

## Podstawowe struktury danych

1

Wygenerowano przez Doxygen 1.8.6

Cz, 19 mar 2015 07:40:06



# Spis treści

<b>1</b>	<b>Sprawozdanie</b>	<b>1</b>
1.1	Zadanie . . . . .	1
1.2	Wyniki . . . . .	1
1.3	Podsumowanie . . . . .	3
<b>2</b>	<b>Indeks hierarchiczny</b>	<b>5</b>
2.1	Hierarchia klas . . . . .	5
<b>3</b>	<b>Indeks klas</b>	<b>7</b>
3.1	Lista klas . . . . .	7
<b>4</b>	<b>Indeks plików</b>	<b>9</b>
4.1	Lista plików . . . . .	9
<b>5</b>	<b>Dokumentacja klas</b>	<b>11</b>
5.1	Dokumentacja klasy benchmark . . . . .	11
5.1.1	Opis szczegółowy . . . . .	11
5.1.2	Dokumentacja funkcji składowych . . . . .	11
5.1.2.1	analyzer . . . . .	12
5.1.2.2	test . . . . .	12
5.2	Dokumentacja klasy list . . . . .	12
5.2.1	Opis szczegółowy . . . . .	14
5.2.2	Dokumentacja konstruktora i destruktor . . . . .	14
5.2.2.1	list . . . . .	14
5.2.2.2	~list . . . . .	14
5.2.3	Dokumentacja funkcji składowych . . . . .	14
5.2.3.1	pop . . . . .	14
5.2.3.2	push . . . . .	15
5.2.3.3	size . . . . .	15
5.2.3.4	test . . . . .	15
5.2.4	Dokumentacja atrybutów składowych . . . . .	15
5.2.4.1	head . . . . .	16
5.3	Dokumentacja struktury node . . . . .	16

5.3.1	Opis szczegółowy . . . . .	16
5.3.2	Dokumentacja konstruktora i destruktora . . . . .	16
5.3.2.1	node . . . . .	16
5.3.3	Dokumentacja atrybutów składowych . . . . .	16
5.3.3.1	data . . . . .	16
5.3.3.2	next . . . . .	17
5.4	Dokumentacja klasy queue . . . . .	17
5.4.1	Opis szczegółowy . . . . .	18
5.4.2	Dokumentacja konstruktora i destruktora . . . . .	18
5.4.2.1	queue . . . . .	18
5.4.2.2	~queue . . . . .	18
5.4.3	Dokumentacja funkcji składowych . . . . .	18
5.4.3.1	pop . . . . .	18
5.4.3.2	push . . . . .	19
5.4.3.3	size . . . . .	19
5.4.3.4	test . . . . .	19
5.4.4	Dokumentacja atrybutów składowych . . . . .	20
5.4.4.1	head . . . . .	20
5.4.4.2	tail . . . . .	20
5.5	Dokumentacja klasy stack . . . . .	20
5.5.1	Opis szczegółowy . . . . .	21
5.5.2	Dokumentacja konstruktora i destruktora . . . . .	21
5.5.2.1	stack . . . . .	21
5.5.2.2	~stack . . . . .	22
5.5.3	Dokumentacja funkcji składowych . . . . .	22
5.5.3.1	pop . . . . .	22
5.5.3.2	push . . . . .	22
5.5.3.3	size . . . . .	23
5.5.3.4	test . . . . .	23
5.5.4	Dokumentacja atrybutów składowych . . . . .	23
5.5.4.1	head . . . . .	23
<b>6</b>	<b>Dokumentacja plików</b>	<b>25</b>
6.1	Dokumentacja pliku benchmark.cpp . . . . .	25
6.2	Dokumentacja pliku benchmark.hh . . . . .	25
6.3	Dokumentacja pliku generator.cpp . . . . .	26
6.3.1	Dokumentacja funkcji . . . . .	27
6.3.1.1	data_generator . . . . .	27
6.4	Dokumentacja pliku generator.hh . . . . .	27
6.4.1	Dokumentacja funkcji . . . . .	28

6.4.1.1	data_generator . . . . .	28
6.5	Dokumentacja pliku list.cpp . . . . .	28
6.6	Dokumentacja pliku list.hh . . . . .	29
6.7	Dokumentacja pliku main.cpp . . . . .	30
6.7.1	Dokumentacja funkcji . . . . .	31
6.7.1.1	main . . . . .	31
6.8	Dokumentacja pliku queue.cpp . . . . .	31
6.9	Dokumentacja pliku queue.hh . . . . .	31
6.10	Dokumentacja pliku stack.cpp . . . . .	32
6.11	Dokumentacja pliku stack.hh . . . . .	33
6.12	Dokumentacja pliku strona.dox . . . . .	34
<b>Indeks</b>		<b>35</b>



# Rozdział 1

## Sprawozdanie

Data

19.03.2015r.

