Podstawowe struktury danych

1

Wygenerowano przez Doxygen 1.8.6

Cz, 19 mar 2015 00:47:18

Spis treści

Rozdział 1

Sprawozdanie

Data

19.03.2015r.

Wersja

0.1

1.1 Zadanie

Celem ćwiczenie było stworzenie trzech abstrakcyjnych typów danych :

-Kolejki (LIFO)

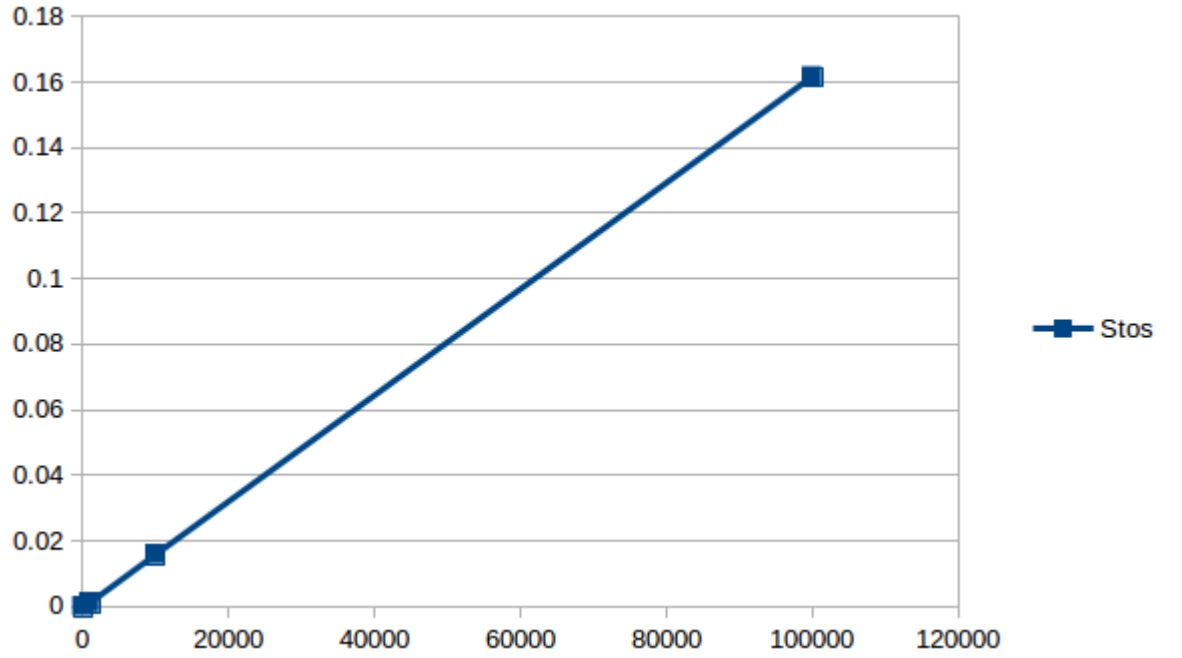
-Stosu (FIFO)

