

Benchmark + Mnozenie

0.3

Wygenerowano przez Doxygen 1.8.6

Cz, 16 kwi 2015 11:41:06

Spis treści

1	Indeks hierarchiczny	1
1.1	Hierarchia klas	1
2	Indeks klas	3
2.1	Lista klas	3
3	Indeks plików	5
3.1	Lista plików	5
4	Dokumentacja klas	7
4.1	Dokumentacja szablonu klasy <code>Assoctab< typeK, typeV ></code>	7
4.1.1	Opis szczegółowy	9
4.1.2	Dokumentacja konstruktora i destruktoru	9
4.1.2.1	<code>Assoctab</code>	9
4.1.2.2	<code>~Assoctab</code>	9
4.1.2.3	<code>Assoctab</code>	9
4.1.2.4	<code>~Assoctab</code>	10
4.1.3	Dokumentacja funkcji składowych	10
4.1.3.1	<code>h</code>	10
4.1.3.2	<code>h</code>	10
4.1.3.3	<code>operator[]</code>	10
4.1.3.4	<code>operator[]</code>	10
4.1.3.5	<code>pop</code>	10
4.1.3.6	<code>pop</code>	10
4.1.3.7	<code>push</code>	10
4.1.3.8	<code>push</code>	10
4.1.3.9	<code>size</code>	11
4.1.3.10	<code>size</code>	11
4.1.3.11	<code>test</code>	11
4.1.3.12	<code>wyczysc_dane</code>	11
4.1.3.13	<code>wyczysc_dane</code>	12
4.1.3.14	<code>wykonaj_program</code>	13

4.1.3.15	wykonaj_program	13
4.1.3.16	wyswietl	13
4.1.3.17	wyswietl	13
4.1.3.18	wyswietl_liste	14
4.1.3.19	wyswietl_liste	14
4.1.4	Dokumentacja atrybutów składowych	14
4.1.4.1	rozmiar	14
4.1.4.2	tab	14
4.2	Dokumentacja klasy Benchmark	14
4.2.1	Opis szczegółowy	15
4.2.2	Dokumentacja funkcji składowych	15
4.2.2.1	rozpocznij_pomiar	15
4.2.2.2	testuj	15
4.2.2.3	testuj_strukture	16
4.2.2.4	zakoncz_pomiar	17
4.2.3	Dokumentacja atrybutów składowych	17
4.2.3.1	czas_pomiaru	17
4.2.3.2	t1	17
4.2.3.3	t2	18
4.3	Dokumentacja szablonu klasy Lista< type, type2 >	18
4.3.1	Opis szczegółowy	19
4.3.2	Dokumentacja konstruktora i destruktora	19
4.3.2.1	Lista	19
4.3.3	Dokumentacja funkcji składowych	19
4.3.3.1	daj	19
4.3.3.2	pop	19
4.3.3.3	pop	20
4.3.3.4	push	20
4.3.3.5	push_front	20
4.3.3.6	size	20
4.3.3.7	wyczyszc_dane	20
4.3.3.8	wykonaj_program	20
4.3.3.9	wyswietl	21
4.3.3.10	wyswietl	21
4.3.4	Dokumentacja atrybutów składowych	21
4.3.4.1	first	21
4.4	Dokumentacja klasy Lista_tab	21
4.4.1	Opis szczegółowy	22
4.4.2	Dokumentacja konstruktora i destruktora	23
4.4.2.1	Lista_tab	23

4.4.2.2	<code>~Lista_tab</code>	23
4.4.3	Dokumentacja funkcji składowych	23
4.4.3.1	<code>mergesort</code>	23
4.4.3.2	<code>pop</code>	23
4.4.3.3	<code>push</code>	23
4.4.3.4	<code>size</code>	24
4.4.3.5	<code>test</code>	24
4.4.3.6	<code>wyczisc_dane</code>	24
4.4.3.7	<code>wykonaj_program</code>	25
4.4.4	Dokumentacja atrybutów składowych	25
4.4.4.1	<code>iterator</code>	25
4.4.4.2	<code>rozmiar</code>	25
4.4.4.3	<code>tab</code>	26
4.5	Dokumentacja struktury <code>Lista< type, type2 >::pole</code>	26
4.5.1	Opis szczegółowy	26
4.5.2	Dokumentacja konstruktora i destruktora	26
4.5.2.1	<code>pole</code>	26
4.5.3	Dokumentacja atrybutów składowych	27
4.5.3.1	<code>next</code>	27
4.5.3.2	<code>val1</code>	27
4.5.3.3	<code>val2</code>	27
4.6	Dokumentacja klasy <code>Program</code>	27
4.6.1	Opis szczegółowy	28
4.6.2	Dokumentacja konstruktora i destruktora	28
4.6.2.1	<code>Program</code>	28
4.6.2.2	<code>~Program</code>	28
4.6.3	Dokumentacja funkcji składowych	28
4.6.3.1	<code>getRozmiar_tab</code>	28
4.6.3.2	<code>test</code>	28
4.6.3.3	<code>wczytaj_dane</code>	29
4.6.3.4	<code>wczytaj_dane</code>	29
4.6.3.5	<code>wyczisc_dane</code>	29
4.6.3.6	<code>wykonaj_program</code>	30
4.6.3.7	<code>wykonaj_program</code>	30
4.6.3.8	<code>wyswietl_dane</code>	30
4.6.3.9	<code>zapisz_dane</code>	30
4.6.4	Dokumentacja atrybutów składowych	31
4.6.4.1	<code>plik_we</code>	31
4.6.4.2	<code>plik_wy</code>	31
4.6.4.3	<code>rozmiar_tab</code>	31

4.6.4.4	tab	31
4.7	Dokumentacja klasy Tabx2	31
4.7.1	Opis szczegółowy	32
4.7.2	Dokumentacja funkcji składowych	32
4.7.2.1	wykonaj_program	32
5	Dokumentacja plików	35
5.1	Dokumentacja pliku assoctab.hh	35
5.1.1	Dokumentacja definicji	36
5.1.1.1	HASH	36
5.1.1.2	TAB	36
5.2	assoctab.hh	36
5.3	Dokumentacja pliku benchmark.cpp	38
5.4	benchmark.cpp	38
5.5	Dokumentacja pliku benchmark.hh	39
5.6	benchmark.hh	40
5.7	Dokumentacja pliku hashtab.cpp	41
5.8	hashtab.cpp	41
5.9	Dokumentacja pliku lista.hh	42
5.10	lista.hh	42
5.11	Dokumentacja pliku lista_tab.cpp	45
5.12	lista_tab.cpp	46
5.13	Dokumentacja pliku lista_tab.hh	47
5.13.1	Dokumentacja definicji	48
5.13.1.1	LISTA__TAB_HH	48
5.14	lista_tab.hh	49
5.15	Dokumentacja pliku main.cpp	49
5.15.1	Dokumentacja funkcji	50
5.15.1.1	main	50
5.16	main.cpp	50
5.17	Dokumentacja pliku program.cpp	50
5.18	program.cpp	51
5.19	Dokumentacja pliku program.hh	52
5.20	program.hh	53
5.21	Dokumentacja pliku tabx2.cpp	53
5.22	tabx2.cpp	54
5.23	Dokumentacja pliku tabx2.hh	54
5.23.1	Opis szczegółowy	55
5.24	tabx2.hh	55
5.25	Dokumentacja pliku tmp.hh	56

5.25.1 Dokumentacja definicji	56
5.25.1.1 HASH	56
5.25.1.2 TAB	56
5.26 tmp.hh	57
Indeks	59

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

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

Benchmark	14
Lista< type, type2 >::pole	26
Program	27
Assoctab< typeK, typeV >	7
Assoctab< typeK, typeV >	7
Lista< type, type2 >	18
Lista< typeK, typeV >	18
Lista_tab	21
Tabx2	31

Rozdział 2

Indeks klas

2.1 Lista klas

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

Assoctab< typeK, typeV >	7
Benchmark	
Klasa Benchmark	14
Lista< type, type2 >	18
Lista_tab	21
Lista< type, type2 >::pole	
Struktura pole	26
Program	
Modeluje klasę Program	27
Tabx2	31

Rozdział 3

Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

assoctab.hh	Definicja klasy Assoctab	36
benchmark.cpp	Plik zawiera metody klasy Benchmark	38
benchmark.hh	Definicja klasy Benchmark	40
hashtab.cpp	Plik zawiera metody klasy Hashtab	41
lista.hh	Definicja klasy Lista	42
lista_tab.cpp	Zawiera definicje metod klasy Lista	46
lista_tab.hh	Definicja klasy Lista_tab	49
main.cpp	50
program.cpp	Plik zawiera metody klasy Program	51
program.hh	Definicja klasy Program	53
tabx2.cpp	Plik zawiera metody klasy Tabx2	54
tabx2.hh	Definicja klasy Tabx2	55
tmp.hh	Definicja klasy Assoctab	57

Rozdział 4

Dokumentacja klas

4.1 Dokumentacja szablonu klasy `Assoctab< typeK, typeV >`

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

Diagram dziedziczenia dla `Assoctab< typeK, typeV >`

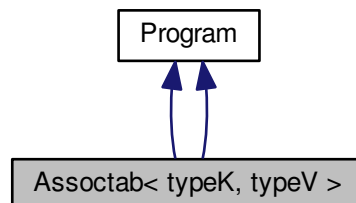
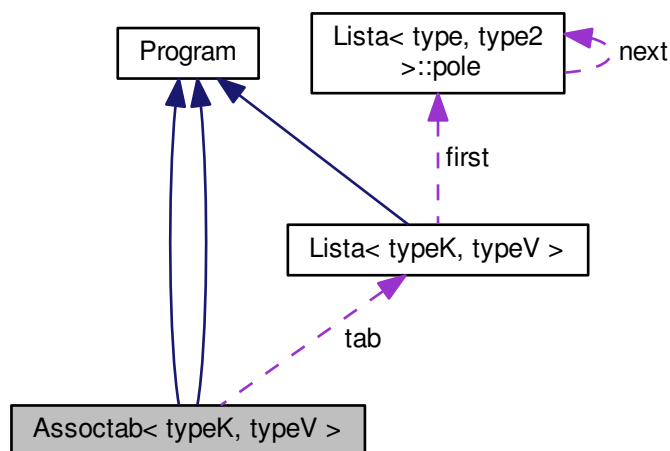


Diagram współpracy dla `Assoctab< typeK, typeV >`:



Metody publiczne

- `Assoctab ()`
Konstruktor bezparametryczny.
- `~Assoctab ()`
Destruktor.
- `void push (typeK klucz, typeV wartosc)`
Procedura push.
- `void pop (typeK klucz)`
Procedura pop.
- `int h (typeK klucz)`
Metoda hash.
- `int size ()`
Metoda size.
- `bool wykonaj_program (char *nazwa_pliku, int ilosc_danych)`
Metoda wykonaj_program.
- `void test ()`
Metoda test()
- `void wyczyszc_dane (int ile)`
Metoda wyczyszc_dane.
- `void wyswietl_liste (typeK klucz)`
Metoda wyswietl.
- `void wyswietl (typeK klucz)`
Metoda wyswietl.
- `const typeV & operator[] (typeK klucz) const`
- `typeV & operator[] (typeK klucz)`
- `Assoctab ()`
Konstruktor bezparametryczny.
- `~Assoctab ()`

Destruktor.

- void `push` (typeK klucz, typeV wartosc)

Procedura push.

- void `pop` (typeK klucz)

Procedura pop.

- int `h` (typeK klucz)

Metoda hash.

- int `size` ()

Metoda size.

- bool `wykonaj_program` (char *nazwa_pliku, int ilosc_danych)

Metoda wykonaj_program.

- void `wyczysc_dane` (int ile)

Metoda wyczysc_dane.

