PAMSI_LAB

Wygenerowano przez Doxygen 1.8.6

Śr, 13 maj 2015 21:38:19

Spis treści

1 Indeks hierarchiczny

1.1 Hierarchia klas

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

2 Indeks klas

2.1 Lista klas

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

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

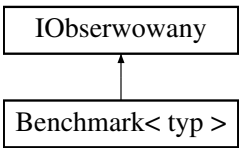
4 Dokumentacja klas

4.1 Dokumentacja szablonu klasy Benchmark< typ >

Modeluje pojęcie Benchmarku.

`#include <Benchmark.hh>`

Diagram dziedziczenia dla Benchmark< typ >



Metody publiczne

- Benchmark (const unsigned int ileProb, unsigned int ∗const ileDanych, const unsigned int ilePowtorzen)

  Konstruktor 2 argumentowy.
- void Test (Framework ∗I, std::string const nazwaPliku)

  Testowanie algorytmu.
- void DodajObserwatora (IObserwator ∗nowyObserwator)

  Dodaje Obserwatora.
- void UsunObserwatora (IObserwator ∗obserwator)

  Usuwa Obserwatora.
- void PowiadomObserwatorow ()

  Powiadamia Obserwatorów.

Atrybuty prywatne

- unsigned int IleProb

  Ilość prób.
- unsigned int ∗ IleDanych

  Tablica liczności serii.
- unsigned int IlePowtorzen

  Ilość powtórzeń
- std::list< IObserwator ∗ > ListaObserwatorow

  Lista Obserwatorow.

4.1.1 Opis szczegółowy

template<class typ>class Benchmark< typ >

Modeluje pojęcie Benchmarku czyli objektu mierzącego czas wykonywania algoytmu

Definicja w linii 26 pliku Benchmark.hh.

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 template<class typ > Benchmark< typ >::Benchmark (const unsigned int *ileProb,* unsigned int ∗const *ileDanych,* const unsigned int *ilePowtorzen*) [inline]

Tworzy objekt klasy Benchmark i inicjuje nową statystykę dla objektu

Parametry

in	*ileProb*	- ilość prób, które zostaną wykonane
in	*ileDanych*	- wkaźnik na tablice z licznościami kolejnych serii

in	*ilePowtorzen*	- ilość powtórzeń każdej serii

Definicja w linii 71 pliku Benchmark.hh.

4.1.3 Dokumentacja funkcji składowych

4.1.3.1 template<class typ > void Benchmark< typ >::DodajObserwatora (IObserwator * *nowyObserwator*) [inline], [virtual]

Dodaje obserwatora do listy obserwatorów danego objektu

Parametry

in	*nowyObserwator*	- wskaźnik na objekt będący obserwatorem

Implementuje IObserwowany.

Definicja w linii 108 pliku Benchmark.hh.

4.1.3.2 template<class typ > void Benchmark< typ >::PowiadomObserwatorow ( ) [inline], [virtual]

Wywołuje u wszystkich aktywnych obserwatorów metodę Aktualizuj.

Implementuje IObserwowany.

Definicja w linii 128 pliku Benchmark.hh.

4.1.3.3 template<class typ > void Benchmark< typ >::Test (Framework * *I*, std::string const *nazwaPliku*) [inline]

Metoda testuje algorytm w okreslonej liczbie serii i powtórzeniach pomiary zapisuje do pliku podanego pez użytkownika

Parametry

in	*I*	- objekt klasy na której zostanie przeprowadzony test
in	*nazwaPliku*	- nazwa pliku z danymi do wczytania

Definicja w linii 87 pliku Benchmark.hh.

4.1.3.4 template<class typ > void Benchmark< typ >::UsunObserwatora (IObserwator * *obserwator*) [inline], [virtual]

Usuwa danego obserwatora z listy obserwatorów

Parametry

in	*obserwator*	- wskaźnik na obserwatora który ma zostać usunięty

Implementuje IObserwowany.

Definicja w linii 119 pliku Benchmark.hh.

4.1.4 Dokumentacja atrybutów składowych

4.1.4.1 template<class typ > unsigned int* Benchmark< typ >::IleDanych [private]

Tablica z licznościami elementów dla kojenych serii

Definicja w linii 42 pliku Benchmark.hh.

4.1.4.2 template<class typ > unsigned int Benchmark< typ >::IlePowtorzen [private]

Ilość powtórzeń każdej serii

Definicja w linii 50 pliku Benchmark.hh.

4.1.4.3 template<class typ > unsigned int Benchmark< typ >::IleProb [private]

Ilość powtórzeń każdej seriii

Definicja w linii 34 pliku Benchmark.hh.

4.1.4.4 template<class typ > std::list<IObserwator*> Benchmark< typ >::ListaObserwatorow [private]

Lista aktywnych obserwatorów danego objektu

Definicja w linii 57 pliku Benchmark.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

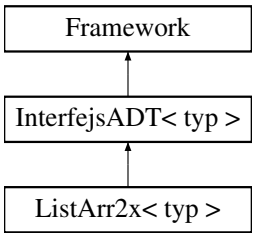
- Benchmark.hh

4.2 Dokumentacja klasy Framework

Modeluje interfejs programu.

`#include <Framework.hh>`

Diagram dziedziczenia dla Framework



Metody publiczne

- virtual void WczytajDane (const char *nazwaPliku, unsigned int n)=0

  Wczytanie danych z pliku.
- virtual void Start (const unsigned int k)=0

  Wykonanie części obliczeniowej programu.
- virtual void Zwolnij ()=0

  Zwalnia pamięć po teście.
- virtual void Pokaz ()=0

4.2.1 Opis szczegółowy

Modeluje interfejs do programów wykonywanch w ramach kursu.

Definicja w linii 24 pliku Framework.hh.

4.2.2 Dokumentacja funkcji składowych

4.2.2.1 virtual void Framework::Pokaz ( ) [pure virtual]

Implementowany w ListArr2x< typ >.

4.2.2.2 virtual void Framework::Start (const unsigned int *k*) [pure virtual]

Metoda w której implementowana jest część obliczeniowa programu, której czas wykonania zostanie zmierzony.

Parametry

in	*k*	- ilość elementów dla których mają zostać wykonane obliczenia.

Implementowany w ListArr2x< typ > i InterfejsADT< typ >.

4.2.2.3 virtual void Framework::WczytajDane (const char * *nazwaPliku*, unsigned int *n*) [pure virtual]

Wczytuje zadaną ilość danych do przetworzenia z pliku o zadanej nazwie.