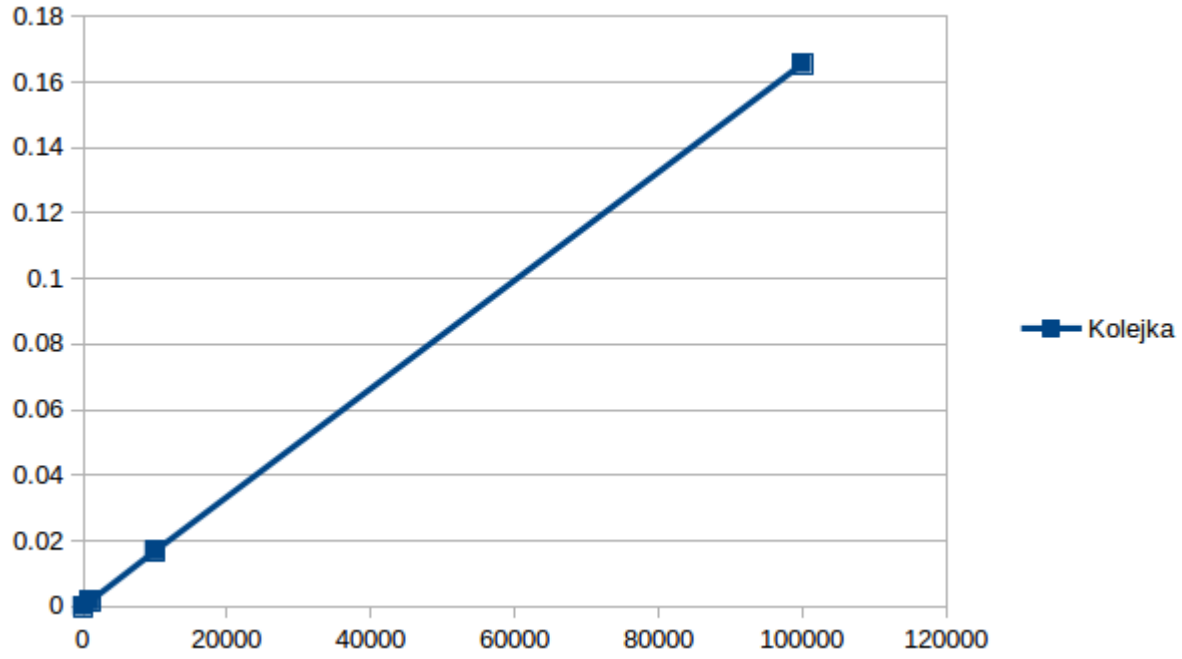
-Kolejki

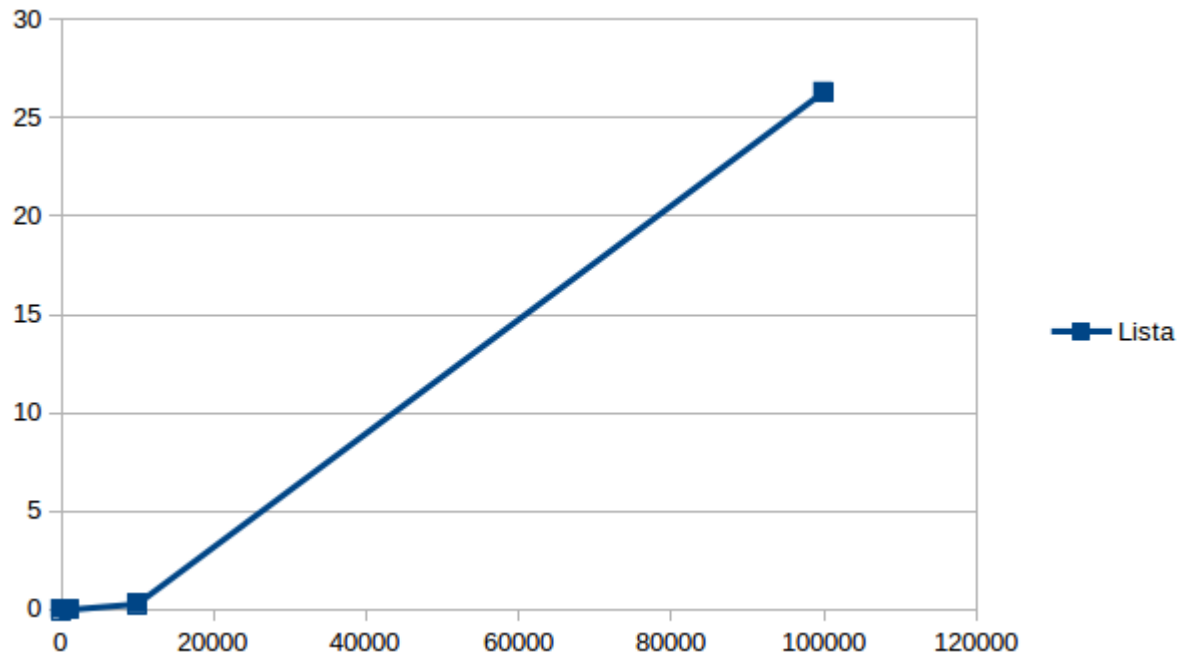
Oraz mierzenie czasu jaki potrzeba aby zapełnić każdy z nich określoną ilością danych

1.2 Wyniki

Dla stu tysięcy operacji komputer jest w stanie przeprowadzić badane operacje w rozsądnym czasie. Na podstawie otrzymanych danych mamy :


0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
20000
40000
60000
80000
100000
120000
Stos


0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
20000
40000
60000
80000
100000
120000
Kolejka



1.3 Podsumowanie

Uzyskane charakterystyki wskazują na liniową złożoność oblizeniową co jest zgodne z oczekiwaniami.

Stos i kolejka zostały zrealizowane za pomocą listy , natomiast sama lista dodawała następny element zawsze na swój koniec (musiała za każdym razem "przechodzić" przez wszystkie swoje elementy) nie dziwi więc zatem że czas wykonywania algorytmu w tym przypadku był bardzo długi.

Uważam iż powinno się dodatkowo opróżniać pamięć po każdorazowym wykonaniu się algorytmu dla każdej ilości danych, co należy zmienić jednak nie jest to łatwe zadanie.

Rozdział 2

Indeks hierarchiczny

2.1 Hierarchia klas

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

Rozdział 3

Indeks klas

3.1 Lista klas

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

Rozdział 4

Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

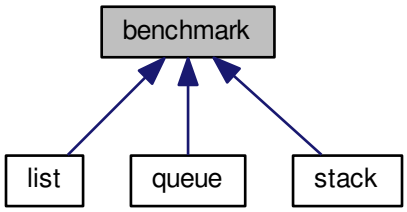
Rozdział 5

Dokumentacja klas

5.1 Dokumentacja klasy benchmark

`#include <benchmark.hh>`

Diagram dziedziczenia dla benchmark



Metody publiczne

- void analyze (int repeat, int data_amount)

  Metoda analyze zlicza czas wykonywania funkcji test()
  Uwaga! do poprawnego działania wymagane jest posiadanie programu gnuplot.

