PAMSI

0.1

Wygenerowano przez Doxygen 1.8.6

Wt, 19 maj 2015 16:48:48

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

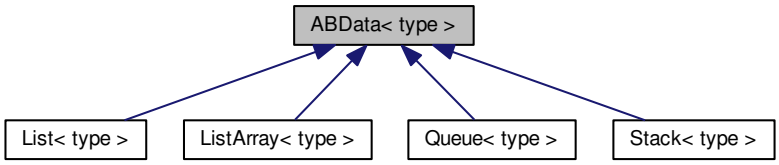
Dokumentacja klas

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 >

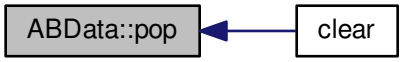
Definicja w linii 16 pliku abdata.hh.

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]

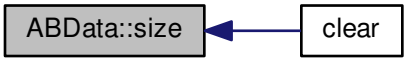
Implementowany w ListArray< type >, List< type >, List< AssocData< typeKey, type > >, Stack< type >, Stack< Observer * > i Queue< type >.

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.

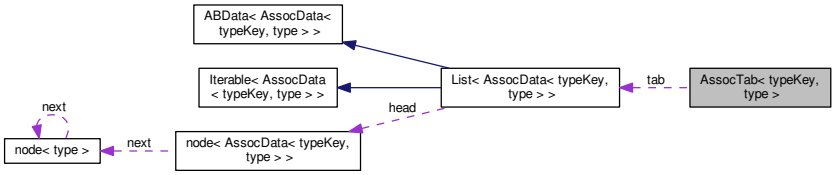
Dokumentacja dla tej struktury została wygenerowana z pliku:

- 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*)

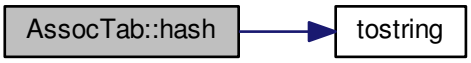
Dokonuje haszowania podanego klucza na wartosc liczbowa.

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*)

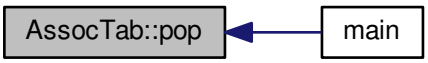
Usuwa z tablicy element odpowiadajacy podanemu kluczowi.

Parametry

in	*toremoveKey*	Klucz odpowiadajacy elementowi, ktory chcemy usunac

Definicja w linii 148 pliku assoctab.hh.

Oto graf wywoływań tej funkcji:



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

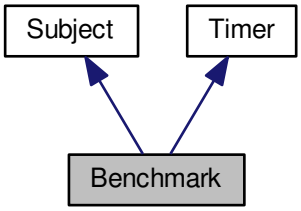
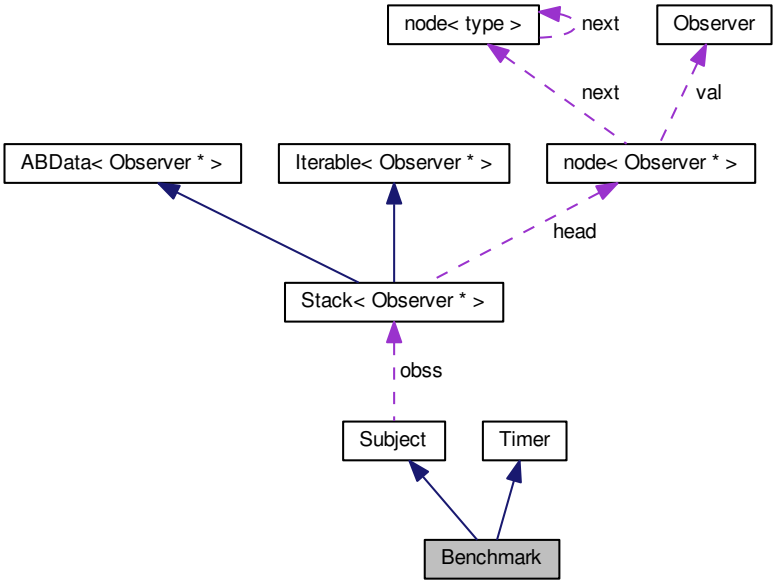


Diagram współpracy dla Benchmark:



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.

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 17 pliku benchmark.hh.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Benchmark::Benchmark ( ) [inline]

Definicja w linii 35 pliku benchmark.hh.

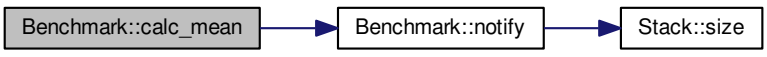
4.4.3 Dokumentacja funkcji składowych

4.4.3.1 void Benchmark::calc_mean ( )

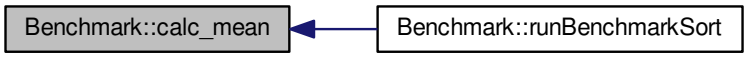
Dzieli sume pomiarow przez ich liczbe i zapisuje do zmiennej mean. Wysyla powiadomienie do obserwatorow.

Definicja w linii 19 pliku benchmark.cpp.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

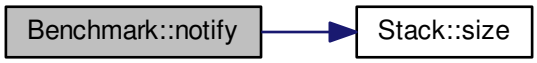


4.4.3.2 void Benchmark::notify ( ) [virtual]

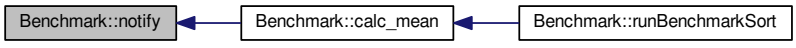
Implementuje Subject.

Definicja w linii 8 pliku benchmark.cpp.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



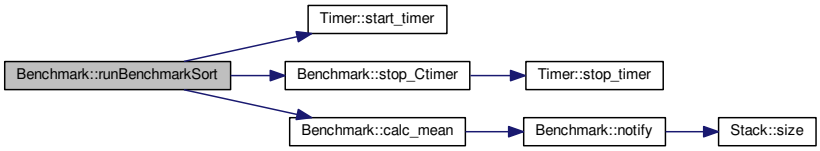
4.4.3.3 template<typename type > void Benchmark::runBenchmarkSort (void(*)(Iterable< type > &, int, int) *f,* Iterable< type > & *container,* int *dataCount,* int *repeats*)

Parametry

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

Definicja w linii 74 pliku benchmark.hh.

