

Podstawowe struktury danych

1

Wygenerowano przez Doxygen 1.8.6

Cz, 9 kwi 2015 01:19:58

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 klasy benchmark	7
4.1.1 Opis szczegółowy	7
4.1.2 Dokumentacja funkcji składowych	7
4.1.2.1 analityze	8
4.1.2.2 test	8
4.2 Dokumentacja klasy list	8
4.2.1 Opis szczegółowy	9
4.2.2 Dokumentacja konstruktora i destruktoru	9
4.2.2.1 list	9
4.2.2.2 ~list	10
4.2.3 Dokumentacja funkcji składowych	10
4.2.3.1 pop	10
4.2.3.2 push	10
4.2.3.3 size	11
4.2.3.4 test	12
4.2.4 Dokumentacja atrybutów składowych	13
4.2.4.1 head	13
4.2.4.2 tail	13
4.3 Dokumentacja klasy list_array	13
4.3.1 Opis szczegółowy	14
4.3.2 Dokumentacja konstruktora i destruktoru	15
4.3.2.1 list_array	15

4.3.2.2	<code>~list_array</code>	15
4.3.3	Dokumentacja funkcji składowych	15
4.3.3.1	<code>analyzer</code>	15
4.3.3.2	<code>heapsort</code>	16
4.3.3.3	<code>pop</code>	16
4.3.3.4	<code>push</code>	17
4.3.3.5	<code>quicksort</code>	17
4.3.3.6	<code>quicksort_left</code>	18
4.3.3.7	<code>size</code>	18
4.3.3.8	<code>test</code>	18
4.3.4	Dokumentacja atrybutów składowych	19
4.3.4.1	<code>n</code>	19
4.3.4.2	<code>temp</code>	19
4.3.4.3	<code>tmp</code>	19
4.4	Dokumentacja struktury <code>node</code>	19
4.4.1	Opis szczegółowy	20
4.4.2	Dokumentacja konstruktora i destruktor	20
4.4.2.1	<code>node</code>	20
4.4.3	Dokumentacja atrybutów składowych	20
4.4.3.1	<code>data</code>	20
4.4.3.2	<code>next</code>	20
4.5	Dokumentacja klasy <code>queue</code>	20
4.5.1	Opis szczegółowy	22
4.5.2	Dokumentacja konstruktora i destruktor	22
4.5.2.1	<code>queue</code>	22
4.5.2.2	<code>~queue</code>	22
4.5.3	Dokumentacja funkcji składowych	22
4.5.3.1	<code>pop</code>	22
4.5.3.2	<code>push</code>	23
4.5.3.3	<code>size</code>	23
4.5.3.4	<code>test</code>	24
4.5.4	Dokumentacja atrybutów składowych	25
4.5.4.1	<code>head</code>	25
4.5.4.2	<code>tail</code>	25
4.6	Dokumentacja klasy <code>stack</code>	25
4.6.1	Opis szczegółowy	26
4.6.2	Dokumentacja konstruktora i destruktor	26
4.6.2.1	<code>stack</code>	26
4.6.2.2	<code>~stack</code>	27
4.6.3	Dokumentacja funkcji składowych	27

4.6.3.1	pop	27
4.6.3.2	push	27
4.6.3.3	size	28
4.6.3.4	test	28
4.6.4	Dokumentacja atrybutów składowych	29
4.6.4.1	head	29
5	Dokumentacja plików	31
5.1	Dokumentacja pliku benchmark.cpp	31
5.2	Dokumentacja pliku benchmark.hh	31
5.3	Dokumentacja pliku generator.cpp	32
5.3.1	Dokumentacja funkcji	33
5.3.1.1	data_generator	33
5.4	Dokumentacja pliku generator.hh	33
5.4.1	Dokumentacja funkcji	34
5.4.1.1	data_generator	34
5.5	Dokumentacja pliku list.cpp	34
5.6	Dokumentacja pliku list.hh	35
5.7	Dokumentacja pliku list_array.cpp	36
5.8	Dokumentacja pliku list_array.hh	37
5.9	Dokumentacja pliku main.cpp	37
5.9.1	Dokumentacja funkcji	38
5.9.1.1	main	38
5.10	Dokumentacja pliku queue.cpp	38
5.11	Dokumentacja pliku queue.hh	39
5.12	Dokumentacja pliku stack.cpp	40
5.13	Dokumentacja pliku stack.hh	41
Indeks		43

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

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

benchmark	7
list	8
list_array	13
queue	20
stack	25
node	19

Rozdział 2

Indeks klas

2.1 Lista klas

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

benchmark	7
list	8
list_array	13
node	19
queue	20
stack	25

Rozdział 3

Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

benchmark.cpp	Deklaracja funkcji z klasy Benchmark	31
benchmark.hh	Definicja klasy Benchmark	31
generator.cpp	Deklaracja funkcji generującej liczby losowe	32
generator.hh	Definicja generatora liczb losowych	33
list.cpp	Deklaracja klasy list	34
list.hh	Definicja klasy lista	35
list_array.cpp	Deklaracja klasy list_array	36
list_array.hh	Definicja klasy list_array	37
main.cpp	37
queue.cpp	Deklaracja klasy queue	38
queue.hh	Definicja klasy stack	39
stack.cpp	Deklaracja klasy stack	40
stack.hh	Definicja klasy stack	41

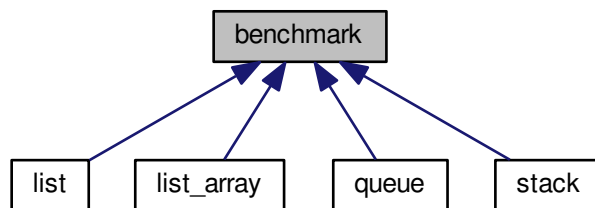
Rozdział 4

Dokumentacja klas

4.1 Dokumentacja klasy benchmark

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

Diagram dziedziczenia dla benchmark



Metody publiczne

- virtual void [analyze](#) (const char *name_output, int repeat, int data_amount)

Metoda analyze zlicza czas wykonywania funkcji [test\(\)](#)

Metody prywatne

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

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

4.1.1 Opis szczegółowy

Definicja w linii 11 pliku benchmark.hh.

