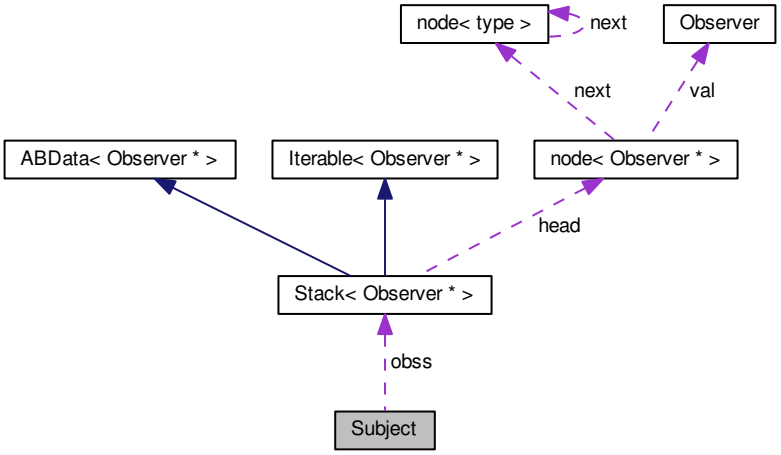



Diagram współpracy dla Subject:




### Metody publiczne

- void addObs (Observer *toadd)
- virtual void notify ()=0

### Atrybuty chronione

- Stack< Observer * > obss

### 4.16.1 Opis szczegółowy

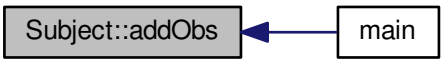
Definicja w linii 19 pliku observer.hh.

### 4.16.2 Dokumentacja funkcji składowych

#### 4.16.2.1 void Subject::addObs ( Observer * *toadd* ) [inline]

Definicja w linii 23 pliku observer.hh.

Oto graf wywoływań tej funkcji:




#### 4.16.2.2 virtual void Subject::notify ( ) [pure virtual]

Implementowany w Benchmark.

### 4.16.3 Dokumentacja atrybutów składowych

#### 4.16.3.1 Stack<Observer*> Subject::obss [protected]

Definicja w linii 21 pliku observer.hh.

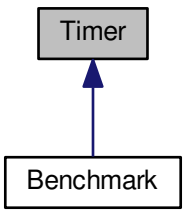
Dokumentacja dla tej klasy została wygenerowana z pliku:

- observer.hh

## 4.17 Dokumentacja klasy Timer

`#include <timer.hh>`

Diagram dziedziczenia dla Timer



### Metody publiczne

- Timer ()

  *Konstruktor bezparametryczny.*
- void start_timer ()

  *Zapisuje moment rozpoczecia pomiaru do zmiennej start.*
- void stop_timer ()

  *Konczy pomiar czasu.*
- double getTime ()

  *Akcesor do zmiennej time.*

### Atrybuty chronione

- timeval start

  *Zmienne start, end.*
- timeval end
- double atime

  *Zmienna time.*

## 4.17.1 Opis szczegółowy

Definicja w linii 12 pliku timer.hh.

## 4.17.2 Dokumentacja konstruktora i destruktora

### 4.17.2.1 Timer::Timer ( ) [inline]

Definicja w linii 32 pliku timer.hh.

## 4.17.3 Dokumentacja funkcji składowych

### 4.17.3.1 double Timer::getTime ( )

**Zwraca**

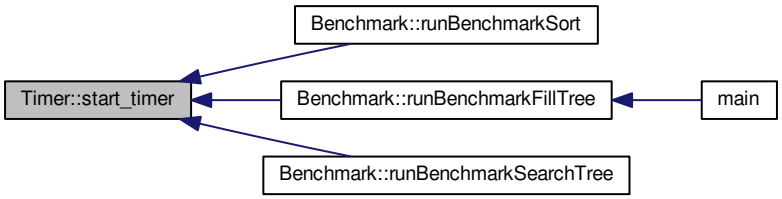
Zwraca wartosc zmiennej time

Definicja w linii 19 pliku timer.cpp.

### 4.17.3.2 void Timer::start_timer ( )

Definicja w linii 8 pliku timer.cpp.

Oto graf wywoływań tej funkcji:

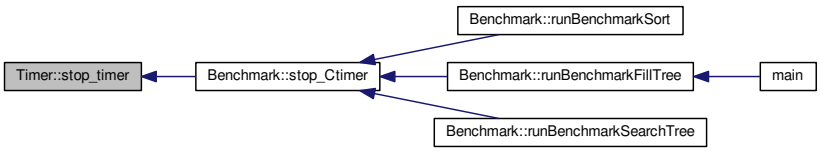



#### 4.17.3.3 void Timer::stop_timer ( )

Zapisuje moment zakonczenia pomiaru do zmiennej end, oblicza zmierzony czas i zapisuje do zmiennej time.

Definicja w linii 13 pliku timer.cpp.

Oto graf wywoływań tej funkcji:




### 4.17.4 Dokumentacja atrybutów składowych

#### 4.17.4.1 double Timer::atime [protected]

Przechowuje zmierzony czas

Definicja w linii 26 pliku timer.hh.

#### 4.17.4.2 timeval Timer::end [protected]

Definicja w linii 19 pliku timer.hh.

#### 4.17.4.3 timeval Timer::start [protected]

Przechowuja informacje o poczatku i koncu pomiaru czasu

Definicja w linii 19 pliku timer.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

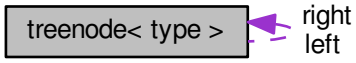
- timer.hh
- timer.cpp

## 4.18 Dokumentacja szablonu struktury treenode< type >

Wezel drzewa.

`#include <binarytree.hh>`

Diagram współpracy dla treenode< type >:




### Metody publiczne

- treenode (type elem)
  *Konstruktor parametryczny.*

### Atrybuty publiczne

- type val
  *Przechowywana wartosc.*
- treenode * left
  *Wskaznik na lewe dziecko.*
- treenode * right
  *Wskaznik na prawe dziecko.*

### 4.18.1 Opis szczegółowy

**template<typename type>struct treenode< type >**

Definicja w linii 18 pliku binarytree.hh.

### 4.18.2 Dokumentacja konstruktora i destruktora

#### 4.18.2.1 template<typename type> treenode< type >::treenode ( type *elem* ) [inline]

Tworzy wezel o podanej wartosci ze wskaznikami ustawionymi na NULL

**Parametry**

| | | |
|---|---|---|
| in | *elem* | Zadana wartosc |

Definicja w linii 39 pliku binarytree.hh.

### 4.18.3 Dokumentacja atrybutów składowych

#### 4.18.3.1 template<typename type> treenode* treenode< type >::left

Definicja w linii 26 pliku binarytree.hh.

#### 4.18.3.2 template<typename type> treenode* treenode< type >::right

Definicja w linii 30 pliku binarytree.hh.

#### 4.18.3.3 template<typename type> type treenode< type >::val

Definicja w linii 22 pliku binarytree.hh.

Dokumentacja dla tej struktury została wygenerowana z pliku:

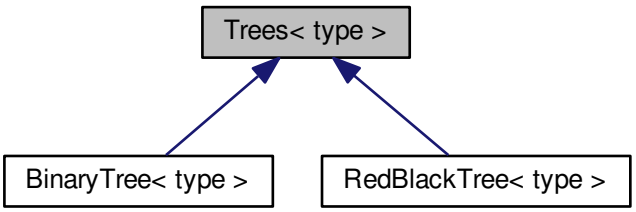
- binarytree.hh

## 4.19 Dokumentacja szablonu klasy Trees< type >

Klasa abstrakcyjna zawierajaca metody wirtualne drzew.

`#include <trees.hh>`

Diagram dziedziczenia dla Trees< type >




### Metody publiczne

- virtual void insert (const type elem)=0
- virtual bool remove (const type elem)=0
- virtual bool search (const type elem)=0
- virtual void clear ()=0

### 4.19.1 Opis szczegółowy

template<class type>class Trees< type >

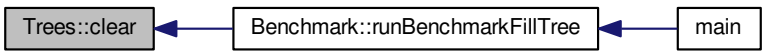
Definicja w linii 13 pliku trees.hh.

### 4.19.2 Dokumentacja funkcji składowych

#### 4.19.2.1 template<class type> virtual void Trees< type >::clear ( ) [pure virtual]

Implementowany w BinaryTree< type > i RedBlackTree< type >.

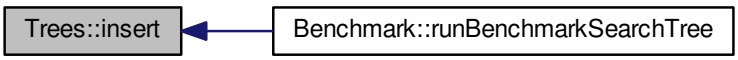
Oto graf wywoływań tej funkcji:




#### 4.19.2.2 template<class type> virtual void Trees< type >::insert ( const type *elem* ) [pure virtual]

Implementowany w BinaryTree< type > i RedBlackTree< type >.

Oto graf wywoływań tej funkcji:




#### 4.19.2.3 template<class type> virtual bool Trees< type >::remove ( const type *elem* ) [pure virtual]

Implementowany w BinaryTree< type > i RedBlackTree< type >.

#### 4.19.2.4 template<class type> virtual bool Trees< type >::search ( const type *elem* ) [pure virtual]

Implementowany w BinaryTree< type > i RedBlackTree< type >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- trees.hh

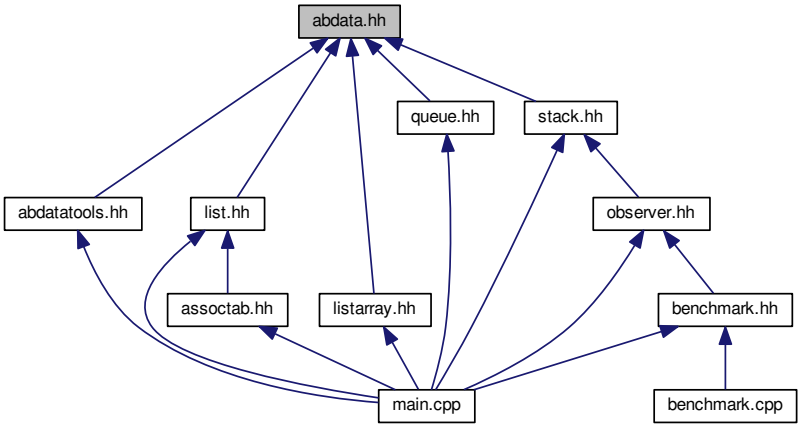
# Rozdział 5

# Dokumentacja plików

## 5.1 Dokumentacja pliku abdata.hh

Definicja wirtualnej klasy ABData.

