

PAMSI_LAB

Wygenerowano przez Doxygen 1.8.6

Cz, 2 kwi 2015 15:46:27

Spis treści

1 Indeks hierarchiczny	1
1.1 Hierarchia klas	1
2 Indeks klas	1
2.1 Lista klas	2
3 Indeks plików	2
3.1 Lista plików	2
4 Dokumentacja klas	3
4.1 Dokumentacja szablonu klasy Benchmark< typ >	3
4.1.1 Opis szczegółowy	3
4.1.2 Dokumentacja konstruktora i destruktora	4
4.1.3 Dokumentacja funkcji składowych	4
4.1.4 Dokumentacja atrybutów składowych	4
4.2 Dokumentacja struktury Kolejka< typ >::Element	5
4.2.1 Opis szczegółowy	5
4.2.2 Dokumentacja konstruktora i destruktora	5
4.2.3 Dokumentacja atrybutów składowych	5
4.3 Dokumentacja struktury Lista< typ >::Element	5
4.3.1 Opis szczegółowy	6
4.3.2 Dokumentacja konstruktora i destruktora	6
4.3.3 Dokumentacja atrybutów składowych	6
4.4 Dokumentacja struktury Stos< typ >::Element	6
4.4.1 Opis szczegółowy	7
4.4.2 Dokumentacja konstruktora i destruktora	7
4.4.3 Dokumentacja atrybutów składowych	7
4.5 Dokumentacja klasy Framework	7
4.5.1 Opis szczegółowy	8
4.5.2 Dokumentacja funkcji składowych	8
4.6 Dokumentacja szablonu klasy InterfejsADT< typ >	9
4.6.1 Opis szczegółowy	9
4.6.2 Dokumentacja funkcji składowych	9
4.7 Dokumentacja szablonu klasy Kolejka< typ >	10
4.7.1 Opis szczegółowy	11
4.7.2 Dokumentacja konstruktora i destruktora	12
4.7.3 Dokumentacja funkcji składowych	12
4.7.4 Dokumentacja atrybutów składowych	14
4.8 Dokumentacja szablonu klasy Lista< typ >	14

4.8.1	Opis szczegółowy	15
4.8.2	Dokumentacja konstruktora i destruktora	15
4.8.3	Dokumentacja funkcji składowych	15
4.8.4	Dokumentacja atrybutów składowych	18
4.9	Dokumentacja szablonu klasy ListArr1 < typ >	18
4.9.1	Opis szczegółowy	19
4.9.2	Dokumentacja konstruktora i destruktora	19
4.9.3	Dokumentacja funkcji składowych	19
4.9.4	Dokumentacja atrybutów składowych	21
4.10	Dokumentacja szablonu klasy ListArr2x< typ >	22
4.10.1	Opis szczegółowy	23
4.10.2	Dokumentacja konstruktora i destruktora	23
4.10.3	Dokumentacja funkcji składowych	23
4.10.4	Dokumentacja atrybutów składowych	24
4.11	Dokumentacja klasy Statystyka	24
4.11.1	Opis szczegółowy	25
4.11.2	Dokumentacja konstruktora i destruktora	25
4.11.3	Dokumentacja funkcji składowych	25
4.11.4	Dokumentacja atrybutów składowych	26
4.12	Dokumentacja szablonu klasy Stos< typ >	26
4.12.1	Opis szczegółowy	27
4.12.2	Dokumentacja konstruktora i destruktora	27
4.12.3	Dokumentacja funkcji składowych	27
4.12.4	Dokumentacja atrybutów składowych	28
5	Dokumentacja plików	29
5.1	Dokumentacja pliku Benchmark.hh	29
5.1.1	Opis szczegółowy	29
5.2	Dokumentacja pliku Framework.hh	29
5.2.1	Opis szczegółowy	29
5.3	Dokumentacja pliku InterfejsADT.hh	30
5.4	Dokumentacja pliku Kolejka.hh	30
5.4.1	Opis szczegółowy	30
5.5	Dokumentacja pliku Lista.hh	30
5.5.1	Opis szczegółowy	30
5.6	Dokumentacja pliku ListArr1.hh	31
5.6.1	Opis szczegółowy	31
5.7	Dokumentacja pliku ListArr2x.hh	31
5.7.1	Opis szczegółowy	31
5.8	Dokumentacja pliku main.cpp	31

5.8.1	Opis szczegółowy	32
5.8.2	Dokumentacja definicji	32
5.8.3	Dokumentacja funkcji	32
5.9	Dokumentacja pliku Pliki.cpp	32
5.9.1	Opis szczegółowy	32
5.9.2	Dokumentacja funkcji	33
5.10	Dokumentacja pliku Pliki.hh	33
5.10.1	Opis szczegółowy	33
5.10.2	Dokumentacja funkcji	34
5.11	Dokumentacja pliku Statystyka.cpp	34
5.11.1	Opis szczegółowy	34
5.12	Dokumentacja pliku Statystyka.hh	34
5.12.1	Opis szczegółowy	35
5.13	Dokumentacja pliku Stos.hh	35
5.13.1	Opis szczegółowy	35
Indeks		36