Metody prywatne

- virtual void test (unsigned long int length)=0

  Metoda test funkcja wirtualna , której czas działania ma być aproksymowany przez metoda analyze()

5.1.1 Opis szczegółowy

Definicja w linii 11 pliku benchmark.hh.

5.1.2 Dokumentacja funkcji składowych

5.1.2.1 void benchmark::analyze (int *repeat*, int *data_amount*)

Metoda analyze zlicza czas fukcji test()

Przykład wywołania funkcji :

analyze(100,7) -> Przeprowadza analize czesu trwania funkcji test() dla 1 miliona danych , każdy czas trwania funkcji jest ustalany na podstawie średniej arytmetycznej ze 100 prób.

Parametry

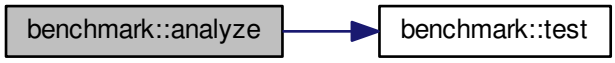
in	*repeat*	- ilość powtórzeń testu
in	*data_amount*	- ilosc wynikowych danych podawana jako potega liczby 10

Zwraca

plik.dat z czasami poszczegolnych pomiarow oraz ilosc testowanych danych oraz plik plot.png bedacy graficznym przedstawieniem danych na wykresie

Definicja w linii 15 pliku benchmark.cpp.

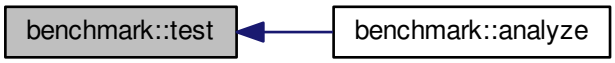
Oto graf wywołań dla tej funkcji:



5.1.2.2 virtual void benchmark::test (unsigned long int *length*) [private], [pure virtual]

Implementowany w list, queue i stack.

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z plików:

- benchmark.hh
- benchmark.cpp

5.2 Dokumentacja klasy list

`#include <list.hh>`

Diagram dziedziczenia dla list

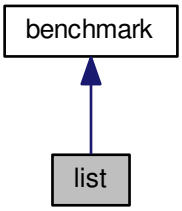
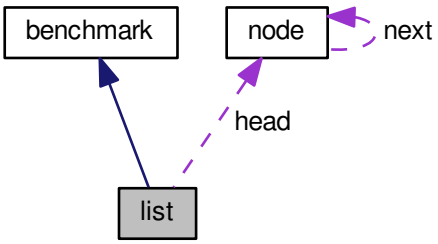


Diagram współpracy dla list:



Metody publiczne

- list ()

  Konstruktor inicjalizujący zmienną wskaźnikową , która domyślnie ma pokazywać na NULL.
- ~list ()

  Destruktor usuwa wszystkie elementy ze stosu za pomocą funkcji pop.
- void test (unsigned long int length)

  Metoda test() realizuje operacje zapelniania stosu ustalonymi danymi, czas będzie zliczany.

Metody prywatne

- void push (int insert)

  Metoda push() dodaje daną na stos.
- void pop ()

  Metoda pop() definiuje usuwanie elementu ze stosu.
- unsigned size ()

  Metoda size() zwraca ilość elementów stosu.

Atrybuty prywatne

- node * head

  Pole będące pierwszym wskaźnikiem na elementy stosu.

5.2.1 Opis szczegółowy

Definicja w linii 35 pliku list.hh.

5.2.2 Dokumentacja konstruktora i destruktora

5.2.2.1 list::list ( )

Definicja w linii 8 pliku list.cpp.

5.2.2.2 list::~list ( )

Definicja w linii 12 pliku list.cpp.

Oto graf wywołań dla tej funkcji:

5.2.3 Dokumentacja funkcji składowych

5.2.3.1 void list::pop ( ) [private]

Metoda pop() usuwa z listy ostatni element lub zwraca komunikat o błędzie w przypadku próby usnięcia elementu z pustego stosu.

Definicja w linii 44 pliku list.cpp.

Oto graf wywoływań tej funkcji:

5.2.3.2 void list::push (int *insert*) [private]

Metoda push() wczytuje liczbę naturalną na liste

Przykład wywołania funkcji :

push(10) - Na początek listy zostanie wprowadzona liczba 10.

Parametry

in	*insert*	- dodawany element

Definicja w linii 24 pliku list.cpp.

Oto graf wywoływań tej funkcji:

5.2.3.3 unsigned list::size ( ) [private]

Metoda size() zwraca ilość elementów znajdujących się na liście.

Definicja w linii 59 pliku list.cpp.

5.2.3.4 void list::test (unsigned long int *length*) [virtual]

Metoda test() realizuje wczytywanie zadanej ilości danych do listy.

Parametry

in	*length*	- ilość dodawanych lementów

Implementuje benchmark.

Definicja w linii 74 pliku list.cpp.

Oto graf wywołań dla tej funkcji:

5.2.4 Dokumentacja atrybutów składowych

5.2.4.1 node* list::head [private]

Definicja w linii 42 pliku list.hh.