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




### Komponenty

- class ABData< type >

  *Modeluje klase wirtualna ABData, ktora jest interfejsem.*

### 5.1.1 Opis szczegółowy

Klasa ABData modeluje interfejs abstrakcyjnych typow danych posiadajacych metody push(), pop() i size()

Definicja w pliku abdata.hh.

## 5.2 abdata.hh

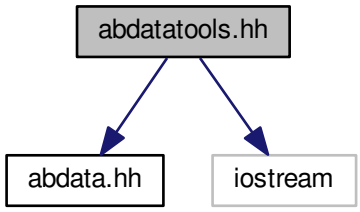
```
00001 #ifndef ABDATA_HH
```

```
00002 #define ABDATA_HH
00003
00015 template <class type>
00016 class ABData{
00017 public:
00018   virtual void push(const type elem)=0;
00019   virtual void pop()=0;
00020   virtual unsigned int size()=0;
00021 };
00022
00023 #endif
```

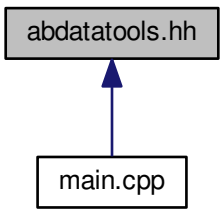
## 5.3 Dokumentacja pliku abdatatools.hh

```
#include "abdata.hh"
#include <iostream>
```

Wykres zależności załączania dla abdatatools.hh:




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




### Funkcje

- template<typename type >
  bool fillFromFile (ABData< type > *item, const int amount, const char *fileName)

  *Wypelnia zadana strukture zadana iloscia danych wczytywana z zadanego pliku.*
- template<typename type >
  void clear (ABData< type > *item)

  *Usuwa wszystkie dane znajdujace sie w strukturze.*

### 5.3.1 Dokumentacja funkcji

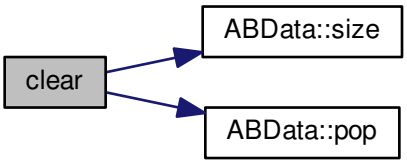
#### 5.3.1.1 template<typename type > void clear ( ABData< type > * *item* )

**Parametry**

| | | |
|---|---|---|
| in | *item | Wskaznik do obiektu typu dziedziczacego z ABData, ktory chcemy wyczyscic |

Definicja w linii 43 pliku abdatatools.hh.

Oto graf wywołań dla tej funkcji:



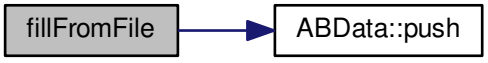

#### 5.3.1.2 template<typename type > bool fillFromFile ( ABData< type > * *item*, const int *amount*, const char * *fileName* )

**Parametry**

| | | |
|---|---|---|
| in | *item | Wskaznik do obiektu typu dziedziczacego z ABData, ktory chcemy wypelnic |
| in | *amount* | Ilosc danych, jakie chcemy wczytac do obiektu |
| in | *fileName* | Nazwa pliku, z ktorego wczytujemy dane |

Definicja w linii 21 pliku abdatatools.hh.

Oto graf wywołań dla tej funkcji:




## 5.4 abdatatools.hh

```
00001 #ifndef ABDATATOOLS_HH
00002 #define ABDATATOOLS_HH
00003
00004 #include "abdata.hh"
00005 #include <iostream>
00006
00007 /*
00008  *!\file
00009  * \brief Plik zawiera definicje funkcji operujacych na obiektach o klasie nadrzednej
00010  * ABData.
00011  */
```

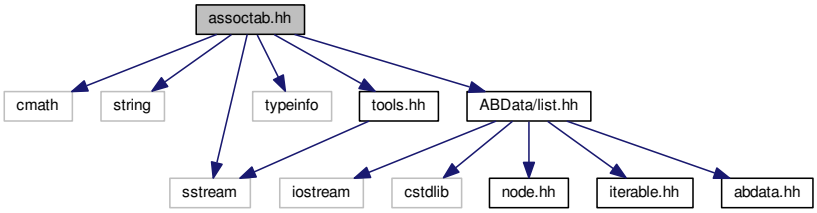
```
00012
00020 template <typename type>
00021 bool fillFromFile(ABData<type> *item, const int amount, const char* fileName){
00022   ifstream inputFile;
00023   inputFile.open(fileName);
00024   if(inputFile.good()==false){
00025     std::cerr<<"Blad odczytu pliku!"<<std::endl;
00026     return false;
00027   }
00028   type tmp;
00029   for(int i=0; i<amount; i++){
00030     inputFile >> tmp;
00031     item->push(tmp);
00032     }
00033   inputFile.close();
00034   return true;
00035 }
00036
00042 template <typename type>
00043 void clear(ABData<type> *item){
00044   while(item->size() > 0)
00045     item->pop();
00046 }
00047
00048 #endif
```

## 5.5 Dokumentacja pliku assoctab.hh

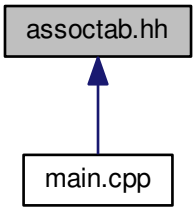
Definicja klasy AssocTab.

```
#include <cmath>
#include <string>
#include <sstream>
#include <typeinfo>
#include "ABData/list.hh"
#include "tools.hh"
```

Wykres zależności załączania dla assoctab.hh:



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




### Komponenty

- class AssocTab< typeKey, type >

### Definicje

- #define TAB 1000
- #define HASH 0.6180339887

### 5.5.1 Dokumentacja definicji

#### 5.5.1.1 #define HASH 0.6180339887

Definicja w linii 12 pliku assoctab.hh.

#### 5.5.1.2 #define TAB 1000

Definicja w linii 11 pliku assoctab.hh.

## 5.6 assoctab.hh

```
00001 #ifndef ASSOCTAB_HH
00002 #define ASSOCTAB_HH
00003
00004 #include <cmath>
00005 #include <string>
00006 #include <sstream>
00007 #include <typeinfo>
00008 #include "ABData/list.hh"
00009 #include "tools.hh"
00010
00011 #define TAB 1000
00012 #define HASH 0.6180339887  //Donald Knuth hashing const
00013
00019 template <class typeKey, class type>
00020 class AssocTab{
00021
00025   List<AssocData<typeKey, type> > *tab;
00026
00030   int counter;
00031
00032 public:
00038   AssocTab(){
00039     tab = new List<AssocData<typeKey, type> > [
```

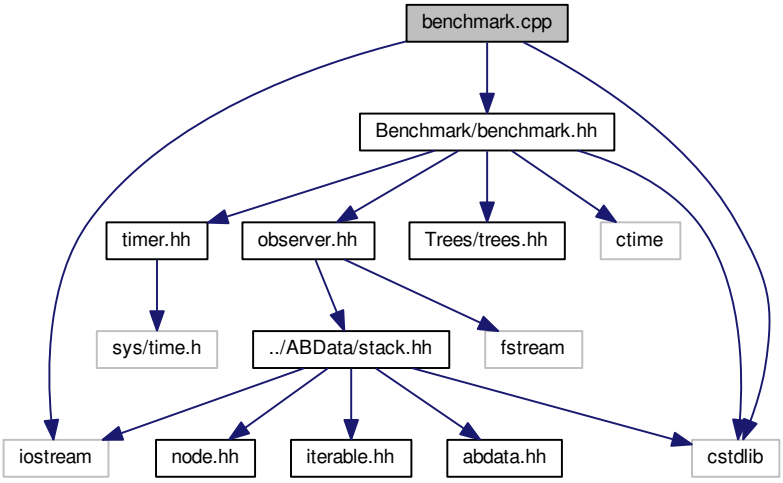
```
      TAB];
00040     counter = TAB;
00041   }
00049   AssocTab(unsigned int howmany){
00050     tab = new List<AssocData<typeKey, type> > [howmany];
00051     counter = howmany;
00052   }
00053
00060   ~AssocTab(){delete[] tab;}
00061
00070   void push(typeKey ikey, type toaddVal);
00071
00079   void pop(typeKey toremoveKey);
00080
00088   int hash(typeKey tohashKey);
00089
00097   unsigned int size(){return counter;}
00098
00110   type& operator [] (const typeKey klucz);
00111 };
00112
00113 template <class typeKey, class type>
00114 void AssocTab<typeKey, type>::push(typeKey ikey, type toaddVal){
00115   AssocData<typeKey, type> toadd(ikey, toaddVal);
00116   tab[hash(ikey)].push(toadd);
00117 }
00118
00119 template <class typeKey, class type>
00120 int AssocTab<typeKey, type>::hash(typeKey tohashKey){
00121   string tohash;
00122   if(typeid(typeKey) == typeid(string))
00123     tohash = tohashKey;
00124   else
00125     tohash = tostring(tohashKey);
00126   double val=0; double add;
00127   for(unsigned int i=0; i<tohash.length(); i++){
00128     add = tohash[i]*(i+1);
00129     val+=add;
00130   }
00131   val*=HASH;
00132   val-=(int)val;
00133   return floor(counter*val);
00134 }
00135
00136 template <class typeKey, class type>
00137 type& AssocTab<typeKey, type>::operator [] (const typeKey klucz){
00138   for(unsigned int i=0; i<tab[hash(klucz)].size(); i++)
00139     if(tab[hash(klucz)][i].key == klucz)
00140       return tab[hash(klucz)][i].val;
00141   AssocData<typeKey, type> created(klucz);
00142   tab[hash(klucz)].push(created);
00143   return tab[hash(klucz)][0].val;
00144
00145 }
00146
00147 template <class typeKey, class type>
00148 void AssocTab<typeKey, type>::pop(typeKey toremoveKey){
00149   for(unsigned int i=0; i<tab[hash(toremoveKey)].size(); i++)
00150     if(tab[hash(toremoveKey)][i].key == toremoveKey)
00151       tab[hash(toremoveKey)].pop(i);
00152 }
00153 #endif
```

## 5.7 Dokumentacja pliku benchmark.cpp

Ciala metod klasy Benchmark.

```
#include "Benchmark/benchmark.hh"
#include <cstdlib>
#include <iostream>
```

Wykres zależności załączania dla benchmark.cpp:




## 5.8 benchmark.cpp

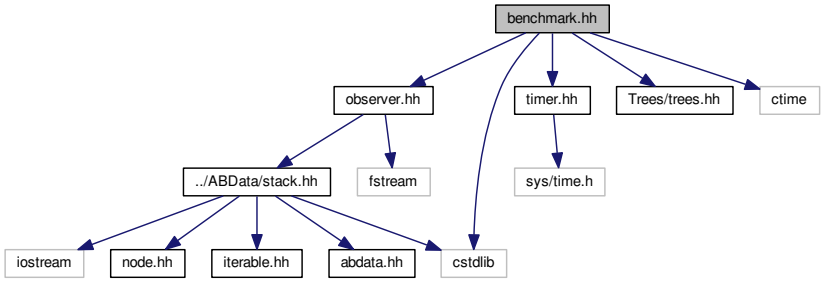
```
00001 #include "Benchmark/benchmark.hh"
00002 #include <cstdlib>
00003 #include <iostream>
00008 void Benchmark::notify(){
00009   for(unsigned int i=0; i<obss.size();i++)
00010     obss[i]->update(amount, mean);
00011 }
00012
00013 void Benchmark::stop_Ctimer(){
00014   stop_timer();
00015   total+=atime;
00016   counter++;
00017 }
00018
00019 void Benchmark::calc_mean(){
00020   mean=total/counter;
00021   std::cout << mean << "   " << amount << "    " << std::endl;
00022   notify();
00023 }
00024
00025 template<typename type>
00026 void Benchmark::runBenchmarkSort(void (*f)(
      Iterable<type>&, int, int), Iterable<type> &container, int dataCount, int
      repeats){
00027   amount = dataCount;
00028   total=0;
00029   mean=0;
00030   counter=0;
00031   for(int i=1; i<=repeats; i++){
00032     start_timer();
00033     (*f)(container, 0, amount-1);
00034     stop_Ctimer();
00035   }
00036   calc_mean();
00037 }
00038
00039
00040
```

## 5.9 Dokumentacja pliku benchmark.hh

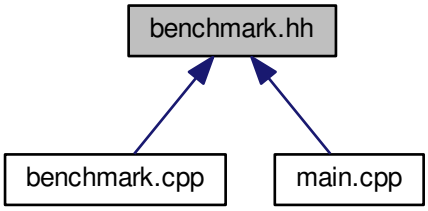
Definicja klasy Benchmark.

```
#include "observer.hh"
#include "timer.hh"
#include "Trees/trees.hh"
#include <cstdlib>
#include <ctime>
```

Wykres zależności załączania dla benchmark.hh:




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




### Komponenty

- class Benchmark

  *Klasa Benchmark.*

## 5.10 benchmark.hh

```
00001 #ifndef BENCHMARK_HH
00002 #define BENCHMARK_HH
00003
00004 #include "observer.hh"
00005 #include "timer.hh"
00006 #include "Trees/trees.hh"
00007 #include <cstdlib>
00008 #include <ctime>
00009
00020 class Benchmark: public Subject, public Timer{
00024   double total;
00028   double mean;
00032   int counter;
00036   int amount;
00037 public:
```

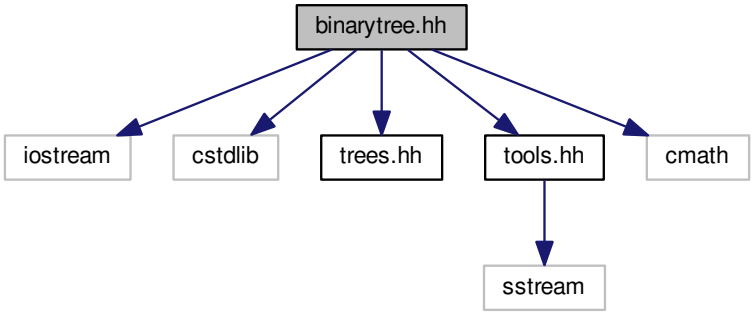
```
00038   Benchmark(){
00039     total = 0;
00040     mean = 0;
00041     counter = 0;
00042     amount = 0;
00043   }
00047   void notify();
00054   void stop_Ctimer();
00055
00061   void calc_mean();
00062
00071   template<typename type>
00072   void runBenchmarkSort(void (*f)(Iterable<type>&, int, int),
      Iterable<type> &container, int dataCount, int repeats);
00073
00082   template<typename type>
00083   void runBenchmarkFillTree(void (Trees<type>::*f)(type),
      Trees<type> &tree, int dataCount, int repeats, char* dataFile);
00084
00093   template<typename type>
00094   void runBenchmarkSearchTree(bool (Trees<type>::*f)(type),
      Trees<type> &tree, int dataCount, int repeats, char* dataFile);
00095 };
00096
00097 template<typename type>
00098 void Benchmark::runBenchmarkFillTree(void (
      Trees<type>::*f)(type), Trees<type> &tree, int dataCount, int repeats, char* dataFile
      ){
00099   amount = dataCount;
00100   total=0;
00101   mean=0;
00102   counter=0;
00103   ifstream input;
00104   input.open(dataFile);
00105   type tmp;
00106   for(int i=1; i<=repeats; i++){
00107     tree.clear();
00108     start_timer();
00109     for(int j=1; j<=dataCount; j++){
00110       input >> tmp;
00111       (tree.*f)(tmp);
00112     }
00113     stop_Ctimer();
00114   }
00115   calc_mean();
00116   input.close();
00117 }
00118
00119 template<typename type>
00120 void Benchmark::runBenchmarkSearchTree(bool (
      Trees<type>::*f)(type), Trees<type> &tree, int dataCount, int repeats, char* dataFile
      ){
00121   amount = dataCount;
00122   total=0;
00123   mean=0;
00124   counter=0;
00125   ifstream input;
00126   input.open(dataFile);
00127   type tmp;
00128   for(int j=1; j<=dataCount; j++){
00129     input >> tmp;
00130     tree.insert(j);
00131   }
00132   input.close();
00133   srand(time(NULL));
00134   for(int i=1; i<=repeats; i++){
00135     start_timer();
00136     for(int j=1; j<=dataCount; j++)
00137       (tree.*f)(48830);
00138     stop_Ctimer();
00139   }
00140   calc_mean();
00141 }
00142
00143
00144 #endif
```

## 5.11 Dokumentacja pliku binarytree.hh

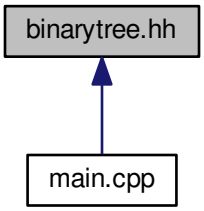
Definicja klasy drzewa binarnego.

```
#include <iostream>
#include <cstdlib>
#include "trees.hh"
#include "tools.hh"
#include <cmath>
```

Wykres zależności załączania dla binarytree.hh:




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




### Komponenty

- struct treenode< type >

  *Wezel drzewa.*
- class BinaryTree< type >

  *Klasa BinaryTree - drzewo binarne.*

## 5.12 binarytree.hh

```
00001 #ifndef BINARYTREE_HH
00002 #define BINARYTREE_HH
00003
00004 #include <iostream>
00005 #include <cstdlib>
00006 #include "trees.hh"
00007 #include "tools.hh"
```

```
00008 #include <cmath>
00017 template <typename type>
00018 struct treenode{
00022   type val;
00026   treenode *left;
00030   treenode *right;
00031
00039   treenode(type elem){
00040     val = elem;
00041     left = NULL;
00042     right = NULL;
00043   }
00044 };
00045
00049 template <class type>
00050 class BinaryTree: public Trees<type>{
00051 private:
00055   int numberOfNodes;
00059   treenode<type> *root;
00065   void insert(const type elem, treenode<type> *leaf);
00071   void print(treenode<type> *root);
00072
00078   treenode<type> * remove(treenode<type> * node, const type elem);
00079
00086   treenode<type> * findMin(treenode<type> *
      node);
00087
00093   bool rotateLeft(treenode<type> *node);
00101   bool rotateRight(treenode<type> *node);
00102
00110   void balance(treenode<type> *root);
00111
00119   void deleteTree(treenode<type> *node);
00120
00130   int height(treenode<type> *node);
00131
00132 public:
00136   BinaryTree(){
00137     numberOfNodes = 0;
00138     root = NULL;
00139   }
00143   ~BinaryTree(){
00144     deleteTree(root);
00145   }
00153   void insert(const type elem);
00164   bool remove(const type elem);
00175   bool search(const type elem);
00182   void print();
00183
00189   int height();
00190
00196   void clear(){deleteTree(root); root=NULL; }
00197 };
00198
00199 template <class type>
00200 void BinaryTree<type>::insert(const type elem,
      treenode<type> *leaf){
00201   if(elem < leaf->val){
00202     if(leaf->left!=NULL)
00203       insert(elem, leaf->left);
00204     else
00205       leaf->left = new treenode<type>(elem);
00206   }
00207   else if(elem >= leaf->val){
00208     if(leaf->right!=NULL)
00209       insert(elem, leaf->right);
00210     else
00211       leaf->right = new treenode<type>(elem);
00212   }
00213 }
00214
00215
00216
00217 template <class type>
00218 void BinaryTree<type>::insert(const type elem){
00219   if(root!=NULL)
00220     insert(elem, root);
00221   else
00222     root = new treenode<type>(elem);
00223   ++numberOfNodes;
00224   if(height(root)>2*log2(numberOfNodes)) //USTAWIONO 2
00225     balance(root);
00226 }
00227
00228 template <class type>
00229 bool BinaryTree<type>::remove(const type elem){
00230   if(root == NULL)
```