Wersja

0.1

### 1.1 Zadanie

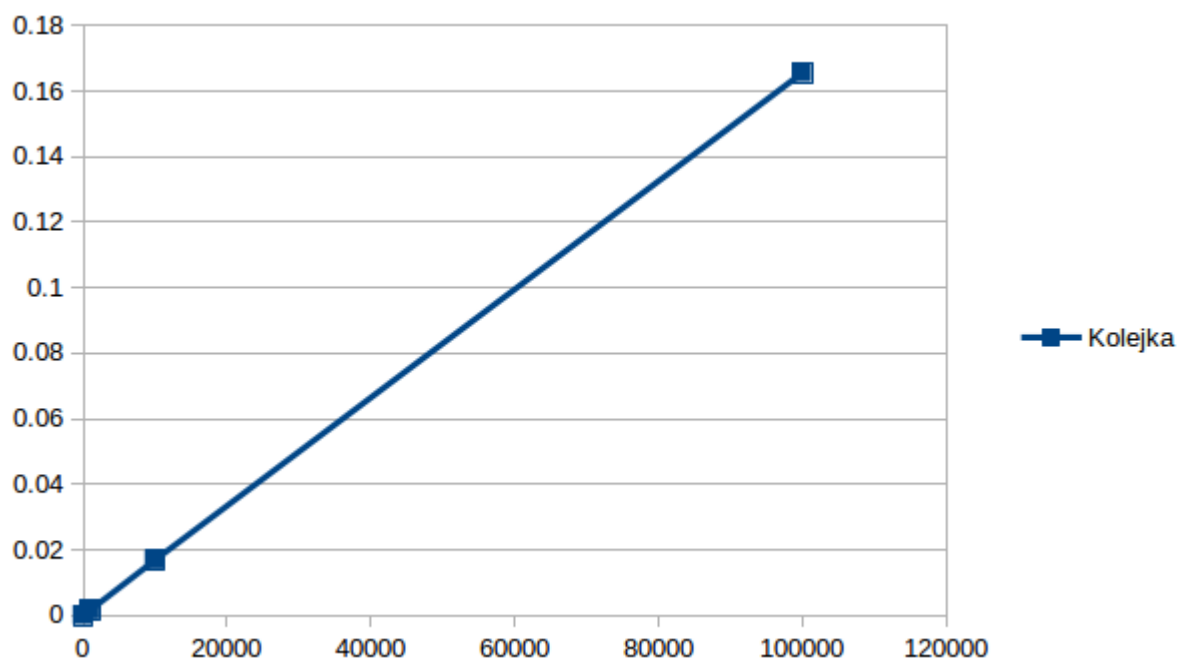
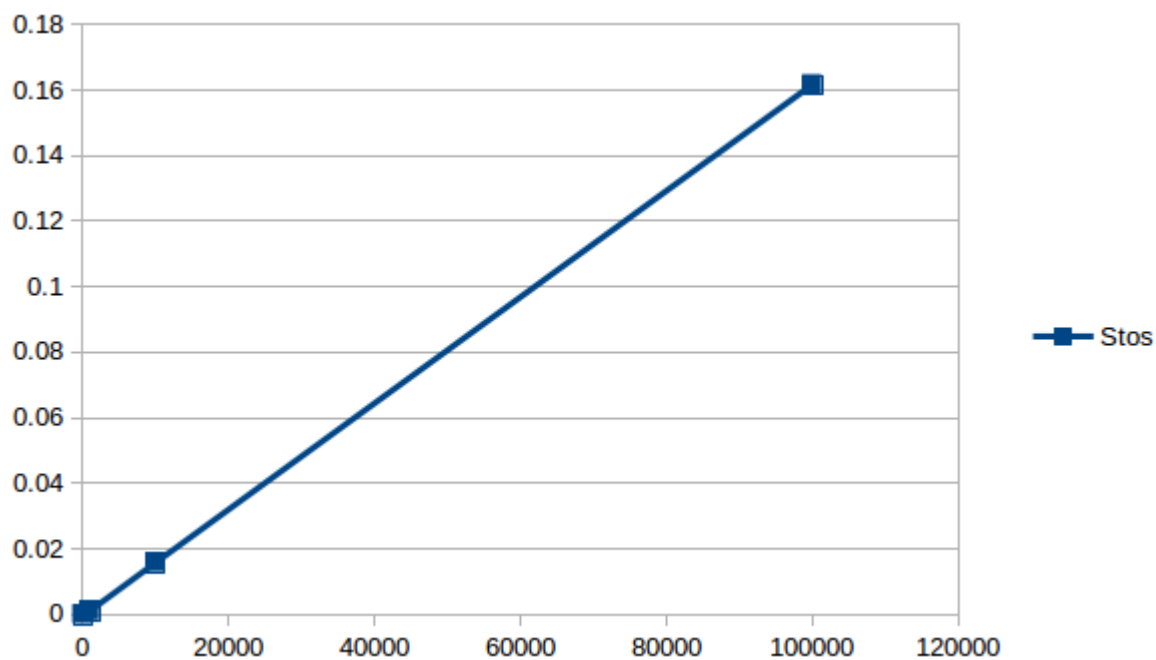
Celem ćwiczenia było stworzenie trzech abstrakcyjnych typów danych :

