Politechnika Śląska
Wydział Automatyki Elektroniki i Informatyki

Implementacja kontenerów z biblioteki STL

Wojciech Janota

3 listopada 2020

1 Opis projektu

Głównym założeniem projetku było stworzenie odpowiedników kontenerów z biblioteki Standard Template Library. Założyłem, że zaimplementuję następujące kontenery:

- vector
- queue
- priority-queue
- stack
- list
- set
- binary tree

2 Opis budowy klas

2.1 Container

Zgodnie ze złożonymi wcześniej wstępnymi szkicami projektu główna klasa bazowa, `container`, odpowiada za zarządzanie pamięcią i podstawowe operacje na wewnętrznej tablicy dynamicznej. Po niej dziedziczą klasy `vector, queue, priority-queue, stack, set`. Jest w niej zaimplementowany konstruktor i destruktor, które są wspólne dla wszystkich kontenerów po niej dziedziczących, podstawowe "getery"i śetery"dla wewnętrznych zmiennych, operator[], podstawowe funkcje operujące na wewnętrznej tablicy dynamicznej (dodawanie, usuwanie elementów) oraz funkcja sortująca, jako metoda chroniona. Klasa jest abstrakcyjna, aby uniemożliwić stworzenie obiektu na jej podstawie.

2.2 Vector

`Vector` to kontener dziedziczący po `container` będący bardziej zaawansowaną tablicą dynamiczną. Zawiera w sobie mechanizmy obsługi pamięci, inteligentnie zarządza swoją wielkością. Dodawanie elementów na początek tego kontenera lub w środku jest bardzo kosztowne czasowo, ponieważ wymagane jest przesunięcie wszystkich elementów od indeksu na którym wstawiany jest nowy element o jedną pozycję w prawo. Jest za to bardzo łatwy w obsłudze (jak tablica dynamiczna), w miarę łatwy w implementacji i w większości zastosowań dość szybki.

2.3 Queue

`Queue`, czyli kolejka to kontener FIFO (First In First Out). Umożliwia dodawanie elementów na koniec, pobranie i wyrzucenie pierwszego elementu i podstawowe metody takie jak zwrócenie wielkości kontenera.

2.4 Priority queue

Jest to kolejka priorytetowa, na podstawie podanego priorytetu (MIN, MAX) na początku kolejki zawsze znajduje się największy lub najmniejszy element. Reszta działania jest identyczna jak w przypadku zwykłej kolejki.

2.5 Stack

`Stack`, czyli stos to kontener FILO (First In Last Out). Możliwe jest dodanie elementu na górę stosu i zdjęcie elementu ze stosu.

2.6 Set

`Set` to kontener samosortujący, poprzez wstawienie elementu w odpowiednie miejsce. Zaimplementowałem odpowiednik STL-owego multiseta sortującego tylko rosnąco, zwracającego także indeks pierwszego elementu nie mniejszego i nie większego od podanego.

2.7 List

Lista jest odpowiednikiem vectora, używającym listy jako wewnętrzego sposobu przechowywania danych. Umozliwia to dodawanie elementu na początku i na końcu w czasie stałym, jednak powoduje również, że operacje pobierania elementu z konkretnego indeksu są wykonywane w czasie liniowym.

2.8 Binary tree

Drzewo binarne jest strukturą sortująca, przechowującą kolejne elementy jako kolejne węzły w drzewie binarnym. Moja implementacja umożliwia wstawianie do drzewa, wypisanie drzewa w kolejności in-order, usuwanie poszczególnych węzłów oraz pobranie prawego i lewego syna podanego elementu.

3 Szczegółowy opis klas, zależności i metod

Szczegółowa dokumentacja została wygenerowana przez narzędzie doxygen.

4 Problemy

W związku z bardzo ograniczonym czasem na napisanie projektu, nie udało się zaimplementować mapy, a binary tree mogłoby zostać jeszcze udoskonalone. Używanie szablonów momentami bywa nieintuicyjne, jednak udało się zaimplementować powyższe kontenery w ich użyciem z czego jestem zadowolony.

5 Testowanie

Do testowania wycieków pamięci użyłem narzędzia valgrind. Projekt okazał się od nich wolny. Przetestowane zostały także wszystkie metody poszczególnych klas. Kod został sformatowany z użyciem clang-format w standardzie LLVM.

6 Wnioski

Po napisaniu w miarę funkcjonalnych odpowiedników kontenerów z STL mogę stwierdzić, że znacząco ułatwiają one pracę z użyciem języka C++. Pisząc je należy pamiętać o pewnych niuansach, które pominięte mogą okazać się katasrofalne, jak np. niepoprawne poszerzanie i zwężanie vectora może spowodować błąd SIGSEGV, ponieważ nagle program odwoła się do niepoprawnej pamięci. Muszę również stwierdzić, że pisanie projektu w tak krótkim czasie jest o wiele trudniejsze i ciężej jest rozłożyć efektywnie pracę. Rozumiem, że wynika to z obecnej sytuacji, jednak chcę zwrócić uwagę na problem, jaki tak krótki czas może sprawić.

7 Dokuemntacja wygenerowana przez doxygen

Kontenery STL

0.1

Wygenerowano przez Doxygen 1.8.18

Rozdział 1

Implementacja kontenerów z biblioteki STL

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 szablonu klasy binary_tree< T >

`#include <binary_tree.h>`

Metody publiczne

- binary_tree ()
  Construct a new binary tree object.
- ~binary_tree ()
  Destroy the binary tree object.
- void print () const noexcept
  wypisuje drzewo w porządku in-order
- T left (const T &item) const
  zwraca lewego syna ojca
- T right (const T &item) const
  zwraca prawego syna ojca
- void delete_item (const T &item)
  usuwa element
- void clear ()
  czyści całe drzewo z pamięci
- void insert (const T &item)
  wstawia wartość do drzewa

5.1.1 Dokumentacja konstruktora i destruktora

5.1.1.1 binary_tree()

```
template<class T >
binary_tree< T >::binary_tree
```

Construct a new binary tree object.

5.1.1.2 ~binary_tree()

```
template<class T >
binary_tree< T >::~binary_tree
```

Destroy the binary tree object.