4.1.2 Dokumentacja funkcji składowych

4.1.2.1 void benchmark::analyzer (const char * name_output, int repeat, int data_amount) [virtual]

Metoda analyzer zlicza czas funkcji [test\(\)](#)

Przykład wywołania funkcji :

analyzer("Plik_wynikowy",100,7) -> Przeprowadza analize czasu trwania funkcji [test\(\)](#) dla 1 miliona danych (ilość danych należy podać jako potęgę 10) , każdy czas trwania funkcji jest ustalany na podstawie średniej arytmetycznej ze 100 prób , wyniki zapisuje do pliku o nazwie Plik_wynikowy.

Uwaga! Aby zmienić tryb rozszerzania tablicy z dodawania 1 elementu na mnożenie rozmiaru przez 2 należy odcommentować odpowiednie pole w funkcji [list::array test\(\)](#)

Parametry

in	<i>name_output</i>	- nazwa pliku wynikowego
in	<i>repeat</i>	- ilość powtórzeń testu
in	<i>data_amount</i>	- ilość wynikowych danych podawana jako potęga liczby 10

Zwraca

plik.dat z czasami poszczególnych pomiarów oraz ilość testowanych danych.

Reimplementowana w [list_array](#).

Definicja w linii 15 pliku benchmark.cpp.

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

Implementowany w [list](#), [list_array](#), [stack](#) i [queue](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

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

4.2 Dokumentacja klasy list

```
#include <list.hh>
```

Diagram dziedziczenia dla list

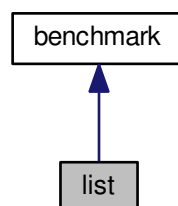
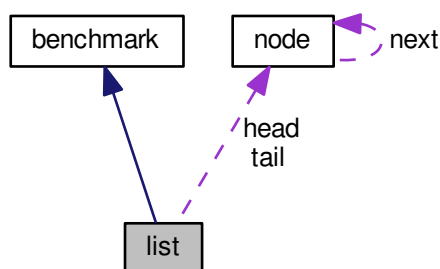


Diagram współpracy dla list:



Metody publiczne

- `list()`
Konstruktor inicjalizujący pustą listę, początek i koniec listy są domyślnie ustawione na NULL.
- `~list()`
Destruktor usuwa wszystkie elementy ze stosu za pomocą funkcji pop.
- `void test (unsigned long int length)`
Metoda `test()` realizuje operacje wypełniania listy danymi.

Metody prywatne

- `void push (int insert, unsigned int where)`
Metoda `push()` dodaje element na listę
- `void pop (unsigned int whence)`
Metoda `pop()` definiuje usuwanie elementu z listy.
- `unsigned size()`
Metoda `size()` zwraca ilość elementów znajdujących się na liście.

Atrybuty prywatne

- `node * head`
Pole będące wskaźnikiem na pierwszy element listy.
- `node * tail`
Pole będące wskaźnikiem na ostatni element listy.

4.2.1 Opis szczegółowy

Definicja w linii 46 pliku list.hh.

4.2.2 Dokumentacja konstruktora i destruktora

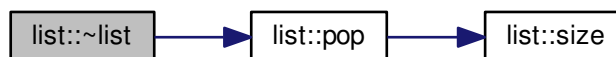
4.2.2.1 `list::list()`

Definicja w linii 9 pliku list.cpp.

4.2.2.2 list::~list ()

Definicja w linii 15 pliku list.cpp.

Oto graf wywołań dla tej funkcji:



4.2.3 Dokumentacja funkcji składowych

4.2.3.1 void list::pop (unsigned int whence) [private]

Metoda `pop()` usuwa z listy wybrany element , lub zwraca błąd jeżeli lista jest już pusta.

Parametry

<code>in</code>	<code>whence</code>	- numer usuwanego elementu
-----------------	---------------------	----------------------------

Definicja w linii 73 pliku list.cpp.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.2.3.2 void list::push (int insert, unsigned int where) [private]

Metoda `push()` wczytuje liczbę naturalną na liste.

Parametry

<code>in</code>	<code>insert</code>	- wartość dodawanego elementu
<code>in</code>	<code>where</code>	- na które miejsce ów element ma zostać dodany

Elementy listy są numerowane od 0!!

Przykład użycia funkcji:

`push(3,3)` -> wstawia element o wartości 3 na 3 miejsce listy , lub zwraca błąd 3 jeżeli lista jest zbyt krótka

Definicja w linii 29 pliku `list.cpp`.

Oto graf wywołań dla tej funkcji:

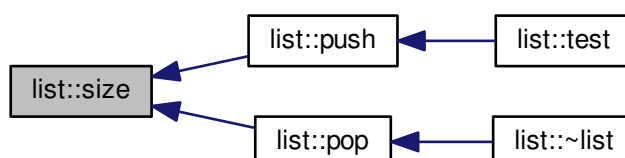


Oto graf wywoływań tej funkcji:

4.2.3.3 `unsigned list::size () [private]`

Definicja w linii 125 pliku `list.cpp`.

Oto graf wywoływań tej funkcji:



4.2.3.4 `void list::test (unsigned long int length)` `[virtual]`

Metoda `test()` realizuje wczytywanie zadanej ilości danych.

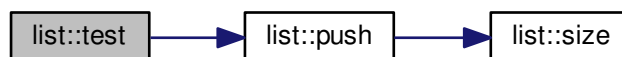
Parametry

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

Implementuje [benchmark](#).

Definicja w linii 143 pliku list.cpp.