- void `wyswietl_liste` (typeK klucz)

Metoda wyswietl.

- void `wyswietl` (typeK klucz)

Metoda wyswietl.

Atrybuty publiczne

- `Lista< typeK, typeV > * tab`

Wskaźnik na dynamicznie alokowana tablice z danymi.

- int `rozmiar`

Aktualny rozmiar tablicy.

Dodatkowe Dziedziczone Składowe

4.1.1 Opis szczegółowy

`template<class typeK, class typeV>class Assoctab< typeK, typeV >`

Definicja w linii 18 pliku `assoctab.hh`.

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 `template<class typeK, class typeV > Assoctab< typeK, typeV >::Assoctab () [inline]`

Ustawia początek listy na NULL

Definicja w linii 37 pliku `assoctab.hh`.

4.1.2.2 `template<class typeK, class typeV > Assoctab< typeK, typeV >::~~Assoctab () [inline]`

Usuwa dynamicznie utworzona tablice danych oraz przypisuje wskaźnikowi wartość NULL.

Definicja w linii 48 pliku `assoctab.hh`.

4.1.2.3 `template<class typeK, class typeV > Assoctab< typeK, typeV >::Assoctab () [inline]`

Ustawia początek listy na NULL

Definicja w linii 37 pliku `tmp.hh`.

4.1.2.4 `template<class typeK , class typeV > Assoctab< typeK, typeV >::~Assoctab () [inline]`

Usuwa dynamicznie utworzona tablice danych oraz przypisuje wskaźnikowi wartosc NULL.

Definicja w linii 48 pliku [tmp.hh](#).

4.1.3 Dokumentacja funkcji składowych

4.1.3.1 `template<class typeK , class typeV > int Assoctab< typeK, typeV >::h (typeK klucz)`

Dokonuje haszowania podanego klucza na wartosc liczbową.

Definicja w linii 156 pliku [assoctab.hh](#).

4.1.3.2 `template<class typeK , class typeV > int Assoctab< typeK, typeV >::h (typeK klucz)`

Dokonuje haszowania podanego klucza na wartosc liczbową.

4.1.3.3 `template<class typeK , class typeV > const typeV & Assoctab< typeK, typeV >::operator[] (typeK klucz) const`

Definicja w linii 196 pliku [assoctab.hh](#).

4.1.3.4 `template<class typeK , class typeV > typeV & Assoctab< typeK, typeV >::operator[] (typeK klucz)`

Definicja w linii 200 pliku [assoctab.hh](#).

4.1.3.5 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::pop (typeK klucz)`

Usuwa z tablicy wartosc odpowiadajaca danemu kluczowi.

4.1.3.6 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::pop (typeK klucz)`

Usuwa z tablicy wartosc odpowiadajaca danemu kluczowi.

Definicja w linii 152 pliku [assoctab.hh](#).

4.1.3.7 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::push (typeK klucz, typeV wartosc)`

Dodaje element o podanej wartosci na miejsce odczytywane przez klucz. Do wyboru 2 metody push - po osiagnieciu maksymalnego rozmiaru tablicy, jedna z nich zwieksza rozmiar tablicy o 1, a druga podwaja aktualny rozmiar tablicy. Wyboru metody nalezy dokonac poprzez odkomentowanie odpowiedniej metody w pliku .cpp.

Parametry

in	x	String, ktory chcemy dodac na koniec listy.
----	---	---

Definicja w linii 148 pliku [assoctab.hh](#).

4.1.3.8 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::push (typeK klucz, typeV wartosc)`

Dodaje element o podanej wartosci na miejsce odczytywane przez klucz. Do wyboru 2 metody push - po osiagnieciu maksymalnego rozmiaru tablicy, jedna z nich zwieksza rozmiar tablicy o 1, a druga podwaja aktualny rozmiar tablicy. Wyboru metody nalezy dokonac poprzez odkomentowanie odpowiedniej metody w pliku .cpp.

Parametry

<code>in</code>	<code>x</code>	String, który chcemy dodać na koniec listy.
-----------------	----------------	---

4.1.3.9 `template<class typeK , class typeV > int Assoctab< typeK, typeV >::size ()`

Daje informacje o rozmiarze tablicy (liczbie jej elementów).

Zwraca

Rozmiar tablicy (liczba jej elementów)

Definicja w linii 167 pliku [assoctab.hh](#).

4.1.3.10 `template<class typeK , class typeV > int Assoctab< typeK, typeV >::size ()`

Daje informacje o rozmiarze tablicy (liczbie jej elementów).

Zwraca

Rozmiar tablicy (liczba jej elementów)

4.1.3.11 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::test () [inline],[virtual]`

Przeprowadza test - odwołuje się do pojedynczego klucza

Reimplementowana z [Program](#).

Definicja w linii 104 pliku [assoctab.hh](#).

Oto graf wywołań dla tej funkcji:

4.1.3.12 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::wyczysc_dane (int ile) [inline],[virtual]`

Usuwa zadana ilość elementów listy za pomocą metody pop

Parametry

<code>in</code>	<code>ile</code>	Liczba elementów, które chcemy usunąć.
-----------------	------------------	--

Implementuje [Program](#).

Definicja w linii 107 pliku [tmp.hh](#).

4.1.3.13 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::wyczysc_dane (int ile) [inline],
[virtual]`

Usuwa zadana ilosc elementow listy za pomoca metody pop

Parametry

<i>in</i>	<i>ile</i>	Liczba elementow, ktore chcemy usunac.
-----------	------------	--

Implementuje [Program](#).

Definicja w linii 116 pliku [assoctab.hh](#).

4.1.3.14 `template<class typeK , class typeV > bool Assoctab< typeK, typeV >::wykonaj_program (char * nazwa_pliku, int ilosc_danych) [virtual]`

Wczytuje zadana ilosc danych do tablicy asocjacyjnej

Zwracane wartości

<i>TRUE</i>	Poprawnie wykonano program
<i>FALSE</i>	Nie wczytano danych

Implementuje [Program](#).

Definicja w linii 171 pliku [assoctab.hh](#).

4.1.3.15 `template<class typeK , class typeV > bool Assoctab< typeK, typeV >::wykonaj_program (char * nazwa_pliku, int ilosc_danych) [inline],[virtual]`

Wykonuje zadany program - dodanie zadanej ilosci danych z pliku do listy za pomoca metody [push\(\)](#)

Zwracane wartości

<i>TRUE</i>	Poprawnie wykonano program
<i>FALSE</i>	Nie wczytano danych

Implementuje [Program](#).

Definicja w linii 98 pliku [tmp.hh](#).

4.1.3.16 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::wyswietl (typeK klucz)`

Wyswietla elementy dokładnie odpowiadające podanemu kluczowi

Parametry

<i>in</i>	<i>Klucz</i>	
-----------	--------------	--

4.1.3.17 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::wyswietl (typeK klucz)`

Wyswietla elementy dokładnie odpowiadające podanemu kluczowi

Parametry

<i>in</i>	<i>Klucz</i>	
-----------	--------------	--

Definicja w linii 192 pliku [assoctab.hh](#).

Oto graf wywoływań tej funkcji:



4.1.3.18 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::wyswietl_liste (typeK klucz)`

Wyswietla wszystkie elementy, ktorzych klucze po haszowaniu maja te sama wartosc

4.1.3.19 `template<class typeK , class typeV > void Assoctab< typeK, typeV >::wyswietl_liste (typeK klucz)`

Wyswietla wszystkie elementy, ktorzych klucze po haszowaniu maja te sama wartosc

Definicja w linii 188 pliku [assoctab.hh](#).

4.1.4 Dokumentacja atrybutów składowych

4.1.4.1 `template<class typeK , class typeV > int Assoctab< typeK, typeV >::rozmiar`

Definicja w linii 29 pliku [assoctab.hh](#).

4.1.4.2 `template<class typeK , class typeV > Lista< typeK, typeV > * Assoctab< typeK, typeV >::tab`

Definicja w linii 23 pliku [assoctab.hh](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

- [assoctab.hh](#)
- [tmp.hh](#)

4.2 Dokumentacja klasy Benchmark

Klasa [Benchmark](#).

```
#include <benchmark.hh>
```

Metody publiczne

- void [rozpocznij_pomiar](#) ()
Procedura rozpocznij_pomiar.
- void [zakoncz_pomiar](#) ()
Procedura zakoncz_pomiar.
- double [testuj](#) ([Program](#) &program, char *dane, int ilosc_danych, int ilosc_testow)
Metoda testuj.
- double [testuj_strukture](#) ([Program](#) &program, char *dane, int ilosc_danych, int ilosc_testow)
Metoda testuj_strukture.

Atrybuty prywatne

- timeval [t1](#)
Zmienne t1, t2.
- timeval [t2](#)
- double [czas_pomiaru](#)
Zmienna czas_pomiaru.

4.2.1 Opis szczegółowy

Jest to klasa służąca do testowania programów.

Definicja w linii [23](#) pliku [benchmark.hh](#).

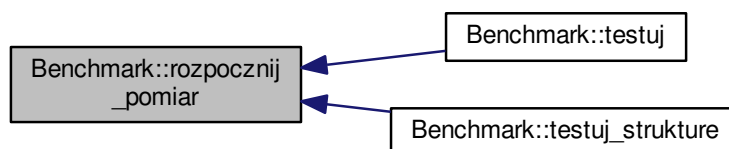
4.2.2 Dokumentacja funkcji składowych

4.2.2.1 void Benchmark::rozpocznij_pomiar ()

Rozpoczyna pomiar czasu.

Definicja w linii [7](#) pliku [benchmark.cpp](#).

Oto graf wywoływań tej funkcji:



4.2.2.2 double Benchmark::testuj (Program & program, char * dane, int ilosc_danych, int ilosc_testow)

Dokonuje testów wybranego programu. Dane wczytywane są do tablicy.

Parametry

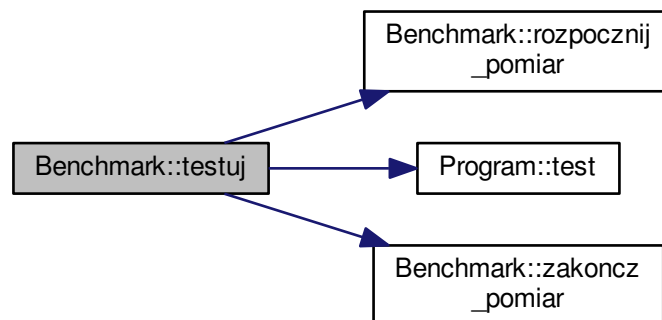
in	<i>program</i>	Program wybrany do testowania.
in	<i>dane</i>	Wskaźnik na nazwę pliku z danymi.
in	<i>ilosc_danych</i>	Ilość danych, które chcemy pobrać do testu.
in	<i>ilosc_testow</i>	Ilość testów, jakie chcemy przeprowadzić.

Zwraca

Metoda zwraca średni czas wykonania programu dla podanych parametrów.

Definicja w linii [17](#) pliku [benchmark.cpp](#).

Oto graf wywołań dla tej funkcji:



4.2.2.3 `double Benchmark::testuj_strukture (Program & program, char * dane, int ilosc_danych, int ilosc_testow)`

Dokonuje testow wybranego programu. Dane wczytywane sa do odpowiedniej struktury.