Oto graf wywołań dla tej funkcji:

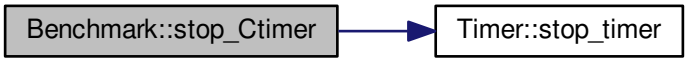


4.4.3.4 void Benchmark::stop_Ctimer ( )

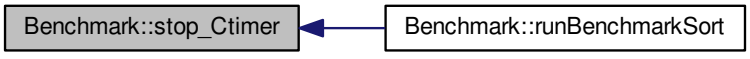
Zapisuje moment zakonczenia pomiaru do zmiennej end, oblicza zmierzony czas i zapisuje do zmiennej time, zwieksza counter o 1.

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 33 pliku benchmark.hh.

4.4.4.2 int Benchmark::counter [private]

Definicja w linii 29 pliku benchmark.hh.

4.4.4.3 double Benchmark::mean [private]

Definicja w linii 25 pliku benchmark.hh.

4.4.4.4 double Benchmark::total [private]

Definicja w linii 21 pliku benchmark.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

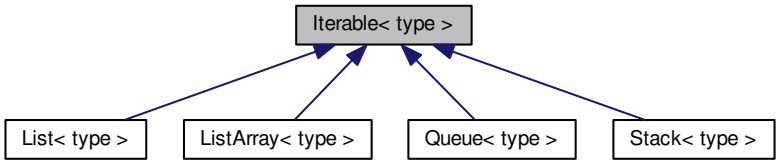
- benchmark.hh
- benchmark.cpp

4.5 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

4.5.1 Opis szczegółowy

template<class type>class Iterable< type >

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

Definicja w linii 15 pliku iterable.hh.

4.5.2 Dokumentacja funkcji składowych

4.5.2.1 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

4.6 Dokumentacja szablonu klasy List< type >

`#include <list.hh>`

Diagram dziedziczenia dla List< type >

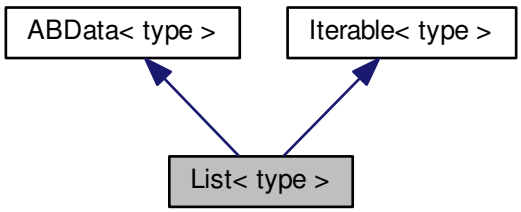
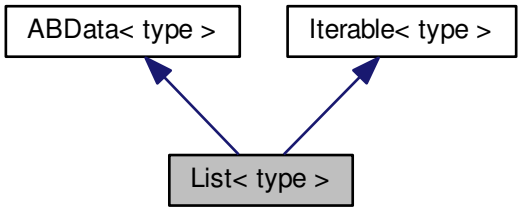


Diagram współpracy dla List< type >:



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.

4.6.1 Opis szczegółowy

template<class type>class List< type >

Definicja w linii 15 pliku list.hh.

4.6.2 Dokumentacja konstruktora i destruktora

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

Ustawia poczatek listy na NULL

Definicja w linii 34 pliku list.hh.

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 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.

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

Usuwa pierwszy element listy.

Implementuje ABData< type >.

Definicja w linii 97 pliku list.hh.

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

Usuwa element o wybranym indeksie z listy.

Definicja w linii 109 pliku list.hh.

4.6.3.4 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.

4.6.3.5 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.

4.6.4 Dokumentacja atrybutów składowych

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

Wskaznik na pierwszy element listy

Definicja w linii 21 pliku list.hh.

4.6.4.2 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

4.7 Dokumentacja szablonu klasy ListArray< type >

`#include <listarray.hh>`

Diagram dziedziczenia dla ListArray< type >

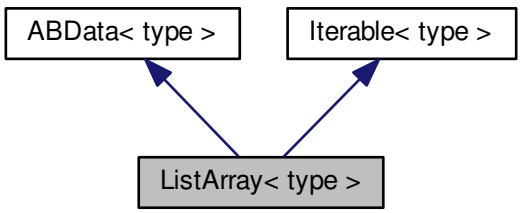
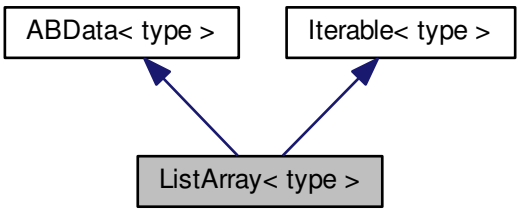


Diagram współpracy dla ListArray< type >:



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.

4.7.1 Opis szczegółowy

template<class type>class ListArray< type >

Definicja w linii 13 pliku listarray.hh.

4.7.2 Dokumentacja konstruktora i destruktora

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

Ustawia wskaznik na tablice na NULL, iterator na 0

Definicja w linii 36 pliku listarray.hh.

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

Usuwa dynamicznie utworzona tablice danych

Definicja w linii 47 pliku listarray.hh.

4.7.3 Dokumentacja funkcji składowych

4.7.3.1 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.

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

Usuwa ostatni element listy.

Implementuje ABData< type >.

Definicja w linii 110 pliku listarray.hh.

4.7.3.3 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.

4.7.3.4 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.

4.7.4 Dokumentacja atrybutów składowych

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

Przechowuje informacje o liczbie elementow znajdujacych sie na liscie

Definicja w linii 19 pliku listarray.hh.

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

Przechowuje informacje o aktualnej pozycji ostatniego elementu w tablicy

Definicja w linii 25 pliku listarray.hh.

4.7.4.3 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

4.8 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.

4.8.1 Opis szczegółowy

template<typename type>struct node< type >

Definicja w linii 24 pliku node.hh.

4.8.2 Dokumentacja konstruktora i destruktora

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

Definicja w linii 36 pliku node.hh.

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

Definicja w linii 42 pliku node.hh.

4.8.3 Dokumentacja atrybutów składowych

4.8.3.1 template<typename type> node* node< type >::next

Definicja w linii 32 pliku node.hh.

