Test struktur danych

Wygenerowano przez Doxygen 1.8.6

Cz, 26 mar 2015 11:49:23

Spis treści

Rozdział 1

Strona główna

Czas wykonywania algorytmu wykonującego dodawanie elementów do podatawowych struktour danych

Program realizuje operacje dodawania n liczby elementów do listy, kolejki i stosu i mierzy czas tych operacji

Autor

Mateusz Bencer

Data

2015.03.19

Wersja

1.0

Mail:

209360@pwr.wroc.edu.pl

test

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 udokumentowanych plików z ich krótkimi opisami:

Rozdział 5

Dokumentacja klas

5.1 Dokumentacja szablonu klasy Benchmark< T >

Metody publiczne

- **Benchmark** (unsigned int testPower)
- void **testList** (ArrayImplementation::List< T > *list, ArrayImplementation::Increase inc)
- void **testQueue** (ArrayImplementation::Queue< T > *queue, ArrayImplementation::Increase inc)
- void **testStack** (ArrayImplementation::Stack< T > *stack, ArrayImplementation::Increase inc)
- void **testList** (ListImplementation::List< T > *list)
- void **testQueue** (ListImplementation::Queue< T > *queue)
- void **testStack** (ListImplementation::Stack< T > *stack)

Dokumentacja dla tej klasy została wygenerowana z pliku:

- inc/Benchmark.h

5.2 Dokumentacja szablonu klasy Element< T >

Klasa reprezentująca abstrakcyjny "pojemnik na dane".

`#include <Element.h>`

Metody publiczne

- Element (T *data)

  Konstruktor zapamiętujący adres przechowywanego obiektu.
- void setData (const T *data)

  setter do przechowywanej danej
- T * getData () const

  getter do przechowywanej danej
- Element< T > * next () const

  zwraca wskaźnik do kolejnego elementu na liście
- Element< T > * prev () const

  zwraca wskaźnik do poprzedniego elementu na liście
- void setNext (Element< T > *next)

  ustawia kolejny element listy
- void setPrev (Element< T > *prev)

ustawia poprzedni element listy

- ∼Element ()

  zwalnianie elementu przechowywanego przez klase Element

5.2.1 Opis szczegółowy

template<class T>class Element< T >

Klasa reprezentująca abstrakcyjny "pojemnik na dane".

Klasa może przechowywać dane zdeterminowane przez typ szablonu, zawiera wskaźnik do kolejnego i poprzedniego elementu (implementacja elementu listy).

5.2.2 Dokumentacja konstruktora i destruktora

5.2.2.1 template<typename T > Element< T >::Element (T * *data*)

Konstruktor zapamiętujący adres przechowywanego obiektu.

Parametry

data	obiekt/zmienna do przechowania

5.2.2.2 template<typename T > Element< T >::∼Element ( )

zwalnianie elementu przechowywanego przez klase Element

Destruktor zapewniający zwalnianie elementu przechowywanego przez klase Element

5.2.3 Dokumentacja funkcji składowych

5.2.3.1 template<typename T > T * Element< T >::getData ( ) const

getter do przechowywanej danej

Metoda do pobrania wskaźnka przechowywanego danej

Zwraca

wskaźnik do zmiennej przechowywanej przez klase

5.2.3.2 template<typename T > Element< T > * Element< T >::next ( ) const

zwraca wskaźnik do kolejnego elementu na liście

Zwraca

wskaźnik do kolejnego elementu na liście

5.2.3.3 template<typename T > Element< T > * Element< T >::prev ( ) const

zwraca wskaźnik do poprzedniego elementu na liście

Zwraca

wskaźnik do poprzedniego elementu na liście

5.2.3.4 template<typename T > void Element< T >::setData (const T * *data*)

setter do przechowywanej danej

Metoda do ustawiania danej przechowywanej przez klase

obiekt/zmięnna, która będzie przechowywana

5.2.3.5 template<typename T > void Element< T >::setNext (Element< T > * *next*)

ustawia kolejny element listy

Parametry

next	kolejny element listy

5.2.3.6 template<typename T > void Element< T >::setPrev (Element< T > * *prev*)

ustawia poprzedni element listy

Parametry

prev	poprzedni element listy

Dokumentacja dla tej klasy została wygenerowana z pliku:

- inc/Element.h

5.3 Dokumentacja szablonu klasy ArrayImplementation::List< T >

Klasa reprezentująca podstawy konterner danych - Listę zaimplementowaną na tablicy.