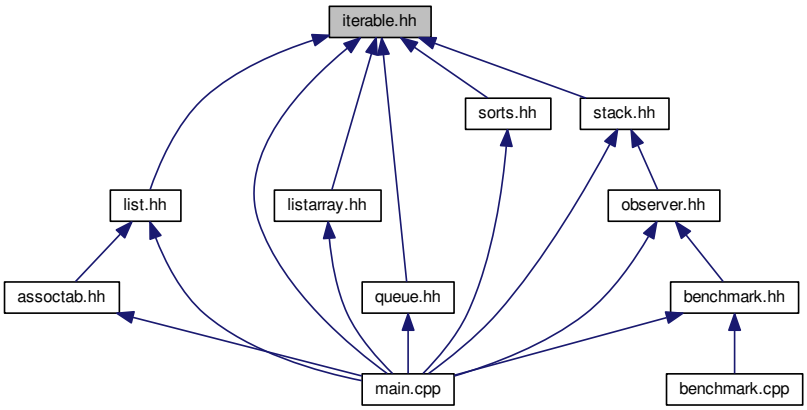
```
00231     return false;
00232   if(elem < root->val)
00233     root->left = remove(root->left, elem);
00234   else if(elem > root->val)
00235     root->right = remove(root->right, elem);
00236   else{
00237     if(root->left == NULL){
00238       treenode<type> *tmp = root;
00239       root = root->right;
00240       delete tmp;
00241       return true;
00242     }
00243     else if(root->right == NULL){
00244       treenode<type> *tmp = root;
00245       root = root->left;
00246       delete tmp;
00247       return true;
00248     }
00249     treenode<type> *tmp = findMin(root->right);
00250     root->val = tmp->val;
00251     root->right = remove(root->right, tmp->val);
00252   }
00253   return true;
00254 }
00255
00256 template <class type>
00257 treenode<type> * BinaryTree<type>::remove(
     treenode<type> *root, const type elem){
00258   if(root == NULL)
00259     return root;
00260   if(elem < root->val)
00261     root->left = remove(root->left, elem);
00262   else if(elem > root->val)
00263     root->right = remove(root->right, elem);
00264   else{
00265     if(root->left == NULL){
00266       treenode<type> *tmp = root;
00267       root = root->right;
00268       delete tmp;
00269       return root;
00270     }
00271     else if(root->right == NULL){
00272       treenode<type> *tmp = root;
00273       root = root->left;
00274       delete tmp;
00275       return root;
00276     }
00277     treenode<type> *tmp = findMin(root->right);
00278     root->val = tmp->val;
00279     root->right = remove(root->right, tmp->val);
00280   }
00281   return root;
00282 }
00283
00284 template <class type>
00285 bool BinaryTree<type>::search(const type elem){
00286   treenode<type> *ptr = root;
00287   while(true){
00288     if(ptr == NULL)
00289       return false;
00290     else if(elem == ptr->val)
00291       return true;
00292     else if(elem < ptr->val)
00293       ptr = ptr->left;
00294     else
00295       ptr = ptr->right;
00296   }
00297 }
00298
00299 template <class type>
00300 void BinaryTree<type>::print(){
00301   if(root != NULL){
00302     std::cout << root->val << " ";
00303     print(root->left);
00304     print(root->right);
00305   }
00306 }
00307
00308 template <class type>
00309 void BinaryTree<type>::print(treenode<type> *root){
00310   if(root != NULL){
00311     std::cout << root-> val << " ";
00312     print(root->left);
00313     print(root->right);
00314   }
00315 }
00316
```

```
00317 template <class type>
00318 treenode<type> * BinaryTree<type>::findMin(
      treenode<type> *node){
00319   treenode<type> *ptr = node;
00320   while(ptr->left != NULL)
00321     ptr = ptr->left;
00322   return ptr;
00323 }
00324
00325 template <class type>
00326 bool BinaryTree<type>::rotateLeft(treenode<type> *
      node){
00327   treenode<type> *ptr;
00328   if(node==NULL || node->right==NULL)
00329     return false;
00330   ptr=node->right;
00331   node->right=ptr->right;
00332   ptr->right=ptr->left;
00333   ptr->left=node->left;
00334   node->left=ptr;
00335
00336   substitute(node->val, ptr->val);
00337   return true;
00338 }
00339 template<class type>
00340 bool BinaryTree<type>::rotateRight(treenode<type> *
      node){
00341   treenode<type> *ptr;
00342   if(node==NULL || node->left==NULL)
00343     return false;
00344   ptr=node->left;
00345   node->left=ptr->left;
00346   ptr->left=ptr->right;
00347   ptr->right=node->right;
00348   node->right=ptr;
00349
00350   substitute(node->val, ptr->val);
00351   return true;
00352 }
00353
00354 template <class type>
00355 void BinaryTree<type>::balance(treenode<type> *root){
00356   treenode<type> *ptr;
00357   int nodecount, i;
00358   for(ptr=root, nodecount=0; ptr!=NULL; ptr=ptr->right, ++nodecount)
00359     while(rotateRight(ptr)==true)
00360       {}
00361   for(i=nodecount/2; i>0; i/=2){
00362     int j;
00363     for(ptr=root, j=0; j<i; ++j, ptr=ptr->right)
00364       rotateLeft(ptr);
00365   }
00366 }
00367
00368 template <class type>
00369 void BinaryTree<type>::deleteTree(treenode<type> *
      node){
00370   if(node){
00371     deleteTree(node->left);
00372     deleteTree(node->right);
00373     delete node;
00374   }
00375 }
00376
00377 template <class type>
00378 int BinaryTree<type>::height(treenode<type> *
      node){
00379   if(node==NULL)
00380     return 0;
00381   return 1+max(height(node->left),height(node->right));
00382 }
00383
00384 template <class type>
00385 int BinaryTree<type>::height(){
00386   return height(root);
00387 }
00388 #endif
```

## 5.13 Dokumentacja pliku iterable.hh

Plik zawiera definicje klasy Iterable.

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




## Komponenty

- class Iterable< type >

  *Modeluje klase wirtualna Iterable.*

## Funkcje

- template<class type >
  void display (Iterable< type > &todisplay, unsigned int howmany)

  *Funkcja display.*

### 5.13.1 Dokumentacja funkcji

#### 5.13.1.1 template<class type > void display ( Iterable< type > & *todisplay,* unsigned int *howmany* )

Pozwala na wyswietlenie zadanej ilosci danych obiektu typu iterable

**Parametry**

| | | |
|---|---|---|
| in | *todisplay* | Referencja do obiektu typu Iterable |
| in | *howmany* | Ilosc danych do wyswietlenia |

Definicja w linii 29 pliku iterable.hh.

## 5.14 iterable.hh

```
00001 #ifndef ITERABLE_HH
00002 #define ITERABLE_HH
00003
00014 template <class type>
00015 class Iterable{
00016 public:
00017   virtual type& operator [](const unsigned int index)=0;
00018 };
00019
00028 template <class type>
00029 void display(Iterable<type> &todisplay, unsigned int howmany){
00030   for(unsigned int i=0; i<howmany; i++)
```

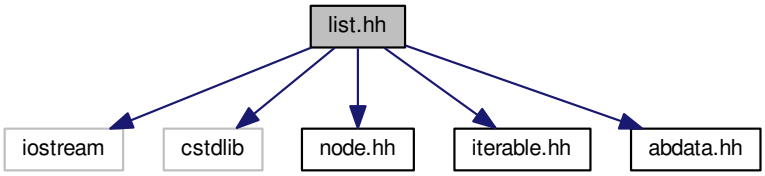
```
00031     std::cout<<todisplay[i]<<std::endl;
00032 }
00033
00034 #endif
```

## 5.15 Dokumentacja pliku list.hh

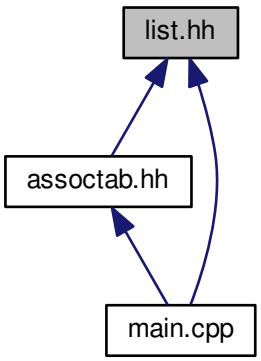
Definicja klasy List.

```
#include <iostream>
#include <cstdlib>
#include "node.hh"
#include "iterable.hh"
#include "abdata.hh"
```

Wykres zależności załączania dla list.hh:




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




### Komponenty

- class List< type >

## 5.16 list.hh

```
00001 #ifndef LIST_HH
00002 #define LIST_HH
00003
00004 #include <iostream>
00005 #include <cstdlib>
00006 #include "node.hh"
00007 #include "iterable.hh"
00008 #include "abdata.hh"
00014 template <class type>
00015 class List: public ABData<type>, public Iterable<type>{
00021   node<type> *head;
00027   int iterator;
00028 public:
00034   List(){
00035     head = NULL;
00036     iterator = 0;
00037   }
00038
00046   void push(const type elem);
00047
00053   void pop();
00054
00060   void pop(unsigned int index);
00061
00069   unsigned int size();
00070
00079   type& operator [] (const unsigned int index);
00080 };
00081 /*************************************************/
00082 /*                                               */
00083 /*               END OF CLASS                    */
00084 /*                                               */
00085 /*************************************************/
00086 template <class type>
00087 void List<type>::push(const type elem){
00088   node<type> *toadd = new node<type>;
00089   toadd->val = elem;
00090   node<type> *ptr = head;
00091   head = toadd;
00092   toadd->next = ptr;
00093   iterator++;
00094 }
00095
00096 template <class type>
00097 void List<type>::pop(){
00098   if(!head)
00099     std::cerr<<"Lista jest pusta!"<<std::endl;
00100   else{
00101     node<type> *ptr = head;
00102     head = head->next;
00103     delete ptr;
00104     iterator--;
00105   }
00106 }
00107
00108 template <class type>
00109 void List<type>::pop(unsigned int index){
00110   if(!head)
00111     std::cerr<<"Lista jest pusta!"<<std::endl;
00112   else{
00113     node<type> *ptr = head;
00114     if(index==0){
00115       head=head->next;
00116       delete ptr;
00117       iterator--;
00118     }
00119     else{
00120       ptr=ptr->next;
00121       node<type> *prev = head;
00122       unsigned int i=1;
00123     for(; i<index && ptr->next; i++){
00124       ptr = ptr->next;
00125       prev = prev->next;
00126     }
00127       if(i==index){
00128     prev->next=ptr->next;
00129     delete ptr;
00130     iterator--;
00131       }
00132       else
00133     std::cerr<<"Brak elementu o podanym indeksie!"<<std::endl;
00134     }
00135   }
00136 }
```

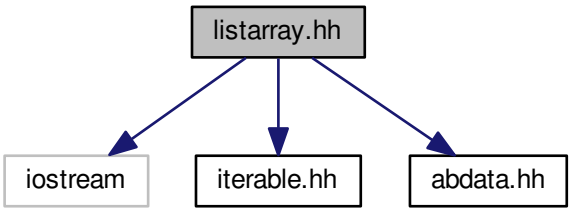
```
00137
00138 template <class type>
00139 unsigned int List<type>::size(){
00140   return iterator;
00141 }
00142
00143 template <class type>
00144 type& List<type>::operator [] (const unsigned int index){
00145   if(index >= size()){
00146     std::cerr<<"Brak elementu o żądanym indeksie!"<<std::endl;
00147     exit(1);
00148   }
00149   else{
00150     node<type> *ptr = head;
00151     for(unsigned int i=1; i<=index; i++)
00152       ptr=ptr->next;
00153     return ptr->val;
00154     }
00155 }
00156
00157 #endif
```