5.1.2 Dokumentacja funkcji składowych

5.1.2.1 clear()

```
template<class T >
void binary_tree< T >::clear
```

czyści całe drzewo z pamięci

5.1.2.2 delete_item()

```
template<class T >
void binary_tree< T >::delete_item (
            const T & item )
```

usuwa element

Parametry

item	element

5.1.2.3 insert()

```
template<class T >
void binary_tree< T >::insert (
            const T & item )
```

wstawia wartość do drzewa

Parametry

item	

5.1.2.4 left()

```
template<class T >
T binary_tree< T >::left (
            const T & item ) const  [inline]
```

zwraca lewego syna ojca

Parametry

item	ojciec

Zwraca

T lewy syn ojca

5.1.2.5 print()

```
template<class T >
void binary_tree< T >::print  [noexcept]
```

wypisuje drzewo w porządku in-order

5.1.2.6 right()

```
template<class T >
T binary_tree< T >::right (
            const T & item ) const  [inline]
```

zwraca prawego syna ojca

Parametry

item	ojciec

Zwraca

T prawy syn ojca

Dokumentacja dla tej klasy została wygenerowana z pliku:

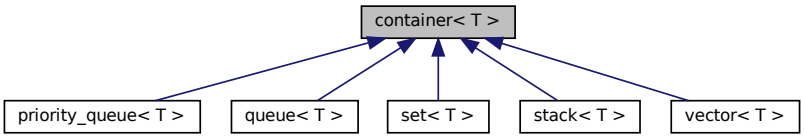
- include/binary_tree.h

5.2 Dokumentacja szablonu klasy container< T >

bazowa klasa zarządzająca pamięcią, implementująca podstawowe funkcjonalności itd.

`#include <container.h>`

Diagram dziedziczenia dla container< T >



Metody publiczne

- container ()

  Konstruktor bezparametryczny, inicjalizuje pusty kontener.
- virtual ~container ()=0

  Destruktor zwalnia pamięć alokowaną dynamicznie, jest wirtualna, aby niemożliwe było stworzenie obiektu z klasy container.
- size_t size_of () const noexcept

  funkcja zwracająca wielkość kontenera
- bool contains (const T &item) const noexcept

  funkcja sprawdzająca, czy dany element wystepuje w kontenerze
- bool empty () const noexcept

  funkcja sprawdzająca, czy kontener jest pusty
- T begin () const noexcept

  funkcja zwracająca pierwszy element kontenera
- T end () const noexcept

  funkcja zwracająca ostatni element kontenera
- void clear () noexcept

  funkcja usuwająca wszystkie elementy z kontenera, lecz nie zwalniająca pamięci
- void delete_item (const size_t &index) noexcept

  funkcja usuwająca element na podanym indeksie
- size_t count (const T &item) const noexcept

  funkcja zliczająca wystąpienia danego elementu w kontenerze
- int find (const T &item) const noexcept

  funkcja znajdująca indeks elementu
- T & operator[ ] (const size_t index)

  operator wyłuskania []

Metody chronione

- void add (const T item)

  funkcja dodająca element na koniec kontenera
- void add (const T item, const size_t index)

  przeładowana funkcja add(), dodająca element w konkretnym indeksie, "rozpychająca" pozostałe elementy
- T & get (const size_t index)

  funkcja pobierająca element z podanego indeksu
- void resize (const size_t step)

  funkcja zmieniająca wielkość kontenera
- void sort (const bool ascendingOrder=true)

  funkcja sortująca wewnętrzną tablicę kontenera
- void insert (const T item, const size_t index)

  funkcja wstawiająca element na daną pozycję
- void clear_memory ()

  funkcja czyszcząca pamięć obiektu

5.2.1 Opis szczegółowy

template<class T>
class container< T >

bazowa klasa zarządzająca pamięcią, implementująca podstawowe funkcjonalności itd.

5.2.2 Dokumentacja konstruktora i destruktora

5.2.2.1 container()

```
template<class T >
container< T >::container
```

Konstruktor bezparametryczny, inicjalizuje pusty kontener.

5.2.2.2 ~container()

```
template<class T >
container< T >::~container  [pure virtual]
```

Destruktor zwalnia pamięć alokowaną dynamicznie, jest wirtualna, aby niemożliwe było stworzenie obiektu z klasy container.

5.2.3 Dokumentacja funkcji składowych

5.2.3.1 add() [1/2]

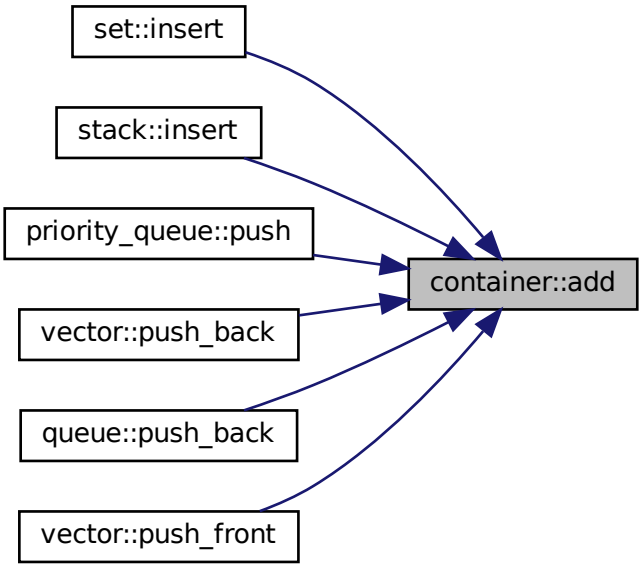
```
template<class T >
void container< T >::add (
            const T item )  [protected]
```

funkcja dodająca element na koniec kontenera

Parametry

item	element do dodania

Oto graf wywoływań tej funkcji:



5.2.3.2 add() [2/2]

```
template<class T >
void container< T >::add (
            const T item,
            const size_t index )  [protected]
```

przeładowana funkcja add(), dodająca element w konkretnym indeksie, "rozpychająca" pozostałe elementy

Parametry

item	element do dodania
index	indeks na którym na znaleźć się ten element

5.2.3.3 begin()

```
template<class T >
T container< T >::begin  [inline], [noexcept]
```

funkcja zwracająca pierwszy element kontenera

Zwraca

T pierwszy element kontenera

5.2.3.4 clear()

```
template<class T >
void container< T >::clear  [noexcept]
```

funkcja usuwająca wszystkie elementy z kontenera, lecz nie zwalniająca pamięci

5.2.3.5 clear_memory()

```
template<class T >
void container< T >::clear_memory  [protected]
```

funkcja czyszcząca pamięć obiektu

5.2.3.6 contains()

```
template<class T >
bool container< T >::contains (
            const T & item ) const  [noexcept]
```

funkcja sprawdzająca, czy dany element wystepuje w kontenerze

Parametry

item	element do sprawdzenia

Zwraca

true element jest w kontenerze

false elementu nie ma w kontenerze

5.2.3.7 count()

```
template<class T >
size_t container< T >::count (
            const T & item ) const  [noexcept]
```

funkcja zliczająca wystąpienia danego elementu w kontenerze

Parametry

item	szukany element

Zwraca

const size_t liczba wystapień danego elementu

5.2.3.8 delete_item()