`#include <List.h>`

Metody publiczne

- List ()
  Konstruktor zerujący aktualny rozmiar i przydzielający pamięć tablicy przechowującej dane w ilości _MAX_SIZE.
- List (unsigned int max_size)
  Konstruktor zerujący rozmiar i przydzielający pamięć tablicy przechowującej dane w ilości określonej przez parametr max_size.
- T * pop (unsigned int index)
  zwraca element o indeksie określonej przez parametr index (równocześnie usuwając go z listy)
- void push (T *elem, unsigned int index, Increase inc)
  dodaje element o indkesie index do listy
- unsigned int size ()
  zwraca aktualny rozmiar listy
- unsigned short isEmpty ()
  zwraca 1, gdy lista jest pusta. 0, gdy są jakieś elementy
- ~List ()
  destruktor czyszczący liste

5.3.1 Opis szczegółowy

template<class T>class ArrayImplementation::List< T >

Klasa reprezentująca podstawy konterner danych - Listę zaimplementowaną na tablicy.

Klasa reprezentująca podstawy kontener - listę. Jest to implemetacja listy oparta na tablicy.

5.3.2 Dokumentacja konstruktora i destruktora

5.3.2.1 template<typename T > ArrayImplementation::List< T >::List (unsigned int *max_size*)

Konstruktor zerujący rozmiar i przydzielający pamięć tablicy przechowującej dane w ilości określonej przez parametr max_size.

Parametry

max_size	maksymalna liczba elementów w liście

5.3.2.2 template<typename T > ArrayImplementation::List< T >::~List ( )

destruktor czyszczący liste

Destruktor usuwawa wszystkie elementy z listy

5.3.3 Dokumentacja funkcji składowych

5.3.3.1 template<typename T > unsigned short ArrayImplementation::List< T >::isEmpty ( )

zwraca 1, gdy lista jest pusta. 0, gdy są jakieś elementy

Zwraca

zwraca informacje, czy w liście są jakieś elementy

5.3.3.2 template<typename T > T * ArrayImplementation::List< T >::pop (unsigned int *index*)

zwraca element o indeksie określonej przez parametr index (równocześnie usuwając go z listy)

Parametry

index	określa indeks elementu znajdującego się na liście, który zostanie zwrócony

Zwraca

element listy o pozycji index

5.3.3.3 template<typename T > void ArrayImplementation::List< T >::push (T * *elem,* unsigned int *index,* Increase *inc*)

dodaje element o indkesie index do listy

Warunek wstępny

Indeksowanie zaczyna się od 1

Parametry

elem	element umieszczany do listy
index	określa pozycję na liście dodawanego elementu (numeracja od 1!)
inc	określa sposób powiększania się listy w razie braku miejsca

5.3.3.4 template<typename T > unsigned int ArrayImplementation::List< T >::size ( )

zwraca aktualny rozmiar listy

Zwraca

rozmiar listy

Dokumentacja dla tej klasy została wygenerowana z pliku:

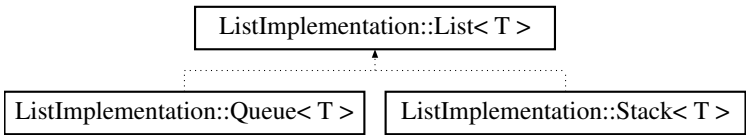
- inc/ArrayImplementation/List.h

5.4 Dokumentacja szablonu klasy ListImplementation::List< T >

Klasa reprezentująca podstawy konterner danych z którego korzystają inne - Listę

`#include <List.h>`

Diagram dziedziczenia dla ListImplementation::List< T >



Metody publiczne

- List ()
  Konstruktor zerujący pola klasy i przydzielający pamięć na _head i _tail.
- Element< T > * pop (Direction dir)
  zwraca element z początku(zależy od użytej struktury danych) listy
- void push (Element< T > *elem, Direction dir)
  dodaje element na początek/koniec(zależy od implementacji) listy
- unsigned int size ()
  zwraca rozmiar użytej struktury danych
- unsigned short isEmpty ()
  zwraca 1, gdy kontener jest pusty, 0 - gdy jest już jakiś element
- virtual ~List ()
  wirtualny destruktor czyszczący liste