## 5.17 Dokumentacja pliku listarray.hh

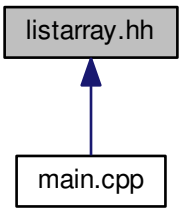
Definicja klasy ListArray.

```
#include <iostream>
#include "iterable.hh"
#include "abdata.hh"
```

Wykres zależności załączania dla listarray.hh:




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



### Komponenty

- class ListArray< type >

## 5.18 listarray.hh

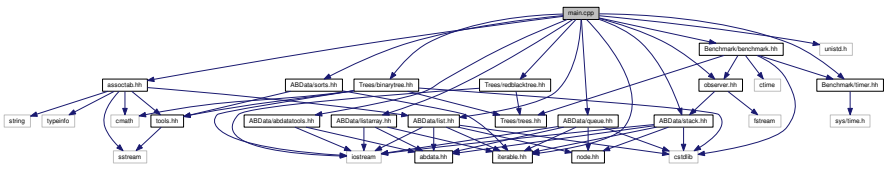
```
00001 #ifndef LISTARRAY_HH
00002 #define LISTARRAY_HH
00003
00004 #include <iostream>
00005 #include "iterable.hh"
00006 #include "abdata.hh"
00007
00012 template <class type>
00013 class ListArray: public ABData<type>, public Iterable<type>{
00019   int counter;
00025   int iterator;
00029   type *tab;
00030 public:
00036   ListArray(){
00037     tab = NULL;
00038     iterator = 0;
00039     counter = 0;
00040   }
00041
00047   ~ListArray(){delete[] tab;}
00048
00056   void push(const type elem);
00057
00063   void pop();
00064
00072   unsigned int size();
00073
00082   type& operator [] (const unsigned int index);
00083 };
00084
00085 template <class type>
00086 void ListArray<type>::push(const type elem){
00087   if(counter==0){
00088     tab = new type [1];
00089     counter=1;
00090     iterator=0;
00091     tab[iterator]=elem;
00092   }
00093   else{
00094     if(iterator<counter-1){
00095       tab[++iterator]=elem;
00096     }
00097     else if(iterator>=counter-1){
00098       type *tmp = new type[2*counter];
00099       for(int i=0;i<=iterator;i++)
00100     tmp[i] = tab[i];
00101       delete [] tab;
00102       tab = tmp;
00103       tab[++iterator]=elem;
00104       counter*=2;
00105     }
00106   }
00107 }
00108
00109 template <class type>
00110 void ListArray<type>::pop(){
00111  if(counter == 0){
00112     cerr<<"Lista jest pusta!"<<endl;
00113   }
00114   iterator--;
00115   if(iterator<0.25*(counter-1)){
00116     type *tmp = new type[iterator+1];
00117     for(int i=0;i<=iterator;i++){
00118       tmp[i]=tab[i];
00119     }
00120     delete [] tab;
00121     tab = tmp;
00122     counter = iterator+1;
00123   }
00124 }
00125
00126 template <class type>
00127 unsigned int ListArray<type>::size(){
00128   return iterator+1;
00129 }
00130
```

```
00131 template <class type>
00132 type& ListArray<type>::operator [] (const unsigned int index){
00133   if(index >= size()){
00134     std::cerr<<"Brak elementu o żądanym indeksie!"<<std::endl;
00135     exit(1);
00136   }
00137   else
00138     return tab[index];
00139 }
00140
00141 #endif
```

## 5.19 Dokumentacja pliku main.cpp

```
#include "ABData/list.hh"
#include "ABData/stack.hh"
#include "ABData/queue.hh"
#include "ABData/iterable.hh"
#include "Benchmark/timer.hh"
#include "Benchmark/benchmark.hh"
#include "Benchmark/observer.hh"
#include "Trees/binarytree.hh"
#include "ABData/sorts.hh"
#include "ABData/abdatatools.hh"
#include "ABData/listarray.hh"
#include "assoctab.hh"
#include "Trees/redblacktree.hh"
#include "unistd.h"
```

Wykres zależności załączania dla main.cpp:




### Funkcje

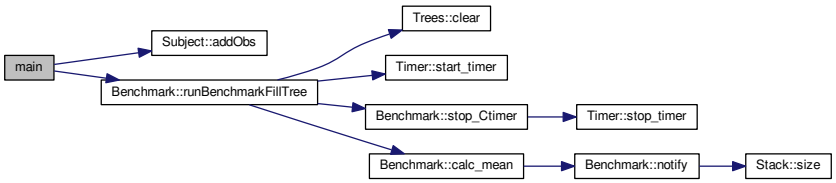
- int main ()

### 5.19.1 Dokumentacja funkcji

#### 5.19.1.1 int main ( )

Definicja w linii 18 pliku main.cpp.

Oto graf wywołań dla tej funkcji:



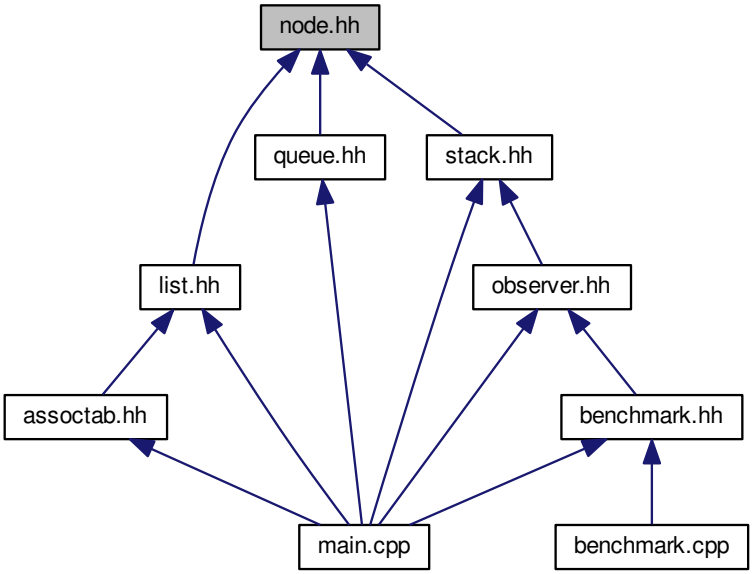

## 5.20 main.cpp

```
00001 #include "ABData/list.hh"
00002 #include "ABData/stack.hh"
00003 #include "ABData/queue.hh"
00004 #include "ABData/iterable.hh"
00005 #include "Benchmark/timer.hh"
00006 #include "Benchmark/benchmark.hh"
00007 #include "Benchmark/observer.hh"
00008 #include "Trees/binarytree.hh"
00009 #include "ABData/sorts.hh"
00010 #include "ABData/abdatatools.hh"
00011 #include "ABData/listarray.hh"
00012 #include "assoctab.hh"
00013 #include "Trees/redblacktree.hh"
00014 #include "unistd.h"
00015 using namespace std;
00016
00017
00018 int main(){
00019   Benchmark test;
00020   SaveToFile saver;
00021   test.addObs(&saver);
00022   // BinaryTree<int> object;
00023   RedBlackTree<int> object;
00024   for(int i=1; i<=10000000; i*=10)
00025     // test.runBenchmarkSearchTree(&Trees<int>::search, object, i, 10, (char*)"dane.dat");
00026     test.runBenchmarkFillTree(&Trees<int>::insert, object, i, 10, (
      char*)"dane.dat");
00027
00028   return 0;
00029 }
```

## 5.21 Dokumentacja pliku node.hh

Struktura node.

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




### Komponenty

- struct AssocData< typeKey, type >
- struct node< type >

### 5.21.1 Opis szczegółowy

Jest to struktura skladowa klasy List, zawierajaca przechowywana wartosc oraz wskaznik na zmienna typu node.

Definicja w pliku node.hh.

## 5.22 node.hh

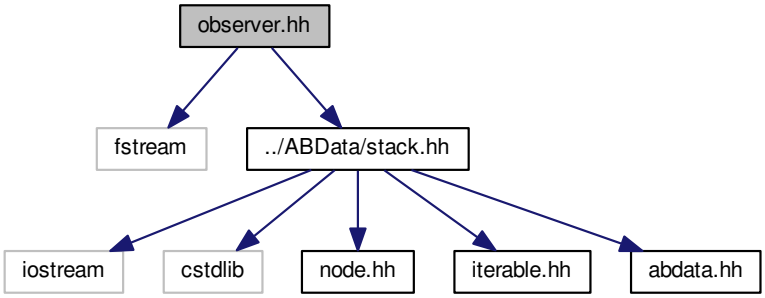
```
00001 #ifndef NODE_HH
00002 #define NODE_HH
00003
00004
00005 template<typename typeKey, class type>
00006 struct AssocData{
00007   typeKey key;
00008   type val;
00009
00010   AssocData(){}
00011   AssocData(typeKey k){key=k;}
00012   AssocData(typeKey k, type v){key=k; val=v;}
00013 };
00014
00015
00023 template <typename type>
00024 struct node{
00028   type val;
00032   node *next;
00036   node(){
00037     next=NULL;
00038   }
```

```
00042   node(type elem){
00043     val=elem;
00044     next=NULL;
00045   }
00046 };
00047
00048 #endif
```

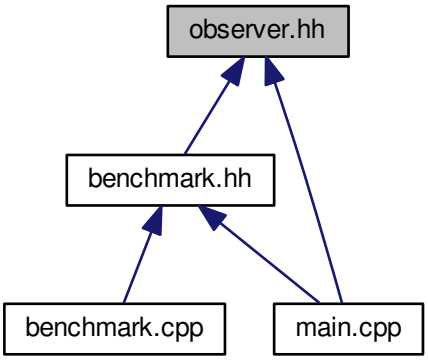
## 5.23 Dokumentacja pliku observer.hh

```
#include <fstream>
#include "../ABData/stack.hh"
```

Wykres zależności załączania dla observer.hh:




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




### Komponenty

- class Observer
- class Subject
- class SaveToFile

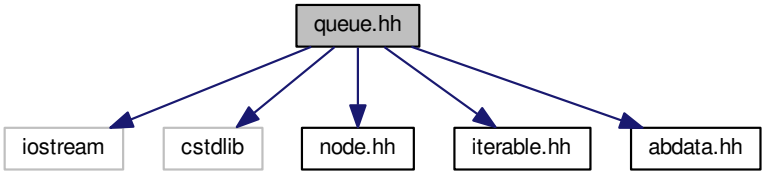
## 5.24 observer.hh

```
00001 #ifndef OBSERVER_HH
00002 #define OBSERVER_HH
00003
00004 #include <fstream>
00005 #include "../ABData/stack.hh"
00006 using namespace std;
00007 class Subject;
00008
00009 class Observer{
00010 public:/*
00011   Subject *model;
00012
00013   Observer(Subject *mod){
00014     model=mod;
00015     }*/
00016   virtual void update(int dataNumber, double mean)=0;
00017 };
00018
00019 class Subject{
00020 protected:
00021   Stack<Observer*> obss;
00022 public:
00023   void addObs(Observer* toadd){obss.push(toadd);}
00024   virtual void notify()=0;
00025 };
00026
00027 class SaveToFile: public Observer{
00028 public:
00029   /* SaveToFile(Benchmark *mod){
00030     model=mod;
00031     }*/
00032   void update(int dataNumber, double mean){
00033     ofstream wyniki;
00034     wyniki.open("wyniki.csv",ios::app);
00035     wyniki<<endl<<dataNumber<<","<<mean;
00036     wyniki.close();
00037   }
00038 };
00039
00040 /*void Subject::notify(){
00041   for(unsigned int i=0; i<obss.size();i++)
00042     obss[i]->update();
00043     }*/
00044
00045
00046 #endif
```

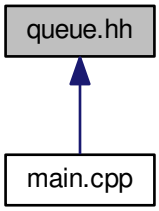
## 5.25 Dokumentacja pliku queue.hh

```
#include <iostream>
#include <cstdlib>
#include "node.hh"
#include "iterable.hh"
#include "abdata.hh"
```

Wykres zależności załączania dla queue.hh:



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




### Komponenty

- class Queue< type >

## 5.26 queue.hh

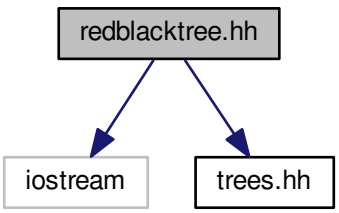
```
00001 #ifndef QUEUE_HH
00002 #define QUEUE_HH
00003
00004 #include <iostream>
00005 #include <cstdlib>
00006 #include "node.hh"
00007 #include "iterable.hh"
00008 #include "abdata.hh"
00009
00010 template <class type>
00011 class Queue: public ABData<type>, public Iterable<type>{
00017   node<type> *head;
00023   int iterator;
00024
00025 public:
00031   Queue(){
00032     head = NULL;
00033     iterator = 0;
00034   }
00035
00043   void push(const type elem);
00044
00050   void pop();
00051
00059   unsigned int size();
00060
00069   type& operator [] (const unsigned int index);
00070
00071   void display(){
00072     node<type> *ptr = head;
00073     while(ptr){
00074       std::cout<<ptr->val<<std::endl;
00075       ptr=ptr->next;
00076     }
00077   }
00078 };
00079
00080
00081 template <class type>
00082 void Queue<type>::push(const type elem){
00083   node<type> *toadd = new node<type>;
00084   toadd->val = elem;
00085   if(head == NULL){
00086     head = toadd;
00087   }
00088   else{
00089     node<type> *ptr = head;
00090     while(ptr->next)
00091       ptr=ptr->next;
```

```
00092     ptr->next = toadd;
00093   }
00094   iterator++;
00095 }
00096
00097 template <class type>
00098 void Queue<type>::pop(){
00099   if(!head)
00100     std::cerr<<"Kolejka jest pusta!"<<std::endl;
00101   else{
00102     node<type> *ptr = head;
00103     head = head->next;
00104     delete ptr;
00105     iterator--;
00106   }
00107 }
00108
00109 template <class type>
00110 unsigned int Queue<type>::size(){
00111   return iterator;
00112 }
00113
00114 template <class type>
00115 type& Queue<type>::operator [] (const unsigned int index){
00116   if(index >= size()){
00117     std::cerr<<"Brak elementu o żądanym indeksie!"<<std::endl;
00118     exit(1);
00119   }
00120   else{
00121     node<type> *ptr = head;
00122     for(unsigned int i=1; i<=index; i++)
00123       ptr=ptr->next;
00124     return ptr->val;
00125     }
00126 }
00127
00128 #endif
```

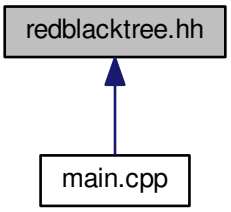
## 5.27 Dokumentacja pliku redblacktree.hh

```
#include <iostream>
#include "trees.hh"
```

Wykres zależności załączania dla redblacktree.hh:



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




### Komponenty

- struct rbtreenode< type >
- class RedBlackTree< type >

  *Klasa RedBlackTree - drzewo czerwono-czarne.*

### Definicje

- #define REBBLACKTREE_HH

### 5.27.1 Dokumentacja definicji

#### 5.27.1.1 #define REBBLACKTREE_HH

Definicja w linii 2 pliku redblacktree.hh.

## 5.28 redblacktree.hh

```
00001 #ifndef REDBLACKTREE_HH
00002 #define REBBLACKTREE_HH
00003
00004 #include <iostream>
00005 #include "trees.hh"
00006
00007 template <typename type>
00008 struct rbtreenode{
00012   type val;
00016   rbtreenode *left;
00020   rbtreenode *right;
00024   rbtreenode *parent;
00031   char color;
00039   rbtreenode(type elem){val=elem; left=NULL; right=NULL;
      color='b';}
00040 };
00041
00045 template <typename type>
00046 class RedBlackTree: public Trees<type>{
00047 private:
00051   rbtreenode<type> *root;
00055   rbtreenode<type> *sentinel;
00056
00060   void print(rbtreenode<type> *root);
00064   rbtreenode<type> * remove(rbtreenode<type> *
      root, const type elem);
00068   void init(rbtreenode<type> *toInit, type data);
00076   rbtreenode<type>* findSuitableParent(type data);
```

```
00083   void createBindings(rbtreenode<type> *parent,
      rbtreenode<type> *child);
00089   rbtreenode<type> * findMin(rbtreenode<type> *
      node);
00095   void setNewRoot(rbtreenode<type> *elem);
00099   void rotateRight(rbtreenode<type> *elem);
00103   void rotateLeft(rbtreenode<type> *elem);
00104
00105   void correct(rbtreenode<type> *elem);
00106   void correctLeft(rbtreenode<type> *elem);
00107   void correctRight(rbtreenode<type> *elem);
00108
00116   void deleteTree(rbtreenode<type> *node);
00117
00118 public:
00122   RedBlackTree(){
00123     root = NULL;
00124     sentinel = new rbtreenode<type>(type());
00125     sentinel->color = 'b';
00126     sentinel->left = NULL;
00127     sentinel->right = NULL;
00128   }
00132   ~RedBlackTree(){
00133     deleteTree(root);
00134   }
00142   void insert(const type data);
00146   bool remove(const type elem);
00157   bool search(const type elem);
00164   void print();
00170   void clear(){deleteTree(root); root=NULL;}
00171
00172 };
00173 /*****************************************/
00174
00175 /*****************************************/
00176 template <class type>
00177 void RedBlackTree<type>::insert(const type data){
00178   rbtreenode<type> *elem = new rbtreenode<type>(type());
00179   init(elem, data);
00180   rbtreenode<type> *parent = findSuitableParent(data);
00181   createBindings(parent, elem);
00182   correct(elem);
00183 }
00184 /*****************************************/
00185
00186 /*****************************************/
00187 template <class type>
00188 bool RedBlackTree<type>::search(const type elem){
00189   rbtreenode<type> *ptr = root;
00190   while(true){
00191     if(ptr == NULL || ptr == sentinel)
00192       return false;
00193     else if(elem == ptr->val)
00194       return true;
00195     else if(elem < ptr->val)
00196       ptr = ptr->left;
00197     else
00198       ptr = ptr->right;
00199   }
00200 }
00201 /*****************************************/
00202
00203 /*****************************************/
00204 template <class type>
00205 bool RedBlackTree<type>::remove(const type elem){
00206   if(root == NULL || root == sentinel)
00207     return false;
00208   if(elem < root->val)
00209     root->left = remove(root->left, elem);
00210   else if(elem > root->val)
00211     root->right = remove(root->right, elem);
00212   else{
00213     if(root->left == NULL || root->left == sentinel){
00214       rbtreenode<type> *tmp = root;
00215       root = root->right;
00216       delete tmp;
00217       return true;
00218     }
00219     else if(root->right == NULL || root->right == sentinel){
00220       rbtreenode<type> *tmp = root;
00221       root = root->left;
00222       delete tmp;
00223       return true;
00224     }
00225     rbtreenode<type> *tmp = findMin(root->right);
00226     root->val = tmp->val;
00227     root->right = remove(root->right, tmp->val);
```

```
00228   }
00229   return true;
00230 }
00231 /****************************************/
00232
00233 /****************************************/
00234 template <class type>
00235 rbtreenode<type> * RedBlackTree<type>::remove(
      rbtreenode<type> *root, const type elem){
00236   if(root == NULL || root == sentinel)
00237     return root;
00238   if(elem < root->val)
00239     root->left = remove(root->left, elem);
00240   else if(elem > root->val)
00241     root->right = remove(root->right, elem);
00242   else{
00243     if(root->left == NULL || root->left == sentinel){
00244       rbtreenode<type> *tmp = root;
00245       root = root->right;
00246       delete tmp;
00247       return root;
00248     }
00249     else if(root->right == NULL || root->right == sentinel){
00250       rbtreenode<type> *tmp = root;
00251       root = root->left;
00252       delete tmp;
00253       return root;
00254     }
00255     rbtreenode<type> *tmp = findMin(root->right);
00256     root->val = tmp->val;
00257     root->right = remove(root->right, tmp->val);
00258   }
00259   return root;
00260 }
00261 /****************************************/
00262
00263 /****************************************/
00264 template <class type>
00265 void RedBlackTree<type>::print(){
00266   if(root != NULL && root != sentinel){
00267     std::cout << root->val << " ";
00268     print(root->left);
00269     print(root->right);
00270   }
00271 }
00272 /****************************************/
00273
00274 /****************************************/
00275 template <class type>
00276 void RedBlackTree<type>::print(rbtreenode<type> *root){
00277   if(root != NULL && root != sentinel){
00278     std::cout << root-> val << " ";
00279     print(root->left);
00280     print(root->right);
00281   }
00282 }
00283 /****************************************/
00284
00285 /****************************************/
00286 template <class type>
00287 void RedBlackTree<type>::init(rbtreenode<type> *toInit, type data){
00288   toInit->color = 'r';
00289   toInit->val = data;
00290   toInit->right = sentinel;
00291   toInit->left = sentinel;
00292 }
00293 /****************************************/
00294
00295 /****************************************/
00296 template <class type>
00297 rbtreenode<type>* RedBlackTree<type>::findSuitableParent
      (type data){
00298   rbtreenode<type> *parent = NULL;
00299   rbtreenode<type> *x = root;
00300   while(x != NULL){
00301     if(x != sentinel){
00302       parent = x;
00303       if(data < x->val)
00304     x = x->left;
00305       else
00306     x = x->right;
00307     }
00308     else
00309       break;
00310   }
00311   return parent;
00312 }
```

```
00313 /****************************************/
00314
00315 /****************************************/
00316 template <class type>
00317 void RedBlackTree<type>::rotateRight(
      rbtreenode<type> *elem){
00318   if(elem != NULL && elem != sentinel){
00319     if(elem->left != sentinel){
00320       rbtreenode<type> *tmp = elem->left;
00321       elem->left = tmp->right;
00322       tmp->right->parent = elem;
00323       tmp->right = elem;
00324       if(elem->parent != NULL){
00325     if(elem->parent->left == elem)
00326       elem->parent->left = tmp;
00327     else
00328       elem->parent->right = tmp;
00329     tmp->parent = elem->parent;
00330       }
00331       elem->parent = tmp;
00332       if(elem == root)
00333     setNewRoot(tmp);
00334     }
00335   }
00336 }
00337 /****************************************/
00338
00339 /****************************************/
00340 template <class type>
00341 void RedBlackTree<type>::rotateLeft(
      rbtreenode<type> *elem){
00342   if(elem != NULL && elem != sentinel){
00343     if(elem->right != sentinel){
00344       rbtreenode<type> *tmp = elem->right;
00345       elem->right = tmp->left;
00346       tmp->left->parent = elem;
00347       tmp->left = elem;
00348       if(elem->parent != NULL){
00349     if(elem->parent->left == elem)
00350       elem->parent->left = tmp;
00351     else
00352       elem->parent->right = tmp;
00353     tmp->parent = elem->parent;
00354       }
00355       elem->parent = tmp;
00356       if(elem == root)
00357     setNewRoot(tmp);
00358     }
00359   }
00360 }
00361 /****************************************/
00362
00363 /****************************************/
00364 template <class type>
00365 void RedBlackTree<type>::setNewRoot(
      rbtreenode<type> *elem){
00366   elem->parent = sentinel;
00367   elem->color = 'b';
00368   root = elem;
00369 }
00370 /****************************************/
00371
00372 /****************************************/
00373 template <class type>
00374 void RedBlackTree<type>::createBindings(
      rbtreenode<type> *parent, rbtreenode<type> *child){
00375   if(parent == NULL){
00376     child->parent = NULL;
00377     root = child;
00378     root->color = 'b';
00379   }
00380   else{
00381     child->parent = parent;
00382     if(child->val >= parent->val)
00383       parent->right = child;
00384     else
00385       parent->left = child;
00386   }
00387 }
00388 /****************************************/
00389
00390 /****************************************/
00391 template <class type>
00392 void RedBlackTree<type>::correct(rbtreenode<type> *elem){
00393   while(elem != root && elem->parent->color == 'r'){
00394     if(elem->parent == elem->parent->parent->left)
00395       correctLeft(elem);
```