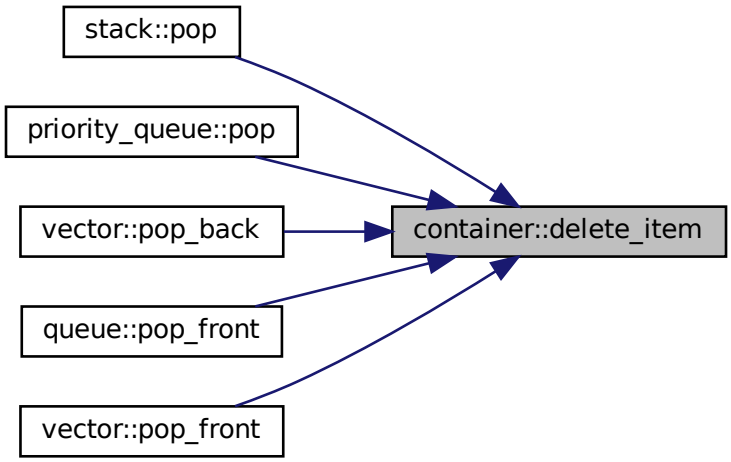
```
template<class T >
void container< T >::delete_item (
            const size_t & index )  [noexcept]
```

funkcja usuwająca element na podanym indeksie

Parametry

index	indeks wystapienia elementu

Oto graf wywoływań tej funkcji:



5.2.3.9 empty()

```
template<class T >
bool container< T >::empty  [inline], [noexcept]
```

funkcja sprawdzająca, czy kontener jest pusty

Zwraca

true kontener jest pusty

false kontener nie jest pusty

5.2.3.10 end()

```
template<class T >
T container< T >::end  [inline], [noexcept]
```

funkcja zwracająca ostatni element kontenera

Zwraca

T ostatni element kontenera

5.2.3.11 find()

```
template<class T >
int container< T >::find (
            const T & item ) const  [noexcept]
```

funkcja znajdująca indeks elementu

Parametry

item	szukany element

Zwraca

const int indeks szukanego elementu, -1 jeżeli element nie występuje

5.2.3.12 get()

```
template<class T >
T & container< T >::get (
            const size_t index )  [protected]
```

funkcja pobierająca element z podanego indeksu

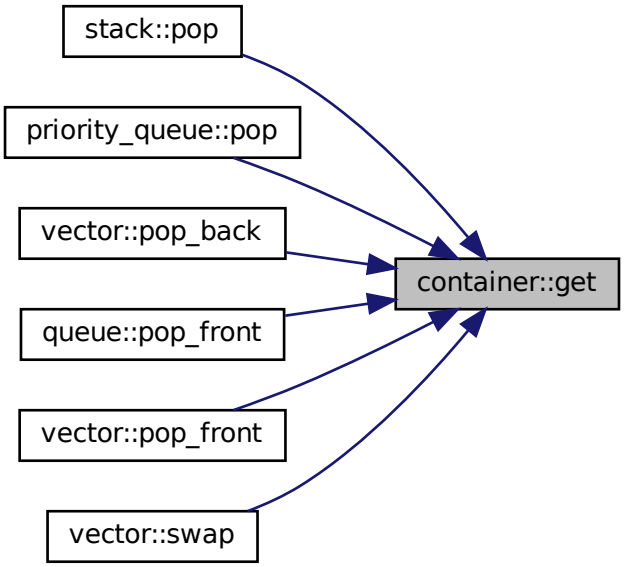
Parametry

index	indeks na którym jest element

Zwraca

T element, który ma zostać zwrócony

Oto graf wywoływań tej funkcji:



5.2.3.13 insert()

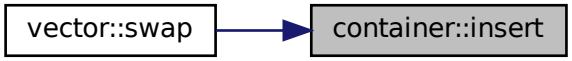
```
template<class T >
void container< T >::insert (
            const T item,
            const size_t index )  [protected]
```

funkcja wstawiająca element na daną pozycję

Parametry

item	element do wstawienia
index	indeks na którym ma być wstawiony element

Oto graf wywoływań tej funkcji:



5.2.3.14 operator[]()

```
template<class T >
T & container< T >::operator[] (
            const size_t index )
```

operator wyłuskania []

Parametry

index	indeks elementu

Zwraca

T& referencja

5.2.3.15 resize()

```
template<class T >
void container< T >::resize (
            const size_t step )  [protected]
```

funkcja zmieniająca wielkość kontenera

Parametry

step	kolejna wielkość kontenera

5.2.3.16 size_of()

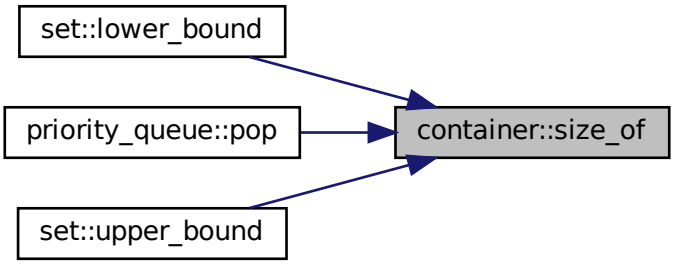
```
template<class T >
size_t container< T >::size_of  [inline], [noexcept]
```

funkcja zwracająca wielkość kontenera

Zwraca

const size_t wielkość kontenera, 0 jeżeli pusty

Oto graf wywoływań tej funkcji:



5.2.3.17 sort()

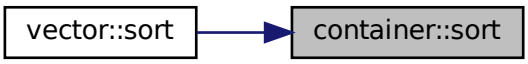
```
template<class T >
void container< T >::sort (
            const bool ascendingOrder = true )  [protected]
```

funkcja sortująca wewnętrzną tablicę kontenera

Parametry

ascendingOrder	jeżeli True, sortuje rosnąco, jeżeli false malejąco, domyślnie True