- Kolejki (LIFO)
- Stosu (FIFO)
- Kolejki

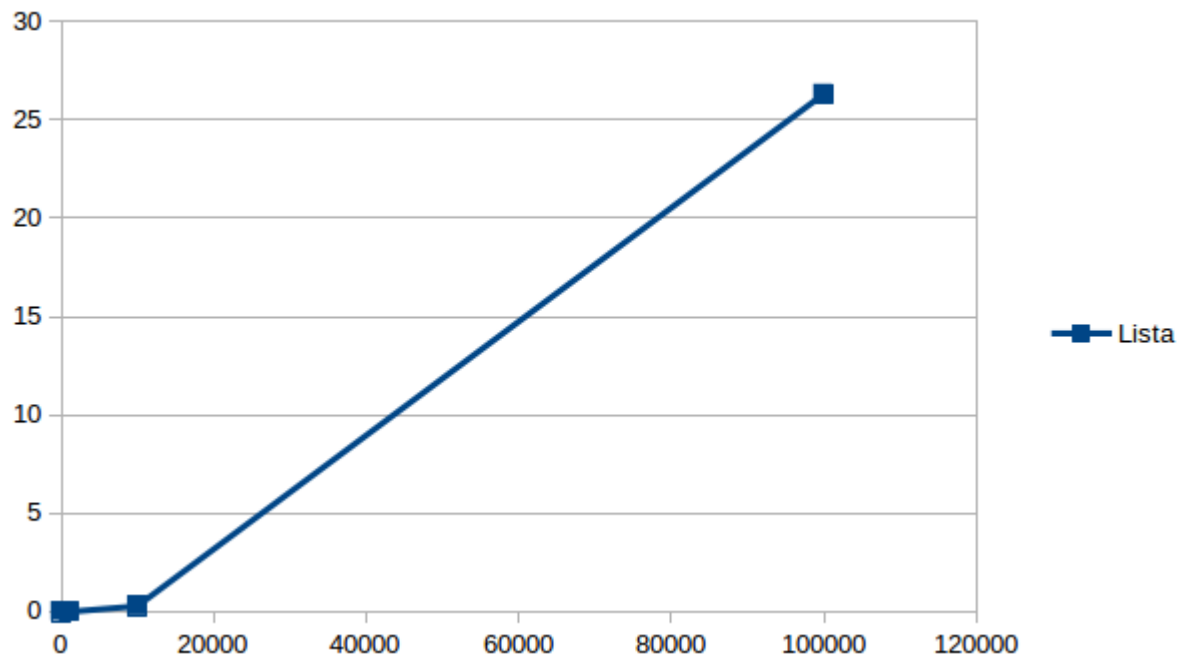
Oraz mierzenie czasu jaki potrzeba aby zappełnić każdy z nich określoną ilością danych

### 1.2 Wyniki

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







### 1.3 Podsumowanie

Uzyskane charakterystyki wskazują na liniową złożoność obliczeniową 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.