Oto graf wywołań dla tej funkcji:



4.2.4 Dokumentacja atrybutów składowych

4.2.4.1 `node* list::head` [private]

Definicja w linii 55 pliku list.hh.

4.2.4.2 `node* list::tail` [private]

Definicja w linii 61 pliku list.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

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

4.3 Dokumentacja klasy list_array

```
#include <list_array.hh>
```

Diagram dziedziczenia dla list_array

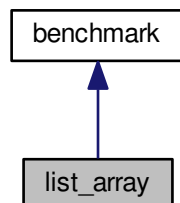
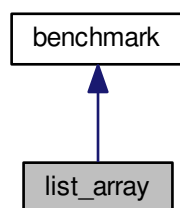


Diagram współpracy dla list_array:



Metody publiczne

- void **push** (int insert, unsigned int where, unsigned int extend, char x)
*Metoda **push()** dodaje element na listę*
- void **pop** (unsigned int whence)
*Metoda **pop()** definiuje usuwanie elementu z listy.*
- unsigned **size** ()
*Metoda **size()** zwraca ilość elementów na liście.*
- **list_array** ()
Konstruktor inicjalizuje pustą listę , wskaźnik na listę jest domyślnie ustawiony na NULL.
- **~list_array** ()
Destruktor usuwa listę
- void **test** (unsigned long int length)
*Metoda **test()** realizuje operacje wypełniania listy danymi.*
- void **quicksort** (int left, int right)
*Metoda **quicksort(int left, int right)** przeprowadza operację sortowania szybkiego (piwot to srodkowy element tablicy).*
- void **quicksort_left** (int left, int right)
*Metoda **quicksort(int left, int right)** przeprowadza operację sortowania szybkiego (piwot to pierwszy element tablicy).*
- void **analyze** (const char *name_output, int repeat, int data_amount)
*Metoda **analyze** zlicza czas wykonywania funkcji *.*
- void **heapsort** ()
Procedura heapsort.

Atrybuty publiczne

- unsigned long int **n**
Ilość elementów na liście.
- int * **tmp**
Uchwyt do listy.
- unsigned long int **temp**
Rozmiar tablicy.

4.3.1 Opis szczegółowy

Definicja w linii 15 pliku list_array.hh.

4.3.2 Dokumentacja konstruktora i destruktor

4.3.2.1 `list_array::list_array ()`

Definicja w linii 8 pliku `list_array.cpp`.

4.3.2.2 `list_array::~~list_array ()`

Definicja w linii 16 pliku `list_array.cpp`.

4.3.3 Dokumentacja funkcji składowych

4.3.3.1 `void list_array::analize (const char * name_output, int repeat, int data_amount) [virtual]`

Metoda `analize` zlicza czas fukcji `quicksort()`

Przykład wywołania funkcji :

`analize("Plik_wynikowy",100,7) ->` Przeprowadza analize czasu trwania funkcji `test()` dla 1 miliona danych (ilość danych należy podać jako potęgę 10) , każdy czas trwania funkcji jest ustalany na podstawie średniej arytmetycznej ze 100 prób , wyniki zapisuje do pliku o nazwie `Plik_wynikowy`.

Uwaga! Aby zmienić tryb rozszerzania tablicy z dodawania 1 elemntu na mnożenie rozmiaru przez 2 należy odkomentować odpowiednie pole w funkcji `list::array test()`

Parametry

in	<i>name_output</i>	- nazwa pliku wynikowego
in	<i>repeat</i>	- ilość powtórzeń testu
in	<i>data_amount</i>	- ilość wynikowych danych podawana jako potęga liczby 10

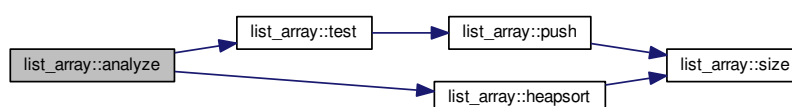
Zwraca

plik.dat z czasami poszczególnych pomiarów oraz ilość testowanych danych.

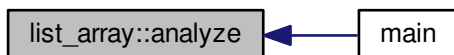
Reimplementowana z `benchmark`.

Definicja w linii 342 pliku `list_array.cpp`.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.3.3.2 void list_array::heapsort ()

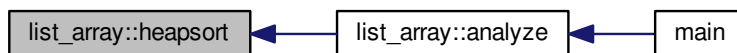
Przeprowadza operację sortowania przez kopcowanie na całej liście

Definicja w linii 279 pliku `list_array.cpp`.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.3.3.3 void list_array::pop (unsigned int whence)

Metoda `pop()` usuwa z listy ostatni/pierwszy element lub zwraca komunikat o błędzie w przypadku próby usunięcia elementu z pustej listy.

Parametry

<code>in</code>	<code>whence</code>	- numer usuwanego elementu
-----------------	---------------------	----------------------------

Definicja w linii 160 pliku `list_array.cpp`.

Oto graf wywołań dla tej funkcji:



4.3.3.4 void list_array::push (int *insert*, unsigned int *where*, unsigned int *extend*, char *x*)

Metoda `push()` wczytuje liczbę naturalną na początek, koniec lub w wybrane miejsce listy;

Parametr *x* może przyjmować 2 wartości '*' lub '+' co odpowiada powiększeniu tablicy o *x* lub *x* razy

Przykład wywołania funkcji :

`push(10,2,4,'*')` - Na drugie miejsce w liście zostanie wstawiona liczba 10 , w przypadku przekroczenia rozmiaru listy jej rozmiar zostanie zwiększony 4 razy.