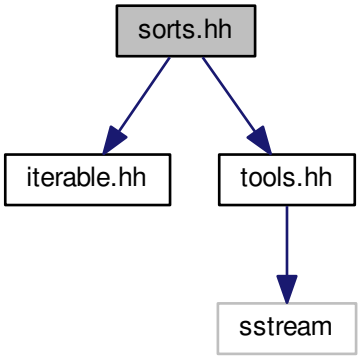
```
00396     else
00397       correctRight(elem);
00398   }
00399 }
00400 /****************************************/
00401
00402 /****************************************/
00403 template <class type>
00404 void RedBlackTree<type>::correctLeft(
      rbtreenode<type> *elem){
00405   if(elem->parent->parent->right->color == 'r'){
00406     elem = elem->parent->parent;
00407     elem->right->color = 'b';
00408     elem->left->color = 'b';
00409     if(elem != root)
00410       elem->color = 'r';
00411     correct(elem);
00412   }
00413   else{
00414     if(elem == elem->parent->right){
00415       rotateLeft(elem->parent);
00416       rotateRight(elem->parent);
00417       elem->color = 'b';
00418       elem->left->color = 'r';
00419       elem->right->color = 'r';
00420     }
00421     else{
00422       rotateRight(elem->parent->parent);
00423       elem = elem->parent;
00424       elem->color = 'b';
00425       elem->left->color = 'r';
00426       elem->right->color = 'r';
00427     }
00428   }
00429 }
00430 /****************************************/
00431
00432 /****************************************/
00433 template <class type>
00434 void RedBlackTree<type>::correctRight(
      rbtreenode<type> *elem){
00435   if(elem->parent->parent->left->color == 'r'){
00436     elem = elem->parent->parent;
00437     elem->right->color = 'b';
00438     elem->left->color = 'b';
00439     elem->color = 'r';
00440     correct(elem);
00441   }
00442   else{
00443     if(elem == elem->parent->left){
00444       elem = elem->parent;
00445       rotateRight(elem);
00446     }
00447     elem->parent->color = 'b';
00448     elem->parent->parent->color = 'r';
00449     rotateLeft(elem->parent->parent);
00450   }
00451   root->color = 'b';
00452 }
00453 /****************************************/
00454
00455 /****************************************/
00456 template <class type>
00457 rbtreenode<type> * RedBlackTree<type>::findMin(
      rbtreenode<type> *node){
00458   rbtreenode<type> *ptr = node;
00459   while(ptr->left != NULL && ptr->left != sentinel)
00460     ptr = ptr->left;
00461   return ptr;
00462 }
00463 /****************************************/
00464
00465 /****************************************/
00466 template <class type>
00467 void RedBlackTree<type>::deleteTree(
      rbtreenode<type> *node){
00468   if(node && node!=sentinel){
00469     deleteTree(node->left);
00470     deleteTree(node->right);
00471     delete node;
00472   }
00473 }
00474 #endif
```

## 5.29 Dokumentacja pliku sorts.hh

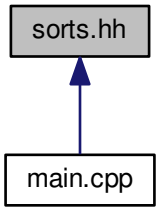
W pliku znajduja sie definicje metod sortujacych obiekty dziedziczace z klasy Iterable - takie ktore maja zdefiniowane operatory indeksowania []. Przykladowe wywolanie metody sortujace caly obiekt: Stack stos; insertsort(stos, stos.-size()-1)

```
#include "iterable.hh"
#include "tools.hh"
```

Wykres zależności załączania dla sorts.hh:




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




### Funkcje

- template<typename type >
  void insertsort (Iterable< type > &tosort, int left, int right)

  *Sortowanie przez wstawianie.*
- template<typename type >
  void quicksort (Iterable< type > &tosort, int left, int right)

  *Sortowanie szybkie.*

### 5.29.1 Dokumentacja funkcji

#### 5.29.1.1 template<typename type > void insertsort ( Iterable< type > & *tosort,* int *left,* int *right* )

Dokonuje sortowania obiektu stosujac metode sortowania przez wstawianie

**Parametry**

| | | |
|---|---|---|
| in | *&tosort* | Referencja do obiektu typu Iterable, ktory chcemy posortowac |
| in | *left* | Poczatek zakresu sortowania |
| in | *right* | Koniec zakresu sortowania |

Definicja w linii 26 pliku sorts.hh.

#### 5.29.1.2 template<typename type > void quicksort ( Iterable< type > & *tosort,* int *left,* int *right* )

Dokonuje sortowania obiektu stosujac metode sortowania szybkiego

**Parametry**

| | | |
|---|---|---|
| in | *&tosort* | Referencja do obiektu typu Iterable, ktory chcemy posortowac |
| in | *left* | Poczatek zakresu sortowania |
| in | *right* | Koniec zakresu sortowania |

Definicja w linii 46 pliku sorts.hh.

Oto graf wywołań dla tej funkcji:


## 5.30 sorts.hh

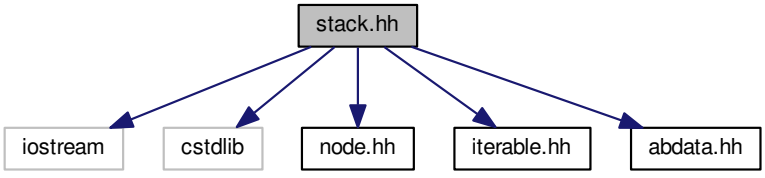
```
00001 #ifndef SORTS_HH
00002 #define SORTS_HH
00003
00004 #include "iterable.hh"
00005 #include "tools.hh"
00006
00025 template<typename type>
00026 void insertsort(Iterable<type> &tosort, int left, int right){
00027   int i,j; int temp;
00028   for(i=left; i<=right; ++i){
00029     temp=tosort[i];
00030     for(j=i; j>left && temp<tosort[j-1]; --j)
00031       tosort[j] = tosort[j-1];
00032     tosort[j]=temp;
00033   }
00034 }
00035
00045 template<typename type>
00046 void quicksort(Iterable<type> &tosort, int left, int right){
00047   int i=(right+left)/2;
00048   int j=0;
00049
00050   if(tosort[right]<tosort[left])
00051     substitute(tosort[right],tosort[left]);
00052   if(tosort[i] < tosort[left])
00053     substitute(tosort[i],tosort[left]);
00054   if(tosort[right]<tosort[i])
```

```
00055     substitute(tosort[right],tosort[i]);
00056
00057   int piwot=tosort[i];
00058   i=left; j = right;
00059   do{
00060     while(tosort[i]<piwot) i++;
00061     while(tosort[j]>piwot) j--;
00062     if(i<=j){
00063       substitute(tosort[i],tosort[j]);
00064       i++; j--;
00065     }
00066   }while(i<=j);
00067
00068   if(j>left)
00069     quicksort(tosort, left,j);
00070   if(i<right)
00071     quicksort(tosort, i,right);
00072 }
00073 #endif
```