4.8.3.2 template<typename type> type node< type >::val

Definicja w linii 28 pliku node.hh.

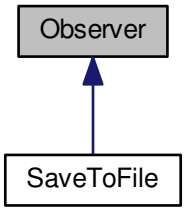
Dokumentacja dla tej struktury została wygenerowana z pliku:

- node.hh

4.9 Dokumentacja klasy Observer

`#include <observer.hh>`

Diagram dziedziczenia dla Observer



Metody publiczne

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

4.9.1 Opis szczegółowy

Definicja w linii 9 pliku observer.hh.

4.9.2 Dokumentacja funkcji składowych

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

Implementowany w SaveToFile.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- observer.hh

4.10 Dokumentacja szablonu klasy Queue< type >

`#include <queue.hh>`

Diagram dziedziczenia dla Queue< type >

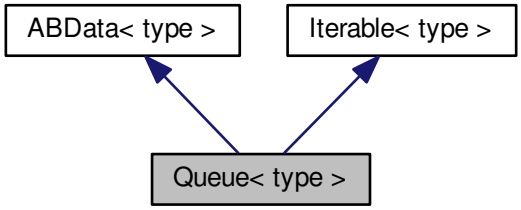
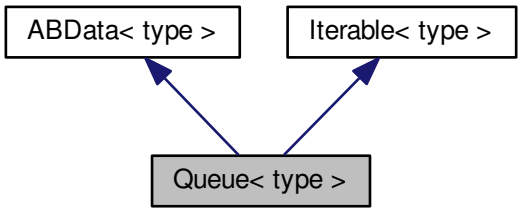


Diagram współpracy dla Queue< type >:



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.

4.10.1 Opis szczegółowy

template<class type>class Queue< type >

Definicja w linii 11 pliku queue.hh.

4.10.2 Dokumentacja konstruktora i destruktora

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

Ustawia poczatek listy na NULL

Definicja w linii 31 pliku queue.hh.

4.10.3 Dokumentacja funkcji składowych

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

Definicja w linii 71 pliku queue.hh.

4.10.3.2 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.

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

Usuwa pierwszy element stosu.

Implementuje ABData< type >.

Definicja w linii 98 pliku queue.hh.

4.10.3.4 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.

4.10.3.5 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.

4.10.4 Dokumentacja atrybutów składowych

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

Wskaznik na pierwszy element kolejki

Definicja w linii 17 pliku queue.hh.

4.10.4.2 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

4.11 Dokumentacja klasy SaveToFile

`#include <observer.hh>`

Diagram dziedziczenia dla SaveToFile

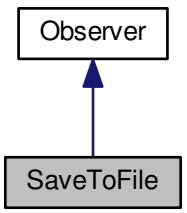
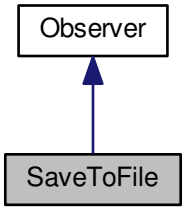


Diagram współpracy dla SaveToFile:



Metody publiczne

- void update (int dataNumber, double mean)

4.11.1 Opis szczegółowy

Definicja w linii 27 pliku observer.hh.

4.11.2 Dokumentacja funkcji składowych

4.11.2.1 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

4.12 Dokumentacja szablonu klasy Stack< type >

```
#include <stack.hh>
```

Diagram dziedziczenia dla Stack< type >

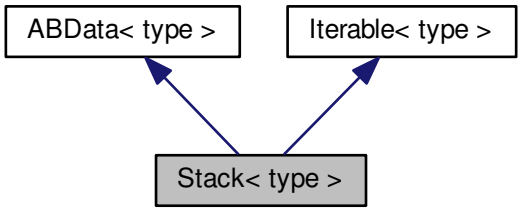
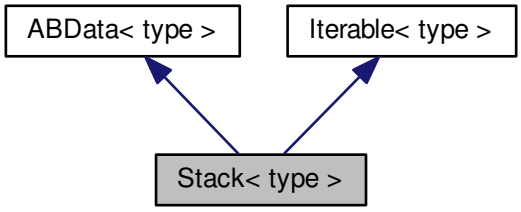


Diagram współpracy dla Stack< type >:



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.

4.12.1 Opis szczegółowy

template<class type>class Stack< type >

Definicja w linii 12 pliku stack.hh.

4.12.2 Dokumentacja konstruktora i destruktora

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

Ustawia poczatek listy na NULL

Definicja w linii 33 pliku stack.hh.

4.12.3 Dokumentacja funkcji składowych

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

Definicja w linii 74 pliku stack.hh.

4.12.3.2 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.

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

Usuwa pierwszy element stosu.

Implementuje ABData< type >.

Definicja w linii 94 pliku stack.hh.

4.12.3.4 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.

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

Daje informacje o rozmiarze stosu (liczbie jego elementow).

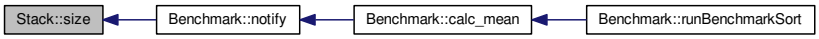
Zwraca

Rozmiar stosu (liczba jego elementow)

Implementuje ABData< type >.

Definicja w linii 106 pliku stack.hh.

Oto graf wywoływań tej funkcji:



4.12.4 Dokumentacja atrybutów składowych

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

Wskaznik na pierwszy element stosu

Definicja w linii 19 pliku stack.hh.