5.4.1 Opis szczegółowy

template<class T>class ListImplementation::List< T >

Klasa reprezentująca podstawy konterner danych z którego korzystają inne - Listę

Klasa reprezentująca podstawy kontener - listę. Jest to podstawowa implementacja listy stanowiąca klase bazową dla listy, stosu i kolejki

5.4.2 Dokumentacja konstruktora i destruktora

5.4.2.1 template<typename T > ListImplementation::List< T >::~List ( ) [virtual]

wirtualny destruktor czyszczący liste

Destruktor usuwawa wszystkie elementy z listy

5.4.3 Dokumentacja funkcji składowych

5.4.3.1 template<typename T > unsigned short ListImplementation::List< T >::isEmpty ( )

zwraca 1, gdy kontener jest pusty, 0 - gdy jest już jakiś element

Zwraca

zwraca informacje, czy w kontenerze są już jakieś elementy

5.4.3.2 template<typename T > Element< T > * ListImplementation::List< T >::pop (Direction *dir*)

zwraca element z początku(zależy od użytej struktury danych) listy

Parametry

dir	określa czy zdjąć element z początku (Front), czy z końca (Back) listy

Zwraca

element będący na początku/końcu listy

5.4.3.3 template<typename T > void ListImplementation::List< T >::push (Element< T > * *elem,* Direction *dir*)

dodaje element na początek/koniec(zależy od implementacji) listy

Parametry

elem	element umieszczany na początku/końcu listy
dir	określa czy włożyć element na początek(Front), na koniec (Back) listy

5.4.3.4 template<typename T > unsigned int ListImplementation::List< T >::size ( )

zwraca rozmiar użytej struktury danych

Zwraca

rozmiar użytej struktury danych

Dokumentacja dla tej klasy została wygenerowana z pliku:

- inc/LinkedListImplementation/List.h

5.5 Dokumentacja szablonu klasy ArrayImplementation::Queue< T >

Klasa reprezentująca podstawy konterner danych kolejke zaimplementowaną na tablicy.

`#include <Queue.h>`

Metody publiczne

- Queue ()

  Konstruktor alokujący pamięć na kolejkę oraz zerujący indeks elementu na końcu kolejki.
- Queue (unsigned int max_size)

  Konstruktor alokujący pamięć na kolejkę w ilości max_size oraz zerujący indeks elementu na końcu kolejki.
- T * pop ()

  zwraca element z początku kolejki
- void push (T *elem, Increase inc)

  dodaje element na koniec kolejki
- ~Queue ()

  Destruktor usuwa tablicę przechowującą elementy kolejki.

5.5.1 Opis szczegółowy

template<class T>class ArrayImplementation::Queue< T >

Klasa reprezentująca podstawy konterner danych kolejke zaimplementowaną na tablicy.

Kolejka jest strukturą danych typu typu FIFO, First In, First Out; pierwszy na wejściu, pierwszy na wyjściu

5.5.2 Dokumentacja konstruktora i destruktora

5.5.2.1 template<typename T > ArrayImplementation::Queue< T >::Queue (unsigned int *max_size*)

Konstruktor alokujący pamięć na kolejkę w ilości max_size oraz zerujący indeks elementu na końcu kolejki.

Parametry

max_size	maksymalna liczba elementów w kolejce

5.5.3 Dokumentacja funkcji składowych

5.5.3.1 template<typename T > T * ArrayImplementation::Queue< T >::pop ( )

zwraca element z początku kolejki

Zwraca

element będący na początku kolejki

5.5.3.2 template<typename T > void ArrayImplementation::Queue< T >::push (T * *elem*, Increase *inc*)

dodaje element na koniec kolejki

Parametry

elem	element umieszczany na końcu kolejki
inc	określa sposób powiększania się kolejki w razie braku miejsca

Dokumentacja dla tej klasy została wygenerowana z pliku:

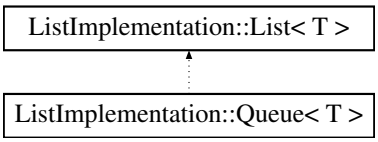
- inc/ArrayImplementation/Queue.h

5.6 Dokumentacja szablonu klasy ListImplementation::Queue< T >

Klasa reprezentująca podstawy konterner danych kolejke.