Mimo wszystko uważam że algorytm dodający elementy do struktury lista jest bardzo wysoce nieefektywny, i w zasadzie stos czy kolejka są zrealizowane za pomocą listy czyli de facto implementacja listy jako oddzielnej klasy (dodatkowo takiej w której funkcja push() przechodzi przez wszystkie elementy) jest bezsensowna i o ile dobrze rozumiem postawione zadanie zbędna.

Uważam również iż powinno się dodatkowo opróżniać pamięć po każdorazowym wykonaniu się algorytmu dla każdej ilości danych, co należałoby zmienić jednak wymagało by to przerywania zliczania czasu.



## Rozdział 2

# Indeks hierarchiczny

### 2.1 Hierarchia klas

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

benchmark . . . . .	11
list . . . . .	12
queue . . . . .	17
stack . . . . .	20
node . . . . .	16



## Rozdział 3

# Indeks klas

### 3.1 Lista klas

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

<a href="#">benchmark</a>	11
<a href="#">list</a>	12
<a href="#">node</a>	16
<a href="#">queue</a>	17
<a href="#">stack</a>	20



## Rozdział 4

# Indeks plików

### 4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

<a href="#">benchmark.cpp</a>		
	Deklaracja funkcji z klasy Benchmark . . . . .	25
<a href="#">benchmark.hh</a>		
	Definicja klasy Benchmark . . . . .	25
<a href="#">generator.cpp</a>		
	Deklaracja funkcji generującej liczby losowe . . . . .	26
<a href="#">generator.hh</a>		
	Definicja generatora liczb losowych . . . . .	27
<a href="#">list.cpp</a>		
	Deklaracja klasy list . . . . .	28
<a href="#">list.hh</a>		
	Definicja klasy stack . . . . .	29
<a href="#">main.cpp</a>		30
<a href="#">queue.cpp</a>		
	Deklaracja klasy queue . . . . .	31
<a href="#">queue.hh</a>		
	Definicja klasy stack . . . . .	31
<a href="#">stack.cpp</a>		
	Deklaracja klasy stack . . . . .	32
<a href="#">stack.hh</a>		
	Definicja klasy stack . . . . .	33





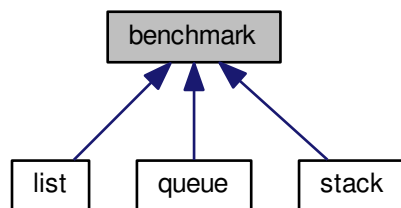
## Rozdział 5

# 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	<i>repeat</i>	- ilość powtórzeń testu
in	<i>data_amount</i>	- 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

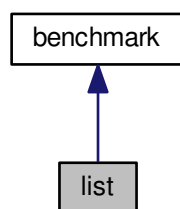
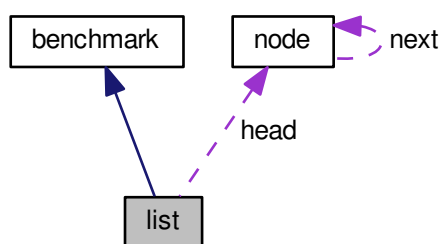


Diagram współpracy dla list:



## 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 destruktor

#### 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 usunięcia 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 listę

Przykład wywołania funkcji :

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

Parametry

in	<i>insert</i>	- dodawany element
----	---------------	--------------------

Definicja w linii 24 pliku list.cpp.

Oto graf wywołań 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	<i>length</i>	- ilość dodawanych elementów
----	---------------	------------------------------

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

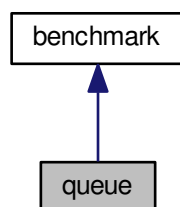
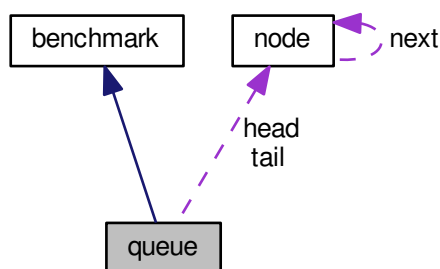


Diagram współpracy dla queue:



## 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 destruktor

#### 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 usunięcia 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	<i>insert</i>	- 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	<i>length</i>	- ilość 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