4.12.4.2 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

4.13 Dokumentacja klasy Subject

`#include <observer.hh>`

Diagram dziedziczenia dla Subject

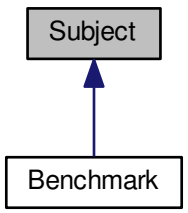
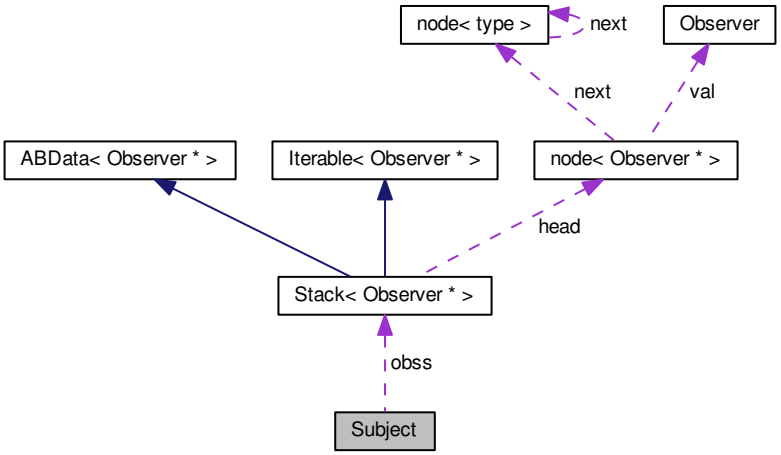


Diagram współpracy dla Subject:



Metody publiczne

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

Atrybuty chronione

- Stack< Observer ∗ > obss

4.13.1 Opis szczegółowy

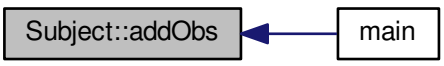
Definicja w linii 19 pliku observer.hh.

4.13.2 Dokumentacja funkcji składowych

4.13.2.1 void Subject::addObs (Observer ∗ *toadd*) [inline]

Definicja w linii 23 pliku observer.hh.

Oto graf wywoływań tej funkcji:



4.13.2.2 virtual void Subject::notify ( ) [pure virtual]

Implementowany w Benchmark.

4.13.3 Dokumentacja atrybutów składowych

4.13.3.1 Stack<Observer*> Subject::obss [protected]

Definicja w linii 21 pliku observer.hh.

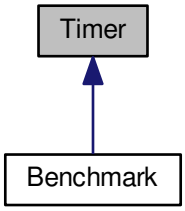
Dokumentacja dla tej klasy została wygenerowana z pliku:

- observer.hh

4.14 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 time

  Zmienna time.

4.14.1 Opis szczegółowy

Definicja w linii 12 pliku timer.hh.

4.14.2 Dokumentacja konstruktora i destruktora

4.14.2.1 Timer::Timer ( ) [inline]

Definicja w linii 32 pliku timer.hh.

4.14.3 Dokumentacja funkcji składowych

4.14.3.1 double Timer::getTime ( )

Zwraca

Zwraca wartosc zmiennej time

Definicja w linii 19 pliku timer.cpp.

4.14.3.2 void Timer::start_timer ( )

Definicja w linii 8 pliku timer.cpp.

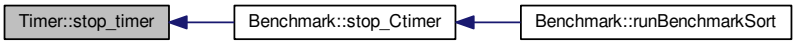
Oto graf wywoływań tej funkcji:

4.14.3.3 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:



4.14.4 Dokumentacja atrybutów składowych

4.14.4.1 timeval Timer::end [protected]

Definicja w linii 19 pliku timer.hh.

4.14.4.2 timeval Timer::start [protected]

Przechowuja informacje o poczatku i koncu pomiaru czasu

Definicja w linii 19 pliku timer.hh.

4.14.4.3 double Timer::time [protected]

Przechowuje zmierzony czas

Definicja w linii 26 pliku timer.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

- timer.hh
- timer.cpp

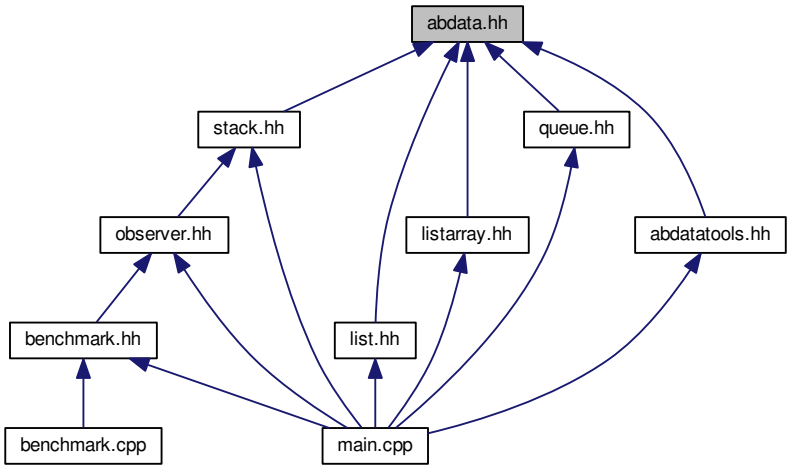
Rozdział 5

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.

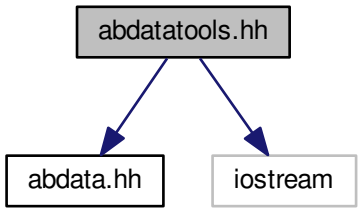
5.2 abdata.hh

```
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
```

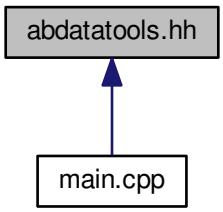
5.3 Dokumentacja pliku abdatatools.hh

```
#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

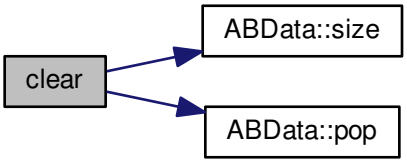
5.3.1.1 template<typename type > void clear (ABData< type > * item)

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:



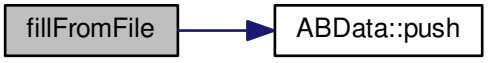
5.3.1.2 template<typename type > bool fillFromFile (ABData< type > * item, const int amount, const char * fileName)

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
```

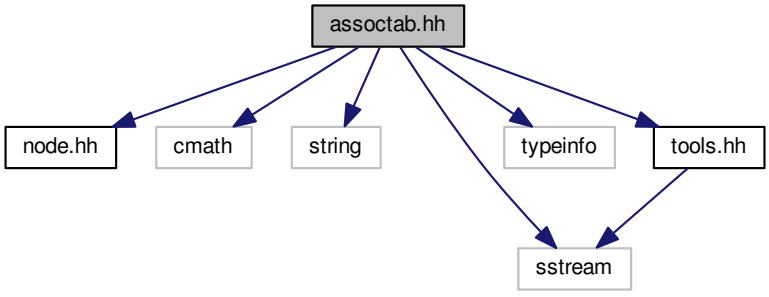
```
00004 #include "abdata.hh"
00005 #include <iostream>
00006
00007 /*
00008  *!\file
00009  * \brief Plik zawiera definicje funkcji operujacych na obiektach o klasie nadrzednej
00010  * ABData.
00011  */
00012
00020 template <typename type>
00021 bool fillFromFile(ABData<type> *item, const int amount, 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

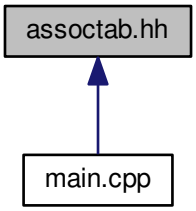
Definicja klasy AssocTab.