Parametry

in	<i>insert</i>	- wartość dodawanego elementu
in	<i>where</i>	- na które miejsce ów element ma zostać dodany
in	<i>extend</i>	- parametr ustalający o ile powiększyć listę
in	<i>x</i>	- typ zwiększania tablicy '+'/'-'

Definicja w linii 29 pliku list_array.cpp.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.3.3.5 void list_array::quicksort (int *left*, int *right*)

Aby posortować całą tablicę, należy jako argumenty podać 0 i `size()-1`. `quicksort(0, size()-1)`

Parametry

in	<i>left</i>	- początek zakresu (pierwszy element) sortowania
in	<i>right</i>	- koniec zakresu (ostatni element) sortowania

Definicja w linii 228 pliku list_array.cpp.

4.3.3.6 void list_array::quicksort_left (int left, int right)

Aby posortować całą tablicę, należy jako argumenty podać 0 i `size()-1`. quicksort(0,`size()-1`)

Parametry

in	<i>left</i>	- początek zakresu (pierwszy element) sortowania
in	<i>right</i>	- koniec zakresu (ostatni element) sortowania

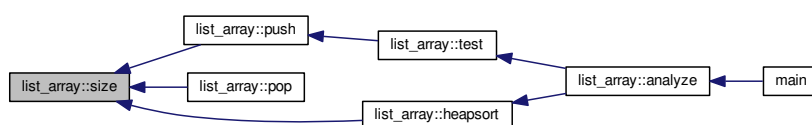
Definicja w linii 253 pliku list_array.cpp.

4.3.3.7 unsigned list_array::size ()

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

Definicja w linii 199 pliku list_array.cpp.

Oto graf wywoływań tej funkcji:



4.3.3.8 void list_array::test (unsigned long int length) [virtual]

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

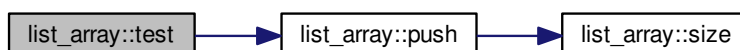
Parametry

in	<i>length</i>	- ilość dodawanych lementów
----	---------------	-----------------------------

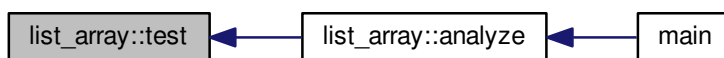
Implementuje `benchmark`.

Definicja w linii 207 pliku list_array.cpp.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.3.4 Dokumentacja atrybutów składowych

4.3.4.1 unsigned long int list_array::n

Definicja w linii 22 pliku list_array.hh.

4.3.4.2 unsigned long int list_array::temp

Definicja w linii 30 pliku list_array.hh.

4.3.4.3 int* list_array::tmp

Definicja w linii 26 pliku list_array.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

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

4.4 Dokumentacja struktury node

```
#include <list.hh>
```

Diagram współpracy dla node:



Metody publiczne

- [node](#) (int element)
Konstruktor węzła.

Atrybuty publiczne

- `int data`

Pole do którego dopisywane są dane.

- `node * next`

Pole będące wskaźnikiem na następny element.

4.4.1 Opis szczegółowy

Definicja w linii 16 pliku list.hh.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 `node::node (int element) [inline]`

Konstruktor inicjalizuje nowy węzeł z wartością równą element oraz wskaźnikiem na NULL

Parametry

<code>in</code>	<code>element</code>	- element dodawany na koniec listy
-----------------	----------------------	------------------------------------

Definicja w linii 38 pliku list.hh.

4.4.3 Dokumentacja atrybutów składowych

4.4.3.1 `int node::data`

Definicja w linii 22 pliku list.hh.

4.4.3.2 `node* node::next`

Definicja w linii 28 pliku list.hh.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- `list.hh`

4.5 Dokumentacja klasy queue

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

Diagram dziedziczenia dla queue

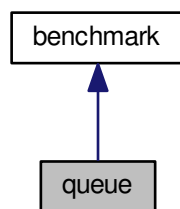
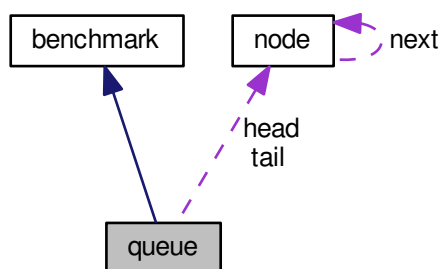


Diagram współpracy dla queue:



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.

4.5.1 Opis szczegółowy

Definicja w linii 18 pliku `queue.hh`.

4.5.2 Dokumentacja konstruktora i destruktor

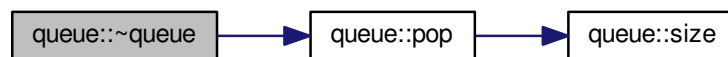
4.5.2.1 `queue::queue ()`

Definicja w linii 9 pliku `queue.cpp`.

4.5.2.2 `queue::~~queue ()`

Definicja w linii 14 pliku `queue.cpp`.

Oto graf wywołań dla tej funkcji:



4.5.3 Dokumentacja funkcji składowych

4.5.3.1 `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 47 pliku `queue.cpp`.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.5.3.2 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

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

Definicja w linii 27 pliku `queue.cpp`.

Oto graf wywoływań tej funkcji:

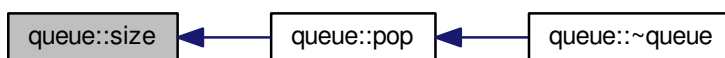


4.5.3.3 unsigned queue::size () [private]

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

Definicja w linii 68 pliku `queue.cpp`.

Oto graf wywoływań tej funkcji:



4.5.3.4 `void queue::test (unsigned long int length)` `[virtual]`

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

Parametry

<code>in</code>	<code>length</code>	- ilość danych do wstawienia
-----------------	---------------------	------------------------------

Implementuje [benchmark](#).

Definicja w linii 83 pliku `queue.cpp`.

Oto graf wywołań dla tej funkcji:



4.5.4 Dokumentacja atrybutów składowych

4.5.4.1 `node* queue::head` `[private]`

Definicja w linii 27 pliku `queue.hh`.