Parametry

in	*nazwaPliku*	- nazwa pliku z danymi
in	*n*	- ilość danych do wczytania

Implementowany w ListArr2x< typ > i InterfejsADT< typ >.

4.2.2.4 virtual void Framework::Zwolnij ( ) [pure virtual]

Zwalnia pamięć zajmowaną przez objekty wykorzytsane do testów

Implementowany w ListArr2x< typ > i InterfejsADT< typ >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

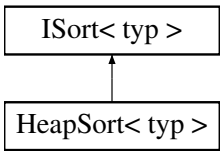
- Framework.hh

4.3 Dokumentacja szablonu klasy HeapSort< typ >

Heap Sort.

`#include <HeapSort.hh>`

Diagram dziedziczenia dla HeapSort< typ >



Metody publiczne

- void Sort (const int lewy, const int prawy, Iterable< typ > &tab)

  Sortowanie przez kopcowanie.

Metody prywatne

- void Kopcuj (const int rozmiarKopca, const int i, Iterable< typ > &tab)

  Porównuje el. kopca.
- void BudujKopiec (const int rozmiar, Iterable< typ > &tab)

  Tworzy kopiec.

4.3.1 Opis szczegółowy

template<class typ>class HeapSort< typ >

Plik zawiera definicję klasy HeapSort

HeapSort

Klasa modelująca objekt potrafiący wykonać sortowanie przez kopcowanie na kontenerze Iterable.

Definicja w linii 19 pliku HeapSort.hh.

4.3.2 Dokumentacja funkcji składowych

4.3.2.1 template<class typ > void HeapSort< typ >::BudujKopiec (const int *rozmiar,* Iterable< typ > & *tab*) [inline], [private]

Tworzy kopiec z tablicy o podanym rozmiarze

Parametry

in	*rozmiar*	- rozmiar tablicy
in	*tab*	- referencja do sortowanego kontenera

Definicja w linii 57 pliku HeapSort.hh.

4.3.2.2 template<class typ > void HeapSort< typ >::Kopcuj (const int *rozmiarKopca,* const int *i,* Iterable< typ > & *tab*) [inline], [private]

Porównuje i ustawia elementy kopca w odpowiedniej kolejności

Parametry

in	*rozmiarKopca*	- rozmiar kopca który sortujemy
in	*i*	- numer gałęzi kopca
in	*tab*	- referencja do sortowanego kontenera

Definicja w linii 31 pliku HeapSort.hh.

4.3.2.3 template<class typ > void HeapSort< typ >::Sort (const int *lewy,* const int *prawy,* Iterable< typ > & *tab*) [inline], [virtual]

Realizuje algorytm sortowania przez kopcowanie Alotytm sortuje tablicę od początku - pierwszy argument jest ignorowany.

Parametry

in	*lewy*	- indekst pierwszego elementu do posortowania
in	*prawy*	- indeks ostatniego elementu do posortowania
in	*tab*	- referencja do kontenera

Implementuje ISort< typ >.

Definicja w linii 76 pliku HeapSort.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

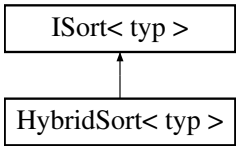
- HeapSort.hh

4.4 Dokumentacja szablonu klasy HybridSort< typ >

Definicja klasy HybridSort.

`#include <HybridSort.hh>`

Diagram dziedziczenia dla HybridSort< typ >



Metody publiczne

- void Sort (const int lewy, const int prawy, Iterable< typ > &tab)

  Sortowanie hybrydowe.

4.4.1 Opis szczegółowy

template<class typ>class HybridSort< typ >

Plik zawiera definicję klasy HybridSort

The HybridSort class

Klasa modeluje objekt potrafiący sortować kontener typu Iterable, algorytm wykorzystuje zotymalizowany (mediana z trzech algorytm sortowania szybkiego oraz algirytm sortowania przez wstawianie

Definicja w linii 22 pliku HybridSort.hh.

4.4.2 Dokumentacja funkcji składowych

4.4.2.1 template<class typ> void HybridSort< typ >::Sort (const int *lewy*, const int *prawy*, Iterable< typ > & *tab*) [inline], [virtual]

Metoda realizuje algorytm sortowania hybrydowego bazujący na zoptymalizowanym ze względu na wybór pivota (mediana z trzech) algorytmowi Sortowania Szybkiego oraz jako algorytm pomocniczy wykorzystane zostało sortowanie przez wstawianie.

Parametry

in	*lewy*	- indeks pierwszego elementu z listy do posortowania
in	*prawy*	- indeks ostatniego elementu z listy do posortowania
in	*tab*	- referencja do sortowanego kontenera

Implementuje ISort< typ >.

Definicja w linii 39 pliku HybridSort.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

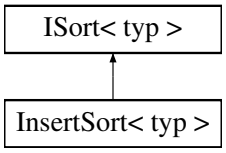
- HybridSort.hh

4.5 Dokumentacja szablonu klasy InsertSort< typ >

Insert Sort.

`#include <InsertSort.hh>`

Diagram dziedziczenia dla InsertSort< typ >



Metody publiczne

- void Sort (const int lewy, const int prawy, Iterable< typ > &tab)

  Sortowanie przez wstawianie.

4.5.1 Opis szczegółowy

template<class typ>class InsertSort< typ >

Plik zawiera definicję klasy InsertSort

InsertSort

Klasa modelująca objekt potrafiący wykonać sortowanie przez wstawianie na kontenerze Iterable.

Definicja w linii 19 pliku InsertSort.hh.

4.5.2 Dokumentacja funkcji składowych

4.5.2.1 template<class typ> void InsertSort< typ >::Sort (const int *lewy*, const int *prawy*, Iterable< typ > & *tab*) [inline], [virtual]

Metoda realizuje algorytm sortowania przez wstawianie.

Parametry

in	*lewy*	- indekst pierwszego elementu do posortowania
in	*prawy*	- indeks ostatniego elementu do posortowania
in	*tab*	- referencja do kontenera

Implementuje ISort< typ >.

Definicja w linii 33 pliku InsertSort.hh.