```
#include "node.hh"
#include <cmath>
#include <string>
#include <sstream>
#include <typeinfo>
#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 "node.hh"
00005 #include <cmath>
00006 #include <string>
00007 #include <sstream>
00008 #include <typeinfo>
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> > [
```

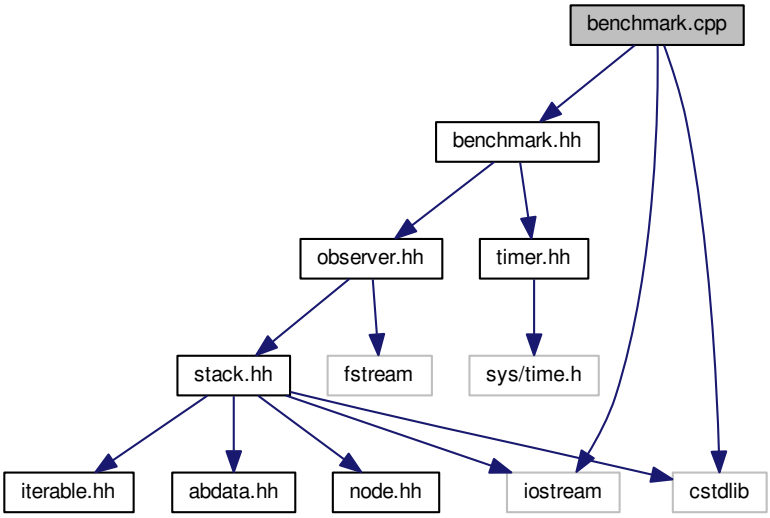
```
      TAB];
00040     counter = TAB;
00041   }
00049   AssocTab(unsigned int howmany){
00050     tab = new List<AssocData<typeKey, type> > [howmany];
00051     counter = howmany;
00052   }
00053
00060   ~AssocTab(){delete[] tab;}
00061
00070   void push(typeKey ikey, type toaddVal);
00071
00079   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.hh"
#include <cstdlib>
#include <iostream>
```

Wykres zależności załączania dla benchmark.cpp:



5.8 benchmark.cpp

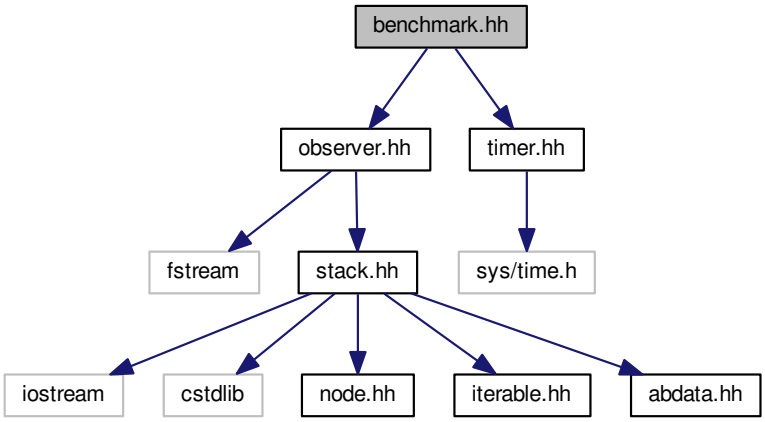
```
00001 #include "benchmark.hh"
00002 #include <cstdlib>
00003 #include <iostream>
00008 void Benchmark::notify(){
00009   for(unsigned int i=0; i<obss.size();i++)
00010     obss[i]->update(amount, mean);
00011 }
00012
00013 void Benchmark::stop_Ctimer(){
00014   stop_timer();
00015   total+=time;
00016   counter++;
00017 }
00018
00019 void Benchmark::calc_mean(){
00020   mean=total/counter;
00021   std::cout << mean << "  " << amount << "    " << std::endl;
00022   notify();
00023 }
00024
```

5.9 Dokumentacja pliku benchmark.hh

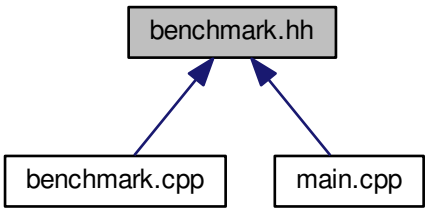
Definicja klasy Benchmark.

```
#include "observer.hh"
#include "timer.hh"
```

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

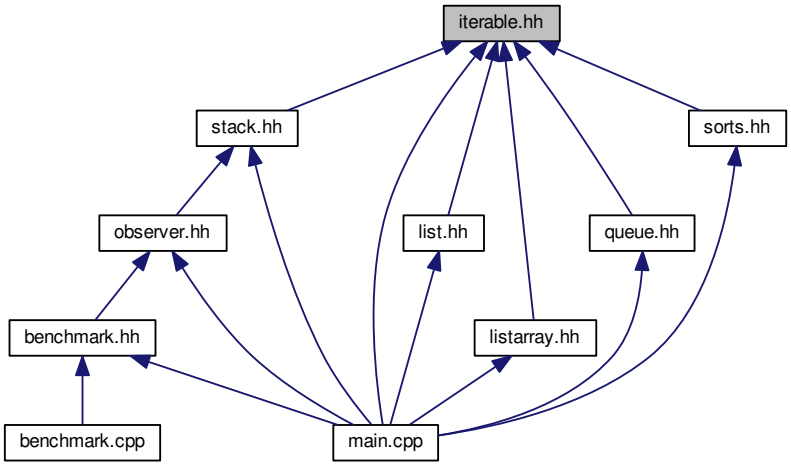
```
00001 #ifndef BENCHMARK_HH
00002 #define BENCHMARK_HH
00003
00004 #include "observer.hh"
00005 #include "timer.hh"
00006
00017 class Benchmark: public Subject, public Timer{
00021   double total;
00025   double mean;
00029   int counter;
00033   int amount;
00034 public:
00035   Benchmark(){
```