4.5.4.2 `node* queue::tail` `[private]`

Definicja w linii 31 pliku `queue.hh`.

Dokumentacja dla tej klasy została wygenerowana z plików:

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

4.6 Dokumentacja klasy stack

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

Diagram dziedziczenia dla `stack`

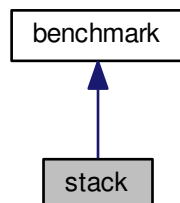
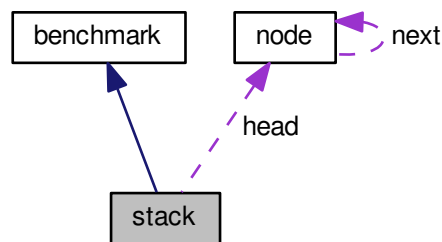


Diagram współpracy dla stack:



Metody publiczne

- `stack()`
Konstruktor inicjalizujący pusty stos, początek stosu jest domyślnie ustawiony 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 wskaźnikiem na pierwszy element stosu.

4.6.1 Opis szczegółowy

Definicja w linii 15 pliku stack.hh.

4.6.2 Dokumentacja konstruktora i destruktora

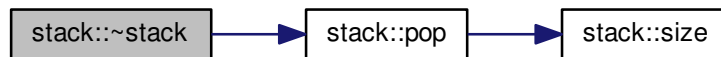
4.6.2.1 `stack::stack ()`

Definicja w linii 8 pliku stack.cpp.

4.6.2.2 `stack::~~stack ()`

Definicja w linii 12 pliku stack.cpp.

Oto graf wywołań dla tej funkcji:



4.6.3 Dokumentacja funkcji składowych

4.6.3.1 `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 39 pliku stack.cpp.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

4.6.3.2 `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>	- wartość dodawanego elementu
-----------------	---------------------	-------------------------------

Definicja w linii 22 pliku `stack.cpp`.

Oto graf wywołań tej funkcji:

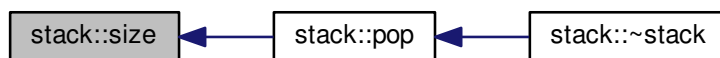


4.6.3.3 `unsigned stack::size ()` `[private]`

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

Definicja w linii 56 pliku `stack.cpp`.

Oto graf wywołań tej funkcji:



4.6.3.4 `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 dodawanych elementów
-----------------	---------------------	------------------------------

Implementuje `benchmark`.

Definicja w linii 71 pliku `stack.cpp`.

Oto graf wywołań dla tej funkcji:



4.6.4 Dokumentacja atrybutów składowych

4.6.4.1 `node* stack::head` `[private]`

Definicja w linii 24 pliku `stack.hh`.

Dokumentacja dla tej klasy została wygenerowana z plików:

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

Rozdział 5

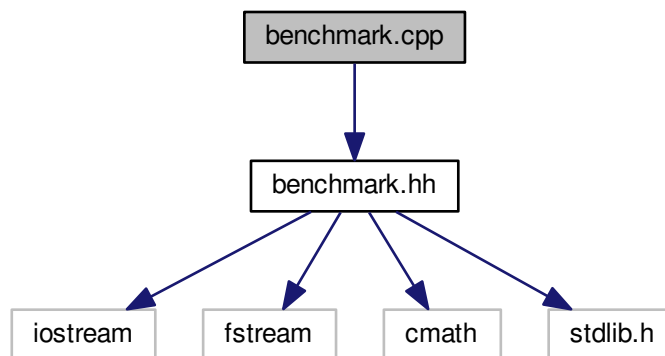
Dokumentacja plików

5.1 Dokumentacja pliku benchmark.cpp

Deklaracja funkcji z klasy Benchmark.

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

Wykres zależności załączania dla benchmark.cpp:

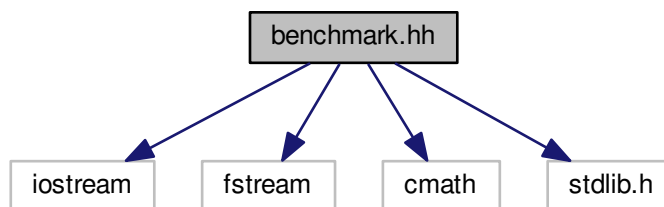


5.2 Dokumentacja pliku benchmark.hh

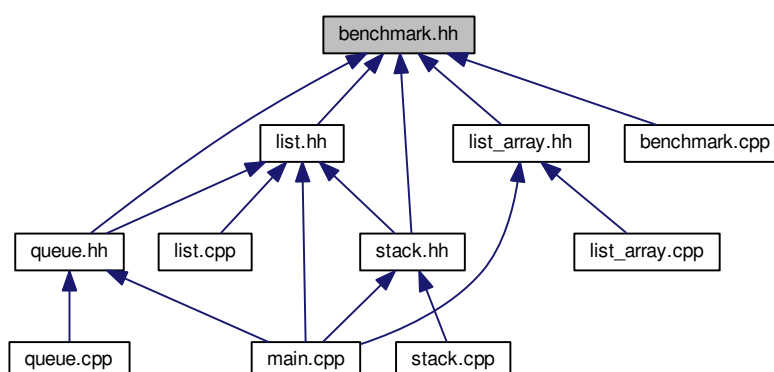
Definicja klasy Benchmark.

```
#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

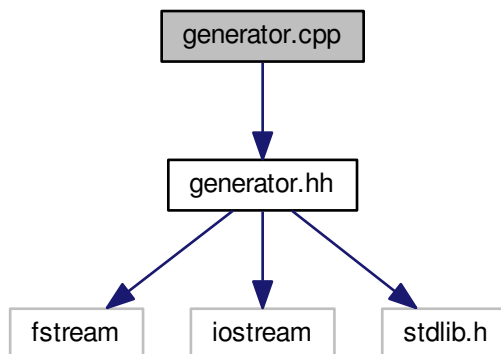
- class `benchmark`

5.3 Dokumentacja pliku `generator.cpp`

Deklaracja funkcji generującej liczby losowe.