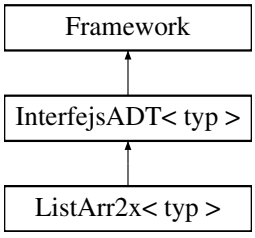
Dokumentacja dla tej klasy została wygenerowana z pliku:

- InsertSort.hh

4.6 Dokumentacja szablonu klasy InterfejsADT< typ >

`#include <InterfejsADT.hh>`

Diagram dziedziczenia dla InterfejsADT< typ >



Metody publiczne

- virtual void push (const typ dana, const unsigned int pole)=0

  Dodaje kolejny element.
- virtual void pop (const unsigned int pole)=0

  Pobiera element.
- virtual unsigned int size () const =0

  Liczność elemetów.
- void WczytajDane (const char ∗nazwaPliku, unsigned int n)=0

  Wczytanie danych z pliku.
- void Start (const unsigned int k)=0

  Wykonanie części obliczeniowej programu.
- virtual void Zwolnij ()=0

  Zwalnia pamięć

4.6.1 Opis szczegółowy

template<class typ>class InterfejsADT< typ >

\ brief Definiuje interfejs użytkownika

Definiuje interfejs użytkownika dla listy, stosu i kolejki.

Definicja w linii 13 pliku InterfejsADT.hh.

4.6.2 Dokumentacja funkcji składowych

4.6.2.1 template<class typ > virtual void InterfejsADT< typ >::pop (const unsigned int *pole*) [pure virtual]

Pobiera element z typu danych

Parametry

in	*pole*	- !!!DOSTEPNE TYLKO DLA LISTY!!! nr pola z ktore pobiera element

Implementowany w ListArr2x< typ >.

4.6.2.2 template<class typ > virtual void InterfejsADT< typ >::push (const typ *dana,* const unsigned int *pole*) [pure virtual]

Dodaje kolejny element do typu danych

Parametry

in	*dana*	- element który chcemy dorzucić do naszego typu
in	*pole*	- !!!DOSTEPNE TYLKO DLA LISTY!!! nr pola na które chcemy dodać element

Implementowany w ListArr2x< typ >.

4.6.2.3 template<class typ > virtual unsigned int InterfejsADT< typ >::size ( ) const [pure virtual]

Informuje o liczności elementów obecnie przechowywanych

Zwracane wartości

zwraca	ilość przechowywanych elementów

Implementowany w ListArr2x< typ >.

4.6.2.4 template<class typ > void InterfejsADT< typ >::Start (const unsigned int *k*) [pure virtual]

Metoda w której implementowana jest część obliczeniowa programu, której czas wykonania zostanie zmierzony.

Parametry

in	*k*	- ilość elementów dla których mają zostać wykonane obliczenia.

Implementuje Framework.

Implementowany w ListArr2x< typ >.

4.6.2.5 template<class typ > void InterfejsADT< typ >::WczytajDane (const char * *nazwaPliku,* unsigned int *n*) [pure virtual]

Wczytuje zadaną ilość danych do przetworzenia z pliku o zadanej nazwie.

Parametry

in	*nazwaPliku*	- nazwa pliku z danymi
in	*n*	- ilość danych do wczytania

Implementuje Framework.

Implementowany w ListArr2x< typ >.

4.6.2.6 template<class typ > virtual void InterfejsADT< typ >::Zwolnij ( ) [pure virtual]

Zwalnia pamięć zajmowaną przez daną strukturę

Implementuje Framework.

Implementowany w ListArr2x< typ >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

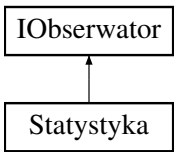
- InterfejsADT.hh

4.7 Dokumentacja klasy IObserwator

Klasa IObserwator.

`#include <IObserwator.hh>`

Diagram dziedziczenia dla IObserwator



Metody publiczne

- virtual void Aktualizuj ()=0

  Aktualizuj.

4.7.1 Opis szczegółowy

Plik zawira definicję klasy IObsereator.

The IObserwator class

Klasa modeluje interfejs objektu będącego obserwatorem.

Definicja w linii 17 pliku IObserwator.hh.

4.7.2 Dokumentacja funkcji składowych

4.7.2.1 virtual void IObserwator::Aktualizuj ( ) [pure virtual]

Aktualizuje dane na podstawie wydarzenie w objekcie obserowanym.

Implementowany w Statystyka.

Dokumentacja dla tej klasy została wygenerowana z pliku:

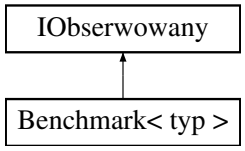
- IObserwator.hh

4.8 Dokumentacja klasy IObserwowany

Interfejs obserwowanego.

`#include <IObserwowany.hh>`

Diagram dziedziczenia dla IObserwowany



Metody publiczne

- virtual void DodajObserwatora (IObserwator ∗nowyObserwator)=0

  Dodaje Obserwatora.
- virtual void UsunObserwatora (IObserwator ∗obserwator)=0

  Usuwa Obserwatora.
- virtual void PowiadomObserwatorow ()=0

  Powiadamia Obserwatorów.

4.8.1 Opis szczegółowy

W pliku zawarta jest definicja interfejsu obserwowanego

The IObserwowany class

Klasa czysto wirtualna modelująca interfejs objektu obserwowanego.

Definicja w linii 19 pliku IObserwowany.hh.

4.8.2 Dokumentacja funkcji składowych

4.8.2.1 virtual void IObserwowany::DodajObserwatora (IObserwator ∗ *nowyObserwator*) [pure virtual]

Dodaje nowego obserwatora do listy oserwatorów danego objektu.

Parametry

in	*nowyObserwator*	- wkaźnik na dodawanego obserwatora

Implementowany w Benchmark< typ >.

4.8.2.2 virtual void IObserwowany::PowiadomObserwatorow ( ) [pure virtual]

Powiadamia obseratorów o wydarzeniu.

Implementowany w Benchmark< typ >.

4.8.2.3 virtual void IObserwowany::UsunObserwatora (IObserwator * *obserwator*) [pure virtual]

Usuwa danego obserwatora z listy obserwatorów danego objektu.

Parametry

in	*obserwator*	- obserwator do usunięcia z listy

Implementowany w Benchmark< typ >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

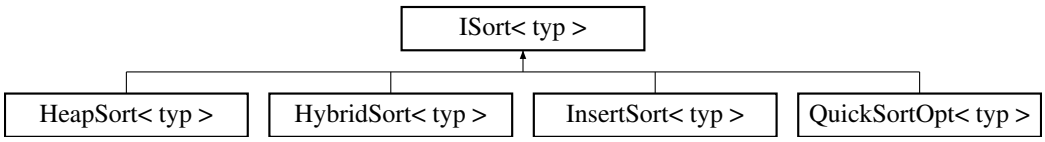
- IObserwowany.hh

4.9 Dokumentacja szablonu klasy ISort< typ >

Interfejs ISort.