Parametry

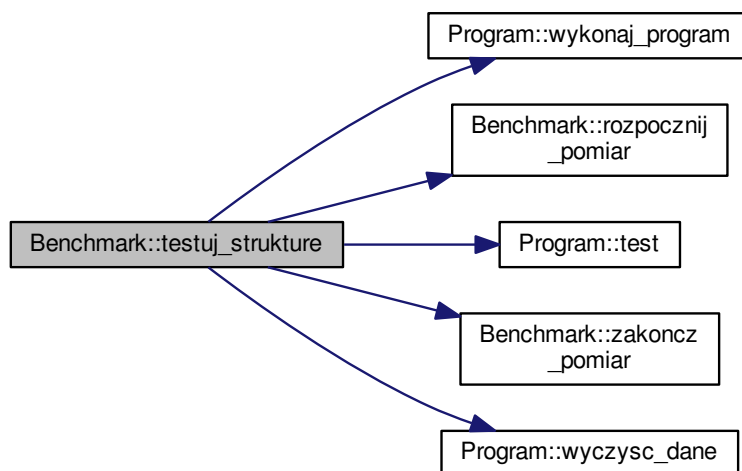
in	<i>program</i>	Program wybrany do testowania.
in	<i>dane</i>	Wskaznik na nazwe pliku z danymi.
in	<i>ilosc_danych</i>	Ilosc danych, ktore chcemy pobrac do testu.
in	<i>ilosc_testow</i>	Ilosc testow, jakie chcemy przeprowadzic.

Zwraca

Metoda zwraca sredni czas wykonania programu dla podanych parametrow.

Definicja w linii 48 pliku [benchmark.cpp](#).

Oto graf wywołań dla tej funkcji:

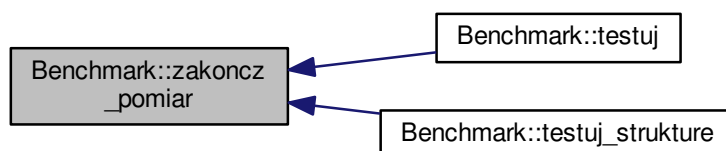


4.2.2.4 void Benchmark::zakoncz_pomiar ()

Konczy pomiar czasu i zapisuje wartosc zmierzona w zmiennej `czas_pomiaru`.

Definicja w linii 11 pliku [benchmark.cpp](#).

Oto graf wywoływań tej funkcji:



4.2.3 Dokumentacja atrybutów składowych

4.2.3.1 double Benchmark::czas_pomiaru [private]

Przechowuje obliczony czas pojedynczego pomiaru (w ms)

Definicja w linii 37 pliku [benchmark.hh](#).

4.2.3.2 timeval Benchmark::t1 [private]

Zmienne przechowujące momenty rozpoczęcia i zakończenia pomiaru czasu.

Definicja w linii 30 pliku [benchmark.hh](#).

4.2.3.3 timeval Benchmark::t2 [private]

Definicja w linii 30 pliku [benchmark.hh](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

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

4.3 Dokumentacja szablonu klasy Lista< type, type2 >

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

Diagram dziedziczenia dla Lista< type, type2 >

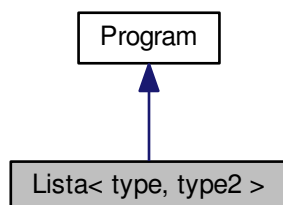
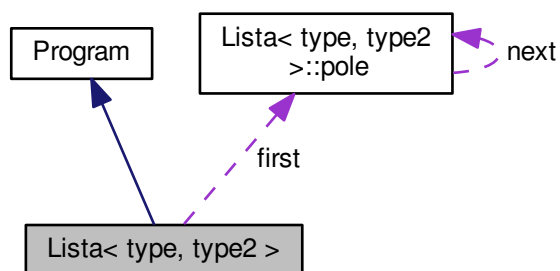


Diagram współpracy dla Lista< type, type2 >:



Komponenty

- struct [pole](#)
Struktura pole.

Metody publiczne

- `Lista ()`
Konstruktor bezparametryczny.
- `void push (type x, type2 y)`
Metoda push.
- `void push_front (type x, type2 y)`
Metoda push.
- `void pop ()`
Procedura pop.
- `int size ()`
Metoda size.
- `bool wykonaj_program (char *nazwa_pliku, int ilosc_danych)`
Metoda wykonaj_program.
- `void wyczyszc_dane (int ile)`
Metoda wyczyszc_dane.
- `void wyswietl ()`
- `void wyswietl (type key)`
- `type2 & daj (type key)`
- `void pop (type key)`
Procedura pop.

Atrybuty publiczne

- `pole * first`

Dodatkowe Dziedziczone Składowe

4.3.1 Opis szczegółowy

`template<class type, class type2>class Lista< type, type2 >`

Definicja w linii 22 pliku `lista.hh`.

4.3.2 Dokumentacja konstruktora i destruktor

4.3.2.1 `template<class type, class type2> Lista< type, type2 >::Lista () [inline]`

Ustawia początek listy na NULL

Definicja w linii 44 pliku `lista.hh`.

4.3.3 Dokumentacja funkcji składowych

4.3.3.1 `template<class type, class type2 > type2 & Lista< type, type2 >::daj (type key)`

Definicja w linii 238 pliku `lista.hh`.

4.3.3.2 `template<class type , class type2 > void Lista< type, type2 >::pop ()`

Usuwa ostatni element listy.

Definicja w linii 178 pliku `lista.hh`.

4.3.3.3 `template<class type, class type2 > void Lista< type, type2 >::pop (type key)`

Usuwa wszystkie elementy listy o podanej wartosci val1.

Parametry

in	Wartosc	val1 elementow do usuniecia
----	---------	-----------------------------

Definicja w linii 258 pliku [lista.hh](#).

4.3.3.4 `template<class type, class type2> void Lista< type, type2 >::push (type x, type2 y)`

Dodaje podana wartosc na koniec listy.

Parametry

in	x	Wartosc, ktora chcemy dodac na koniec listy.
----	---	--

Definicja w linii 148 pliku [lista.hh](#).

4.3.3.5 `template<class type, class type2> void Lista< type, type2 >::push_front (type x, type2 y)`

Dodaje podana wartosc na koniec listy.

Parametry

in	x	Wartosc, ktora chcemy dodac na poczatek listy.
----	---	--

Definicja w linii 164 pliku [lista.hh](#).

4.3.3.6 `template<class type , class type2 > int Lista< type, type2 >::size ()`

Daje informacje o rozmiarze listy (liczbie jej elementow).

Zwraca

Rozmiar listy (liczba jej elementow)

Definicja w linii 200 pliku [lista.hh](#).

4.3.3.7 `template<class type , class type2 > void Lista< type, type2 >::wyczysc_dane (int ile) [virtual]`

Usuwa zadana ilosc elementow listy za pomoca metody pop

Parametry

in	ile	Liczba elementow, ktore chcemy usunac.
----	-----	--

Implementuje [Program](#).

Definicja w linii 218 pliku [lista.hh](#).

4.3.3.8 `template<class type , class type2 > bool Lista< type, type2 >::wykonaj_program (char * nazwa_pliku, int ilosc_danych) [virtual]`

Wykonuje zadany program - dodanie zadanej ilosci danych z pliku do listy za pomoca metody [push\(\)](#)

Zwracane wartości

<i>TRUE</i>	Poprawnie wykonano program
<i>FALSE</i>	Nie wczytano danych

Implementuje [Program](#).

Definicja w linii 214 pliku [lista.hh](#).

4.3.3.9 `template<class type, class type2> void Lista< type, type2 >::wyswietl () [inline]`

Definicja w linii 106 pliku [lista.hh](#).

4.3.3.10 `template<class type, class type2 > void Lista< type, type2 >::wyswietl (type key)`

Definicja w linii 223 pliku [lista.hh](#).

4.3.4 Dokumentacja atrybutów składowych

4.3.4.1 `template<class type, class type2> pole* Lista< type, type2 >::first`

Definicja w linii 37 pliku [lista.hh](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

4.4 Dokumentacja klasy Lista_tab

```
#include <lista_tab.hh>
```

Diagram dziedziczenia dla Lista_tab

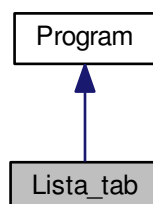
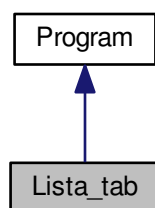


Diagram współpracy dla Lista_tab:



Metody publiczne

- [Lista_tab](#) ()
Konstruktor bezparametryczny.
- [~Lista_tab](#) ()
Destruktor.
- void [push](#) (int x)
Metoda push.
- void [pop](#) ()
Procedura pop.
- int [size](#) ()
Metoda size.
- bool [wykonaj_program](#) (char *nazwa_pliku, int ilosc_danych)
Metoda wykonaj_program.
- void [wyczysc_dane](#) (int ile)
Metoda wyczysc_dane.
- void [mergesort](#) (int beg, int end)
Metoda mergesort.
- void [test](#) ()
Metoda test.

Atrybuty publiczne

- int [rozmiar](#)
Aktualny rozmiar tablicy.
- int [iterator](#)
Iterator, zawierający informacje o liczbie elementów w tablicy.
- int * [tab](#)
Wskaźnik na dynamicznie alokowana tablice z danymi.

Dodatkowe Dziedziczone Składowe

4.4.1 Opis szczegółowy

Definicja w linii 12 pliku [lista_tab.hh](#).

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Lista_tab::Lista_tab () [inline]

Ustawia początek listy na NULL

Definicja w linii 31 pliku [lista_tab.hh](#).

4.4.2.2 Lista_tab::~~Lista_tab () [inline]

Usuwa dynamicznie utworzona tablice danych oraz przypisuje wskaźnikowi wartosc NULL.

Definicja w linii 43 pliku [lista_tab.hh](#).

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 void Lista_tab::mergesort (int beg, int end)

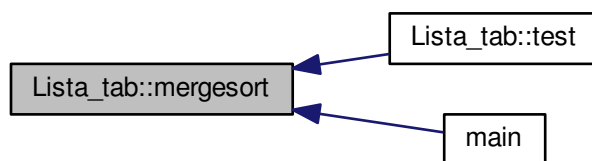
Dokonuje sortowania tablicy przez scalanie

Parametry

in	<i>beg</i>	Początek obszaru sortowania
in	<i>end</i>	Koniec obszaru sortowania

Definicja w linii 100 pliku [lista_tab.cpp](#).

Oto graf wywoływań tej funkcji:



4.4.3.2 void Lista_tab::pop ()

Usuwa ostatni element listy.

Definicja w linii 56 pliku [lista_tab.cpp](#).

4.4.3.3 void Lista_tab::push (int x)

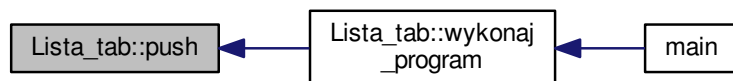
Dodaje podana wartosc na koniec listy. Do wyboru 2 metody push - po osiagnieciu maksymalnego rozmiaru tablicy, jedna z nich zwieksza rozmiar tablicy o 1, a druga podwaja aktualny rozmiar tablicy. Wyboru metody nalezy dokonac poprzedz odkomentowanie odpowiedniej metody w pliku .cpp.

Parametry

<code>in</code>	<code>x</code>	Wartosc, ktora chcemy dodac na koniec listy.
-----------------	----------------	--

Definicja w linii 32 pliku [lista_tab.cpp](#).

Oto graf wywołań tej funkcji:

**4.4.3.4 int Lista_tab::size ()**

Daje informacje o rozmiarze listy (liczbie jej elementow).

Zwraca

Rozmiar listy (liczba jej elementow)

Definicja w linii 73 pliku [lista_tab.cpp](#).

4.4.3.5 void Lista_tab::test () [inline],[virtual]

Wykonuje sortowanie przez scalanie

Reimplementowana z [Program](#).

Definicja w linii 107 pliku [lista_tab.hh](#).

Oto graf wywołań dla tej funkcji:

**4.4.3.6 void Lista_tab::wyczysc_dane (int ile) [virtual]**

Usuwa zadana ilosc elementow listy za pomoca metody pop

Parametry

<i>in</i>	<i>ile</i>	Liczba elementow, ktore chcemy usunac.
-----------	------------	--

Implementuje [Program](#).