`#include <Queue.h>`

Diagram dziedziczenia dla ListImplementation::Queue< T >



Metody publiczne

- Queue ()

  Konstruktor wywołujący konstruktor klasy bazowej List.
- Element< T > * pop ()

  zwraca element z początku kolejki
- void push (Element< T > *elem)

  dodaje element na koniec kolejki
- virtual ~Queue ()

  Wywołuje destruktor klasy bazowej.

5.6.1 Opis szczegółowy

template<class T>class ListImplementation::Queue< T >

Klasa reprezentująca podstawy konterner danych kolejke.

Kolejka jest strukturą danych typu typu FIFO, First In, First Out; pierwszy na wejściu, pierwszy na wyjściu

5.6.2 Dokumentacja funkcji składowych

5.6.2.1 template<typename T > Element< T > * ListImplementation::Queue< T >::pop ( )

zwraca element z początku kolejki

Zwraca

element będący na początku kolejki

5.6.2.2 template<typename T > void ListImplementation::Queue< T >::push (Element< T > * *elem*)

dodaje element na koniec kolejki

Parametry

elem	element umieszczany na końcu

Dokumentacja dla tej klasy została wygenerowana z pliku:

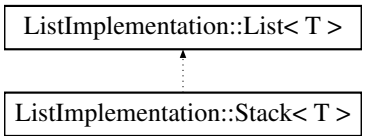
- inc/LinkedListImplementation/Queue.h

5.7 Dokumentacja szablonu klasy ListImplementation::Stack< T >

Klasa reprezentująca podstawy konterner danych - stos.

`#include <Stack.h>`

Diagram dziedziczenia dla ListImplementation::Stack< T >



Metody publiczne

- Stack ()

  Konstruktor wywołujący konstruktor klasy bazowej List.
- Element< T > * pop ()

  zwraca element z wierzchu stosu
- void push (Element< T > *elem)

  dodaje element na wierzch
- virtual ~Stack ()

  Wywołuje destruktor klasy bazowej.

5.7.1 Opis szczegółowy

template<class T>class ListImplementation::Stack< T >

Klasa reprezentująca podstawy konterner danych - stos.

Kolejka jest strukturą danych typu typu LIFO, Last In, First Out; ostatni na wejściu, pierwszy na wyjściu

5.7.2 Dokumentacja funkcji składowych

5.7.2.1 template<typename T > Element< T > * ListImplementation::Stack< T >::pop ( )

zwraca element z wierzchu stosu

Zwraca

element będący na wierzchu stosu

5.7.2.2 template<typename T > void ListImplementation::Stack< T >::push (Element< T > * *elem*)

dodaje element na wierzch

Parametry

elem	element, który zostanie umieszczony na wierzchu stosu

Dokumentacja dla tej klasy została wygenerowana z pliku:

- inc/LinkedListImplementation/Stack.h

5.8 Dokumentacja szablonu klasy ArrayImplementation::Stack< T >

Klasa reprezentująca podstawy konterner danych - stos w implementacji tablicowej.

`#include <Stack.h>`

Metody publiczne

- Stack ()

  Konstruktor przydzielający pamieć na stos w ilości _MAX_SIZE oraz zerujący szczyt stosu.
- Stack (int max_size)

  Konstruktor przydzielający pamieć na stos w ilości max_size oraz zerujący szczyt stosu.
- Element< T > * pop ()

  zwraca element z wierzchu stosu
- void push (T *elem, Increase inc)

  dodaje element na wierzch
- int size ()

  zwraca aktualny rozmiar stosu
- unsigned short isEmpty ()

  zwraca 1, gdy stos jest pusty. 0, gdy są jakieś elementy
- ~Stack ()

  Destruktor usuwa pamięc po tablicy _elements.

5.8.1 Opis szczegółowy

template<class T>class ArrayImplementation::Stack< T >

Klasa reprezentująca podstawy konterner danych - stos w implementacji tablicowej.