`#include <ISort.hh>`

Diagram dziedziczenia dla ISort< typ >



Metody publiczne

- virtual void Sort (const int lewy, const int prawy, Iterable< typ > &tab)=0
  Sort.

4.9.1 Opis szczegółowy

template<class typ>class ISort< typ >

Plik zawiera defincję interfejsu ISortable

Interfejs ISort

Klasa modeluje pojęcie interfejsu ISort, który tworzy interfejs użytkownika dla algorytmów sortujących.

Definicja w linii 21 pliku ISort.hh.

4.9.2 Dokumentacja funkcji składowych

4.9.2.1 template<class typ > virtual void ISort< typ >::Sort (const int *lewy,* const int *prawy,* Iterable< typ > & *tab*) [pure virtual]

Metoda wykonująca agorytm sortowania na kontenerze podanym w argumencie.

Parametry

in	*tab*	- referencja do sortowanego kontenera
in	*lewy*	- początkowy indeks sortowania
in	*prawy*	- końcowy indeks sortowania

Implementowany w QuickSortOpt< typ >, HeapSort< typ >, HybridSort< typ > i InsertSort< typ >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

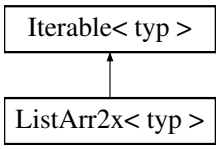
- ISort.hh

4.10 Dokumentacja szablonu klasy Iterable< typ >

Interfejs Iterable.

`#include <Iterable.hh>`

Diagram dziedziczenia dla Iterable< typ >



Metody publiczne

- virtual typ operator[] (unsigned int indeks) const =0

  operator []
- virtual void Zamien (int a, int b)=0

4.10.1 Opis szczegółowy

template<class typ>class Iterable< typ >

Plik zawiera defincję interfejsu Iterable

Interfejs Iterable

Klasa modeluje pojęcie interfejsu Iterable, który umożliwia wgląd w kontener, oraz udostępnia interferjsowi ISortable metodę umożliwiającą kolejkowanie elementów.

Definicja w linii 22 pliku Iterable.hh.

4.10.2 Dokumentacja funkcji składowych

4.10.2.1 template<class typ> virtual typ Iterable< typ >::operator[] (unsigned int *indeks*) const [pure virtual]

Przeciążenie opratora[] w celu umożliwienia przeglądania zawrtości kontenera. Indeksujemy od 0.

Parametry

in	*indeks*	- indeks elementu, którego wartość ma zostać odczytana

Zwracane wartości

zwraca	wartość znajdującą się na danym indeksie kontenera

Implementowany w ListArr2x< typ >.

4.10.2.2 template<class typ> virtual void Iterable< typ >::Zamien (int *a*, int *b*) [pure virtual]

Implementowany w ListArr2x< typ >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

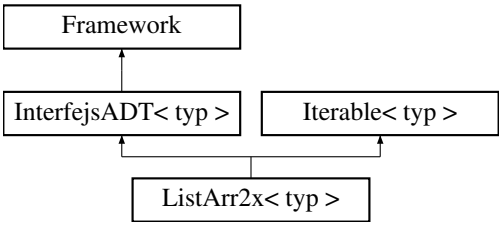
- Iterable.hh

4.11 Dokumentacja szablonu klasy ListArr2x< typ >

Modeluje pojęcie Listy (array)

`#include <ListArr2x.hh>`

Diagram dziedziczenia dla ListArr2x< typ >



Metody publiczne

- ListArr2x ()

  Konstruktor bezarumentowy.
- void push (const typ dana, const unsigned int pole)

  Dodaje element do ListyArr2x.
- void pop (const unsigned int pole)

  Pobiera element z ListyArr2x.
- unsigned int size () const

  Wielkość listy.
- void Start (const unsigned int k)

  Metoda do testów.
- void WczytajDane (const char ∗nazwaPliku, unsigned int n)

  Wczytuje dane z pliku.
- void Zwolnij ()

  Zwalnia pamięć
- void Pokaz ()

  Wyświetla elementy Listy.
- typ operator[] (unsigned int indeks) const

  operator []
- void Zamien (int a, int b)

  Zamienia elmenty listy.

Metody prywatne

- void UsunZListy (const unsigned int pole)

  UsunZListy.
- void DodajDoListy (const typ dana, const unsigned int pole)

  DodajDoListy.

Atrybuty prywatne

- typ * tab

  Wkaźnik na dynamiczną tablicę
- unsigned int RozmiarT

  Rozmiar tablicy.
- unsigned int RozmiarL

  Rozmiar Listy.

4.11.1 Opis szczegółowy

template<class typ>class ListArr2x< typ >

Modeluje pojęcie Listy opartej na dynamicznej tablicy. Dodając elementy zwiększa tablicę dwukrotnie, jeżeli brakuje miejsca. a

Definicja w linii 25 pliku ListArr2x.hh.

4.11.2 Dokumentacja konstruktora i destruktora

4.11.2.1 template<class typ> ListArr2x< typ >::ListArr2x ( ) [inline]

Kontruktor alokujący tablicę jednoelementową z której będzie tworzona lista

Definicja w linii 98 pliku ListArr2x.hh.

4.11.3 Dokumentacja funkcji składowych

4.11.3.1 template<class typ> void ListArr2x< typ >::DodajDoListy (const typ *dana,* const unsigned int *pole*) [inline], [private]

Dodaje daną do listy na określony indeks

Parametry

dana	- wartość która ma zostać umieszczona na liście
pole	- indeks pola na którym ma zostać umieszczona wartość

Definicja w linii 77 pliku ListArr2x.hh.

4.11.3.2 template<class typ> typ ListArr2x< typ >::operator[] (unsigned int *indeks*) const [inline], [virtual]

Przeciążenie opratora[] w celu umożliwienia przeglądania zawrtości listy. Indeksujemy od 0.

Parametry

in	*indeks*	- indeks elementu, którego wartość ma zostać odczytana

Zwracane wartości

zwraca	wartość znajdującą się na danym indeksie listy

Implementuje Iterable< typ >.

Definicja w linii 254 pliku ListArr2x.hh.