Definicja w linii 93 pliku [lista_tab.cpp](#).

4.4.3.7 `bool Lista_tab::wykonaj_program (char * nazwa_pliku, int ilosc_danych) [virtual]`

Wykonuje zadany program - dodanie zadanej ilosci danych z pliku do listy za pomoca metody [push\(\)](#)

Zwracane wartości

<i>TRUE</i>	Poprawnie wykonano program
<i>FALSE</i>	Nie wczytano danych

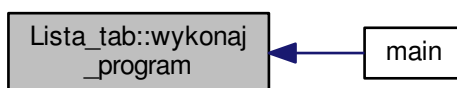
Implementuje [Program](#).

Definicja w linii 77 pliku [lista_tab.cpp](#).

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.4.4 Dokumentacja atrybutów składowych

4.4.4.1 `int Lista_tab::iterator`

Definicja w linii 20 pliku [lista_tab.hh](#).

4.4.4.2 `int Lista_tab::rozmiar`

Definicja w linii 16 pliku [lista_tab.hh](#).

4.4.4.3 int* Lista_tab::tab

Definicja w linii 24 pliku [lista_tab.hh](#).

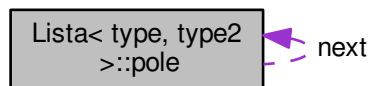
Dokumentacja dla tej klasy została wygenerowana z plików:

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

4.5 Dokumentacja struktury Lista< type, type2 >::pole

Struktura pole.

Diagram współpracy dla Lista< type, type2 >::pole:



Metody publiczne

- [pole \(\)](#)

Atrybuty publiczne

- type [val1](#)
- type2 [val2](#)
- pole * [next](#)

4.5.1 Opis szczegółowy

```
template<class type, class type2>struct Lista< type, type2 >::pole
```

Jest to struktura składowa klasy [Lista](#), zawierająca przechowywana wartość oraz wskaźnik na zmienną typu pole.

Definicja w linii 31 pliku [lista.hh](#).

4.5.2 Dokumentacja konstruktora i destruktor

4.5.2.1 `template<class type, class type2> Lista< type, type2 >::pole::pole () [inline]`

Definicja w linii 35 pliku [lista.hh](#).

4.5.3 Dokumentacja atrybutów składowych

4.5.3.1 `template<class type, class type2> pole* Lista< type, type2 >::pole::next`

Definicja w linii 34 pliku [lista.hh](#).

4.5.3.2 `template<class type, class type2> type Lista< type, type2 >::pole::val1`

Definicja w linii 32 pliku [lista.hh](#).

4.5.3.3 `template<class type, class type2> type2 Lista< type, type2 >::pole::val2`

Definicja w linii 33 pliku [lista.hh](#).

Dokumentacja dla tej struktury została wygenerowana z pliku:

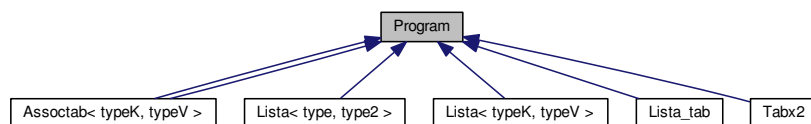
- [lista.hh](#)

4.6 Dokumentacja klasy Program

Modeluje klasę [Program](#).

```
#include <program.hh>
```

Diagram dziedziczenia dla Program



Metody publiczne

- `int getRozmiar_tab ()`
Akcesor [getRozmiar_tab](#).
- `Program ()`
Konstruktor bezparametryczny.
- `~Program ()`
Destruktor.
- `bool wczytaj_dane (char *nazwa_pliku)`
Metoda [wczytaj_dane](#).
- `bool wczytaj_dane (char *nazwa_pliku, int ile_danych)`
Metoda [wczytaj_dane](#).
- `bool zapisz_dane (char *nazwa_pliku)`
- `void wyswietl_dane ()`
Procedura [wyswietl_dane](#).
- `virtual bool wykonaj_program ()`
Wirtualna metoda [wykonaj_program](#).
- `virtual bool wykonaj_program (char *nazwa_pliku, int ilosc_danych)=0`

- virtual void [wyczysc_dane](#) (int ile)=0
- virtual void [test](#) ()

Atrybuty chronione

- int [rozmiar_tab](#)
Zmiana rozmiar_tab.
- int * [tab](#)
Zmienna tablica.
- ifstream [plik_we](#)
Zmienna plik_we.
- ofstream [plik_wy](#)
Zmienna plik_wy.

4.6.1 Opis szczegółowy

Klasa [Program](#) zawiera zmienne oraz metody wspólne dla wszystkich programow. Sa one zwiazane z przechowywaniem i obsluga danych.

Definicja w linii 22 pliku [program.hh](#).

4.6.2 Dokumentacja konstruktora i destruktor

4.6.2.1 `Program::Program () [inline]`

Przypisuje domyslne wartosc 0 dla rozmiaru tablicy danych oraz NULL dla wskaznika.

Definicja w linii 71 pliku [program.hh](#).

4.6.2.2 `Program::~~Program () [inline]`

Usuwa dynamicznie utworzona tablice danych oraz przypisuje wskaznikowi wartosc NULL.

Definicja w linii 79 pliku [program.hh](#).

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 `int Program::getRozmiar_tab () [inline]`

Metoda dajaca mozliwosc odczytu rozmiaru tablicy.

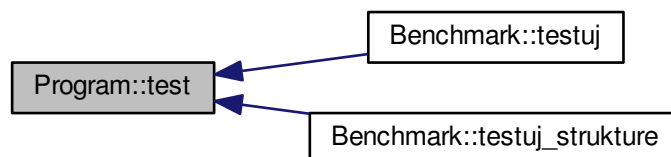
Definicja w linii 63 pliku [program.hh](#).

4.6.3.2 `virtual void Program::test () [inline],[virtual]`

Reimplementowana w [Lista_tab](#) i [Assoctab< typeK, typeV >](#).

Definicja w linii 150 pliku [program.hh](#).

Oto graf wywołań tej funkcji:



4.6.3.3 bool Program::wczytaj_dane (char * nazwa_pliku)

Wczytuje dane z pliku. W pierwszej linii pliku musi znajdować się informacja o ilości wczytywanych danych, dane w kolejnych liniach: ilosc_danych dana1 dana2 ...

Parametry

in	<i>nazwa_pliku</i>	Wskaźnik do nazwy pliku do wczytania.
----	--------------------	---------------------------------------

Zwracane wartości

<i>TRUE</i>	Poprawnie wczytano plik.
<i>FALSE</i>	Błąd podczas wczytywania pliku.

Definicja w linii 8 pliku [program.cpp](#).

4.6.3.4 bool Program::wczytaj_dane (char * nazwa_pliku, int ile_danych)

Wczytuje określoną liczbę danych z pliku. W pierwszej linii pliku musi znajdować się informacja o ilości wczytywanych danych, dane w kolejnych liniach: ilosc_danych dana1 dana2 ...

Parametry

in	<i>nazwa_pliku</i>	Wskaźnik do nazwy pliku do wczytania.
in	<i>ile_danych</i>	Ilość danych, jakie chcemy wczytać.

Zwracane wartości

<i>TRUE</i>	Poprawnie wczytano plik.
<i>FALSE</i>	Błąd podczas wczytywania pliku.

Definicja w linii 26 pliku [program.cpp](#).

4.6.3.5 virtual void Program::wyczyszc_dane (int ile) [pure virtual]

Implementowany w [Assoctab< typeK, typeV >](#), [Assoctab< typeK, typeV >](#), [Lista< type, type2 >](#), [Lista< typeK, typeV >](#) i [Lista_tab](#).

Oto graf wywoływań tej funkcji:



4.6.3.6 `bool Program::wykonaj_program () [virtual]`

Wykonuje program na zadanej liczbie danych.

Reimplementowana w [Tabx2](#).

Definicja w linii 67 pliku [program.cpp](#).

Oto graf wywoływań tej funkcji:



4.6.3.7 `virtual bool Program::wykonaj_program (char * nazwa_pliku, int ilosc_danych) [pure virtual]`

Implementowany w [Assoctab< typeK, typeV >](#), [Assoctab< typeK, typeV >](#), [Lista< type, type2 >](#), [Lista< typeK, typeV >](#) i [Lista_tab](#).

4.6.3.8 `void Program::wyswietl_dane ()`

Wypisuje wczytane dane jedna pod druga na standardowy strumien wyjścia.

Definicja w linii 62 pliku [program.cpp](#).

4.6.3.9 `bool Program::zapisz_dane (char * nazwa_pliku)`

Metoda zapisz_dane

Zapisuje przetworzone dane do pliku. W pierwszej linijce zamieszcza informacje o ilości danych, w kolejnych liniach pojedyncze dane: `ilosc_danych dana1 dana2 ...`

Parametry

<code>in</code>	<code>nazwa_pliku</code>	Wskaźnik do nazwy pliku do zapisu.
-----------------	--------------------------	------------------------------------

Zwracane wartości

<i>TRUE</i>	Poprawnie zapisano plik.
<i>FALSE</i>	Błąd podczas zapisu pliku.

Definicja w linii 47 pliku [program.cpp](#).

4.6.4 Dokumentacja atrybutów składowych

4.6.4.1 `ifstream Program::plik_we` `[protected]`

Zmienna przechowująca strumień wejściowy do otwartego pliku z wczytywanymi danymi.

Definicja w linii 47 pliku [program.hh](#).

4.6.4.2 `ofstream Program::plik_wy` `[protected]`

Zmienna przechowująca strumień wyjściowy do tworzonego pliku z danymi po przetworzeniu.

Definicja w linii 55 pliku [program.hh](#).

4.6.4.3 `int Program::rozmiar_tab` `[protected]`

Zmienna przechowująca informacje o ilości wczytanych danych, która równa jest długości utworzonej tablicy dynamicznej (wskazywanej wskaźnikiem `tab`).

Definicja w linii 31 pliku [program.hh](#).

4.6.4.4 `int* Program::tab` `[protected]`

Zmienna wskaźnikowa wskazująca na dynamicznie tworzoną tablicę z danymi.

Definicja w linii 39 pliku [program.hh](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

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

4.7 Dokumentacja klasy Tabx2

```
#include <tabx2.hh>
```

Diagram dziedziczenia dla Tabx2

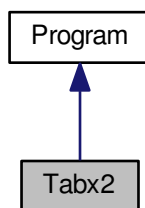
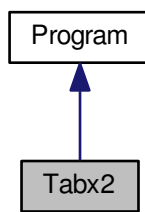


Diagram współpracy dla Tabx2:



Metody publiczne

- virtual bool `wykonaj_program()`
Metoda wirtualna wykonaj_program.

Dodatkowe Dziedziczone Składowe

4.7.1 Opis szczegółowy

Definicja w linii 18 pliku `tabx2.hh`.

4.7.2 Dokumentacja funkcji składowych

4.7.2.1 bool Tabx2::wykonaj_program() [virtual]

Dokonuje przemnożenia przez 2 wszystkich danych znajdujących się w tablicy wskazywanej przez `tab`.

Zwracane wartości

<i>TRUE</i>	Poprawnie dokonano mnożenia wszystkich liczb
<i>FALSE</i>	Rozmiar tablicy danych wynosi 0

Reimplementowana z [Program](#).