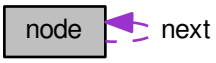
Dokumentacja dla tej klasy została wygenerowana z plików:

- list.hh
- list.cpp

5.3 Dokumentacja struktury node

`#include <list.hh>`

Diagram współpracy dla node:



Metody publiczne

- node ()

  Konstruktor węzła.

Atrybuty publiczne

- int data

  Pole do którego dopisywane są dane.
- node * next

  Pole będące wskaźnikiem na następny element.

5.3.1 Opis szczegółowy

Definicja w linii 15 pliku list.hh.

5.3.2 Dokumentacja konstruktora i destruktora

5.3.2.1 node::node ( ) [inline]

Definicja w linii 28 pliku list.hh.

5.3.3 Dokumentacja atrybutów składowych

5.3.3.1 int node::data

Definicja w linii 20 pliku list.hh.

5.3.3.2 node* node::next

Definicja w linii 24 pliku list.hh.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- list.hh

5.4 Dokumentacja klasy queue

`#include <queue.hh>`

Diagram dziedziczenia dla queue

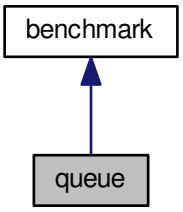
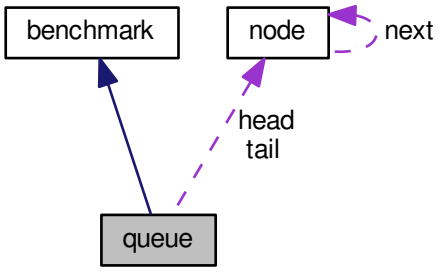


Diagram współpracy dla queue:



Metody publiczne

- queue ()

  Konstruktor inicjalizujący zmienną wskaźnikową , która domyślnie ma pokazywać na NULL.
- ~queue ()

  Destruktor usuwa wszystkie elementy z kolejki za pomocą funkcji pop.
- void test (unsigned long int length)

  Metoda test() realizuje operacje zapelniania kolejki ustalonymi danymi, czas będzie zliczany.

Metody prywatne

- void push (int insert)

  Metoda push() dodaje daną do kolejki.
- void pop ()

  Metoda pop() definiuje usuwanie elementu z kolejki.
- unsigned size ()

  Metoda size() zwraca ilość elementów kolejki.

Atrybuty prywatne

- node ∗ head

  Pole będące pierwszym wskaźnikiem na elementy kolejki.
- node ∗ tail

  Pole będące wskaźnikiem na ostatni element kolejki.

5.4.1 Opis szczegółowy

Definicja w linii 18 pliku queue.hh.

5.4.2 Dokumentacja konstruktora i destruktora

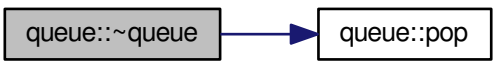
5.4.2.1 queue::queue ( )

Definicja w linii 9 pliku queue.cpp.

5.4.2.2 queue::∼queue ( )

Definicja w linii 14 pliku queue.cpp.

Oto graf wywołań dla tej funkcji:



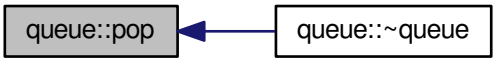
5.4.3 Dokumentacja funkcji składowych

5.4.3.1 void queue::pop ( ) [private]

Metoda pop() usuwa z kolejki pierwszy element lub zwraca komunikat o błędzie w przypadku próby usnięcia elementu z pustej kolejki.

Definicja w linii 39 pliku queue.cpp.

Oto graf wywoływań tej funkcji:



5.4.3.2 void queue::push (int *insert*) [private]

Metoda push() wczytuje liczbę naturalną do kolejki

Przykład wywołania funkcji :

push(10) - Na koniec kolejki zostanie wprowadzona liczba 10.

Parametry

in	*insert*	- dodawany element

Definicja w linii 26 pliku queue.cpp.

Oto graf wywoływań tej funkcji:

5.4.3.3 unsigned queue::size ( ) [private]

Metoda size() zwraca ilość elementów znajdujących się w kolejce.

Definicja w linii 56 pliku queue.cpp.

5.4.3.4 void queue::test (unsigned long int *length*) [virtual]

Metoda test() realizuje wczytywanie zadanej ilości danych do kolejki.

Parametry

in	*length*	- ilosc danych do wstawienia

Implementuje benchmark.

Definicja w linii 70 pliku queue.cpp.

Oto graf wywołań dla tej funkcji:

5.4.4 Dokumentacja atrybutów składowych

5.4.4.1 node* queue::head [private]

Definicja w linii 27 pliku queue.hh.