```
#include "generator.hh"
```

Wykres zależności załączania dla generator.cpp:



Funkcje

- void [data_generator](#) (unsigned long int data_amount)

Generuje liczby losowe.

5.3.1 Dokumentacja funkcji

5.3.1.1 void 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
----	-------------	---

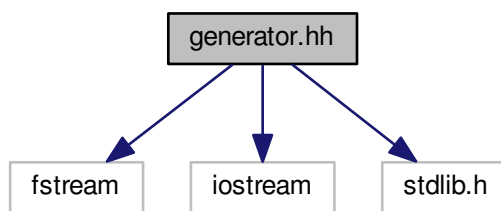
Definicja w linii 9 pliku generator.cpp.

5.4 Dokumentacja pliku generator.hh

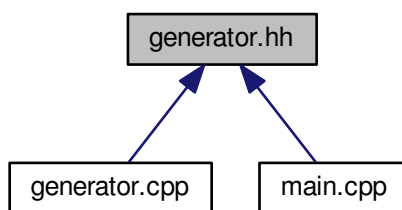
Definicja generatora liczb losowych.

```
#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

- void [data_generator](#) (unsigned long int data_amount)
Generuje liczby losowe.

5.4.1 Dokumentacja funkcji

5.4.1.1 void 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
----	-------------	---

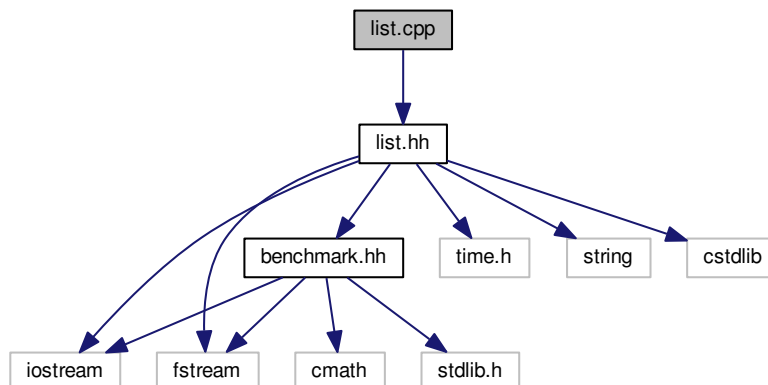
Definicja w linii 9 pliku generator.cpp.

5.5 Dokumentacja pliku list.cpp

Deklaracja klasy list.


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

Wykres zależności załączania dla list.cpp:

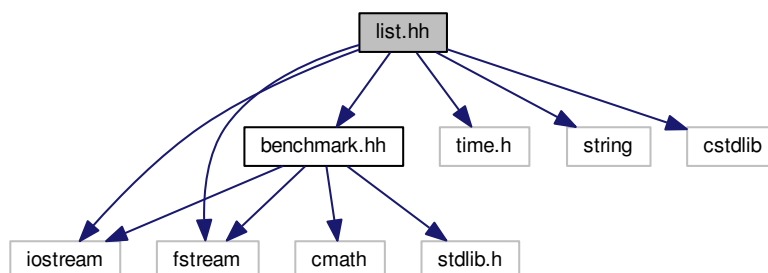


5.6 Dokumentacja pliku list.hh

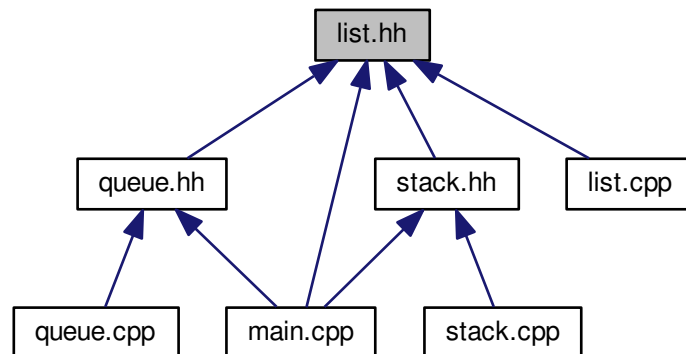
Definicja klasy lista.

```
#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

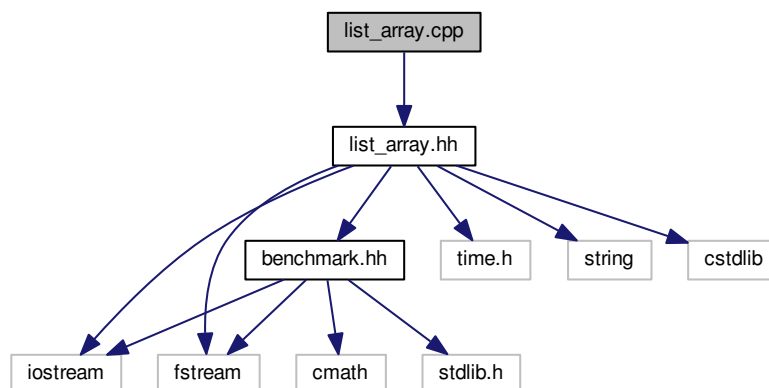
- struct [node](#)
- class [list](#)

5.7 Dokumentacja pliku list_array.cpp

Deklaracja klasy [list_array](#).