Stos jest strukturą danych typu typu FIFO, Frist In, First Out; pierwszy na wejściu, pierwszy na wyjściu

5.8.2 Dokumentacja konstruktora i destruktora

5.8.2.1 template<typename T > ArrayImplementation::Stack< T >::Stack (int *max_size*)

Konstruktor przydzielający pamieć na stos w ilości max_size oraz zerujący szczyt stosu.

Parametry

max_size	maksymalna liczba elementów na stosie

5.8.3 Dokumentacja funkcji składowych

5.8.3.1 template<typename T > unsigned short ArrayImplementation::Stack< T >::isEmpty ( )

zwraca 1, gdy stos jest pusty. 0, gdy są jakieś elementy

Zwraca

zwraca informacje, czy na stosie są jakieś elementy

5.8.3.2 template<typename T > Element< T > * ArrayImplementation::Stack< T >::pop ( )

zwraca element z wierzchu stosu

Zwraca

element będący na wierzchu stosu

5.8.3.3 template<typename T > void ArrayImplementation::Stack< T >::push (T * *elem*, Increase *inc*)

dodaje element na wierzch

Parametry

elem	element, który zostanie umieszczony na wierzchu stosu
inc	określa sposób powiększania się stosu w razie braku miejsca

5.8.3.4 template<typename T > int ArrayImplementation::Stack< T >::size ( )

zwraca aktualny rozmiar stosu

Zwraca

rozmiar stosu

Dokumentacja dla tej klasy została wygenerowana z pliku:

- inc/ArrayImplementation/Stack.h

5.9 Dokumentacja klasy Timer

Klasa do pomiaru różnicy czasów.

`#include <Timer.h>`

Metody publiczne

- Timer ()

  Konstruktor zerujący parametry.
- void startTimer ()

  Zmierzenie czasu rozpoczęcia pomiaru.
- void stopTimer ()

  Zmierzenie czasu zakończenia pomiaru.
- double diffTimeMs ()

  Funkcja zwracająca różnice czasu pomiędzy czasem rozpoczęcia i zakończenia pomiaru.

5.9.1 Opis szczegółowy

Klasa do pomiaru różnicy czasów.

Klasa pozwala na pomiar czasów w danych momentach oraz na zwrócenie czasu, który upłynał pomiędzy tymi momentami

5.9.2 Dokumentacja konstruktora i destruktora

5.9.2.1 Timer::Timer ( )

Konstruktor zerujący parametry.

Konstruktor ten odpowiada za zerowania zmiennych startu i stopu w celu możliwości późniejszego sprawdzenia, czy pomiary czasu konieczne do wyznaczenia różnicy zostały zrealizowane.

5.9.3 Dokumentacja funkcji składowych

5.9.3.1 double Timer::diffTimeMs ( )

Funkcja zwracająca różnice czasu pomiędzy czasem rozpoczęcia i zakończenia pomiaru.

Różnica czasu zwracana jest w milisekundach.

Warunek wstępny

Czas zkończenia pomiaru musi być większy (późniejszy) od czasu jego rozpoczęcia

Zwraca

Zwracana jest różnica czasu zrzutowana do typu double

5.9.3.2 void Timer::startTimer ( )

Zmierzenie czasu rozpoczęcia pomiaru.

Funkcja zapamiętuje bierzący czas, jako czas rozpoczęcia pomiaru.

5.9.3.3 void Timer::stopTimer ( )

Zmierzenie czasu zakończenia pomiaru.

Funkcja zapamiętuje bierzący czas, jako czas zakończenia pomiaru.

Dokumentacja dla tej klasy została wygenerowana z plików:

- inc/Timer.h
- src/Timer.cpp

Rozdział 6

Dokumentacja plików

6.1 Dokumentacja pliku inc/ArrayImplementation/List.h

Deklaracja klasy Lista (w implementacji opartej na tablicy)

```
#include "../Element.h"
#include "Increase.h"
#include <iostream>
```

Komponenty

- class ArrayImplementation::List< T >

  Klasa reprezentująca podstawy konterner danych - Listę zaimplementowaną na tablicy.

Definicje

- #define **DEFAULT_MAX_SIZE** 1

6.1.1 Opis szczegółowy

Deklaracja klasy Lista (w implementacji opartej na tablicy) List.h

6.2 Dokumentacja pliku inc/LinkedListImplementation/List.h

Deklaracja klasy List.