5.4.4.2 node* queue::tail [private]

Definicja w linii 31 pliku queue.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

- queue.hh
- queue.cpp

5.5 Dokumentacja klasy stack

`#include <stack.hh>`

Diagram dziedziczenia dla stack

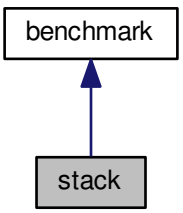
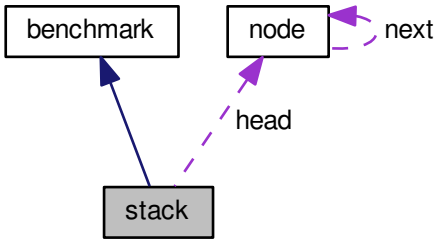


Diagram współpracy dla stack:



Metody publiczne

- stack ()

  Konstruktor inicjalizujący zmienną wskaźnikową , która domyślnie ma pokazywać na NULL.
- ~stack ()

  Destruktor usuwa wszystkie elementy ze stosu za pomocą funkcji pop.
- void test (unsigned long int length)

  Metoda test() realizuje operacje zapelniania stosu ustalonymi danymi, czas będzie zliczany.

Metody prywatne

- void push (int insert)

  Metoda push() dodaje daną na stos.
- void pop ()

  Metoda pop() definiuje usuwanie elementu ze stosu.
- unsigned size ()

  Metoda size() zwraca ilość elementów stosu.

Atrybuty prywatne

- node * head

  Pole będące pierwszym wskaźnikiem na elementy stosu.

5.5.1 Opis szczegółowy

Definicja w linii 18 pliku stack.hh.

5.5.2 Dokumentacja konstruktora i destruktora

5.5.2.1 stack::stack ( )

Definicja w linii 8 pliku stack.cpp.

5.5.2.2 stack::~stack ( )

Definicja w linii 12 pliku stack.cpp.

Oto graf wywołań dla tej funkcji:

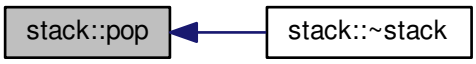
5.5.3 Dokumentacja funkcji składowych

5.5.3.1 void stack::pop ( ) [private]

Metoda pop() usuwa ze stosu ostatni element lub zwraca komunikat o błędzie w przypadku próby usnięcia elementu z pustego stosu.

Definicja w linii 40 pliku stack.cpp.

Oto graf wywoływań tej funkcji:



5.5.3.2 void stack::push (int *insert*) [private]

Metoda push() wczytuje liczbę naturalną na stos

Przykład wywołania funkcji :

push(10) - Na początek stosu zostanie wprowadzona liczba 10.

Parametry

in	*insert*	- dodawany element

Definicja w linii 22 pliku stack.cpp.

Oto graf wywoływań tej funkcji:

5.5.3.3 unsigned stack::size ( ) [private]

Metoda size() zwraca ilość elementów znajdujących się na stosie.

Definicja w linii 53 pliku stack.cpp.

5.5.3.4 void stack::test (unsigned long int *length*) [virtual]

Metoda test() realizuje wczytywanie zadanej ilości danych na stos.

Parametry

in	*length*	- ilosc danych do wstawienie

Implementuje benchmark.

Definicja w linii 68 pliku stack.cpp.

Oto graf wywołań dla tej funkcji:

5.5.4 Dokumentacja atrybutów składowych

5.5.4.1 node* stack::head [private]

Definicja w linii 26 pliku stack.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

- stack.hh
- stack.cpp

Rozdział 6

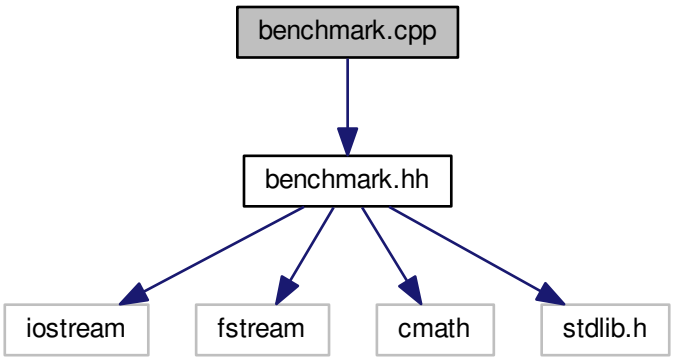
Dokumentacja plików

6.1 Dokumentacja pliku benchmark.cpp

Deklaracja funkzji z klasy Benchmark.

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

Wykres zależności załączania dla benchmark.cpp:

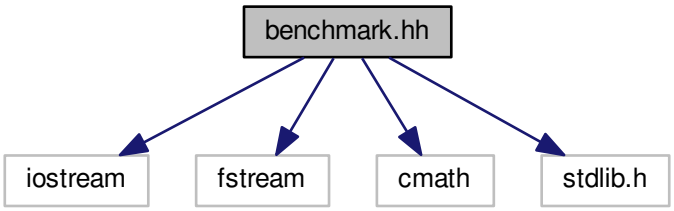


6.2 Dokumentacja pliku benchmark.hh

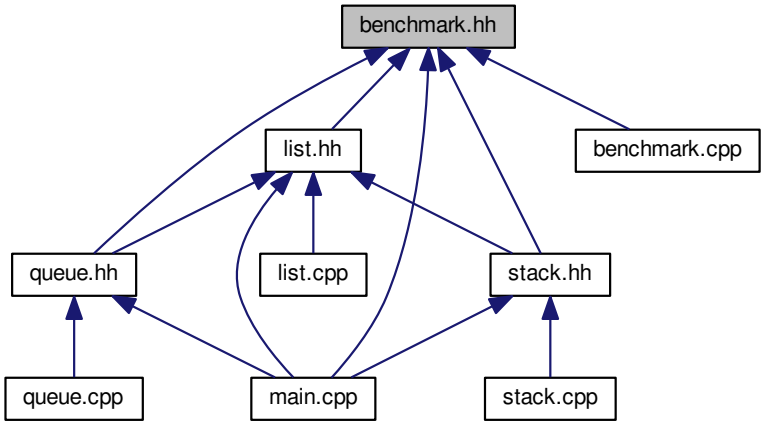
Definicja klasy Benchmark.

```
#include <iostream>
#include <fstream>
#include <cmath>
#include "stdlib.h"
```

Wykres zależności załączania dla benchmark.hh:



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



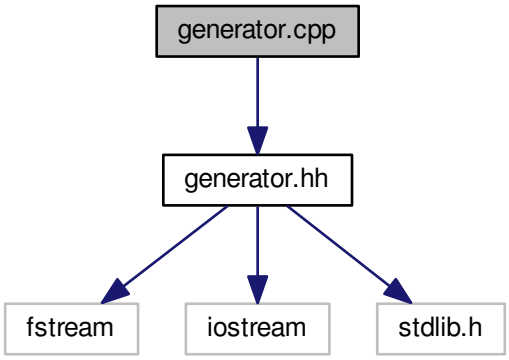
Komponenty

- class benchmark

6.3 Dokumentacja pliku generator.cpp

Deklaracja funckji generujacej liczby losowe.

```
#include "generator.hh"
```
Wykres zależności załączania dla generator.cpp:



Funkcje

- bool data_generator (unsigned long int data_amount)

  Generuje liczby losowe.

6.3.1 Dokumentacja funkcji

6.3.1.1 bool data_generator (unsigned long int *data_amount*)

Funkcja generuje naturalne liczby losowe z przedziału 0-100, ktore nastepnie sa zapisywane do pliku random_-data.dat

Parametry

in	*data_amount*	- ilosc liczb wynikowych ktore chcemy uzyskac

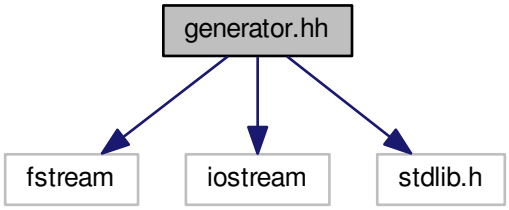
Definicja w linii 9 pliku generator.cpp.

6.4 Dokumentacja pliku generator.hh

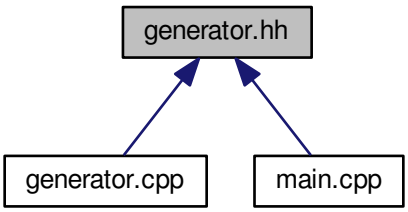
Definicja generatora liczb losowych.

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
```

Wykres zależności załączania dla generator.hh:



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



Funkcje

- bool data_generator (unsigned long int data_amount)

  Generuje liczby losowe.

6.4.1 Dokumentacja funkcji

6.4.1.1 bool data_generator (unsigned long int *data_amount*)

Funkcja generuje naturalne liczby losowe z przedziału 0-100, ktore nastepnie sa zapisywane do pliku random_-data.dat

Parametry

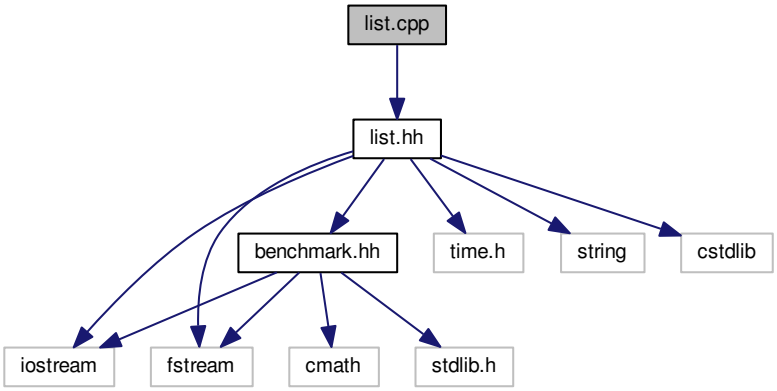
in	*data_amount*	- ilosc liczb wynikowych ktore chcemy uzyskac

Definicja w linii 9 pliku generator.cpp.

6.5 Dokumentacja pliku list.cpp

Deklaracja klasy list.

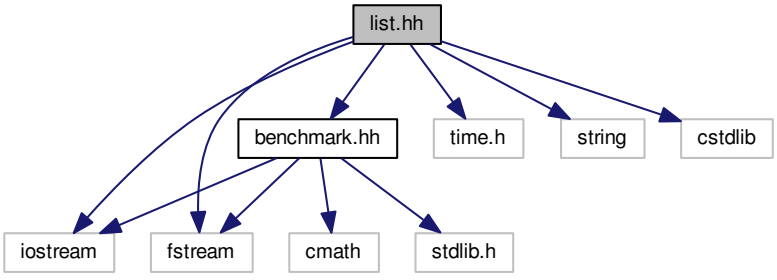
```
#include "list.hh"
```
Wykres zależności załączania dla list.cpp:



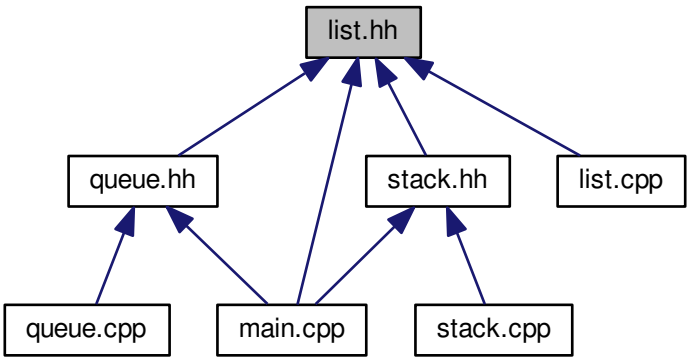
6.6 Dokumentacja pliku list.hh

Definicja klasy stack.