4.11.3.3 template<class typ> void ListArr2x< typ >::Pokaz ( ) [inline], [virtual]

Metoda wypsuje na terminal elementy znajdujące się na liście

Implementuje Framework.

Definicja w linii 238 pliku ListArr2x.hh.

4.11.3.4 template<class typ> void ListArr2x< typ >::pop (const unsigned int *pole*) [inline], [virtual]

Pobiera element z ListyArr2x usuwając go z niej i zmniejszając rozmiar o połowę w przypadku przekroczenia stosunku 1:4 (RozmiarL:RozmiarT)

param[in] - pole - nr pola z którgo chcemy pobrać element (indeksowane od 0)

Implementuje InterfejsADT< typ >.

Definicja w linii 150 pliku ListArr2x.hh.

4.11.3.5 template<class typ> void ListArr2x< typ >::push (const typ *dana*, const unsigned int *pole*) [inline], [virtual]

Dodaje nowy element do ListyArr2x

Parametry

in	*dana*	- element który chcemy umieścić na liście
in	*pole*	- nr pola na którym chcemy umieścić element jeżeli chcesz umieścić na początku listy podaj wartość 0, na końcu warość size()

Implementuje InterfejsADT< typ >.

Definicja w linii 115 pliku ListArr2x.hh.

4.11.3.6 template<class typ> unsigned int ListArr2x< typ >::size ( ) const [inline], [virtual]

Informuje o ilości elementów znajdujących się na LiścieArr2x

Zwracane wartości

-	zwraca liczbę elementów ListyArr2x

Implementuje InterfejsADT< typ >.

Definicja w linii 184 pliku ListArr2x.hh.

4.11.3.7 template<class typ> void ListArr2x< typ >::Start (const unsigned int *k*) [inline], [virtual]

Metoda, której czas wykonanai ma zostać zmierzony

Parametry

in	*k*	- ilość elementów do wczytania

Implementuje InterfejsADT< typ >.

Definicja w linii 194 pliku ListArr2x.hh.

4.11.3.8 template<class typ> void ListArr2x< typ >::UsunZListy (const unsigned int *pole*) [inline], [private]

Usuwa z listy element o podanym indeksie

Parametry

in	*pole*	- indeks elementu do usunięcia.

Definicja w linii 59 pliku ListArr2x.hh.

4.11.3.9 template<class typ> void ListArr2x< typ >::WczytajDane (const char * *nazwaPliku,* unsigned int *n*) [inline], [virtual]

Wczytuje dane z pliku do ListArr2x

param[in] nazwaPliku - nazwa pliku z danymi param[in] n - ilość danych do wczytania, 0 oznacza wszystkie dane z pliku

Implementuje InterfejsADT< typ >.

Definicja w linii 207 pliku ListArr2x.hh.

4.11.3.10 template<class typ> void ListArr2x< typ >::Zamien (int *a,* int *b*) [inline], [virtual]

Zamienia dwa elementy listy o polach podanych w wywołaniu

Parametry

in	*a*	- indeks pierwszego elementu do zamiany
in	*b*	- indeks drugiego elementu do zamiany

Implementuje Iterable< typ >.

Definicja w linii 266 pliku ListArr2x.hh.

4.11.3.11 template<class typ> void ListArr2x< typ >::Zwolnij ( ) [inline], [virtual]

Zwalnia pamięć zaalokowaną przez ListArr2x

Implementuje InterfejsADT< typ >.

Definicja w linii 224 pliku ListArr2x.hh.

4.11.4 Dokumentacja atrybutów składowych

4.11.4.1 template<class typ> unsigned int ListArr2x< typ >::RozmiarL [private]

Aktualny rozmiar ListyArr2x

Definicja w linii 49 pliku ListArr2x.hh.

4.11.4.2 template<class typ> unsigned int ListArr2x< typ >::RozmiarT [private]

Aktualny rozmiar tablicy.

Definicja w linii 41 pliku ListArr2x.hh.

4.11.4.3 template<class typ> typ* ListArr2x< typ >::tab [private]

Wskaźnik na dynamiczną tablicę tworzącą ListęArr2x

Definicja w linii 33 pliku ListArr2x.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

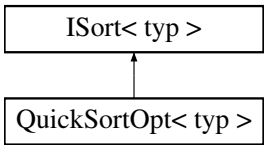
- ListArr2x.hh

4.12 Dokumentacja szablonu klasy QuickSortOpt< typ >

Definicja klasy QuickSortOpt.

`#include <QuickSortOpt.hh>`

Diagram dziedziczenia dla QuickSortOpt< typ >



Metody publiczne

- void Sort (const int lewy, const int prawy, Iterable< typ > &tab)

  Zoptymalizowane Szybkie Sortowanie.

Metody prywatne

- int Partition (int lewy, int prawy, Iterable< typ > &tab)
- int MedianaTrzech (const int a, const int b, const int c, Iterable< typ > &tab) const

  Znajduje mediane.

Przyjaciele

- template<class U >
  class HybridSort

4.12.1 Opis szczegółowy

template<class typ>class QuickSortOpt< typ >

Plik zawiera definicję klasy QuickSortOpt

The QuickSortOpt class

Klasa modeluje objekt potrafiący sortować kontener typu Iterable, algorymem szybkiego syrtowania zoptymalizowanym o wybór pivota na podstawie mediany z trzech.

Definicja w linii 22 pliku QuickSortOpt.hh.

4.12.2 Dokumentacja funkcji składowych

4.12.2.1 template<class typ> int QuickSortOpt< typ >::MedianaTrzech (const int *a*, const int *b*, const int *c*, Iterable< typ > & *tab*) const [inline], [private]

Znajduje mediane wartości z trzech podanych elementów kontenera

Parametry

in	*a*	- indeks pierwszego elementu do liczenia mediany

in	*b*	- indeks drugiego elementu do liczenia mediany
in	*c*	- indeks trzeciego elementu do liczenia mediany
in	*tab*	- referencja do sortowanego kontenera

Zwracane wartości

-	zwraca indeks elementu będącego medianą z trzech wartości podanych elementów

Definicja w linii 65 pliku QuickSortOpt.hh.

4.12.2.2 template<class typ> int QuickSortOpt< typ >::Partition (int *lewy*, int *prawy*, Iterable< typ > & *tab*) [inline], [private]

Partycjonowanie kontenera

Metoda będąca częścią algorytmu Sortowania Szybkiego. Dzieli przekazany fragment kontenera na dwie części - lewy z elementami mniejszymi od wybranego pivota i prawa z elementami większymi od wybranego pivota. Pivot jest dobierany za pomocą liczenia mediany z trzech elementów: pierwszego, środkowego i ostatniego.