1 Indeks hierarchiczny

1.1 Hierarchia klas

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

Benchmark< typ >	3
Kolejka< typ >::Element	5
Lista< typ >::Element	5
Stos< typ >::Element	6
Framework	7
InterfejsADT< typ >	9
Kolejka< typ >	10
Lista< typ >	14
ListArr1< typ >	18
ListArr2x< typ >	22
Stos< typ >	26
Statystyka	24

2 Indeks klas

2.1 Lista klas

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

Benchmark < typ > Modeluje pojęcie Benchmarku	3
Kolejka < typ >::Element Modeluje jeden element Kolejki	5
Lista < typ >::Element Modeluje jeden element Listy	5
Stos < typ >::Element Modeluje jeden element Stosu	6
Framework Modeluje interfejs programu	7
InterfejsADT < typ >	9
Kolejka < typ > Modeluje pojęcie Kolejki	10
Lista < typ > Modeluje pojęcie listy	14
ListArr1 < typ > Modeluje pojęcie Listy (array)	18
ListArr2x < typ > Modeluje pojęcie Listy (array)	22
Statystyka Modeluje pojęcie statystyki	24
Stos < typ > Modeluje pojęcie Stosu	26

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Benchmark.hh Definicja klasy Benchmark	29
Framework.hh Definicja klasy Framework	29
InterfejsADT.hh	30
Kolejka.hh Definicja klasy Kolejka	30
Lista.hh Definicja klasy Lista	30

ListArr1.hh	
Definicja klasy ListaArr1	31
ListArr2x.hh	
Definicja klasy ListArr1	31
main.cpp	
Moduł główny programu	31
Pliki.cpp	
Definicje funkcji obsługi plików	32
Pliki.hh	
Funkcje obsługi plików	33
Statystyka.cpp	
Zawiera definicję metod klasy Statystyka	34
Statystyka.hh	
Zawiera definicję klasy Statystyka	34
Stos.hh	
Zawiera definicję Stosu	35

4 Dokumentacja klas

4.1 Dokumentacja szablonu klasy `Benchmark< typ >`

Modeluje pojęcie Benchmarku.

```
#include <Benchmark.hh>
```

Metody publiczne

- [Benchmark](#) (const unsigned int ileProb, unsigned int *const ileDanych, const unsigned int ilePowtorzen)
Konstruktor 2 argumentowy.
- void [Test](#) ([Framework](#) *f, std::string const nazwaPliku) const
Testowanie algorytmu.

Atrybuty prywatne

- [Statystyka](#) * stat
Statystyki testu.
- unsigned int [ileProb](#)
Ilość prób.
- unsigned int * [ileDanych](#)
Tablica liczności serii.
- unsigned int [ilePowtorzen](#)
Ilość powtórzeń

4.1.1 Opis szczegółowy

```
template<class typ>class Benchmark< typ >
```

Modeluje pojęcie Benchmarku czyli obiektu mierzącego czas wykonywania algoytmu

Definicja w linii 24 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 obiekt klasy [Benchmark](#) i inicjuje nową statystykę dla obiektu

Parametry

in	<i>ileProb</i>	- ilość prób, które zostaną wykonane
in	<i>ileDanych</i>	- wskaźnik na tablice z licznosciami kolejnych serii
in	<i>ilePowtorzen</i>	- ilość powtórzeń każdej serii

Definicja w linii 69 pliku Benchmark.hh.

4.1.3 Dokumentacja funkcji składowych

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

Metoda testuje algorytm w określonej liczbie serii i powtórzeniach pomiary zapisuje do pliku podanego przez użytkownika

Parametry

in	<i>f</i>	- obiekt klasy na której zostanie przeprowadzony test
in	<i>nazwaPliku</i>	- nazwa pliku do którego zostaną zapisane statystyki

Definicja w linii 86 pliku Benchmark.hh.

4.1.4 Dokumentacja atrybutów składowych

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

Tablica z licznosciami elementów dla kolejnych serii

Definicja w linii 47 pliku Benchmark.hh.

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

Ilość powtórzeń każdej serii

Definicja w linii 55 pliku Benchmark.hh.

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

Ilość powtórzeń każdej serii

Definicja w linii 39 pliku Benchmark.hh.

4.1.4.4 `template<class typ> Statystyka* Benchmark< typ >::stat [private]`

Pole przechowuje wyniki testów

Definicja w linii 31 pliku Benchmark.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [Benchmark.hh](#)

4.2 Dokumentacja struktury Kolejka< typ >::Element

Modeluje jeden element Kolejki.