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

Komponenty

- class ListImplementation::List< T >

  Klasa reprezentująca podstawy konterner danych z którego korzystają inne - Listę

Wyliczenia

- enum **Direction** { **Front**, **Back** }

  Enumerator przekazywany do funckji push w celu określenia, czy umieszczamy element na początku lub na końcu listy [na początku (Front), czy na końcu (Back), None - domyślne dla kontenera].

6.2.1 Opis szczegółowy

Deklaracja klasy List. List.h

6.3 Dokumentacja pliku inc/ArrayImplementation/Queue.h

Deklaracja i definicja klasy Queue.

```
#include "../Element.h"
#include "Increase.h"
```

Komponenty

- class ArrayImplementation::Queue< T >

  Klasa reprezentująca podstawy konterner danych kolejke zaimplementowaną na tablicy.

Definicje

- #define **DEFAULT_MAX_SIZE** 1

6.3.1 Opis szczegółowy

Deklaracja i definicja klasy Queue. Queue.h

6.4 Dokumentacja pliku inc/LinkedListImplementation/Queue.h

Deklaracja i definicja klasy Queue.

```
#include "List.h"
```

Komponenty

- class ListImplementation::Queue< T >

  Klasa reprezentująca podstawy konterner danych kolejke.

6.4.1 Opis szczegółowy

Deklaracja i definicja klasy Queue. Queue.h

6.5 Dokumentacja pliku inc/ArrayImplementation/Stack.h

Deklaracja i definicja klasy Stack w wersji tablicowej.

```
#include "List.h"
#include "../Element.h"
#include "Increase.h"
```

Komponenty

- class ArrayImplementation::Stack< T >

  Klasa reprezentująca podstawy konterner danych - stos w implementacji tablicowej.

Definicje

- #define **DEFAULT_MAX_SIZE** 1

6.5.1 Opis szczegółowy

Deklaracja i definicja klasy Stack w wersji tablicowej. Stack.h

6.6 Dokumentacja pliku inc/LinkedListImplementation/Stack.h

Deklaracja i definicja klasy Stack.

```
#include "List.h"
```

Komponenty

- class ListImplementation::Stack< T >

  Klasa reprezentująca podstawy konterner danych - stos.

6.6.1 Opis szczegółowy

Deklaracja i definicja klasy Stack. Stack.h

6.7 Dokumentacja pliku inc/Element.h

Deklaracja i definicja klasy Element.

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

Komponenty

- class Element< T >

  Klasa reprezentująca abstrakcyjny "pojemnik na dane".

6.7.1 Opis szczegółowy

Deklaracja i definicja klasy Element. Element.h

6.8 Dokumentacja pliku inc/Timer.h

Plik zawierający deklaracje klasy Timer służącej do pomiaru różnicy czasów.

```
#include <ctime>
```

Komponenty

- class Timer

  Klasa do pomiaru różnicy czasów.

6.8.1 Opis szczegółowy

Plik zawierający deklaracje klasy Timer służącej do pomiaru różnicy czasów. Timer.h

6.9 Dokumentacja pliku src/main.cpp

Plik zawierający sekwencje operacji do mierzenia czasu operacji mnożenia elementów tablicy przez 2.

```
#include <iostream>
#include "../inc/LinkedListImplementation/List.h"
#include "../inc/LinkedListImplementation/Queue.h"
#include "../inc/LinkedListImplementation/Stack.h"
#include "../inc/ArrayImplementation/List.h"
#include "../inc/ArrayImplementation/Queue.h"
#include "../inc/ArrayImplementation/Stack.h"
#include "../inc/Benchmark.h"
```

Funkcje

- int **main** ()

6.9.1 Opis szczegółowy

Plik zawierający sekwencje operacji do mierzenia czasu operacji mnożenia elementów tablicy przez 2. main.cpp

6.10 Dokumentacja pliku src/Timer.cpp

Plik zawierający definicje funkcji klasy Timer służącej do pomiaru różnicy czasów.

```
#include "../inc/Timer.h"
#include <iostream>
```

6.10.1 Opis szczegółowy

Plik zawierający definicje funkcji klasy Timer służącej do pomiaru różnicy czasów. Timer.cpp

Skorowidz