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/container.h

5.3 Dokumentacja szablonu klasy list< T >

`#include <list.h>`

Metody publiczne

- list ()
  Construct a new list object.
- ~list ()
  Destroy the list object.
- int size_of () const
  zwraca wielkość listy
- bool empty () const
  czy lista jest pusta
- T & front ()
  zwraca pierwszy element
- T & back ()
  zwraca ostatni element
- T * begin ()
  zwraca wskaźnik na pierwszy element
- T * end ()
  zwraca wskaźnik na ostatni element
- void push_back (const T &item)
  dodaje element na koniec
- void push_front (const T &item)
  dodaje element na początek
- void insert (const T &item, const size_t index)
  wstawia element na konkretny indeks

- void pop_back ()

  usuwa element z końca
- void pop_front ()

  usuwa element z początku
- void erase (size_t index)

  usuwa element z podanego indeksu
- void remove (const T &item)

  usuwa wszytskie wystapienia elementu
- T & operator[ ] (const size_t index)

  operator wyłuskania []

5.3.1 Dokumentacja konstruktora i destruktora

5.3.1.1 list()

```
template<class T >
list< T >::list
```

Construct a new list object.

5.3.1.2 ~list()

```
template<class T >
list< T >::~list
```

Destroy the list object.

5.3.2 Dokumentacja funkcji składowych

5.3.2.1 back()

```
template<class T >
T & list< T >::back  [inline]
```

zwraca ostatni element

Zwraca

T& ostatni element

5.3.2.2 begin()

```
template<class T >
T * list< T >::begin  [inline]
```

zwraca wskaźnik na pierwszy element

Zwraca

T∗ wskaźnik na pierwszy element

5.3.2.3 empty()

```
template<class T >
bool list< T >::empty  [inline]
```

czy lista jest pusta

Zwraca

true tak

false nie

5.3.2.4 end()

```
template<class T >
T * list< T >::end  [inline]
```

zwraca wskaźnik na ostatni element

Zwraca

T∗ wskaźnik na ostatni element

5.3.2.5 erase()

```
template<class T >
void list< T >::erase (
            size_t index )
```

usuwa element z podanego indeksu

Parametry

index	indeks

5.3.2.6 front()

```
template<class T >
T & list< T >::front  [inline]
```

zwraca pierwszy element

Zwraca

T& pierwszy element

5.3.2.7 insert()

```
template<class T >
void list< T >::insert (
            const T & item,
            const size_t index )
```

wstawia element na konkretny indeks

Parametry

item	element do wstawienia
index	indeks na którym ma być wstawiony

5.3.2.8 operator[]()

```
template<class T >
T & list< T >::operator[] (
            const size_t index )
```

operator wyłuskania []

Parametry

index	indeks do pobrania

Zwraca

T& referencja do elementu

5.3.2.9 pop_back()

```
template<class T >
void list< T >::pop_back
```

usuwa element z końca

5.3.2.10 pop_front()

```
template<class T >
void list< T >::pop_front
```

usuwa element z początku

5.3.2.11 push_back()

```
template<class T >
void list< T >::push_back (
            const T & item )
```

dodaje element na koniec

Parametry

item	element do dodania

5.3.2.12 push_front()

```
template<class T >
void list< T >::push_front (
            const T & item )
```

dodaje element na początek

Parametry

item	element do dodania

5.3.2.13 remove()

```
template<class T >
void list< T >::remove (
            const T & item )
```

usuwa wszytskie wystapienia elementu

Parametry

item	element do usunięcia

5.3.2.14 size_of()

```
template<class T >
int list< T >::size_of  [inline]
```

zwraca wielkość listy

Zwraca

int wielkość listy

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/list.h

5.4 Dokumentacja szablonu klasy priority_queue< T >

```
#include <priority_queue.h>
```

Diagram dziedziczenia dla priority_queue< T >

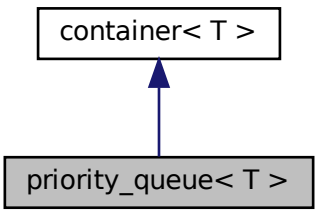
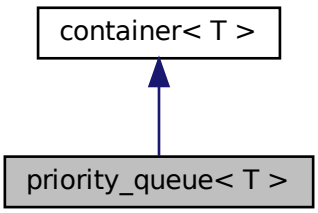


Diagram współpracy dla priority_queue< T >:



Metody publiczne

- void setPriority (priorityType choice)

  ustawia priorytet kolejki
- void push (const T &item)

  wkłada element do kolejki
- T pop ()

  ściąga element z kolejki i zwraca ten element.

Dodatkowe Dziedziczone Składowe

5.4.1 Dokumentacja funkcji składowych

5.4.1.1 pop()

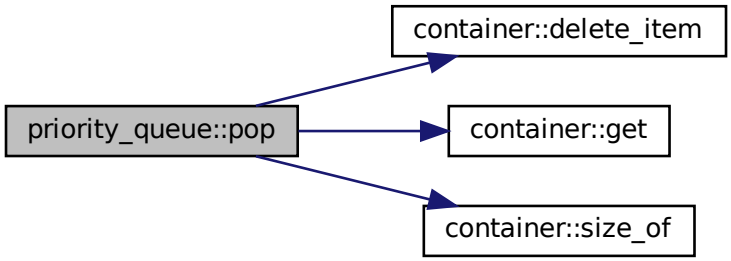
```
template<class T >
T priority_queue< T >::pop
```

ściąga element z kolejki i zwraca ten element.

Zwraca

T ściągnięty element

Oto graf wywołań dla tej funkcji:



5.4.1.2 push()

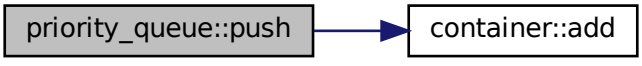
```
template<class T >
void priority_queue< T >::push (
              const T & item )
```

wkłada element do kolejki

Parametry

item	element do włożenia

Oto graf wywołań dla tej funkcji:



5.4.1.3 setPriority()

```
template<class T >
void priority_queue< T >::setPriority (
            priorityType choice )  [inline]
```

ustawia priorytet kolejki

Parametry

choice	jaki priorytet (najmniejszy lub największy)

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/priority_queue.h

5.5 Dokumentacja szablonu klasy queue< T >

```
#include <queue.h>
```

Diagram dziedziczenia dla queue< T >

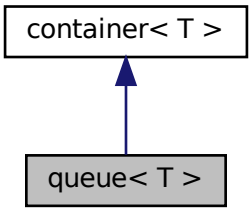
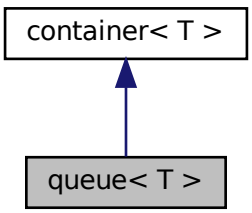