```
00036     total = 0;
00037     mean = 0;
00038     counter = 0;
00039     amount = 0;
00040   }
00044   void notify();
00051   void stop_Ctimer();
00052
00058   void calc_mean();
00059
00068   template<typename type>
00069   void runBenchmarkSort(void (*f)(Iterable<type>&, int, int),
      Iterable<type> &container, int dataCount, int repeats);
00070
00071 };
00072
00073 template<typename type>
00074 void Benchmark::runBenchmarkSort(void (*f)(
      Iterable<type>&, int, int), Iterable<type> &container, int dataCount, int
      repeats){
00075   amount = dataCount;
00076   total=0;
00077   mean=0;
00078   counter=0;
00079   for(int i=1; i<=repeats; i++){
00080     start_timer();
00081     (*f)(container, 0, amount-1);
00082     stop_Ctimer();
00083   }
00084   calc_mean();
00085 }
00086
00087 #endif
```

5.11 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.

5.11.1 Dokumentacja funkcji

5.11.1.1 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.

5.12 iterable.hh

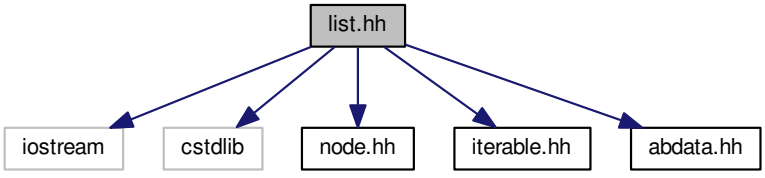
```
00001 #ifndef ITERABLE_HH
00002 #define ITERABLE_HH
00003
00014 template <class type>
00015 class Iterable{
00016 public:
00017   virtual type& operator [](const unsigned int index)=0;
00018 };
00019
00028 template <class type>
00029 void display(Iterable<type> &todisplay, unsigned int howmany){
00030   for(unsigned int i=0; i<howmany; i++)
00031     std::cout<<todisplay[i]<<std::endl;
00032 }
00033
00034 #endif
```

5.13 Dokumentacja pliku list.hh

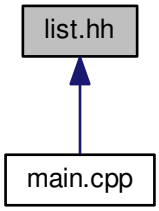
Definicja klasy List.

```
#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 >

5.14 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;
```

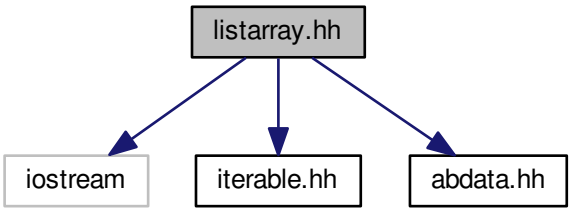
```
00102     head = head->next;
00103     delete ptr;
00104     iterator--;
00105   }
00106 }
00107
00108 template <class type>
00109 void List<type>::pop(unsigned int index){
00110   if(!head)
00111     std::cerr<<"Lista jest pusta!"<<std::endl;
00112   else{
00113     node<type> *ptr = head;
00114     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
```

5.15 Dokumentacja pliku listarray.hh

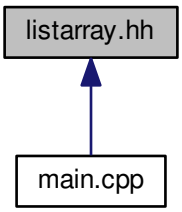
Definicja klasy ListArray.

```
#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 >

5.16 listarray.hh

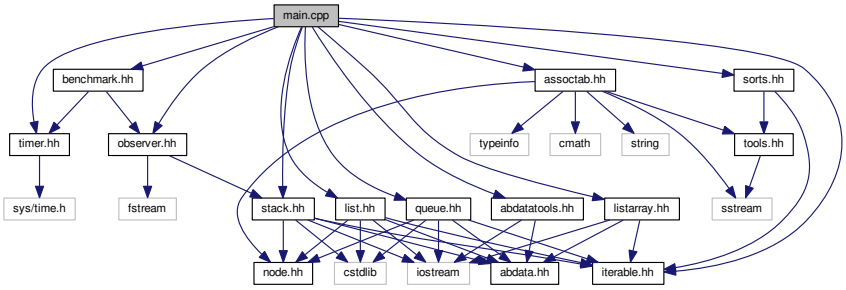
```
00001 #ifndef LISTARRAY_HH
00002 #define LISTARRAY_HH
00003
00004 #include <iostream>
00005 #include "iterable.hh"
00006 #include "abdata.hh"
00007
00012 template <class type>
00013 class ListArray: public ABData<type>, public Iterable<type>{
00019   int counter;
00025   int iterator;
00029   type *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
```

5.17 Dokumentacja pliku main.cpp

```
#include "list.hh"
#include "stack.hh"
#include "queue.hh"
#include "iterable.hh"
#include "timer.hh"
#include "benchmark.hh"
#include "observer.hh"
#include "sorts.hh"
#include "abdatatools.hh"
#include "listarray.hh"
#include "assoctab.hh"
```

Wykres zależności załączania dla main.cpp:



Funkcje

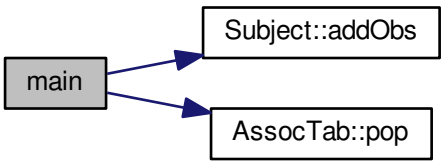
- int main ()

5.17.1 Dokumentacja funkcji

5.17.1.1 int main ( )

Definicja w linii 16 pliku main.cpp.