```
#include <iostream>
#include <time.h>
#include <string>
#include <fstream>
#include <cstdlib>
#include "benchmark.hh"
```
Wykres zależności załączania dla list.hh:



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



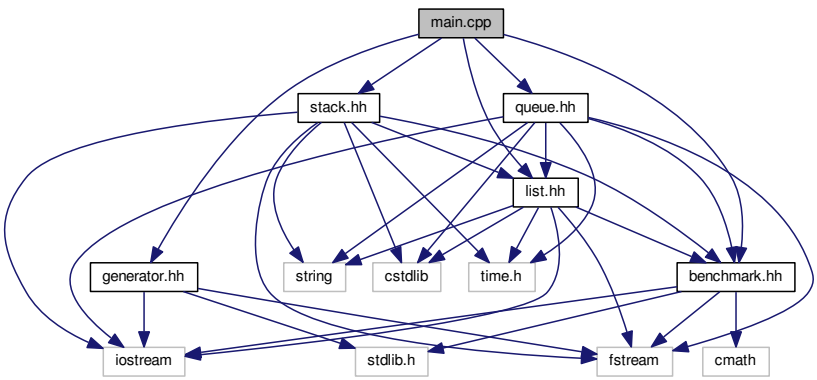
Komponenty

- struct node
- class list

6.7 Dokumentacja pliku main.cpp

```
#include "stack.hh"
#include "benchmark.hh"
#include "generator.hh"
#include "list.hh"
#include "queue.hh"
```

Wykres zależności załączania dla main.cpp:



Funkcje

- int main ()

  Aby wygenerować liczby losowe należy odkomentować linie zawierającą funkcje data_generator()
  Aby Przeprowadzić analizę złożoności obliczeniowej dla stosu należy odkomentować 1 blok (dotyczący klasy stack)
  Aby Przeprowadzić analizę złożoności obliczeniowej dla listy należy odkomentować 2 blok (dotyczący klasy list)
  Aby Przeprowadzić analizę złożoności obliczeniowej dla kolejki należy odkomentować 3 blok (dotyczący klasy queue)
  .

6.7.1 Dokumentacja funkcji

6.7.1.1 int main ( )

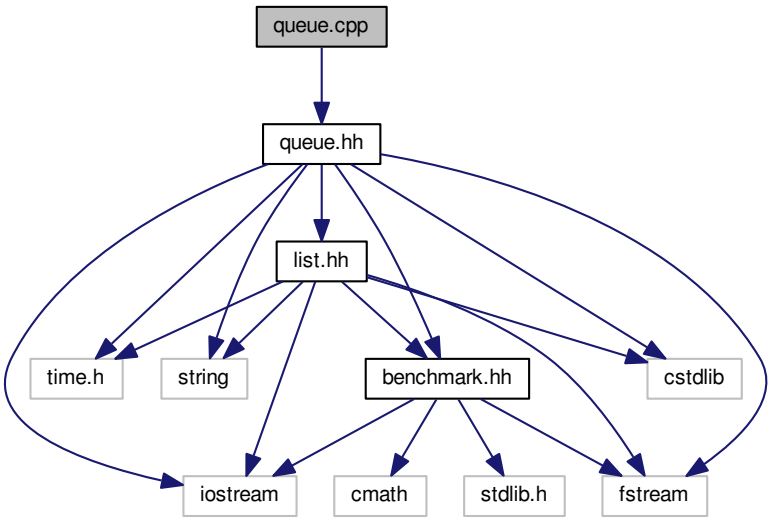
Definicja w linii 13 pliku main.cpp.

6.8 Dokumentacja pliku queue.cpp

Deklaracja klasy queue.