Definicja w linii 7 pliku [tabx2.cpp](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

- [tabx2.hh](#)
- [tabx2.cpp](#)

Rozdział 5

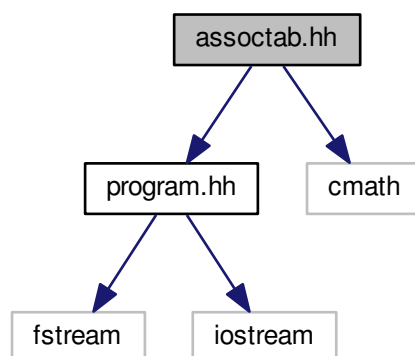
Dokumentacja plików

5.1 Dokumentacja pliku assoctab.hh

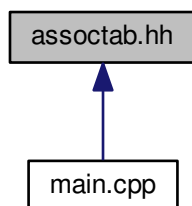
Definicja klasy [Assoctab](#).

```
#include "program.hh"  
#include <cmath>
```

Wykres zależności załączania dla assoctab.hh:



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



Komponenty

- class `Assoctab< typeK, typeV >`

Definicje

- `#define HASH 0.6180339887`
- `#define TAB 10000`

5.1.1 Dokumentacja definicji

5.1.1.1 `#define HASH 0.6180339887`

Definicja w linii 6 pliku `assoctab.hh`.

5.1.1.2 `#define TAB 10000`

Definicja w linii 7 pliku `assoctab.hh`.

5.2 assoctab.hh

```
00001 //assoctab.hh
00002 #ifndef ASSOCTAB_HH
00003 #define ASSOCTAB_HH
00004
00005 //Donald Knuth hashing const
00006 #define HASH 0.6180339887
00007 #define TAB 10000
00008
00009 #include "program.hh"
00010 #include <cmath>
00011
00017 template <class typeK, class typeV>
00018 class Assoctab: public Program{
00019 public:
00023     Lista<typeK, typeV> *tab;
00024
00025
00029     int rozmiar;
00030
00031 public:
00037     Assoctab(){
00038         tab = new Lista<typeK, typeV> [TAB];
00039         rozmiar = TAB;
```

```

00040     }
00041
00048 ~Assoctab() {delete[] tab; tab=NULL; rozmiar=0;}
00049
00050
00062 void push(typeK klucz, typeV wartosc);
00063
00069 void pop(typeK klucz);
00070
00076 int h(typeK klucz);
00077
00078 //int h(string klucz);
00079
00087 int size();
00088
00097 bool wykonaj_program(char* nazwa_pliku,int ilosc_danych);
00098
00104 void test(){
00105     //typeV a = tab[h("fa37")].daj("fa37");
00106     wyswietl("fa37");
00107 };
00108
00116 void wyczyszc_dane(int ile){}
00117
00118
00124 void wyswietl_liste(typeK klucz);
00132 void wyswietl(typeK klucz);
00133
00134 /*
00135  * \brief Przeciazanie operatora []
00136  */
00137 const typeV& operator[] (typeK klucz) const;
00138
00139 /*
00140  * \brief Przeciazanie operatora []
00141  */
00142 typeV& operator[] (typeK klucz);
00143
00144
00145
00146 };
00147
00148 template <class typeK, class typeV> void Assoctab<typeK, typeV>::push(typeK
klucz, typeV wartosc){
00149     tab[h(klucz)].push_front(klucz, wartosc);
00150 }
00151
00152 template <class typeK, class typeV> void Assoctab<typeK, typeV>::pop(typeK klucz
){
00153     tab[h(klucz)].pop(klucz);
00154 }
00155
00156 template<class typeK, class typeV> int Assoctab<typeK, typeV>::h(typeK klucz){
00157     double val=0; double add;
00158     for(unsigned int i=0; i<klucz.length(); i++){
00159         add = klucz[i]*(i+1);
00160         val+=add;
00161     }
00162     val*=HASH;
00163     val-=(int)val;
00164     return floor(rozmiar*val);
00165 }
00166
00167 template <class typeK, class typeV> int Assoctab<typeK, typeV>::size(){
00168     return rozmiar;
00169 }
00170
00171 template <class typeK, class typeV> bool Assoctab<typeK, typeV>::wykonaj_program
(char* nazwa_pliku,int ilosc_danych){
00172     typeK key;
00173     typeV val;
00174     plik_we.open(nazwa_pliku);
00175     if(plik_we.good()==false){
00176         cerr<<"Blad odczytu pliku!"<<endl;
00177         return false;
00178     }
00179     for(int i=0;i<ilosc_danych;i++){
00180         plik_we >> key;
00181         plik_we >> val;
00182         push(key,val);
00183     }
00184     plik_we.close();
00185     return true;
00186 }
00187
00188 template <class typeK, class typeV> void Assoctab<typeK, typeV>::wyswietl_liste
(typeK klucz){

```

```

00189     tab[h(klucz)].wyswietl();
00190 }
00191
00192 template <class typeK, class typeV> void Assoctab<typeK, typeV>::wyswietl(
    typeK klucz){
00193     tab[h(klucz)].wyswietl(klucz);
00194 }
00195
00196 template <class typeK, class typeV> const typeV&
    Assoctab<typeK,typeV>::operator[] (typeK klucz) const{
00197     return tab[h(klucz)].daj(klucz);
00198 }
00199
00200 template <class typeK, class typeV> typeV& Assoctab<typeK,typeV>::operator[]
    (typeK klucz){
00201     return tab[h(klucz)].daj(klucz);
00202 }
00203 #endif

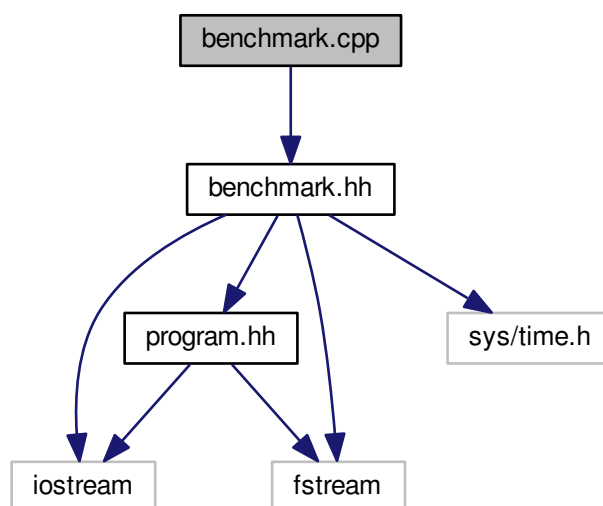
```

5.3 Dokumentacja pliku benchmark.cpp

Plik zawiera metody klasy [Benchmark](#).

```
#include "benchmark.hh"
```

Wykres zależności załączania dla benchmark.cpp:



5.4 benchmark.cpp

```

00001 #include "benchmark.hh"
00002
00007 void Benchmark::rozpoczni_j_pomiar(){
00008     gettimeofday(&t1, NULL);
00009 }
00010
00011 void Benchmark::zakoncz_pomiar(){
00012     gettimeofday(&t2, NULL);
00013     czas_pomiaru = (t2.tv_sec - t1.tv_sec) * 1000.0; // sec to ms
00014     czas_pomiaru += (t2.tv_usec - t1.tv_usec) / 1000.0; // us to ms
00015 }
00016
00017 double Benchmark::testuj(Program &program, char* dane, int ilosc_danych, int

```

```

    ilosc_testow){
00018     double suma=0;
00019     double srednia=0;
00020     ofstream wyniki;
00021     wyniki.open("wyniki.csv",ios::app);
00022
00023     //if(program.wczytaj_dane(dane,ilosc_danych)==false){
00024     //    cerr<<"Niewystarczajaca ilosc danych!"<<endl;
00025     //    return 0;
00026     // }
00027     //char* dane_wy = (char*)"dane_wy.dat"; //do zapisu do pliku
00028     //program.wykonaj_program();
00029     //rozpoczni_j_pomiar();
00030     //program.test();
00031     //program.zapisz_dane(dane_wy);//zapisywanie wynikow do pliku
00032     //zakoncz_pomiar();
00033     //suma+=czas_pomiaru;
00034     for(int i=1;i<ilosc_testow;i++){
00035         //program.wczytaj_dane(dane,ilosc_danych); //zawsze dane od poczatku
00036         rozpoczni_j_pomiar();
00037         //program.wykonaj_program();
00038         program.test();
00039         zakoncz_pomiar();
00040         suma+=czas_pomiaru;
00041     }
00042     srednia=suma/(ilosc_testow);
00043     wyniki<<endl<<ilosc_danych<<","<<srednia;
00044     wyniki.close();
00045     return srednia;
00046 }
00047
00048 double Benchmark::testuj_strukture(Program &program,char* dane, int
    ilosc_danych, int ilosc_testow){
00049     double suma=0;
00050     double srednia=0;
00051     ofstream wyniki;
00052
00053     for(int i=1;i<=ilosc_testow;i++){
00054         program.wykonaj_program(dane,ilosc_danych);
00055         rozpoczni_j_pomiar();
00056         program.test();
00057         zakoncz_pomiar();
00058         program.wyczyszc_dane(ilosc_danych);
00059         suma+=czas_pomiaru;
00060     }
00061     srednia=suma/(ilosc_testow);
00062
00063     wyniki.open("wyniki.csv",ios::app);
00064     wyniki<<endl<<ilosc_danych<<","<<srednia;
00065     wyniki.close();
00066     return srednia;
00067 }

```

5.5 Dokumentacja pliku benchmark.hh

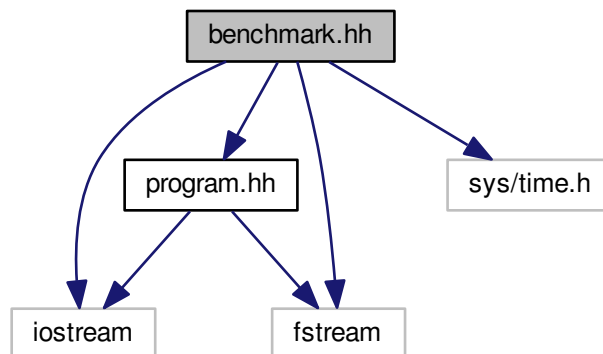
Definicja klasy [Benchmark](#).

```

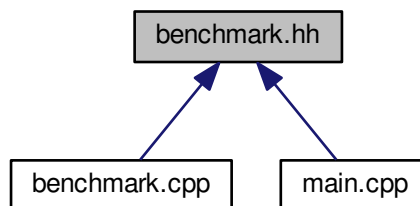
#include <iostream>
#include "program.hh"
#include <sys/time.h>
#include <fstream>

```

Wykres zależności załączania dla benchmark.hh:



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



Komponenty

- class [Benchmark](#)
Klasa [Benchmark](#).