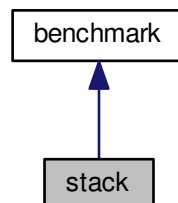
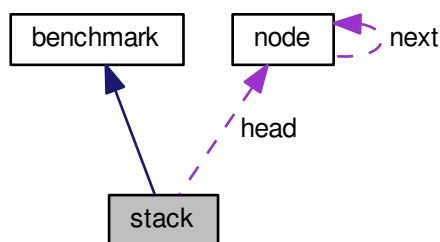


Diagram współpracy dla `stack`:



### 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 usunięcia 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

<code>in</code>	<code>insert</code>	- dodawany element
-----------------	---------------------	--------------------

Definicja w linii 22 pliku `stack.cpp`.

Oto graf wywołań 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

<code>in</code>	<code>length</code>	- ilość 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

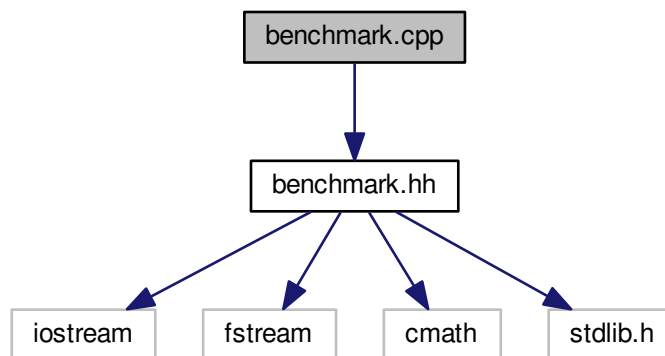
# Dokumentacja plików

### 6.1 Dokumentacja pliku benchmark.cpp

Deklaracja funkcji z klasy Benchmark.

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

Wykres zależności załączania dla benchmark.cpp:

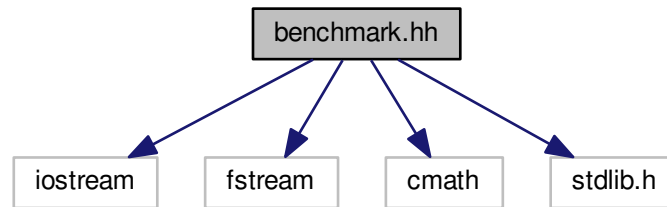


### 6.2 Dokumentacja pliku benchmark.hh

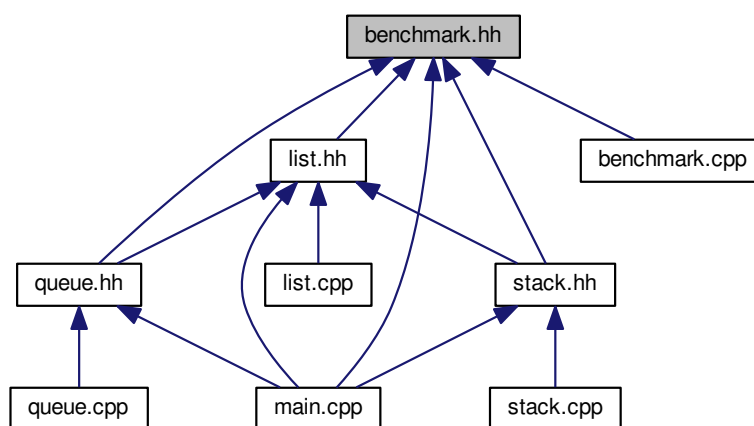
Definicja klasy Benchmark.

```
#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`

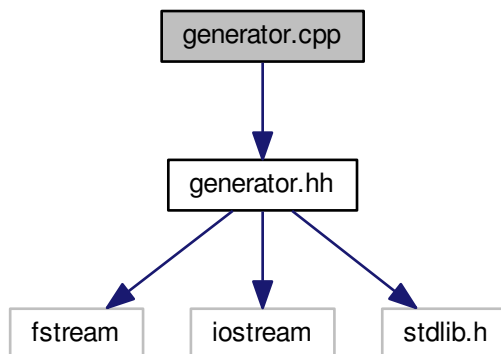
## 6.3 Dokumentacja pliku `generator.cpp`

Deklaracja funkcji generującej 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, które następnie są zapisywane do pliku random\_data.dat

**Parametry**

in	data_amount	- ilość liczb wynikowych które chcemy uzyskać
----	-------------	---

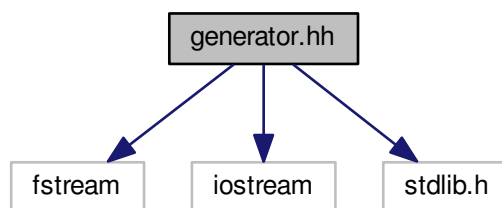
Definicja w linii 9 pliku generator.cpp.

## 6.4 Dokumentacja pliku generator.hh

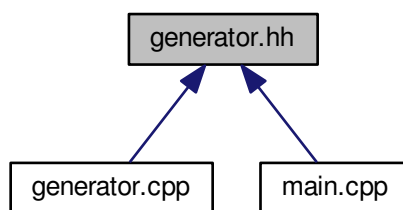
Definicja generatora liczb losowych.