Oto graf wywołań dla tej funkcji:



5.18 main.cpp

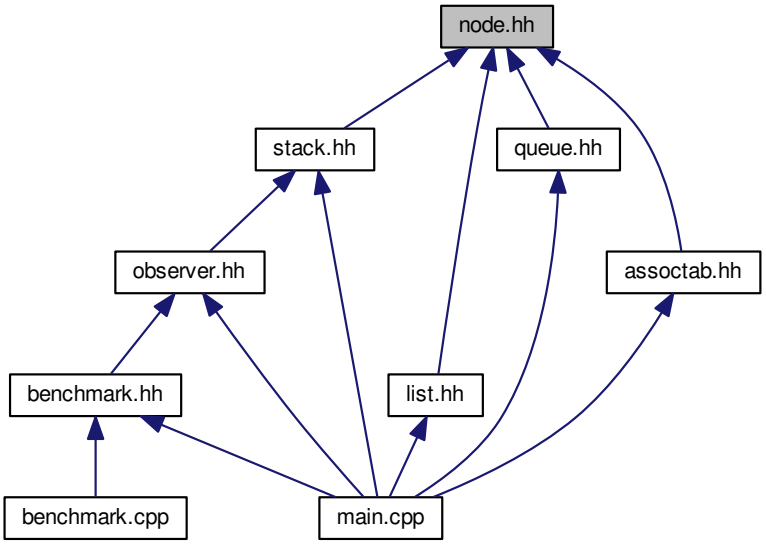
```
00001 #include "list.hh"
00002 #include "stack.hh"
00003 #include "queue.hh"
00004 #include "iterable.hh"
00005 #include "timer.hh"
00006 #include "benchmark.hh"
00007 #include "observer.hh"
00008 #include "sorts.hh"
00009 #include "abdatatools.hh"
00010 #include "listarray.hh"
00011 #include "assoctab.hh"
00012
00013 using namespace std;
00014
00015
00016 int main(){
00017   //ListArray<int> object;
00018   Benchmark test;
```

```
00019   SaveToFile saver;
00020   test.addObs(&saver);
00021   /*
00022   for(int i=10; i<=1000000; i*=10){
00023     fillFromFile(&object, i, "dane.dat");
00024     test.runBenchmarkSort(&quicksort, object, object.size(), 20);
00025     clear(&object);
00026     }
00027   */
00028   AssocTab<string, string> object;
00029   object["kon"] = "rafal";
00030   cout << object["kon"] << endl;
00031   object.pop("kon");
00032
00033   return 0;
00034 }
```

5.19 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 >

5.19.1 Opis szczegółowy

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

Definicja w pliku node.hh.

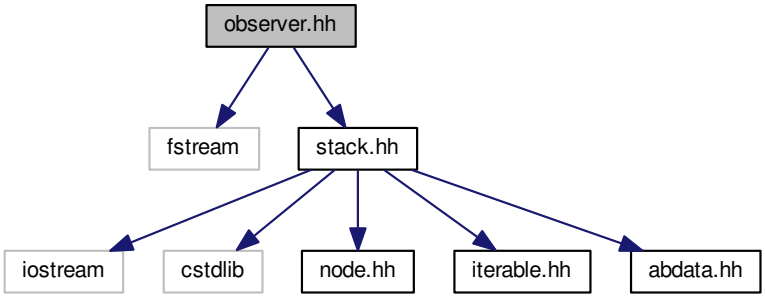
5.20 node.hh

```
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
```

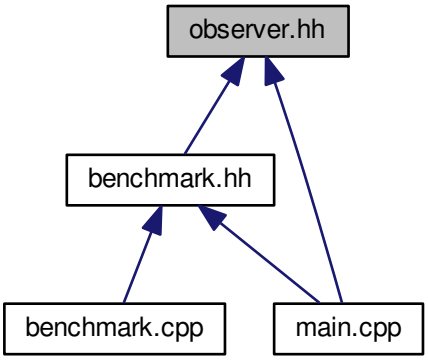
5.21 Dokumentacja pliku observer.hh

```
#include <fstream>
#include "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

5.22 observer.hh

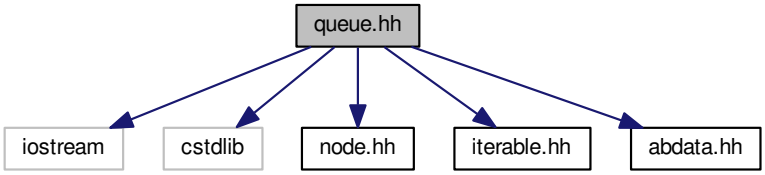
```
00001 #ifndef OBSERVER_HH
00002 #define OBSERVER_HH
00003
00004 #include <fstream>
00005 #include "stack.hh"
00006 using namespace std;
00007 class Subject;
00008
00009 class Observer{
00010 public:/*
00011   Subject *model;
00012
00013   Observer(Subject *mod){
00014     model=mod;
00015     }*/
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
```

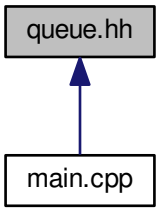
5.23 Dokumentacja pliku queue.hh

```
#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 >

5.24 queue.hh

```
00001 #ifndef QUEUE_HH
00002 #define QUEUE_HH
00003
00004 #include <iostream>
```

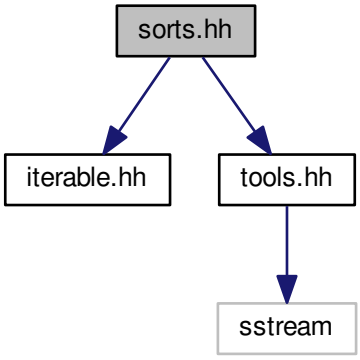
```
00005 #include <cstdlib>
00006 #include "node.hh"
00007 #include "iterable.hh"
00008 #include "abdata.hh"
00009
00010 template <class type>
00011 class Queue: public ABData<type>, public Iterable<type>{
00017   node<type> *head;
00023   int iterator;
00024
00025 public:
00031   Queue(){
00032     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
```