Parametry

in	*lewy*	- indeks pierwszego elementu z kontenera do posortowania
in	*prawy*	- indekst ostatniego elmentu z kontenera do posortowania
in	*tab*	- referencja do sortowanego kontenera

Definicja w linii 38 pliku QuickSortOpt.hh.

4.12.2.3 template<class typ> void QuickSortOpt< typ >::Sort (const int *lewy*, const int *prawy*, Iterable< typ > & *tab*) [inline], [virtual]

Realizuje zoptymalizowany ze względu na wybór pivota algorytm szybkiego sortowania elementów konteneru

Parametry

in	*lewy*	- indeks pierwszego elementu tworzącego kontener do posotrowania
in	*prawy*	- indeks ostatniego elementu tworzącego kontener do posotrowania
in	*tab*	- referencja do sortowanego kontenera

Implementuje ISort< typ >.

Definicja w linii 94 pliku QuickSortOpt.hh.

4.12.3 Dokumentacja przyjaciół i funkcji związanych

4.12.3.1 template<class typ> template<class U > friend class HybridSort [friend]

Definicja w linii 78 pliku QuickSortOpt.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

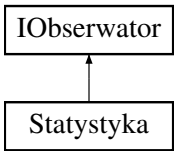
- QuickSortOpt.hh

4.13 Dokumentacja klasy Statystyka

Modeluje pojęcie statystyki.

`#include <Statystyka.hh>`

Diagram dziedziczenia dla Statystyka



Metody publiczne

- Statystyka (const unsigned int iloscProb, unsigned int ∗proby, const unsigned int ilePowtorzen)

  Konstruktor z dwoma pramametrami.
- ∼Statystyka ()

  Destruktor - zwaknia pamięć
- void ZapiszStaty (std::string nazwaPliku) const

  Zapisuje statysykę do pliku.
- void Aktualizuj ()

  Aktualizuj.

Atrybuty prywatne

- unsigned int IleProb

  Ilość prób.
- unsigned int ∗ Proba

  Tablica z rozmiarami prób.
- double ∗ Czas

  Średni czas wykonania danej próby.
- double SumaCzasuProby

  Suma Czasu Proby.
- unsigned int IloscPowtorzen

  Ilość Powtórzeń
- unsigned int LicznikPowtorzen

  Licznik Powtórzeń
- unsigned int LicznikProb

  Licznik Prób.
- Stoper ∗ MojStoper

  Stoper.

4.13.1 Opis szczegółowy

Modeluje pojęcie statystyki, czyli średnich czasów wykonania metody dla różnyuch wielkości prób.

Definicja w linii 27 pliku Statystyka.hh.

4.13.2 Dokumentacja konstruktora i destruktora

4.13.2.1 Statystyka::Statystyka (const unsigned int *iloscProb*, unsigned int ∗ *proby*, const unsigned int *ilePowtorzen*)

Konstruktor z dwoma paramatremi tworzy dynamiczne tablice przechowujące statystykę oraz wypełnia rozmiary prób.

Parametry

in	*iloscProb*	- liczbosc prob w ksperymencie
in	*proby*	- tablica z licznościami prób.

Definicja w linii 12 pliku Statystyka.cpp.

4.13.2.2 Statystyka::∼Statystyka ( ) [inline]

Zwalnia pamięć zaalokowaną na dynamiczne tablicy przechowujące statystykę.

Definicja w linii 108 pliku Statystyka.hh.

4.13.3 Dokumentacja funkcji składowych

4.13.3.1 void Statystyka::Aktualizuj ( ) [virtual]

Aktualizuje pozyskiwane dane dotyczące wyników testu: Jeżeli stoper nie odlicza to uruchamia odliczanie, Jeżeli stper odlicza to go zatrzymuje i sumuje czasy powtórzeń. Gdy nasąpi wykonanie wszystkich pomiarów w próbie to uzupełnia talicę przechowywujacą średnie czasy każdej próby.

Implementuje IObserwator.

Definicja w linii 44 pliku Statystyka.cpp.

4.13.3.2 void Statystyka::ZapiszStaty (std::string *nazwaPliku*) const

Zapisuje statystystykę do pliku o nazwie podanej w argumencie. Plik zapisany zostaje w sposób, gdzie każda nowa linia wygląda następująco: RozmiarPróby,ŚredniCzas czas wyrażony jest w ms.

Parametry

in	*nazwaPliku*	- nazwa pliku do którego ma zostać zapisanaza statystyka

Definicja w linii 25 pliku Statystyka.cpp.

4.13.4 Dokumentacja atrybutów składowych

4.13.4.1 double∗ Statystyka::Czas [private]

wskaźnik na tablica ze średnimi czasami wykonania kolejnych prób.

Definicja w linii 51 pliku Statystyka.hh.

4.13.4.2 unsigned int Statystyka::IleProb [private]

Ilość prób do utworzenia statystyki

Definicja w linii 35 pliku Statystyka.hh.

4.13.4.3 unsigned int Statystyka::IloscPowtorzen [private]

Przechowuje ilość wykonywanych powtórzeń pojedyńczego testu.

Definicja w linii 65 pliku Statystyka.hh.

4.13.4.4 unsigned int Statystyka::LicznikPowtorzen [private]

Zlicza ilosc wykonanych powtórzeń w danej próbie.

Definicja w linii 72 pliku Statystyka.hh.

4.13.4.5 unsigned int Statystyka::LicznikProb [private]

Zlicza ilosć prób wykonanych prób.

Definicja w linii 79 pliku Statystyka.hh.

4.13.4.6 Stoper* Statystyka::MojStoper [private]

Stoper wykorzystywany do pomiaru czasu.

Definicja w linii 86 pliku Statystyka.hh.

4.13.4.7 unsigned int* Statystyka::Proba [private]

Wskaźnik na tablicę zawierającą wielkości danych prób.

Definicja w linii 43 pliku Statystyka.hh.

4.13.4.8 double Statystyka::SumaCzasuProby [private]

Przechowuje sumę czasów pojedyńczych powtórzeń z danej próby.

Definicja w linii 58 pliku Statystyka.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Statystyka.hh
- Statystyka.cpp

4.14 Dokumentacja klasy Stoper

Klasa Stoper.