```
#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, które następnie są zapisywane do pliku random\_data.dat

Parametry

in	data_amount	- ilość liczb wynikowych które chcemy uzyskać
----	-------------	---

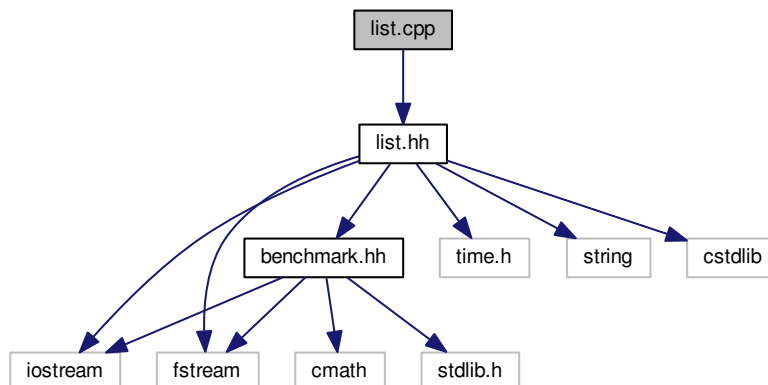
Definicja w linii 9 pliku generator.cpp.

## 6.5 Dokumentacja pliku list.cpp

Deklaracja klasy list.

```
#include "list.hh"
```

Wykres zależności załączania dla list.cpp:

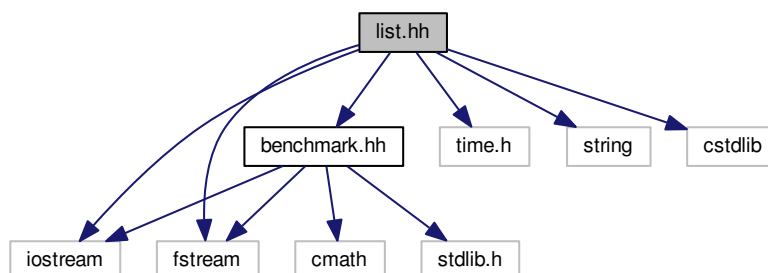


## 6.6 Dokumentacja pliku list.hh

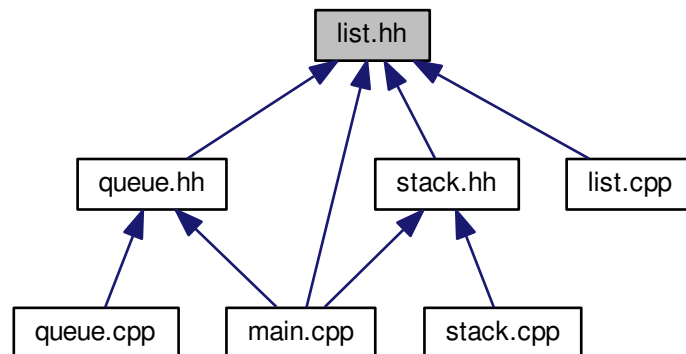
Definicja klasy stack.

```
#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:



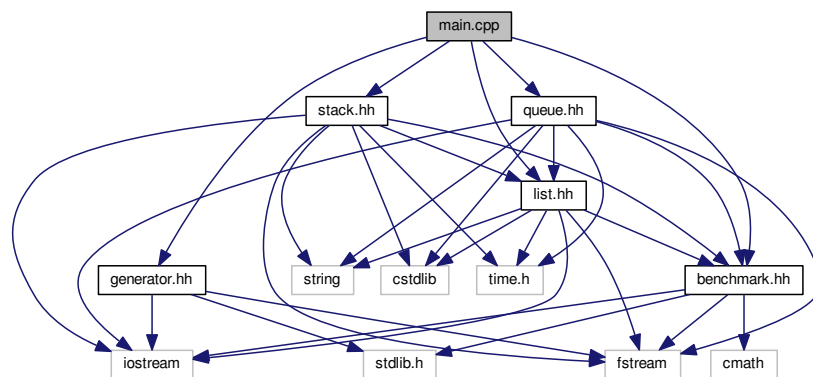
## Komponenty

- 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 funkcję `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 ( )

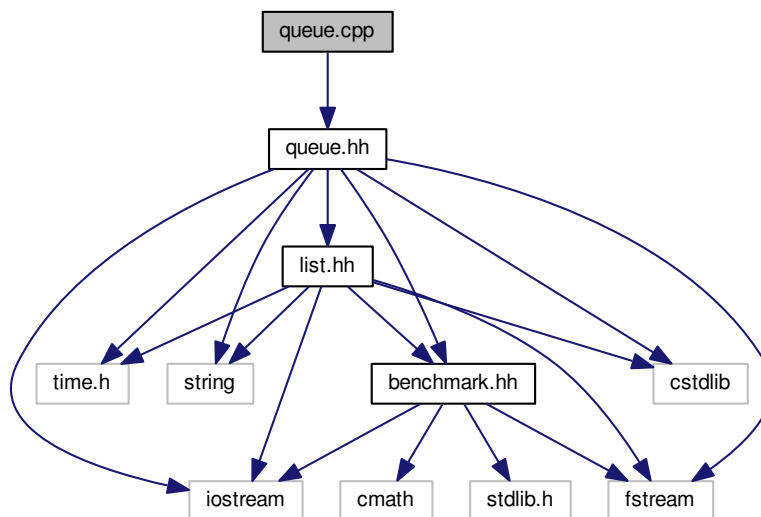
Definicja w linii 13 pliku main.cpp.

## 6.8 Dokumentacja pliku queue.cpp

Deklaracja klasy `queue`.

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

Wykres zależności załączania dla queue.cpp:

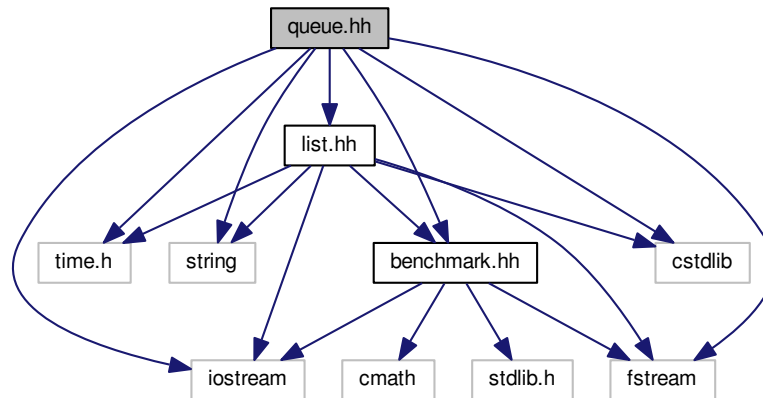


## 6.9 Dokumentacja pliku queue.hh

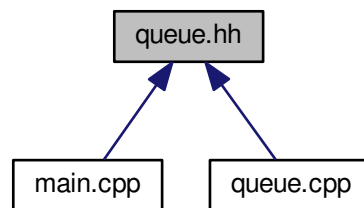
Definicja klasy `stack`.

```
#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

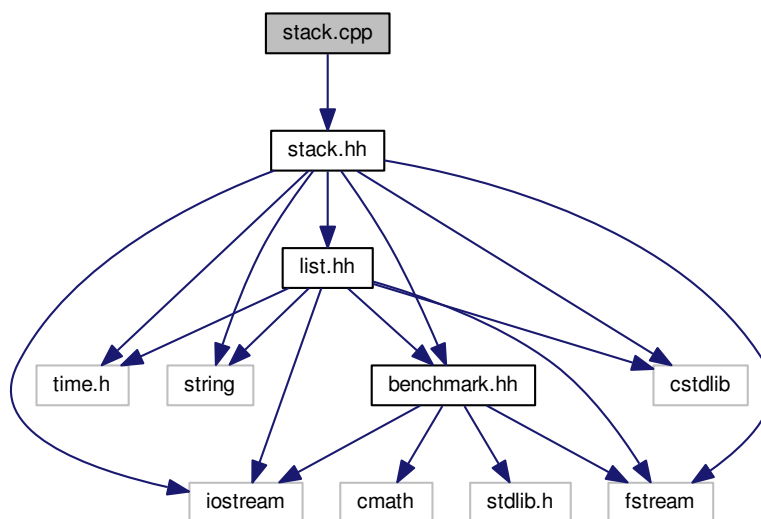
- class `queue`

## 6.10 Dokumentacja pliku stack.cpp

Deklaracja klasy stack.