```
#include "list_array.hh"
```

Wykres zależności załączania dla list_array.cpp:

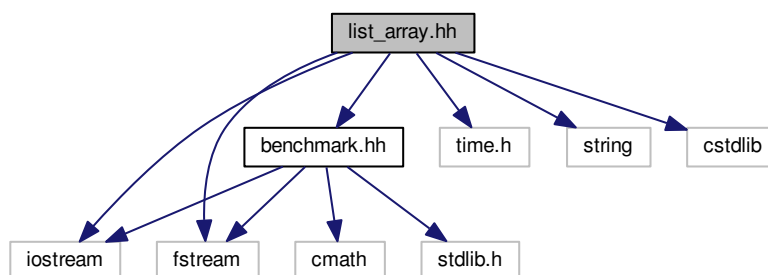


5.8 Dokumentacja pliku list_array.hh

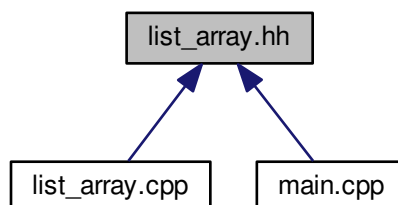
Definicja klasy `list_array`.

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

Wykres zależności załączania dla list_array.hh:



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



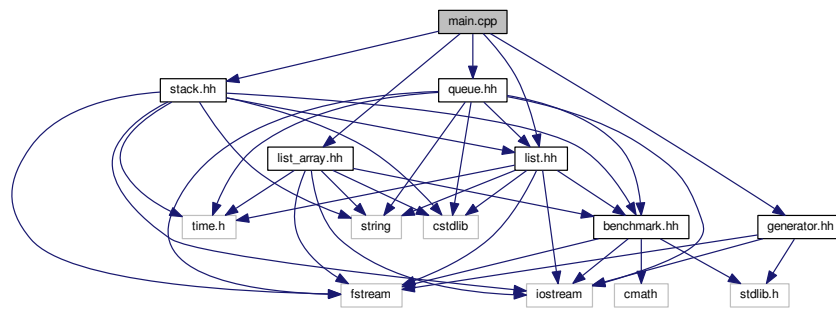
Komponenty

- class `list_array`

5.9 Dokumentacja pliku main.cpp

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

Wykres zależności załączania dla main.cpp:



Funkcje

- int [main](#) ()

Aby wygenerować liczby losowe należy odkomentować linie zawierającą funkcję [data_generator\(\)](#)

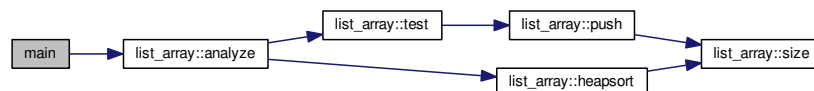
5.9.1 Dokumentacja funkcji

5.9.1.1 int main ()

Generuj Dane losowe (ustawione na 10^7)

Definicja w linii 9 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

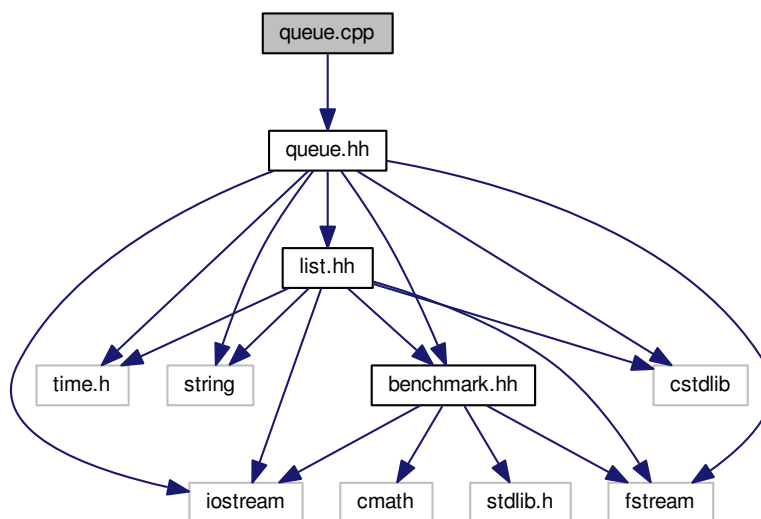


5.10 Dokumentacja pliku queue.cpp

Deklaracja klasy queue.

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

Wykres zależności załączania dla queue.cpp:



5.11 Dokumentacja pliku queue.hh

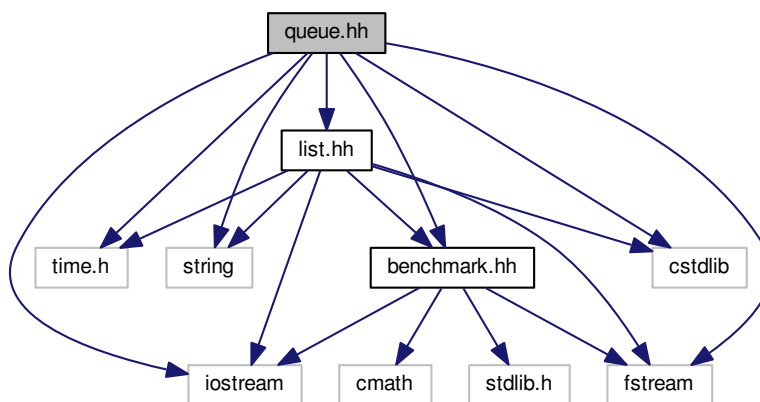
Definicja klasy stack.