`#include <Stoper.hh>`

Metody publiczne

- Stoper ()
  Stoper.
- void Start ()
  Start.
- void Stop ()
  Stop.
- void Reset ()
  Reset.
- double DajPomiar () const
  Pomiar.
- bool CzyOdmierza () const
  Czy Odmierza.

Atrybuty prywatne

- double CzasPoczatkowy
  Czas Początkowy.
- double CzasKoncowy
  Czas Końcowy.
- bool CzyLiczy
  Czy Liczy.

4.14.1 Opis szczegółowy

Plik zawiera definicję klasy Stoper.

The Stoper class

Klasa modeluje stoper niezbędny do odliczania czasu.

Definicja w linii 20 pliku Stoper.hh.

4.14.2 Dokumentacja konstruktora i destruktora

4.14.2.1 Stoper::Stoper ( )

Kontruktor bezarumentowy zeruje czasy i ustawia wartość pola CzyLiczy na false.

Definicja w linii 3 pliku Stoper.cpp.

4.14.3 Dokumentacja funkcji składowych

4.14.3.1 bool Stoper::CzyOdmierza ( ) const

Informuje czy stoper aktualinie liczy czy nie.

Zwracane wartości

true	- gdy odlicza
false	- gdy nie odlicza

Definicja w linii 29 pliku Stoper.cpp.

4.14.3.2 double Stoper::DajPomiar ( ) const

Wyłuskuje czas pomiaru w ms.

Zwracane wartości

zwrca	czas pomiaru wyrażon w ms

Definicja w linii 25 pliku Stoper.cpp.

4.14.3.3 void Stoper::Reset ( )

Resetuje stoper.

Definicja w linii 19 pliku Stoper.cpp.

4.14.3.4 void Stoper::Start ( )

Uruchamia odliczanie czasu.

Definicja w linii 9 pliku Stoper.cpp.

4.14.3.5 void Stoper::Stop ( )

Zatrzymuje odliczanie czasu.

Definicja w linii 14 pliku Stoper.cpp.

4.14.4 Dokumentacja atrybutów składowych

4.14.4.1 double Stoper::CzasKoncowy [private]

Czas w którym odliczanie czasu zostało zatrzymane.

Definicja w linii 34 pliku Stoper.hh.

4.14.4.2 double Stoper::CzasPoczatkowy [private]

Czas w którym stoper zaczął odliczać.

Definicja w linii 27 pliku Stoper.hh.

4.14.4.3 bool Stoper::CzyLiczy [private]

Zmienna przechowuje wartośc true gdy stoper aktualnie odlicza czas, lub false gdy jest zatrzymany.

Definicja w linii 42 pliku Stoper.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Stoper.hh
- Stoper.cpp

5 Dokumentacja plików

5.1 Dokumentacja pliku Benchmark.hh

Definicja klasy Benchmark.

```
#include "Framework.hh"
#include <ctime>
#include "Statystyka.hh"
#include "IObserwowany.hh"
#include <list>
```

Komponenty

- class Benchmark< typ >
  Modeluje pojęcie Benchmarku.

5.1.1 Opis szczegółowy

Plik zawiera definicję klasy Benchmark wraz z definicją jej metod.

Definicja w pliku Benchmark.hh.

5.2 Dokumentacja pliku Framework.hh

Definicja klasy Framework.

```
#include <iostream>
```

Komponenty

- class Framework
  Modeluje interfejs programu.

5.2.1 Opis szczegółowy

Plik zawiera definicję abstrakcyjnej klasy Framework, która tworzy interfejs dla programów implementowanych podczas zajęć laboratoryjnych z PAMSI.

Definicja w pliku Framework.hh.

5.3 Dokumentacja pliku HeapSort.hh

```
#include "ISort.hh"
```

Komponenty

- class HeapSort< typ >

  Heap Sort.

5.4 Dokumentacja pliku HybridSort.hh

```
#include "QuickSortOpt.hh"
#include "InsertSort.hh"
```

Komponenty

- class HybridSort< typ >

  Definicja klasy HybridSort.

5.5 Dokumentacja pliku InsertSort.hh

```
#include "ISort.hh"
```

Komponenty

- class InsertSort< typ >

  Insert Sort.

5.6 Dokumentacja pliku InterfejsADT.hh

```
#include "Framework.hh"
```

Komponenty

- class InterfejsADT< typ >

5.7 Dokumentacja pliku IObserwator.hh

Komponenty

- class IObserwator

  Klasa IObserwator.

5.8 Dokumentacja pliku IObserwowany.hh

```
#include "IObserwator.hh"
```

Komponenty

- class IObserwowany

  Interfejs obserwowanego.

Definicje

- #define IOBSERWOWANY_HH

5.8.1 Dokumentacja definicji

5.8.1.1 #define IOBSERWOWANY_HH

Definicja w linii 2 pliku IObserwowany.hh.

5.9 Dokumentacja pliku ISort.hh

```
#include "ListArr2x.hh"
```

Komponenty

- class ISort< typ >

  Interfejs ISort.

5.10 Dokumentacja pliku Iterable.hh

```
#include <iostream>
```

Komponenty

- class Iterable< typ >

  Interfejs Iterable.

Definicje

- #define ITREABLE_HH

5.10.1 Dokumentacja definicji

5.10.1.1 #define ITREABLE_HH

Definicja w linii 2 pliku Iterable.hh.

5.11 Dokumentacja pliku ListArr2x.hh

Definicja klasy ListArr2x.

```
#include "InterfejsADT.hh"
#include "Iterable.hh"
#include "Pliki.hh"
```

Komponenty

- class ListArr2x< typ >

  Modeluje pojęcie Listy (array)

5.11.1 Opis szczegółowy

Plik zawiera definicję klasy ListaArr2x ujętej w szablon typu wraz z jej składowymi metofdami.

Definicja w pliku ListArr2x.hh.

5.12 Dokumentacja pliku main.cpp

Moduł główny programu.

```
#include "../inc/ListArr2x.hh"
#include "../inc/Statystyka.hh"
#include "../inc/Benchmark.hh"
#include "../inc/Pliki.hh"
#include "../inc/InsertSort.hh"
#include "../inc/HeapSort.hh"
#include "../inc/HybridSort.hh"
```

Funkcje

- int main (int argc, char *argv[])

Zmienne

- const int ILOSC_POWTORZEN = 50

  Ilosc powtórzeń danej próby.
- const int ILOSC_PROB = 9

  Ilość prób.
- const std::string NAZWA_PLIKU_Z_DANYMI = "dane.dat"

5.12.1 Opis szczegółowy

Program wkonuje serię 50 pomiarów czasu wykonania metody start dla różncyh wielkości problemu obliczeniowego. OBSŁUGA PROGRAMU: Aby wywołać program należy w lini poleceń wywołać jego nazę np: "./a.out"

Definicja w pliku main.cpp.