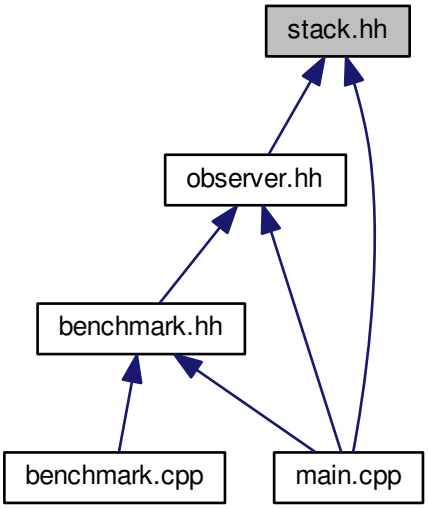
## 5.31 Dokumentacja pliku stack.hh

```
#include <iostream>
#include <cstdlib>
#include "node.hh"
#include "iterable.hh"
#include "abdata.hh"
```

Wykres zależności załączania dla stack.hh:



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




### Komponenty

- class Stack< type >

## 5.32 stack.hh

```
00001 #ifndef STACK_HH
00002 #define STACK_HH
00003
00004 #include <iostream>
00005 #include <cstdlib>
00006 #include "node.hh"
00007 #include "iterable.hh"
00008 #include "abdata.hh"
00009
00010
00011 template <class type>
00012 class Stack: public ABData<type>, public Iterable<type>{
00013
00019   node<type> *head;
00025   int iterator;
00026
00027 public:
00033   Stack(){
00034     head = NULL;
00035     iterator = 0;
00036   }
00037
00045   void push(const type elem);
00046
00052   void pop();
00053
00061   unsigned int size();
00062
00071   type& operator [] (const unsigned int index);
00072
00073
00074 void display(){
00075   node<type> *ptr = head;
00076   while(ptr){
00077     std::cout<<ptr->val<<std::endl;
```

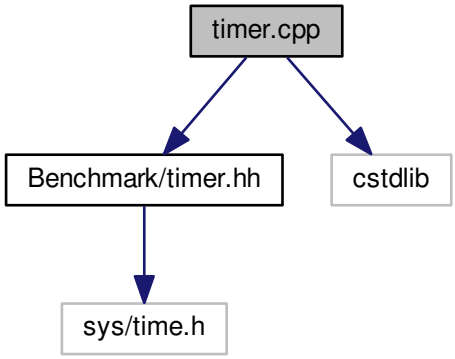
```
00078     ptr=ptr->next;
00079   }
00080 }
00081 };
00082
00083 template <class type>
00084 void Stack<type>::push(const type elem){
00085   node<type> *toadd = new node<type>;
00086   toadd->val = elem;
00087   node<type> *ptr = head;
00088   head = toadd;
00089   toadd->next = ptr;
00090   iterator++;
00091 }
00092
00093 template <class type>
00094 void Stack<type>::pop(){
00095   if(!head)
00096     std::cerr<<"Stos jest pusty!"<<std::endl;
00097   else{
00098     node<type> *ptr = head;
00099     head = head->next;
00100     delete ptr;
00101     iterator--;
00102   }
00103 }
00104
00105 template <class type>
00106 unsigned int Stack<type>::size(){
00107   return iterator;
00108 }
00109
00110 template <class type>
00111 type& Stack<type>::operator [] (const unsigned int index){
00112   if(index >= size()){
00113     std::cerr<<"Brak elementu o żądanym indeksie!"<<std::endl;
00114     exit(1);
00115   }
00116   else{
00117     node<type> *ptr = head;
00118     for(unsigned int i=1; i<=index; i++)
00119       ptr=ptr->next;
00120     return ptr->val;
00121     }
00122 }
00123 #endif
```

## 5.33 Dokumentacja pliku timer.cpp

Ciala metod klasy Timer.

```
#include "Benchmark/timer.hh"
#include <cstdlib>
```

Wykres zależności załączania dla timer.cpp:



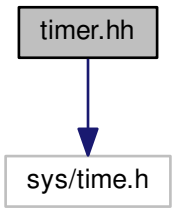

## 5.34 timer.cpp

```
00001 #include "Benchmark/timer.hh"
00002 #include <cstdlib>
00003
00008 void Timer::start_timer(){
00009   gettimeofday(&start, NULL);
00010
00011 }
00012
00013 void Timer::stop_timer(){
00014   gettimeofday(&end, NULL);
00015   atime = (end.tv_sec - start.tv_sec) * 1000.0;      // sec to ms
00016   atime += (end.tv_usec - start.tv_usec) / 1000.0;   // us to ms
00017 }
00018
00019 double Timer::getTime(){
00020   return atime;
00021 }
```

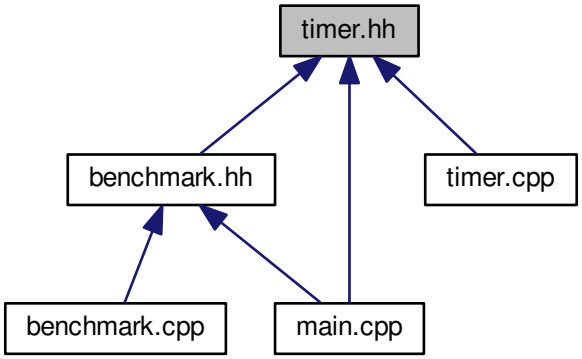
## 5.35 Dokumentacja pliku timer.hh

Klasa Timer.

`#include <sys/time.h>`
Wykres zależności załączania dla timer.hh:




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




### Komponenty

- class Timer

### 5.35.1 Opis szczegółowy

Służy do pomiaru czasu

Definicja w pliku timer.hh.

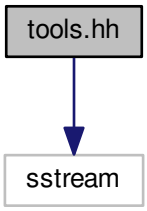
## 5.36 timer.hh

```
00001 #ifndef TIMER_HH
00002 #define TIMER_HH
00003
00004 #include <sys/time.h>
00005
```

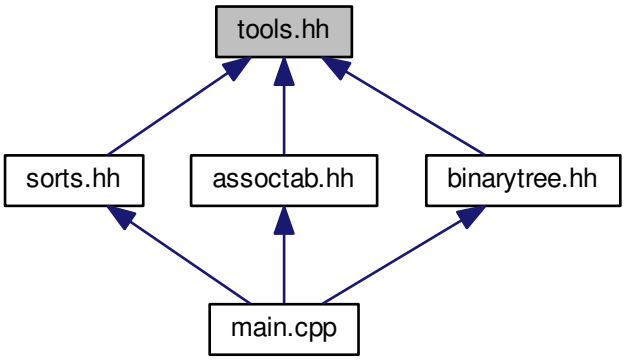
```
00012 class Timer{
00013 protected:
00019   timeval start, end;
00020
00026   double atime;
00027
00028 public:
00032   Timer(){atime=0;}
00036   void start_timer();
00043   void stop_timer();
00044
00050   double getTime();
00051 };
00052
00053 #endif
```

## 5.37 Dokumentacja pliku tools.hh

`#include <sstream>`
Wykres zależności załączania dla tools.hh:




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




### Funkcje

- template<typename type >

void substitute (type &val1, type &val2)

*Plik zawiera definicje roznych przydatnych funkcji.*

- template<typename type >
  std::string tostring (const type &toConvert)

### 5.37.1 Dokumentacja funkcji

#### 5.37.1.1 template<typename type > void substitute ( type & *val1*, type & *val2* )

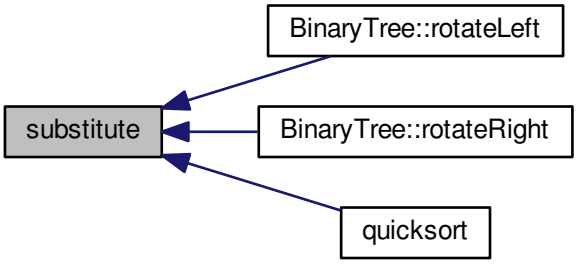
Funkcja zamieia ze soba dwie wartosci podane w argumentach

**Parametry**

| | | |
|---|---|---|
| in | *val1* | Pierwsza wartosc do zamiany |
| in | *val2* | Druga wartosc do zamiany |

Definicja w linii 17 pliku tools.hh.

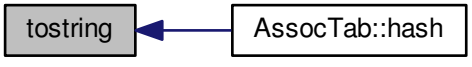
Oto graf wywoływań tej funkcji:




#### 5.37.1.2 template<typename type > std::string tostring ( const type & *toConvert* )

Definicja w linii 24 pliku tools.hh.

Oto graf wywoływań tej funkcji:




## 5.38 tools.hh

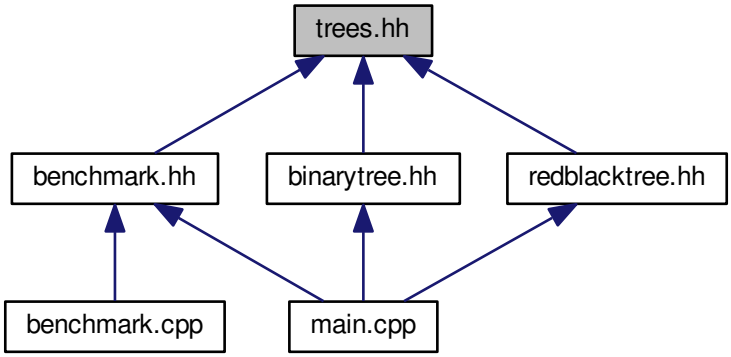
```
00001 #ifndef TOOLS_HH
```

```
00002 #define TOOLS_HH
00003
00004 #include <sstream>
00005
00016 template <typename type>
00017 void substitute(type& val1, type& val2){
00018   type tmp = val1;
00019   val1 = val2;
00020   val2 = tmp;
00021 }
00022
00023 template <typename type>
00024 std::string tostring(const type& toConvert){
00025   std::ostringstream os;
00026   os << toConvert;
00027   return os.str();
00028 }
00029
00030 #endif
```

## 5.39 Dokumentacja pliku trees.hh

Definicja interfejsu dla drzew.

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




### Komponenty

- class Trees< type >

  *Klasa abstrakcyjna zawierajaca metody wirtualne drzew.*

## 5.40 trees.hh

```
00001 #ifndef TREES_HH
00002 #define TREES_HH
00003
00012 template <class type>
00013 class Trees{
00014 public:
00015   virtual void insert(const type elem)=0;
00016   virtual bool remove(const type elem)=0;
00017   virtual bool search(const type elem)=0;
00018   virtual void clear()=0;
00019 };
00020
```

```
00021 #endif
```

# Skorowidz