```

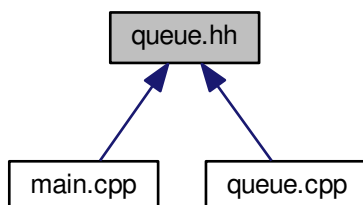
#include <iostream>
#include <time.h>
#include <string>
#include <fstream>
#include <cstdlib>
#include "benchmark.hh"
#include "list.hh"

```

Wykres zależności załączania dla queue.hh:



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



Komponenty

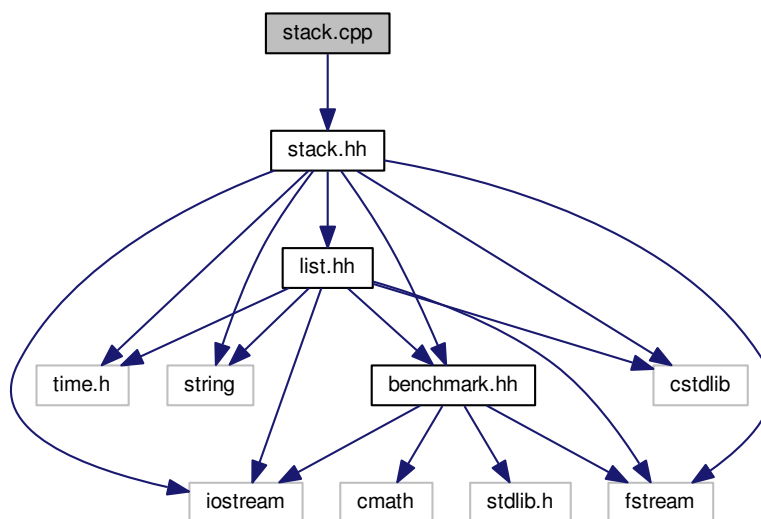
- class `queue`

5.12 Dokumentacja pliku stack.cpp

Deklaracja klasy stack.

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

Wykres zależności załączania dla stack.cpp:

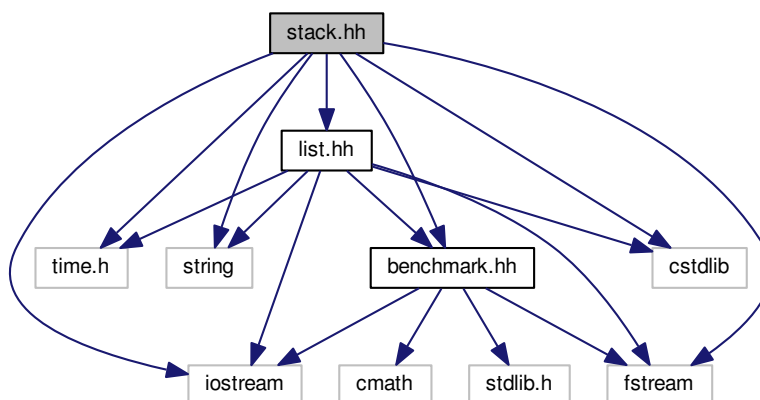


5.13 Dokumentacja pliku stack.hh

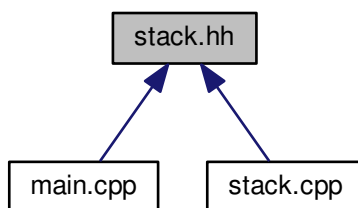
Definicja klasy stack.

```
#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](#)

Skorowidz

- ~list
 - list, [9](#)
- ~list_array
 - list_array, [15](#)
- ~queue
 - queue, [22](#)
- ~stack
 - stack, [26](#)
- analyze
 - benchmark, [7](#)
 - list_array, [15](#)
- benchmark, [7](#)
 - analyze, [7](#)
 - test, [8](#)
- benchmark.cpp, [31](#)
- benchmark.hh, [31](#)
- data
 - node, [20](#)
- data_generator
 - generator.cpp, [33](#)
 - generator.hh, [34](#)
- generator.cpp, [32](#)
 - data_generator, [33](#)
- generator.hh, [33](#)
 - data_generator, [34](#)
- head
 - list, [13](#)
 - queue, [25](#)
 - stack, [29](#)
- heapsort
 - list_array, [16](#)
- list, [8](#)
 - ~list, [9](#)
 - head, [13](#)
 - list, [9](#)
 - pop, [10](#)
 - push, [10](#)
 - size, [11](#)
 - tail, [13](#)
 - test, [11](#)
- list.cpp, [34](#)
- list.hh, [35](#)
- list_array, [13](#)
 - ~list_array, [15](#)
 - analyze, [15](#)
 - heapsort, [16](#)
 - list_array, [15](#)
 - list_array, [15](#)
 - n, [19](#)
 - pop, [16](#)
 - push, [17](#)
 - quicksort, [17](#)
 - quicksort_left, [18](#)
 - size, [18](#)
 - temp, [19](#)
 - test, [18](#)
 - tmp, [19](#)
- list_array.cpp, [36](#)
- list_array.hh, [37](#)
- main
 - main.cpp, [38](#)
- main.cpp, [37](#)
 - main, [38](#)
- n
 - list_array, [19](#)
- next
 - node, [20](#)
- node, [19](#)
 - data, [20](#)
 - next, [20](#)
 - node, [20](#)
- pop
 - list, [10](#)
 - list_array, [16](#)
 - queue, [22](#)
 - stack, [27](#)
- push
 - list, [10](#)
 - list_array, [17](#)
 - queue, [23](#)
 - stack, [27](#)
- queue, [20](#)
 - ~queue, [22](#)
 - head, [25](#)
 - pop, [22](#)
 - push, [23](#)
 - queue, [22](#)
 - size, [23](#)
 - tail, [25](#)
 - test, [23](#)
- queue.cpp, [38](#)

- queue.hh, [39](#)
- quicksort
 - list_array, [17](#)
- quicksort_left
 - list_array, [18](#)
- size
 - list, [11](#)
 - list_array, [18](#)
 - queue, [23](#)
 - stack, [28](#)
- stack, [25](#)
 - ~stack, [26](#)
 - head, [29](#)
 - pop, [27](#)
 - push, [27](#)
 - size, [28](#)
 - stack, [26](#)
 - test, [28](#)
- stack.cpp, [40](#)
- stack.hh, [41](#)
- tail
 - list, [13](#)
 - queue, [25](#)
- temp
 - list_array, [19](#)
- test
 - benchmark, [8](#)
 - list, [11](#)
 - list_array, [18](#)
 - queue, [23](#)
 - stack, [28](#)
- tmp
 - list_array, [19](#)