5.12.2 Dokumentacja funkcji

5.12.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definicja w linii 41 pliku main.cpp.

5.12.3 Dokumentacja zmiennych

5.12.3.1 const int ILOSC_POWTORZEN = 50

Ilośc powtórzeń danej próby

Definicja w linii 29 pliku main.cpp.

5.12.3.2 const int ILOSC_PROB = 9

Ilość prób = ilość rozmiarów prób

Definicja w linii 37 pliku main.cpp.

5.12.3.3 const std::string NAZWA_PLIKU_Z_DANYMI = "dane.dat"

Definicja w linii 39 pliku main.cpp.

5.13 Dokumentacja pliku Pliki.cpp

Definicje funkcji obslugi plikow.

```
#include "../inc/Pliki.hh"
```

Funkcje

- void OtworzPlikIn (const char ∗nazwaPliku, std::fstream &plik)

  Otwiera plik do odczytu.
- void OtworzPlikOut (const char ∗nazwaPliku, std::fstream &plik)

  Otwiera plik do zapisu czyszcąc jego zawartość
- void LosujIntDoPliku (const unsigned int n, const unsigned int zakres)

  Zapisuje n losowych liczb(int) do pliku.

5.13.1 Opis szczegółowy

Plik zawiera definicje funkcji zwiazanych z obsluga plikow.

Definicja w pliku Pliki.cpp.

5.13.2 Dokumentacja funkcji

5.13.2.1 void LosujIntDoPliku (const unsigned int *n,* const unsigned int *zakres*)

Losuje n liczb z zakresu od 1 do podonago przez użytwkonika następnie zapisuje wylosowane dane do pliku o nazwe "dane.dat"

Parametry

in	*n*	- ilość liczb do zapisania
in	*zakres*	- górny zakres wartości liczb

Definicja w linii 27 pliku Pliki.cpp.

5.13.2.2 void OtworzPlikIn (const char * *nazwaPliku*, std::fstream & *plik*)

Otwiera plik i sprawdza czy otwarcie sie powiodlo jezeli nie to koczy program

Parametry

in	*nazwaPliku*	- nazwa pliku ktory chcemy otworzyc
in	*plik*	- strumien powiazany z plikiem

Definicja w linii 11 pliku Pliki.cpp.

5.13.2.3 void OtworzPlikOut (const char * *nazwaPliku*, std::fstream & *plik*)

Otwiera plik i sprawdza czy otwarcie sie powiodlo jezeli nie to koczy program

Parametry

in	*nazwaPliku*	- nazwa pliku ktory chcemy otworzyc
in	*plik*	- strumien powiazany z plikiem

Definicja w linii 19 pliku Pliki.cpp.

5.14 Dokumentacja pliku Pliki.hh

Funkcje obslugi plikow.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
```

Funkcje

- void OtworzPlikIn (const char *nazwaPliku, std::fstream &plik)

  Otwiera plik do odczytu.
- void OtworzPlikOut (const char *nazwaPliku, std::fstream &plik)

  Otwiera plik do zapisu czyszcąc jego zawartość
- void LosujIntDoPliku (const unsigned int n, const unsigned int zakres)

  Zapisuje n losowych liczb(int) do pliku.

5.14.1 Opis szczegółowy

Plik zawiera deklaracje funkcji zwiazanych z obsuga plikow

Definicja w pliku Pliki.hh.

5.14.2 Dokumentacja funkcji

5.14.2.1 void LosujIntDoPliku (const unsigned int *n*, const unsigned int *zakres*)

Losuje n liczb z zakresu od 1 do podonago przez użytwkonika następnie zapisuje wylosowane dane do pliku o nazwe "dane.dat"

Parametry

in	*n*	- ilość liczb do zapisania
in	*zakres*	- górny zakres wartości liczb

Definicja w linii 27 pliku Pliki.cpp.

5.14.2.2 void OtworzPlikIn (const char * *nazwaPliku*, std::fstream & *plik*)

Otwiera plik i sprawdza czy otwarcie sie powiodlo jezeli nie to koczy program

Parametry

in	*nazwaPliku*	- nazwa pliku ktory chcemy otworzyc
in	*plik*	- strumien powiazany z plikiem

Definicja w linii 11 pliku Pliki.cpp.

5.14.2.3 void OtworzPlikOut (const char * *nazwaPliku*, std::fstream & *plik*)

Otwiera plik i sprawdza czy otwarcie sie powiodlo jezeli nie to koczy program

Parametry

in	*nazwaPliku*	- nazwa pliku ktory chcemy otworzyc
in	*plik*	- strumien powiazany z plikiem

Definicja w linii 19 pliku Pliki.cpp.

5.15 Dokumentacja pliku QuickSortOpt.hh

```
#include "ISort.hh"
#include "HybridSort.hh"
```

Komponenty

- class QuickSortOpt< typ >

  Definicja klasy QuickSortOpt.

5.16 Dokumentacja pliku Statystyka.cpp

Zawiera definicję metod klasy Statystyka.

```
#include "../inc/Statystyka.hh"
```

5.16.1 Opis szczegółowy

Plik zawiera definicję metod klasy Statystyka.

Definicja w pliku Statystyka.cpp.

5.17 Dokumentacja pliku Statystyka.hh

Zawiera definicję klasy Statystyka.

```
#include <iostream>
#include "IObserwator.hh"
#include "Stoper.hh"
#include <fstream>
#include <cstdlib>
#include <string>
```

Komponenty

- class Statystyka

  Modeluje pojęcie statystyki.

5.17.1 Opis szczegółowy

Zawiera definicję klasy Statystyka

Definicja w pliku Statystyka.hh.

5.18 Dokumentacja pliku Stoper.cpp

```
#include "../inc/Stoper.hh"
```

5.19 Dokumentacja pliku Stoper.hh

```
#include <iostream>
#include <ctime>
```

Komponenty

- class Stoper

  Klasa Stoper.

Skorowidz