5.6 benchmark.hh

```

00001 //benchmark.hh
00002
00003 #ifndef BENCHMARK_HH
00004 #define BENCHMARK_HH
00005
00006 #include <iostream>
00007 #include "program.hh"
00008 #include <sys/time.h>
00009 #include <fstream>
00010
00016 using namespace std;
00017
00023 class Benchmark{
00024 private:
  
```



```

00030     timeval t1, t2;
00031
00037     double czas_pomiaru;
00038
00039 public:
00045     void rozpocznij_pomiar();
00046
00052     void zakoncz_pomiar();
00053
00066     double testuj(Program &program, char* dane, int ilosc_danych, int ilosc_testow);
00067
00081     double testuj_strukture(Program &program, char* dane, int ilosc_danych, int ilosc_testow);
00082 };
00083
00084 #endif

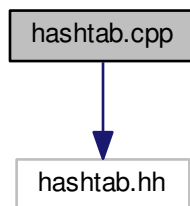
```

5.7 Dokumentacja pliku hashtab.cpp

Plik zawiera metody klasy Hashtab.

```
#include "hashtab.hh"
```

Wykres zależności załączania dla hashtab.cpp:



5.8 hashtab.cpp

```

00001 #include "hashtab.hh"
00002
00003 using namespace std;
00008 void Hashtab::push(string klucz, string wartosc){
00009     tab[h(klucz)] = wartosc;
00010 }
00011
00012 void Hashtab::pop(string klucz){
00013     tab[h(klucz)] = "0";
00014 }
00015
00016 void Hashtab::wyswietl(string klucz){
00017     cout << tab[h(klucz)];
00018 }
00019
00020 int Hashtab::size(){
00021     return rozmiar;
00022 }
00023
00024
00025 int Hashtab::h(string klucz){
00026     int val=0; int add;
00027     for(unsigned int i=0; i<klucz.length(); i++){
00028         add = klucz[i]*(i+1);
00029         val+=add;
00030     }
00031     return val%rozmiar;
00032 }
00033

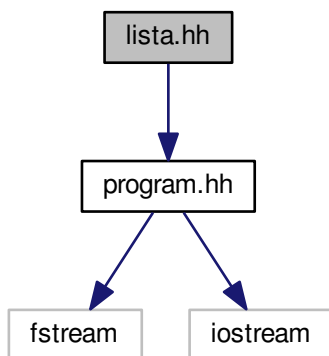
```

5.9 Dokumentacja pliku lista.hh

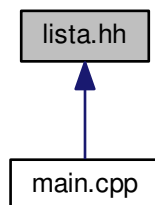
Definicja klasy [Lista](#).

```
#include "program.hh"
```

Wykres zależności załączania dla lista.hh:



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



Komponenty

- class [Lista](#)< type, type2 >
- struct [Lista](#)< type, type2 >::pole

Struktura pole.

5.10 lista.hh

```
00001 //lista.hh
00002 #ifndef LISTA_HH
00003 #define LISTA_HH
00004
00005 #include "program.hh"
```

```

00006
00012 /*
00013 struct pole{
00014     int wartosc;
00015     pole *next;
00016     pole(){wartosc=0; next=NULL;}
00017 };
00018 */
00019
00020
00021 template <class type, class type2>
00022 class Lista: public Program{
00023     //zmien
00031     struct pole{
00032         type val1;
00033         type2 val2;
00034         pole *next;
00035         pole(){next=NULL;}
00036     };
00037 public: pole *first;
00038 public:
00044     Lista(){
00045         first = NULL;
00046     }
00054     void push(type x, type2 y);
00055
00063     void push_front(type x, type2 y);
00064
00070     void pop();
00078     int size();
00079
00089     bool wykonaj_program(char* nazwa_pliku,int ilosc_danych);
00090
00098     void wyczyszc_dane(int ile);
00099
00100     /*
00101     *!
00102     * \brief Metoda wyswietl
00103     *
00104     * Wyswietla wszystkie pola listy
00105     */
00106     void wyswietl(){
00107         if(first == NULL){
00108             cerr<<"Lista jest pusta!"<<endl;
00109         }
00110         else{
00111             pole *wsk = first;
00112             cout<<"Key: "<<wsk->val1<< " , Value:"<<wsk->val2<<endl;
00113             while(wsk->next){
00114                 wsk=wsk->next;
00115                 cout<<"Key: "<<wsk->val1<< " , Value:"<<wsk->val2<<endl;
00116             }
00117         }
00118     }
00119
00120     /*
00121     *!
00122     * \brief Metoda wyswietl z argumentem key
00123     *
00124     * Wyswietla wszystkie pola, ktorzy pole x ma podany klucz
00125     *
00126     * \param[in] Klucz, dla ktorego chcemy wyswietlic wszystkie pola o identycznym kluczu
00127     */
00128     void wyswietl(type key);
00129
00130     /*
00131     *!
00132     * \brief Metoda daj z argumentem key
00133     *
00134     * \return Zwraca wartosc pierwszego napotkanego elementu na liscie, o kluczu rownym key
00135     */
00136     type2& daj(type key);
00137
00145     void pop(type key);
00146 };
00147
00148 template <class type, class type2> void Lista<type, type2>::push(type x, type2 y){
00149     pole *nowe = new pole;
00150     nowe->val1 = x;
00151     nowe->val2 = y;
00152     if(first == NULL){
00153         first = nowe;
00154     }
00155     else{
00156         pole *wsk = first;
00157         while(wsk->next)
00158             wsk=wsk->next;

```

```

00159     wsk->next = nowe;
00160     nowe->next = NULL;
00161 }
00162 }
00163
00164 template <class type, class type2> void Lista<type, type2>::push_front(type x
, type2 y){
00165     pole *nowe = new pole;
00166     nowe->vall = x;
00167     nowe->val2 = y;
00168     if(first == NULL){
00169         first = nowe;
00170     }
00171     else{
00172         pole *wsk = first->next;
00173         first->next = nowe;
00174         nowe->next = wsk;
00175     }
00176 }
00177
00178 template <class type, class type2> void Lista<type, type2>::pop(){
00179     if(first == NULL){
00180         cerr<<"Lista jest pusta!"<<endl;
00181     }
00182     else{
00183         pole *wsk = first;
00184         pole *prev = NULL;
00185         while(wsk->next){
00186             prev=wsk;
00187             wsk=wsk->next;
00188         }
00189         if(prev==NULL){
00190             delete wsk;
00191             wsk = NULL;
00192         }
00193         else{
00194             prev->next = NULL;
00195             delete wsk;
00196         }
00197     }
00198 }
00199
00200 template <class type, class type2> int Lista<type, type2>::size(){
00201     if(first == NULL)
00202         return 0;
00203     else{
00204         pole *wsk = first;
00205         int i=1;
00206         while(wsk->next){
00207             wsk=wsk->next;
00208             i++;
00209         }
00210         return i;
00211     }
00212 }
00213
00214 template <class type, class type2> bool Lista<type, type2>::wykonaj_program
(char* nazwa_pliku,int ilosc_danych){
00215     return true;
00216 }
00217
00218 template <class type, class type2> void Lista<type, type2>::wyczyszc_dane(
int ile){
00219     for(int i=0;i<ile;i++){
00220         pop();
00221     }
00222 }
00223 template<class type, class type2> void Lista<type, type2>::wyswietl(type key){
00224     if(first == NULL){
00225         cerr<<"Lista jest pusta!"<<endl;
00226     }
00227     else{
00228         pole *wsk = first;
00229         if(wsk->vall == key)
00230             cout<<"Key: "<<wsk->vall<<" , Value:"<<wsk->val2<<endl;
00231         while(wsk->next){
00232             wsk=wsk->next;
00233             if(wsk->vall == key)
00234                 cout<<"Key: "<<wsk->vall<<" , Value:"<<wsk->val2<<endl;
00235         }
00236     }
00237 }
00238 template<class type, class type2> type2& Lista<type, type2>::daj(type key){
00239     if(first == NULL){
00240         cerr<<"Brak elementu o podanym kluczu!"<<endl;
00241         //type2 a;
00242         // return a;

```

```

00243     }
00244     else{
00245         pole *wsk = first;
00246         if(wsk->val1 == key)
00247             return wsk->val2;
00248         while(wsk->next){
00249             wsk=wsk->next;
00250             if(wsk->val1 == key)
00251                 return wsk->val2;;
00252         }
00253     }
00254     //type2 a;
00255     //return a;
00256 }
00257
00258 template<class type, class type2> void Lista<type, type2>::pop(type key){
00259     pole *wsk = first;
00260     pole *prev = NULL;
00261     if(first == NULL){
00262         cerr<<"Lista jest pusta!"<<endl;
00263     }
00264     else{
00265         while(wsk->val1==key) {
00266             if(wsk->val1==key && wsk->next==NULL){
00267                 delete wsk;
00268                 wsk = first;
00269             }
00270             else{
00271                 first = wsk->next;
00272                 delete wsk;
00273                 wsk = first;
00274             }
00275         }
00276     }
00277     while(wsk->next){
00278         prev=wsk;
00279         wsk=wsk->next;
00280         if(wsk->val1==key){
00281             prev->next=wsk->next;
00282             delete wsk;
00283             wsk=prev->next;
00284         }
00285         else{
00286             prev=wsk;
00287             wsk=wsk->next;
00288         }
00289     }
00290 }
00291
00292
00293 #endif

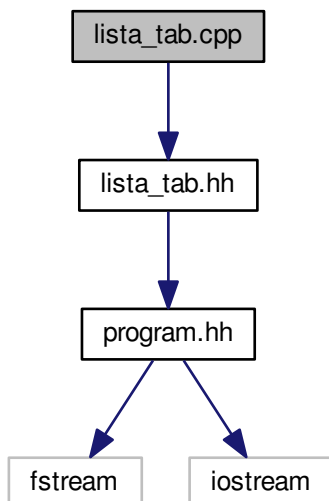
```

5.11 Dokumentacja pliku lista_tab.cpp

Zawiera definicje metod klasy [Lista](#).

```
#include "lista_tab.hh"
```

Wykres zależności załączania dla lista_tab.cpp:



5.12 lista_tab.cpp

```

00001 //lista_tab.cpp
00002 #include "lista_tab.hh"
00003
00007 // ZWIEKSZA SIE O 1
00008 /*void Lista_tab::push(int x){
00009     if(rozmiar==0){
00010         tab = new int [1];
00011         rozmiar=1;
00012         iterator=0;
00013         tab[iterator]=x;
00014     }
00015     else{
00016         if(iterator<rozmiar-1){
00017             tab[++iterator]=x;
00018         }
00019         else if(iterator>=rozmiar-1){
00020             int *tmp = new int[rozmiar+1];
00021             for(int i=0;i<=iterator;i++)
00022                 tmp[i] = tab[i];
00023             delete [] tab;
00024             tab = tmp;
00025             tab[++iterator]=x;
00026             rozmiar++;
00027         }
00028     }
00029 }*/
00030 //TABLICA POWIEKSZANA DWUKROTNIE
00031
00032 void Lista_tab::push(int x){
00033     if(rozmiar==0){
00034         tab = new int [1];
00035         rozmiar=1;
00036         iterator=0;
00037         tab[iterator]=x;
00038     }
00039     else{
00040         if(iterator<rozmiar-1){
00041             tab[++iterator]=x;
00042         }
00043         else if(iterator>=rozmiar-1){
00044             int *tmp = new int[2*rozmiar];

```

```

00045         for(int i=0;i<=iterator;i++)
00046             tmp[i] = tab[i];
00047         delete [] tab;
00048         tab = tmp;
00049         tab[++iterator]=x;
00050         rozmiar*=2;
00051     }
00052 }
00053 }
00054
00055
00056 void Lista_tab::pop(){
00057     if(rozmiar == 0){
00058         cerr<<"Lista jest pusta!"<<endl;
00059     }
00060     iterator--;
00061     if(iterator<0.25*(rozmiar-1)){
00062         int *tmp = new int[iterator+1];
00063         for(int i=0;i<=iterator;i++){
00064             tmp[i]=tab[i];
00065         }
00066         delete [] tab;
00067         tab = tmp;
00068         rozmiar = iterator+1;
00069     }
00070 }
00071
00072
00073 int Lista_tab::size(){
00074     return rozmiar;
00075 }
00076
00077 bool Lista_tab::wykonaj_program(char* nazwa_pliku, int ilosc_danych){
00078     int tmp;
00079     plik_we.open(nazwa_pliku);
00080     if(plik_we.good()==false){
00081         cerr<<"Blad odczytu pliku!"<<endl;
00082         return false;
00083     }
00084     plik_we >> rozmiar_tab; //pozbyc sie pierwszej liczby z pliku z danymi
00085     for(int i=0;i<ilosc_danych;i++){
00086         plik_we >> tmp;
00087         push(tmp);
00088     }
00089     plik_we.close();
00090     return true;
00091 }
00092
00093 void Lista_tab::wyczyszc_dane(int ile){
00094     delete [] tab;
00095     rozmiar = 0;
00096     iterator = 0;
00097     tab = NULL;
00098 }
00099
00100 void Lista_tab::mergesort(int beg, int end){
00101     int tmp[end+1];
00102     int mid = (beg+end+1)/2; int i1, i2,i;
00103     if(mid-beg>1)
00104         mergesort(beg,mid-1);
00105     if(end-mid>0)
00106         mergesort(mid,end);
00107     i1=beg; i2=mid;
00108     for(i=beg; i<=end; i++)
00109         tmp[i]=((i1==mid) || ((i2<=end) && (tab[i1] > tab[i2]))) ? tab[i2++] :
tab[i1++];
00110     for(i=beg; i<=end; i++) tab[i]=tmp[i];
00111 }

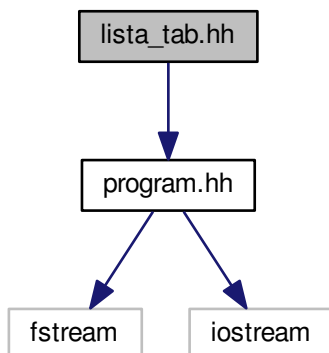
```

5.13 Dokumentacja pliku lista_tab.hh

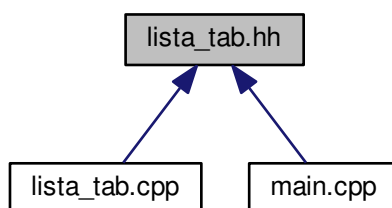
Definicja klasy [Lista_tab](#).

```
#include "program.hh"
```

Wykres zależności załączania dla lista_tab.hh:



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



Komponenty

- class [Lista_tab](#)

Definicje

- `#define LISTA__TAB_HH`

5.13.1 Dokumentacja definicji

5.13.1.1 `#define LISTA__TAB_HH`

Definicja w linii 3 pliku [lista_tab.hh](#).