Diagram współpracy dla queue< T >:



Metody publiczne

- void push_back (const T &item)

  funkcja dodająca element na koniec vectora
- T pop_front ()

  funkcja usuwająca element na początku i zwracająca go

Dodatkowe Dziedziczone Składowe

5.5.1 Dokumentacja funkcji składowych

5.5.1.1 pop_front()

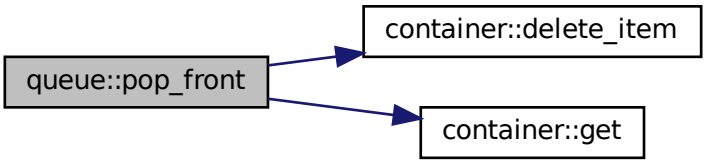
```
template<class T >
T queue< T >::pop_front
```

funkcja usuwająca element na początku i zwracająca go

Zwraca

T pierwszy element

Oto graf wywołań dla tej funkcji:



5.5.1.2 push_back()

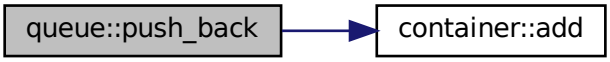
```
template<class T >
void queue< T >::push_back (
            const T & item )
```

funkcja dodająca element na koniec vectora

Parametry

item	referencja na dodawany element, później tworzona jest kopia, więc tutaj warto przekazać referencję

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/queue.h

5.6 Dokumentacja szablonu klasy set< T >

`#include <set.h>`

Diagram dziedziczenia dla set< T >

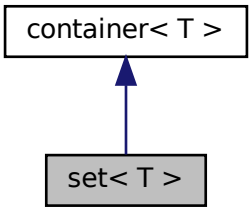
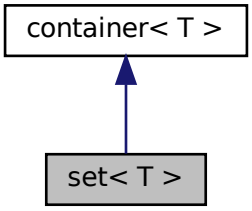


Diagram współpracy dla set< T >:



Metody publiczne

- void insert (const T &item)

  funkcja dodająca element do kontenera
- int lower_bound (const T &item) const noexcept

  funkcja zwracająca indeks pierwszego nie mniejszego niż podany elementu
- int upper_bound (const T &item) const noexcept

  funkcja zwracająca indeks pierwszego nie większego niż podany elementu

Dodatkowe Dziedziczone Składowe

5.6.1 Dokumentacja funkcji składowych

5.6.1.1 insert()

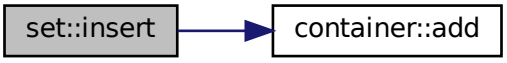
```
template<class T >
void set< T >::insert (
            const T & item )
```

funkcja dodająca element do kontenera

Parametry

item	element do dodania

Oto graf wywołań dla tej funkcji:



5.6.1.2 lower_bound()

```
template<class T >
int set< T >::lower_bound (
            const T & item ) const  [noexcept]
```

funkcja zwracająca indeks pierwszego nie mniejszego niż podany elementu

Parametry

item	element do sprawdzenia

Zwraca

int indeks znalezionego elementu, -1 jeżeli nie znaleziono elementu

Oto graf wywołań dla tej funkcji:

5.6.1.3 upper_bound()

```
template<class T >
int set< T >::upper_bound (
            const T & item ) const  [noexcept]
```

funkcja zwracająca indeks pierwszego nie większego niż podany elementu

Parametry

item	element do sprawdzenia

Zwraca

int indeks znalezionego elementu, -1 jeżeli nie znaleziono elementu

Oto graf wywołań dla tej funkcji:

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/set.h

5.7 Dokumentacja szablonu klasy stack< T >

`#include <stack.h>`

Diagram dziedziczenia dla stack< T >

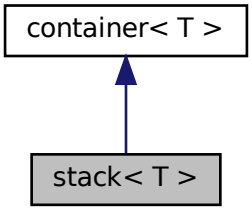
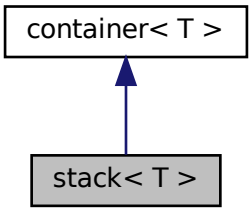


Diagram współpracy dla stack< T >:



Metody publiczne

- void insert (const T &item)

  funkcja wstawiająca element na stos
- T pop ()

  funkcja zwracająca element z góry stosu

Dodatkowe Dziedziczone Składowe

5.7.1 Dokumentacja funkcji składowych

5.7.1.1 insert()

```
template<class T >
void stack< T >::insert (
            const T & item )  [inline]
```

funkcja wstawiająca element na stos

Parametry

item	element do wstawienia

Oto graf wywołań dla tej funkcji:

5.7.1.2 pop()

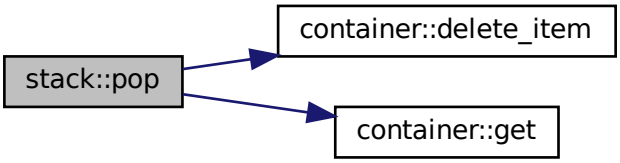
```
template<class T >
T stack< T >::pop  [inline]
```

funkcja zwracająca element z góry stosu

Zwraca

T pierwszy element na stosie

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/stack.h

5.8 Dokumentacja szablonu klasy vector< T >

`#include <vector.h>`

Diagram dziedziczenia dla vector< T >

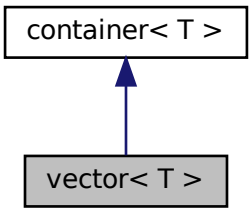
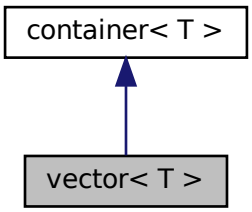


Diagram współpracy dla vector< T >:



Metody publiczne

- void push_back (const T &item)

  funkcja dodająca element na koniec vectora
- void push_front (const T &item)

  funkcja dodająca element na początek vectora
- T pop_back ()

  funkcja usuwająca element na końcu i zwracająca go
- T pop_front ()

  funkcja usuwająca element na początku i zwracająca go
- void swap (const size_t index1, const size_t index2)

  funkcja zamieniająca miejscami dwa elementy
- void sort (const bool ascendingOrder=true)

  funkcja sortująca vector

Dodatkowe Dziedziczone Składowe

5.8.1 Dokumentacja funkcji składowych

5.8.1.1 pop_back()