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

Wykres zależności załączania dla stack.cpp:

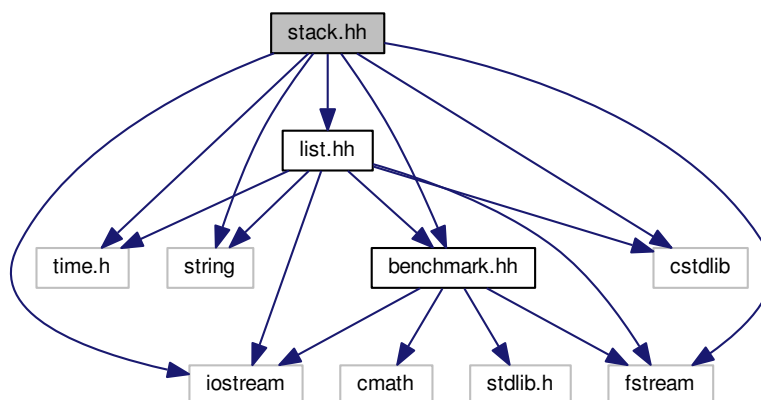


## 6.11 Dokumentacja pliku stack.hh

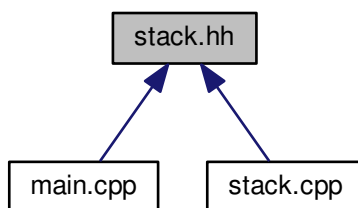
Definicja klasy stack.

```
#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

- ~list
  - list, [14](#)
- ~queue
  - queue, [18](#)
- ~stack
  - stack, [21](#)
- analyze
  - benchmark, [11](#)
- benchmark, [11](#)
  - analyze, [11](#)
  - test, [12](#)
- benchmark.cpp, [25](#)
- benchmark.hh, [25](#)
- data
  - node, [16](#)
- data\_generator
  - generator.cpp, [27](#)
  - generator.hh, [28](#)
- generator.cpp, [26](#)
  - data\_generator, [27](#)
- generator.hh, [27](#)
  - data\_generator, [28](#)
- head
  - list, [15](#)
  - queue, [20](#)
  - stack, [23](#)
- list, [12](#)
  - ~list, [14](#)
  - head, [15](#)
  - list, [14](#)
  - pop, [14](#)
  - push, [14](#)
  - size, [15](#)
  - test, [15](#)
- list.cpp, [28](#)
- list.hh, [29](#)
- main
  - main.cpp, [31](#)
- main.cpp, [30](#)
  - main, [31](#)
- next
  - node, [16](#)
- node, [16](#)
  - data, [16](#)
  - next, [16](#)
  - node, [16](#)
- pop
  - list, [14](#)
  - queue, [18](#)
  - stack, [22](#)
- push
  - list, [14](#)
  - queue, [19](#)
  - stack, [22](#)
- queue, [17](#)
  - ~queue, [18](#)
  - head, [20](#)
  - pop, [18](#)
  - push, [19](#)
  - queue, [18](#)
  - size, [19](#)
  - tail, [20](#)
  - test, [19](#)
- queue.cpp, [31](#)
- queue.hh, [31](#)
- size
  - list, [15](#)
  - queue, [19](#)
  - stack, [23](#)
- stack, [20](#)
  - ~stack, [21](#)
  - head, [23](#)
  - pop, [22](#)
  - push, [22](#)
  - size, [23](#)
  - stack, [21](#)
  - test, [23](#)
- stack.cpp, [32](#)
- stack.hh, [33](#)
- strona.dox, [34](#)
- tail
  - queue, [20](#)
- test
  - benchmark, [12](#)
  - list, [15](#)
  - queue, [19](#)
  - stack, [23](#)