5.14 lista_tab.hh

```

00001 //lista_tab.hh
00002 #ifndef LISTA_TAB_HH
00003 #define LISTA_TAB_HH
00004
00005 #include "program.hh"
00006
00012 class Lista_tab: public Program{
00013 public:
00016     int rozmiar;
00020     int iterator;
00024     int *tab;
00025 public:
00031     Lista_tab(){
00032         tab = NULL;
00033         rozmiar = 0;
00034         iterator = 0;
00035     }
00036
00043     ~Lista_tab(){delete[] tab; tab=NULL; rozmiar=0;
        iterator=0;}
00044
00056     void push(int x);
00062     void pop();
00070     int size();
00071
00081     bool wykonaj_program(char* nazwa_pliku,int ilosc_danych);
00082
00090     void wyczyszc_dane(int ile);
00091
00100     void mergesort(int beg, int end);
00101
00107     void test() {mergesort(0,iterator);};
00108 };
00109
00110 #endif

```

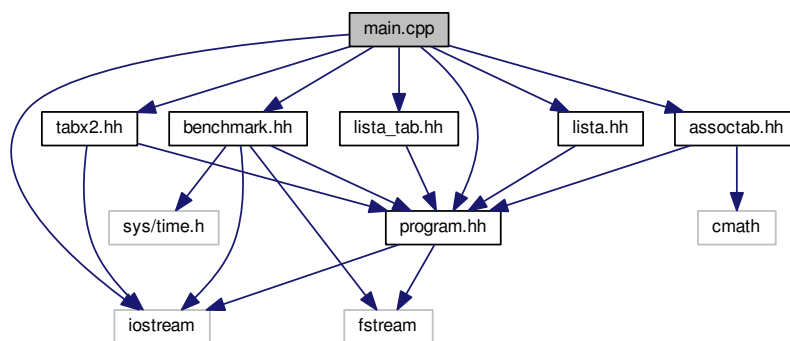
5.15 Dokumentacja pliku main.cpp

```

#include <iostream>
#include "program.hh"
#include "tabx2.hh"
#include "benchmark.hh"
#include "lista.hh"
#include "lista_tab.hh"
#include "assoctab.hh"

```

Wykres zależności załączania dla main.cpp:



Funkcje

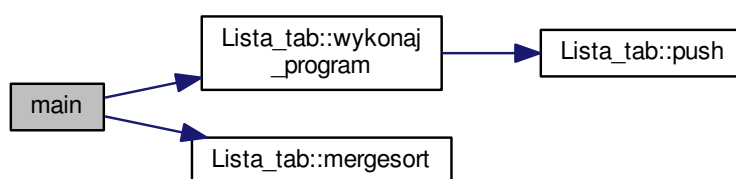
- int [main](#) ()

5.15.1 Dokumentacja funkcji

5.15.1.1 int main ()

Definicja w linii 12 pliku [main.cpp](#).

Oto graf wywołań dla tej funkcji:



5.16 main.cpp

```

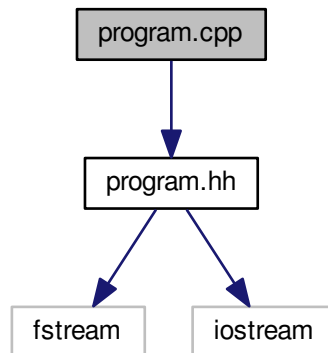
00001 //main.cpp
00002 #include <iostream>
00003 #include "program.hh"
00004 #include "tabx2.hh"
00005 #include "benchmark.hh"
00006 #include "lista.hh"
00007 #include "lista_tab.hh"
00008 #include "assotab.hh"
00009
00010 using namespace std;
00011
00012 int main(){
00013     Lista_tab a;
00014     Benchmark b;
00015     char* dane = (char*)"dane.dat";
00016     int ilosc_testow = 10;
00017
00018     a.wykonaj_program(dane, 10);
00019     for(int i=0; i<=a.iterator; i++)
00020         cout<< a.tab[i]<<endl;
00021     a.mergesort(0,a.iterator);
00022     cout<<endl<<endl;
00023     for(int i=0; i<=a.iterator; i++)
00024         cout<< a.tab[i]<<endl;
00025
00026     /*
00027     for(int ilosc_danych=1; ilosc_danych<=10000000;ilosc_danych*=10){
00028         cout << b.testuj_strukture(a,dane,ilosc_danych,ilosc_testow) << endl;
00029     }
00030     */
00031     return 0;
00032 }
  
```

5.17 Dokumentacja pliku program.cpp

Plik zawiera metody klasy [Program](#).

```
#include "program.hh"
```

Wykres zależności załączania dla program.cpp:



5.18 program.cpp

```

00001 //program.cpp
00002 #include "program.hh"
00003
00004 using namespace std;
00008 bool Program::wczytaj_dane(char* nazwa_pliku) {
00009     if(plik_we.good() == true)
00010        plik_we.close();
00011    plik_we.open(nazwa_pliku);
00012     if(plik_we.good() == false) {
00013         cerr<<"Błąd odczytu pliku!";
00014         return false;
00015     }
00016    plik_we >> rozmiar_tab;
00017     tab=new int [rozmiar_tab];
00018     int i=0;
00019     while(plik_we >> tab[i]){
00020         i++;
00021     }
00022    plik_we.close();
00023     return true;
00024 }
00025
00026 bool Program::wczytaj_dane(char* nazwa_pliku, int ile_danych){
00027     if(plik_we.good() == true)
00028        plik_we.close();
00029    plik_we.open(nazwa_pliku);
00030     if(plik_we.good() == false) {
00031         cerr<<"Błąd odczytu pliku!"<<endl;
00032         return false;
00033     }
00034    plik_we >> rozmiar_tab;
00035     if(ile_danych>rozmiar_tab){
00036        plik_we.close();
00037         return false;
00038     }
00039     rozmiar_tab=ile_danych;
00040     tab=new int [ile_danych];
00041     for(int i=0;i<ile_danych;i++)
00042        plik_we>>tab[i];
00043    plik_we.close();
00044     return true;
00045 }
00046
00047 bool Program::zapisz_dane(char* nazwa_pliku) {
00048     if(plik_wy.good() == true)
00049        plik_wy.close();
00050    plik_wy.open(nazwa_pliku);
00051     if(plik_wy.good() == false) {

```

```

00052     cerr<<"Bład odczytu pliku!";
00053     return false;
00054 }
00055 plik_wy << rozmiar_tab << endl;
00056 for(int i=0;i<rozmiar_tab;i++)
00057     plik_wy << tab[i] << endl;
00058 plik_wy.close();
00059 return true;
00060 }
00061
00062 void Program::wyswietl_dane(){
00063     for(int i=0;i<rozmiar_tab;i++)
00064         cout<<tab[i]<<endl;
00065 }
00066
00067 bool Program::wykonaj_program(){
00068     cerr<<"Nie wybrano programu do wykonania!"<<endl;
00069     return false;
00070 }

```

5.19 Dokumentacja pliku program.hh

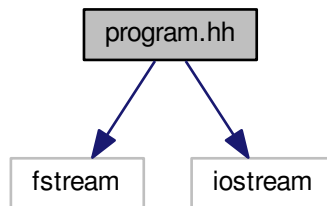
Definicja klasy [Program](#).

```

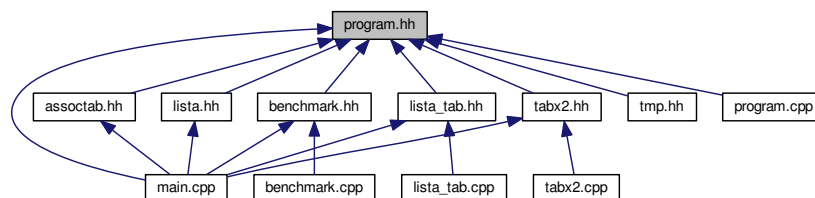
#include <fstream>
#include <iostream>

```

Wykres zależności załączania dla program.hh:



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



Komponenty

- class [Program](#)

Modeluje klasę [Program](#).

5.20 program.hh

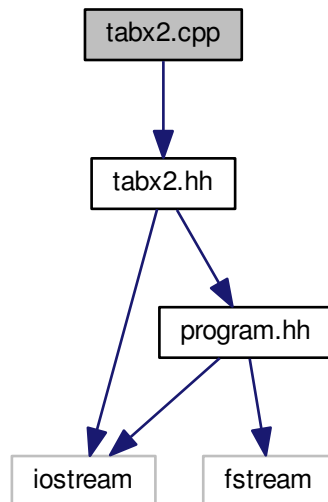
```
00001 //program.hh
00002
00003 #ifndef PROGRAM_HH
00004 #define PROGRAM_HH
00005
00006 #include <fstream>
00007 #include <iostream>
00008
00009 using namespace std;
00010
00022 class Program{
00023 protected:
00031     int rozmiar_tab;
00032
00039     int *tab;
00040
00047     ifstream plik_we;
00048
00055     ofstream plik_wy;
00056
00057 public:
00063     int getRozmiar_tab(){return rozmiar_tab;}
00064
00071     Program(){rozmiar_tab=0;tab=NULL;}
00072
00079     ~Program(){delete[] tab; tab=NULL;}
00080
00096     bool wczytaj_dane(char* nazwa_pliku);
00097
00114     bool wczytaj_dane(char* nazwa_pliku, int ile_danych);
00115
00131     bool zapisz_dane(char* nazwa_pliku);
00132
00138     void wyswietl_dane();
00139
00145     virtual bool wykonaj_program();
00146
00147     virtual bool wykonaj_program(char* nazwa_pliku, int ilosc_danych)=0;
00148     virtual void wyczyszc_dane(int ile)=0;
00149
00150     virtual void test(){};
00151 };
00152
00153 #endif
```

5.21 Dokumentacja pliku tabx2.cpp

Plik zawiera metody klasy [Tabx2](#).

```
#include "tabx2.hh"
```

Wykres zależności załączania dla tabx2.cpp:



5.22 tabx2.cpp

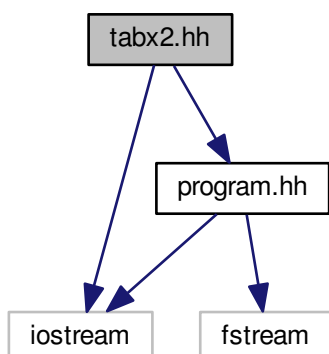
```
00001 #include "tabx2.hh"
00002
00003 using namespace std;
00007 bool Tabx2::wykonaj_program() {
00008     if (rozmiar_tab==0) {
00009         cerr<<"Brak danych wejsciowych!"<<endl;
00010         return false;
00011     }
00012     for(int i=0; i<rozmiar_tab; i++) {
00013         tab[i]*=2;
00014     }
00015     return true;
00016 }
```

5.23 Dokumentacja pliku tabx2.hh

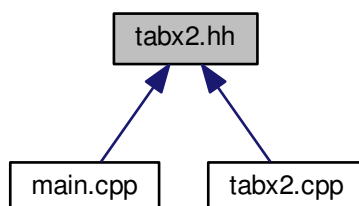
Definicja klasy [Tabx2](#).