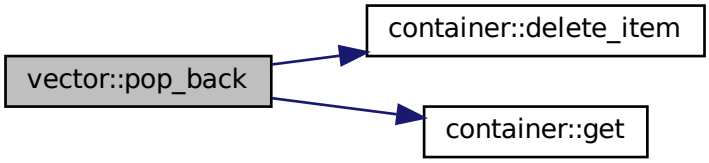
```
template<class T >
T vector< T >::pop_back
```

funkcja usuwająca element na końcu i zwracająca go

Zwraca

T ostatni element

Oto graf wywołań dla tej funkcji:



5.8.1.2 pop_front()

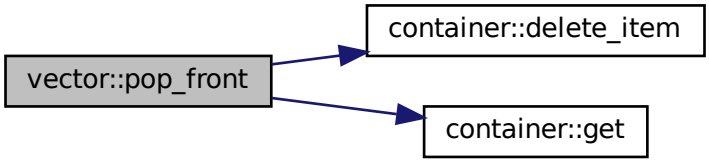
```
template<class T >
T vector< T >::pop_front
```

funkcja usuwająca element na początku i zwracająca go

Zwraca

T pierwszy element

Oto graf wywołań dla tej funkcji:



5.8.1.3 push_back()

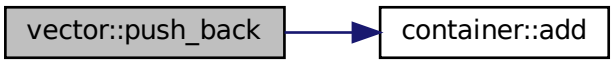
```
template<class T >
void vector< T >::push_back (
            const T & item )
```

funkcja dodająca element na koniec vectora

Parametry

item	referencja na dodawany element, poźniej tworzona jest kopia, więc tutaj warto przekazać referencję

Oto graf wywołań dla tej funkcji:



5.8.1.4 push_front()

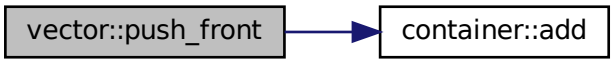
```
template<class T >
void vector< T >::push_front (
            const T & item )
```

funkcja dodająca element na początek vectora

Parametry

item	referencja na dodawany element, później tworzona jest kopia, więc tutaj warto przekazać referencję

Oto graf wywołań dla tej funkcji:



5.8.1.5 sort()

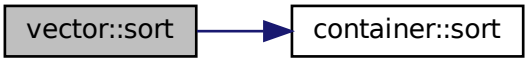
```
template<class T >
void vector< T >::sort (
            const bool ascendingOrder = true )
```

funkcja sortująca vector

Parametry

ascendingOrder	wybór porządku rosnącego lub

Oto graf wywołań dla tej funkcji:



5.8.1.6 swap()

```
template<class T >
void vector< T >::swap (
```

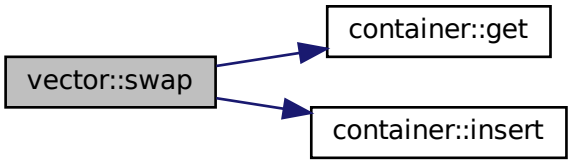
```
const size_t index1,
const size_t index2 )
```

funkcja zamieniająca miejscami dwa elementy

Parametry

index1	indeks pierwszego elementu
index2	indeks drugiego elementu

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/vector.h

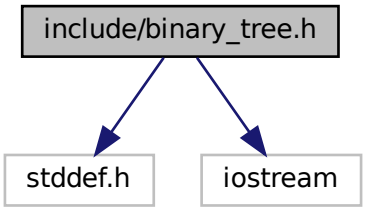
Rozdział 6

Dokumentacja plików

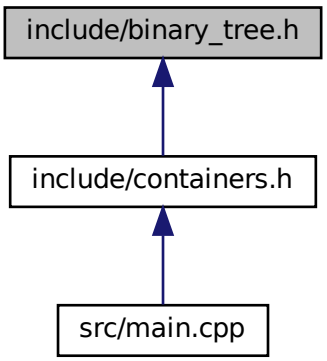
6.1 Dokumentacja pliku include/binary_tree.h

```
#include <stddef.h>
#include <iostream>
```

Wykres zależności załączania dla binary_tree.h:



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



Komponenty

- class binary_tree< T >

6.1.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

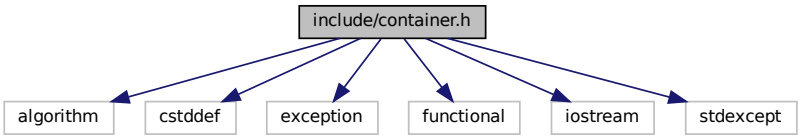
Copyright

Copyright (c) 2020

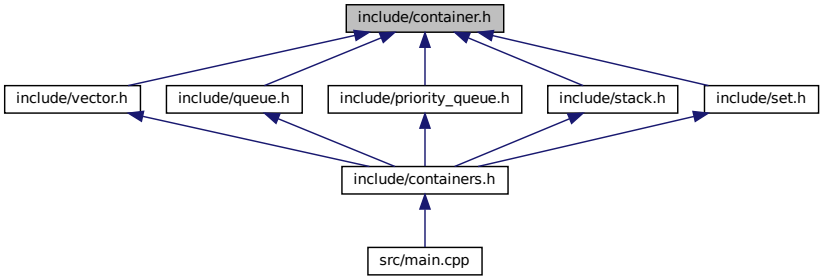
6.2 Dokumentacja pliku include/container.h

```
#include <algorithm>
#include <cstddef>
#include <exception>
#include <functional>
#include <iostream>
#include <stdexcept>
```

Wykres zależności załączania dla container.h:



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



Komponenty

- class container< T >

  bazowa klasa zarządzająca pamięcią, implementująca podstawowe funkcjonalności itd.