```
#include "queue.hh"
```

Wykres zależności załączania dla queue.cpp:

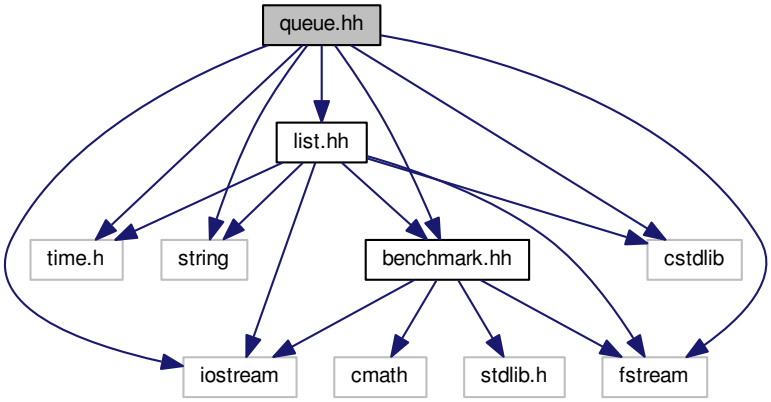


6.9 Dokumentacja pliku queue.hh

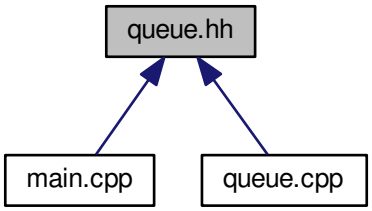
Definicja klasy stack.

```
#include <iostream>
#include <time.h>
#include <string>
#include <fstream>
#include <cstdlib>
#include "benchmark.hh"
#include "list.hh"
```

Wykres zależności załączania dla queue.hh:



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



Komponenty

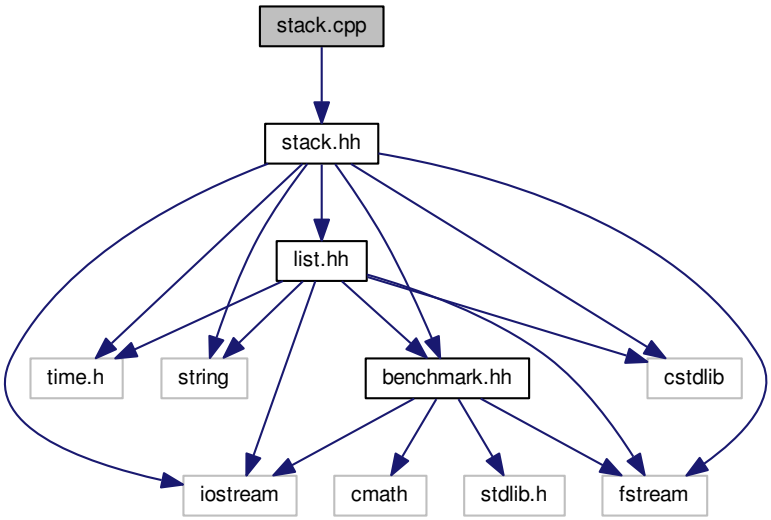
- class queue

6.10 Dokumentacja pliku stack.cpp

Deklaracja klasy stack.

```
#include "stack.hh"
```

Wykres zależności załączania dla stack.cpp:

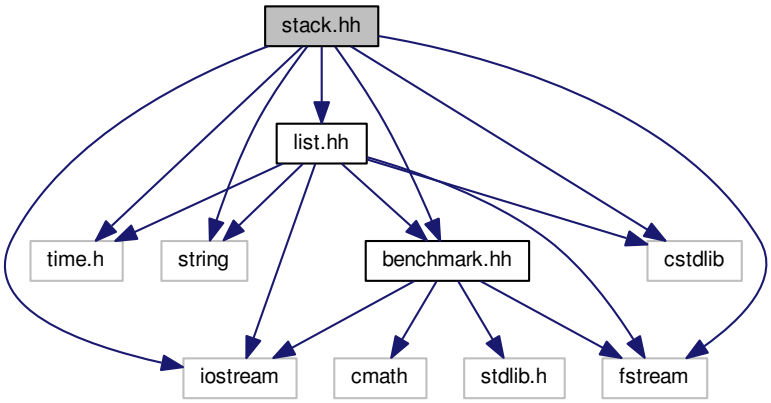


6.11 Dokumentacja pliku stack.hh

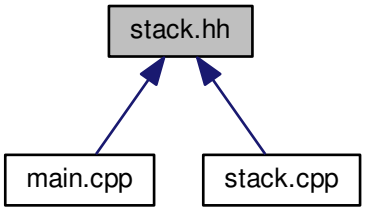
Definicja klasy stack.

```
#include <iostream>
#include <time.h>
#include <string>
#include <fstream>
#include <cstdlib>
#include "benchmark.hh"
#include "list.hh"
```

Wykres zależności załączania dla stack.hh:



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



Komponenty

- class stack

6.12 Dokumentacja pliku strona.dox

Skorowidz