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

Wykres zależności załączania dla tabx2.hh:



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



Komponenty

- class [Tabx2](#)

5.23.1 Opis szczegółowy

Klasa [Tabx2](#) jest klasa pochodna od klasy [Program](#). Definiuje metode podawajajaca kazda liczbe znajdujaca sie w tablicy danych wskazywanej przez zmienna tab klasy [Program](#).

Definicja w pliku [tabx2.hh](#).

5.24 tabx2.hh

```
00001 //mnozenie_tablicy.hh
00002
00003 #ifndef TABX2_HH
00004 #define TABX2_HH
00005
```

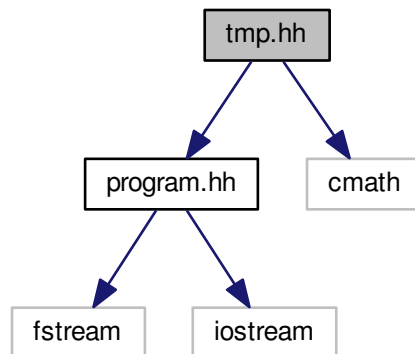
```
00015 #include <iostream>
00016 #include "program.hh"
00017
00018 class Tabx2: public Program{
00019
00020 public:
00030     virtual bool wykonaj_program();
00031 };
00032
00033 #endif
```

5.25 Dokumentacja pliku tmp.hh

Definicja klasy [Assoctab](#).

```
#include "program.hh"
#include <cmath>
```

Wykres zależności załączania dla tmp.hh:



Komponenty

- class [Assoctab](#)< typeK, typeV >

Definicje

- #define [HASH](#) 0.6180339887
- #define [TAB](#) 1

5.25.1 Dokumentacja definicji

5.25.1.1 #define HASH 0.6180339887

Definicja w linii 6 pliku [tmp.hh](#).

5.25.1.2 #define TAB 1

Definicja w linii 7 pliku [tmp.hh](#).

5.26 tmp.hh

```

00001 //assoctab.hh
00002 #ifndef ASSOCTAB_HH
00003 #define ASSOCTAB_HH
00004
00005 //Donald Knuth hashing const
00006 #define HASH 0.6180339887
00007 #define TAB 1
00008
00009 #include "program.hh"
00010 #include <cmath>
00011
00017 template <class typeK, class typeV>
00018 class Assoctab: public Program{
00019 public:
00023     Lista<typeK, typeV> *tab;
00024
00025
00029     int rozmiar;
00030
00031 public:
00037     Assoctab(){
00038         tab = new Lista<typeK, typeV> [TAB];
00039         rozmiar = TAB;
00040     }
00041
00048     ~Assoctab(){delete[] tab; tab=NULL; rozmiar=0;}
00049
00050
00062     void push(typeK klucz, typeV wartosc);
00063
00069     void pop(typeK klucz);
00070
00076     int h(typeK klucz);
00077
00078     //int h(string klucz);
00079
00087     int size();
00088
00098     bool wykonaj_program(char* nazwa_pliku,int ilosc_danych){return true;}
00099
00107     void wyczyszc_dane(int ile){}
00108
00109
00115     void wyswietl_liste(typeK klucz);
00123     void wyswietl(typeK klucz);
00124 };
00125
00126 template <class typeK, class typeV> void Assoctab<typeK, typeV>::push(typeK
klucz, typeV wartosc){
00127     tab[h(klucz)].push(klucz, wartosc);
00128 }
00129
00130 template <class typeK, class typeV> void Assoctab<typeK, typeV>::pop(typeK klucz
){
00131     tab[h(klucz)].pop(klucz);
00132 }
00133
00134 template<class typeK, class typeV> int Assoctab<typeK, typeV>::h(typeK klucz){
00135     double val=0; double add;
00136     for(unsigned int i=0; i<klucz.length(); i++){
00137         add = klucz[i]*(i+1);
00138         val+=add;
00139     }
00140     val*=HASH;
00141     val=(int)val;
00142     return floor(rozmiar*val);
00143 }
00144 /*
00145 template <class typeK, class typeV> int Assoctab<typeK, typeV>::h(typeK klucz){
00146     double val = (double)klucz;
00147     val*=HASH;
00148     val=(int)val;
00149
00150     return floor(rozmiar*val);
00151 }
00152 */
00153
00154 template <class typeK, class typeV> int Assoctab<typeK, typeV>::size(){
00155     return rozmiar;
00156 }
00157
00158 template <class typeK, class typeV> void Assoctab<typeK, typeV>::wyswietl_liste
(typeK klucz){
00159     tab[h(klucz)].wyswietl();

```

```
00160 }
00161
00162 template <class typeK, class typeV> void Assoctab<typeK, typeV>::wyswietl(
    typeK klucz){
00163     tab[h(klucz)].wyswietl(klucz);
00164 }
00165
00166
00167 #endif
```

Skorowidz

- ~Assoctab
 - Assoctab, [9](#)
- ~Lista_tab
 - Lista_tab, [23](#)
- ~Program
 - Program, [28](#)
- Assoctab
 - ~Assoctab, [9](#)
 - Assoctab, [9](#)
 - h, [10](#)
 - pop, [10](#)
 - push, [10](#)
 - rozmiar, [14](#)
 - size, [11](#)
 - tab, [14](#)
 - test, [11](#)
 - wyczysc_dane, [11](#)
 - wykonaj_program, [13](#)
 - wyswietl, [13](#)
 - wyswietl_liste, [14](#)
- Assoctab< typeK, typeV >, [7](#)
- assoctab.hh, [35](#), [36](#)
 - HASH, [36](#)
 - TAB, [36](#)
- Benchmark, [14](#)
 - czas_pomiaru, [17](#)
 - rozpoczniej_pomiar, [15](#)
 - t1, [17](#)
 - t2, [18](#)
 - testuj, [15](#)
 - testuj_strukture, [16](#)
 - zakoncz_pomiar, [17](#)
- benchmark.cpp, [38](#)
- benchmark.hh, [39](#), [40](#)
- czas_pomiaru
 - Benchmark, [17](#)
- daj
 - Lista, [19](#)
- first
 - Lista, [21](#)
- getRozmiar_tab
 - Program, [28](#)
- h
 - Assoctab, [10](#)
- HASH
 - assoctab.hh, [36](#)
 - tmp.hh, [56](#)
 - hashtab.cpp, [41](#)
- iterator
 - Lista_tab, [25](#)
- LISTA__TAB_HH
 - lista_tab.hh, [48](#)
- Lista
 - daj, [19](#)
 - first, [21](#)
 - Lista, [19](#)
 - pop, [19](#)
 - push, [20](#)
 - push_front, [20](#)
 - size, [20](#)
 - wyczysc_dane, [20](#)
 - wykonaj_program, [20](#)
 - wyswietl, [21](#)
- Lista< type, type2 >, [18](#)
- Lista< type, type2 >::pole, [26](#)
- lista.hh, [42](#)
- Lista::pole
 - next, [27](#)
 - pole, [26](#)
 - val1, [27](#)
 - val2, [27](#)
- Lista_tab, [21](#)
 - ~Lista_tab, [23](#)
 - iterator, [25](#)
 - Lista_tab, [23](#)
 - Lista_tab, [23](#)
 - mergesort, [23](#)
 - pop, [23](#)
 - push, [23](#)
 - rozmiar, [25](#)
 - size, [24](#)
 - tab, [25](#)
 - test, [24](#)
 - wyczysc_dane, [24](#)
 - wykonaj_program, [25](#)
- lista_tab.cpp, [45](#), [46](#)
- lista_tab.hh, [47](#), [49](#)
 - LISTA__TAB_HH, [48](#)
- main
 - main.cpp, [50](#)
 - main.cpp, [49](#), [50](#)

- main, [50](#)
- mergesort
 - Lista_tab, [23](#)
- next
 - Lista::pole, [27](#)
- plik_we
 - Program, [31](#)
- plik_wy
 - Program, [31](#)
- pole
 - Lista::pole, [26](#)
- pop
 - Assoctab, [10](#)
 - Lista, [19](#)
 - Lista_tab, [23](#)
- Program, [27](#)
 - ~Program, [28](#)
 - getRozmiar_tab, [28](#)
 - plik_we, [31](#)
 - plik_wy, [31](#)
 - Program, [28](#)
 - rozmiar_tab, [31](#)
 - tab, [31](#)
 - test, [28](#)
 - wczytaj_dane, [29](#)
 - wyczysc_dane, [29](#)
 - wykonaj_program, [30](#)
 - wyswietl_dane, [30](#)
 - zapisz_dane, [30](#)
- program.cpp, [50](#), [51](#)
- program.hh, [52](#), [53](#)
- push
 - Assoctab, [10](#)
 - Lista, [20](#)
 - Lista_tab, [23](#)
- push_front
 - Lista, [20](#)
- rozmiar
 - Assoctab, [14](#)
 - Lista_tab, [25](#)
- rozmiar_tab
 - Program, [31](#)
- rozpocznij_pomiar
 - Benchmark, [15](#)
- size
 - Assoctab, [11](#)
 - Lista, [20](#)
 - Lista_tab, [24](#)
- t1
 - Benchmark, [17](#)
- t2
 - Benchmark, [18](#)
- TAB
 - assoctab.hh, [36](#)
 - tmp.hh, [56](#)
- tab
 - Assoctab, [14](#)
 - Lista_tab, [25](#)
 - Program, [31](#)
- Tabx2, [31](#)
 - wykonaj_program, [32](#)
- tabx2.cpp, [53](#), [54](#)
- tabx2.hh, [54](#), [55](#)
- test
 - Assoctab, [11](#)
 - Lista_tab, [24](#)
 - Program, [28](#)
- testuj
 - Benchmark, [15](#)
- testuj_strukture
 - Benchmark, [16](#)
- tmp.hh, [56](#), [57](#)
 - HASH, [56](#)
 - TAB, [56](#)
- val1
 - Lista::pole, [27](#)
- val2
 - Lista::pole, [27](#)
- wczytaj_dane
 - Program, [29](#)
- wyczysc_dane
 - Assoctab, [11](#)
 - Lista, [20](#)
 - Lista_tab, [24](#)
 - Program, [29](#)
- wykonaj_program
 - Assoctab, [13](#)
 - Lista, [20](#)
 - Lista_tab, [25](#)
 - Program, [30](#)
 - Tabx2, [32](#)
- wyswietl
 - Assoctab, [13](#)
 - Lista, [21](#)
- wyswietl_dane
 - Program, [30](#)
- wyswietl_liste
 - Assoctab, [14](#)
- zakoncz_pomiar
 - Benchmark, [17](#)
- zapisz_dane
 - Program, [30](#)