Definicje

- #define START_SIZE 32
- #define RESIZE_STEP 64

6.2.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

Copyright

6.2.2 Dokumentacja definicji

6.2.2.1 RESIZE_STEP

```
#define RESIZE_STEP 64
```

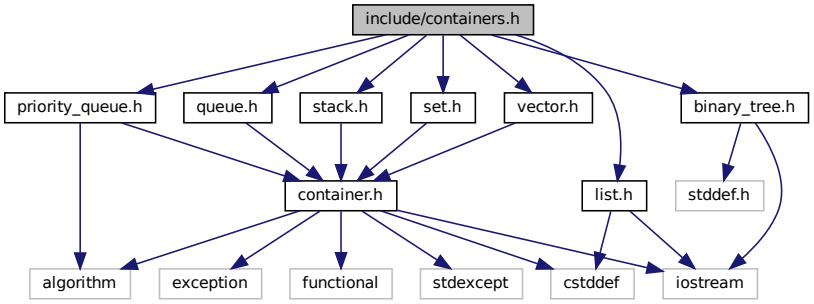
6.2.2.2 START_SIZE

```
#define START_SIZE 32
```

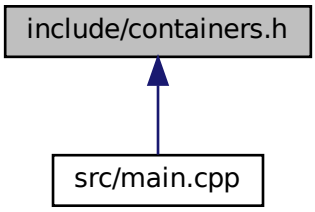
6.3 Dokumentacja pliku include/containers.h

```
#include "vector.h"
#include "queue.h"
#include "priority_queue.h"
#include "stack.h"
#include "set.h"
#include "binary_tree.h"
#include "list.h"
```

Wykres zależności załączania dla containers.h:



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



6.3.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

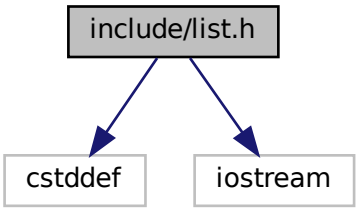
Copyright

Copyright (c) 2020

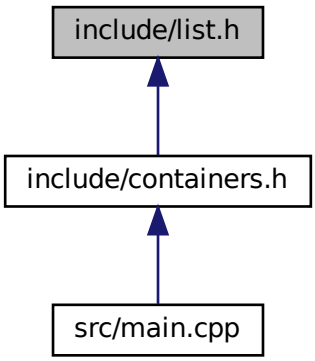
6.4 Dokumentacja pliku include/list.h

```
#include <cstddef>
#include <iostream>
```

Wykres zależności załączania dla list.h:



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



Komponenty

- class list< T >

6.4.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

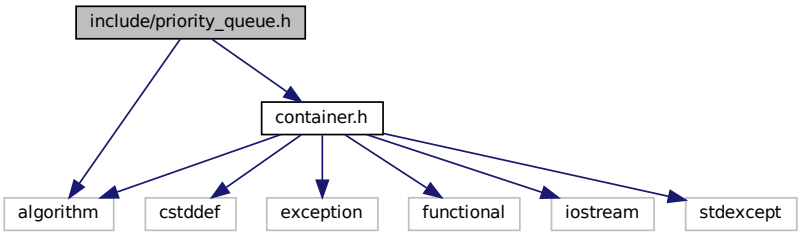
2020-11-02

Copyright

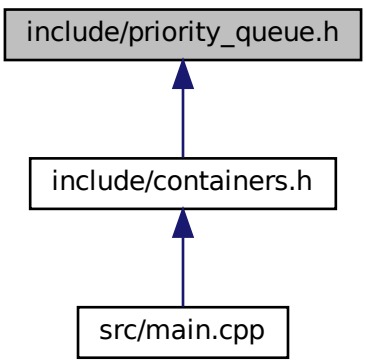
6.5 Dokumentacja pliku include/priority_queue.h

```
#include "container.h"
#include <algorithm>
```

Wykres zależności załączania dla priority_queue.h:



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



Komponenty

- class priority_queue< T >

Definicje typów

- typedef enum priorityType priorityType

Wyliczenia

- enum priorityType { MIN, MAX }

6.5.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

Copyright

Copyright (c) 2020

6.5.2 Dokumentacja definicji typów

6.5.2.1 priorityType

```
typedef enum priorityType priorityType
```

6.5.3 Dokumentacja typów wyliczanych

6.5.3.1 priorityType

```
enum priorityType
```

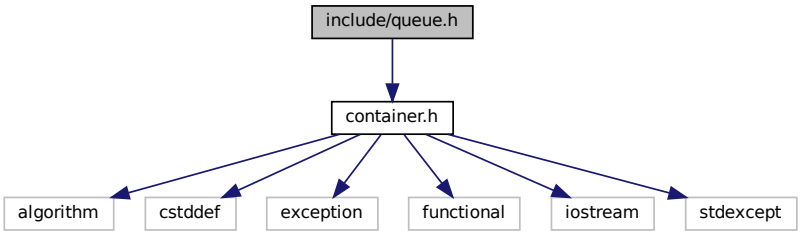
Wartości wyliczeń

MIN	
MAX	

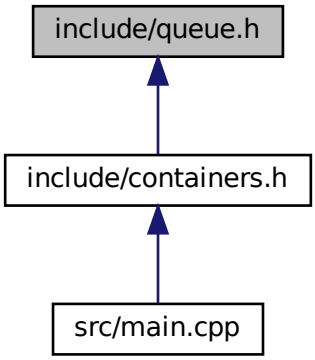
6.6 Dokumentacja pliku include/queue.h

```
#include "container.h"
```

Wykres zależności załączania dla queue.h:



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



Komponenty

- class queue< T >

6.6.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

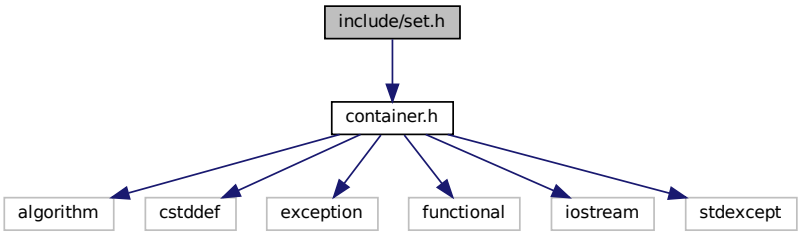
Copyright

Copyright (c) 2020

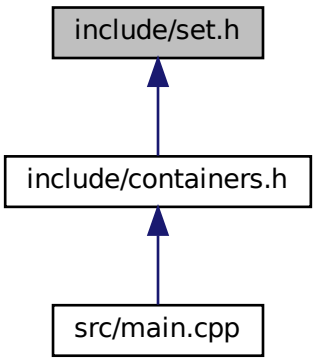
6.7 Dokumentacja pliku include/set.h

```
#include "container.h"
```

Wykres zależności załączania dla set.h:



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



Komponenty

- class set< T >

6.7.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

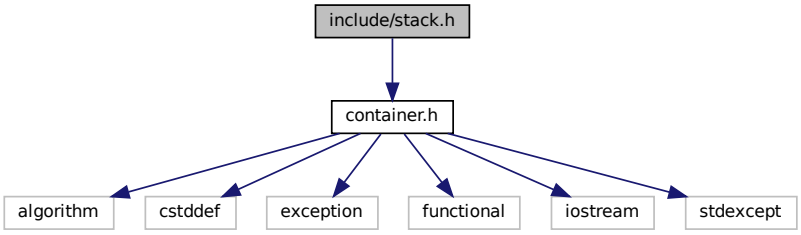
Copyright

Copyright (c) 2020

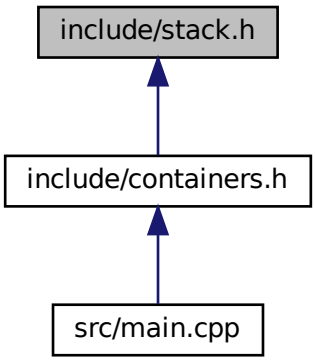
6.8 Dokumentacja pliku include/stack.h

```
#include "container.h"
```

Wykres zależności załączania dla stack.h:



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



Komponenty

- class stack< T >

6.8.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

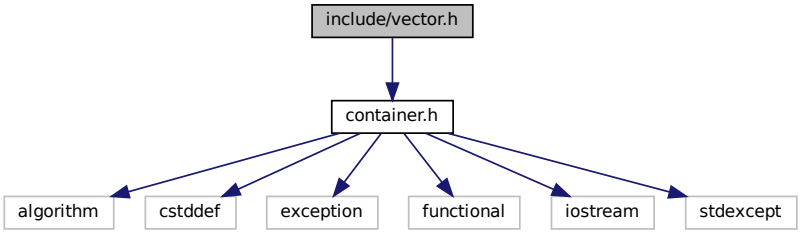
Copyright

Copyright (c) 2020

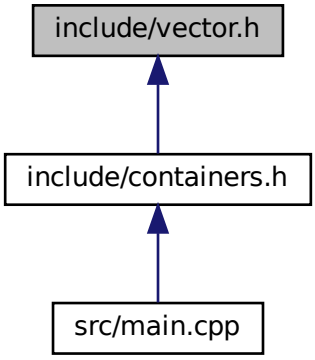
6.9 Dokumentacja pliku include/vector.h

```
#include "container.h"
```

Wykres zależności załączania dla vector.h:



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



Komponenty

- class vector< T >

6.9.1 Opis szczegółowy

Autor

Wojciech Janota

Wersja

0.1

Data

2020-11-02

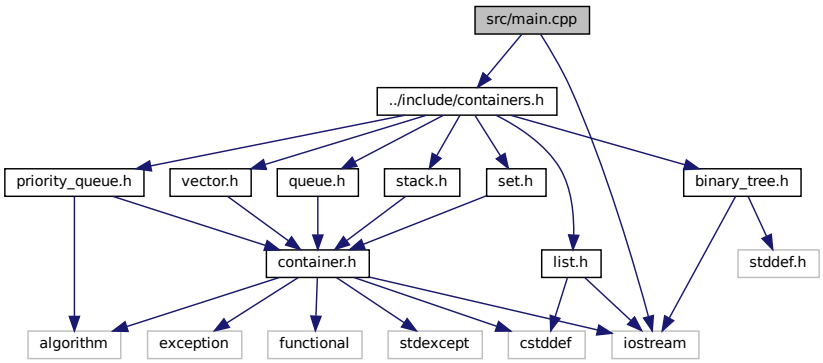
Copyright

6.10 Dokumentacja pliku README.md

6.11 Dokumentacja pliku src/main.cpp

```
#include "../include/containers.h"
#include <iostream>
```

Wykres zależności załączania dla main.cpp:



Funkcje

- int main ()

6.11.1 Dokumentacja funkcji

6.11.1.1 main()

```
int main ( )
```

6.12 Dokumentacja pliku valgrind-out.txt

Zmienne

- a memory error detector ==61528== Copyright (C) 2002-2017
- a memory error and GNU GPL d
- a memory error and GNU GPL by Julian Seward et al ==61528== Using Valgrind-3.16.1-36d6727e1d-20200622X and LibVEX
- rerun with h for copyright info
- rerun with h for copyright LittleEndian
- rerun with h for copyright amd64 cx16 lzcnt rdtscp sse3 ssse3 avx avx2 bmi f16c rdrand Page sizes
- rerun with h for copyright amd64 cx16 lzcnt rdtscp sse3 ssse3 avx avx2 bmi f16c rdrand Page max supported Valgrind library directory
- rerun with h for copyright amd64 cx16 lzcnt rdtscp sse3 ssse3 avx avx2 bmi f16c rdrand Page max supported Valgrind library unless you know exactly what you re doing

6.12.1 Dokumentacja zmiennych

6.12.1.1 al

```
a memory error and GNU GPL by Julian Seward et al ==61528== Using Valgrind-3.16.1-36d6727e1d-20200622X
and LibVEX
```

6.12.1.2 d

```
a memory error and GNU GPL d
```

6.12.1.3 detector

```
a memory error detector ==61528== Copyright (C) 2002-2017
```

6.12.1.4 directory

```
rerun with h for copyright amd64 cx16 lzcnt rdtscp sse3 ssse3 avx avx2 bmi f16c rdrand Page
max supported Valgrind library directory
```

6.12.1.5 doing

```
rerun with h for copyright amd64 cx16 lzcnt rdtscp sse3 ssse3 avx avx2 bmi f16c rdrand Page
max supported Valgrind library unless you know exactly what you re doing
```

6.12.1.6 info

```
rerun with h for copyright info
```

Wartość początkowa:

```
==61528== Command: bin/main
==61528== Parent PID: 59586
==61528==
--61528--
--61528-- Valgrind options:
--61528--    --leak-check=full
--61528--    --show-leak-kinds=all
--61528--    --track-origins=yes
--61528--    --verbose
--61528--    --log-file=valgrind-out.txt
--61528-- Contents of /proc/version:
--61528--   Linux version 5.8.0-25-generic (buildd@lcy01-amd64-022) (gcc (Ubuntu 10.2.0-13ubuntu1) 10.2.0,
       GNU ld (GNU Binutils for Ubuntu) 2.35.1) #26-Ubuntu SMP Thu Oct 15 10:30:38 UTC 2020
--61528--
--61528-- Arch and hwcaps: AMD64
```

6.12.1.7 LittleEndian

```
rerun with h for copyright LittleEndian
```

6.12.1.8 sizes

```
rerun with h for copyright amd64 cx16 lzcnt rdtscp sse3 ssse3 avx avx2 bmi f16c rdrand Page
sizes
```

Indeks