5.25 Dokumentacja pliku sorts.hh

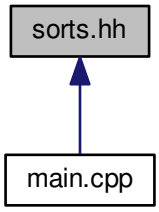
W pliku znajduja sie definicje metod sortujacych obiekty dziedziczace z klasy Iterable - takie ktore maja zdefiniowane operatory indeksowania []. Przykladowe wywolanie metody sortujace caly obiekt: Stack stos; insertsort(stos, stos.-size()-1)

```
#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.

5.25.1 Dokumentacja funkcji

5.25.1.1 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.

5.25.1.2 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:

5.26 sorts.hh

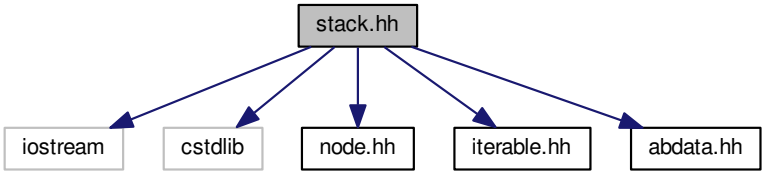
```
00001 #ifndef SORTS_HH
00002 #define SORTS_HH
00003
00004 #include "iterable.hh"
00005 #include "tools.hh"
00006
00025 template<typename type>
00026 void insertsort(Iterable<type> &tosort, int left, int right){
00027   int i,j; int temp;
00028   for(i=left; i<=right; ++i){
00029     temp=tosort[i];
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
```

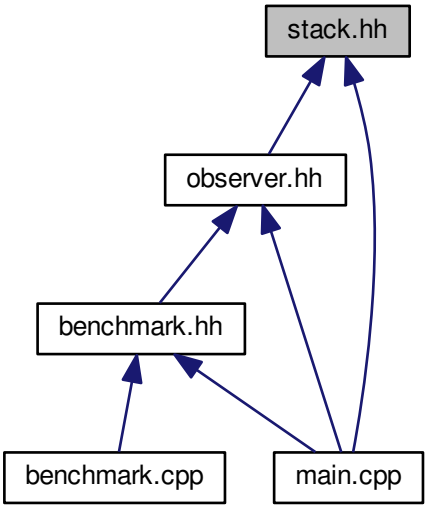
5.27 Dokumentacja pliku stack.hh

```
#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 >

5.28 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;
```

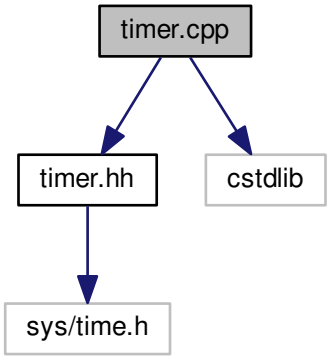
```
00078     ptr=ptr->next;
00079   }
00080 }
00081 };
00082
00083 template <class type>
00084 void Stack<type>::push(const type elem){
00085   node<type> *toadd = new node<type>;
00086   toadd->val = elem;
00087   node<type> *ptr = head;
00088   head = toadd;
00089   toadd->next = ptr;
00090   iterator++;
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
```

5.29 Dokumentacja pliku timer.cpp

Ciala metod klasy Timer.

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

Wykres zależności załączania dla timer.cpp:



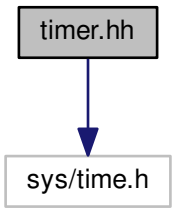
5.30 timer.cpp

```
00001 #include "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   time = (end.tv_sec - start.tv_sec) * 1000.0;      // sec to ms
00016   time += (end.tv_usec - start.tv_usec) / 1000.0;   // us to ms
00017 }
00018
00019 double Timer::getTime(){
00020   return time;
00021 }
```

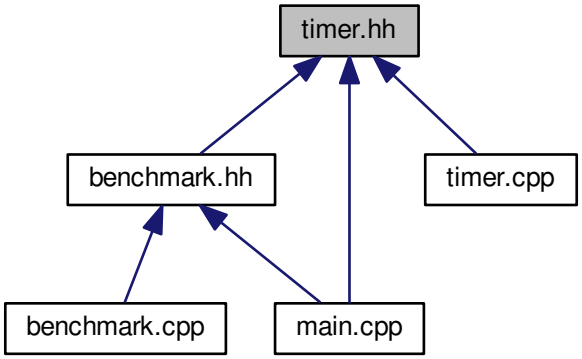
5.31 Dokumentacja pliku timer.hh

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

5.31.1 Opis szczegółowy

Służy do pomiaru czasu

Definicja w pliku timer.hh.

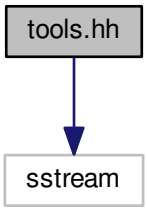
5.32 timer.hh

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

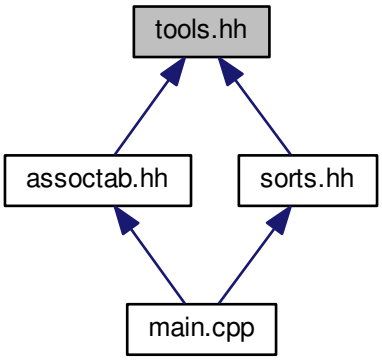
```
00012 class Timer{
00013 protected:
00019   timeval start, end;
00020
00026   double time;
00027
00028 public:
00032   Timer(){time=0;}
00036   void start_timer();
00043   void stop_timer();
00044
00050   double getTime();
00051 };
00052
00053 #endif
```

5.33 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)

5.33.1 Dokumentacja funkcji

5.33.1.1 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

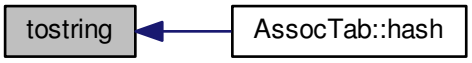
Definicja w linii 17 pliku tools.hh.

Oto graf wywoływań tej funkcji:

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

Definicja w linii 24 pliku tools.hh.

Oto graf wywoływań tej funkcji:



5.34 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
```

Skorowidz