Metody publiczne

- [Element](#) (typ k)
Konstruktor daną przekazywaną w argumencie.

Atrybuty publiczne

- typ [wartosc](#)
Wartosc Elementu.
- [Element](#) * [nastepny](#)
Wskaźnik na kolejny [Element](#) Kolejki.

4.2.1 Opis szczegółowy

```
template<class typ>struct Kolejka< typ >::Element
```

Modeluje jeden nierozłączny element Kolejki - przechowywaną daną oraz wskaźnik na następny element;

Definicja w linii 34 pliku Kolejka.hh.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 `template<class typ > Kolejka< typ >::Element::Element (typ k) [inline]`

Konstruktor zapisujący w Elemencie na końcu Kolejki daną podaną w argumencie i ustawiający wskaźnik na NULL

Parametry

<code>in</code>	<code>k</code>	- dana która ma zostać dodana na koniec Kolejki
-----------------	----------------	---

Definicja w linii 60 pliku Kolejka.hh.

4.2.3 Dokumentacja atrybutów składowych

4.2.3.1 `template<class typ > Element* Kolejka< typ >::Element::nastepny`

Wskaźnik na kolejny [Element](#) Kolejki

Definicja w linii 49 pliku Kolejka.hh.

4.2.3.2 `template<class typ > typ Kolejka< typ >::Element::wartosc`

Wartość Elementu - przechowywanej wartości przez dany [Element](#) Kolejki

Definicja w linii 42 pliku Kolejka.hh.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [Kolejka.hh](#)

4.3 Dokumentacja struktury Lista< typ >::Element

Modeluje jeden element Listy.

Metody publiczne

- [Element](#) (typ k)

Konstruktor daną przekazywaną w argumencie.

Atrybuty publiczne

- typ [wartosc](#)

Wartosc Elementu.

- [Element](#) * [nastepny](#)

Wskaźnik na kolejny [Element](#) Listy.

4.3.1 Opis szczegółowy

```
template<class typ>struct Lista< typ >::Element
```

Modeluje jeden nierozłączny element listy - przechowywaną daną oraz wskaźnik na następny element;

Definicja w linii 33 pliku Lista.hh.

4.3.2 Dokumentacja konstruktora i destruktor

4.3.2.1 `template<class typ> Lista< typ >::Element::Element (typ k) [inline]`

Konstruktor zapisujący w Elemencie na końcu Listy daną podaną w argumencie i ustawiający wskaźnik na NULL

Parametry

<code>in</code>	<code>k</code>	- dana która ma zostać dodana na koniec Listy
-----------------	----------------	---

Definicja w linii 59 pliku Lista.hh.

4.3.3 Dokumentacja atrybutów składowych

4.3.3.1 `template<class typ> Element* Lista< typ >::Element::nastepny`

Wskaźnik na kolejny [Element](#) Listy

Definicja w linii 48 pliku Lista.hh.

4.3.3.2 `template<class typ> typ Lista< typ >::Element::wartosc`

Wartość Elementu - przechowywanej wartości przez dany [Element](#) listy

Definicja w linii 41 pliku Lista.hh.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [Lista.hh](#)

4.4 Dokumentacja struktury Stos< typ >::Element

Modeluje jeden element Stosu.

Metody publiczne

- [Element](#) (typ k)

Konstruktor daną przekazywaną w argumentach.

Atrybuty publiczne

- typ [wartosc](#)

Wartosc Elementu.

- [Element](#) * [nastepny](#)

Wskaźnik na kolejny [Element](#) Stosu.

4.4.1 Opis szczegółowy

```
template<class typ>struct Stos< typ >::Element
```

Modeluje jeden nierozłączny element Stosu - przechowywaną daną oraz wskaźnik na następny element;

Definicja w linii 30 pliku Stos.hh.

4.4.2 Dokumentacja konstruktora i destruktor

4.4.2.1 `template<class typ > Stos< typ >::Element::Element (typ k) [inline]`

Konstruktor zapisujący w Elemencie na końcu Listy daną podaną w argumentach i ustawiający wskaźnik na NULL

Parametry

<code>in</code>	<code>k</code>	- dana która ma zostać dodana na koniec Stosu
-----------------	----------------	---

Definicja w linii 56 pliku Stos.hh.

4.4.3 Dokumentacja atrybutów składowych

4.4.3.1 `template<class typ > Element* Stos< typ >::Element::nastepny`

Wskaźnik na kolejny [Element](#) Stosu

Definicja w linii 45 pliku Stos.hh.

4.4.3.2 `template<class typ > typ Stos< typ >::Element::wartosc`

Wartość Elementu - przechowywanej wartości przez dany [Element](#) Stosu

Definicja w linii 38 pliku Stos.hh.

Dokumentacja dla tej struktury została wygenerowana z pliku:

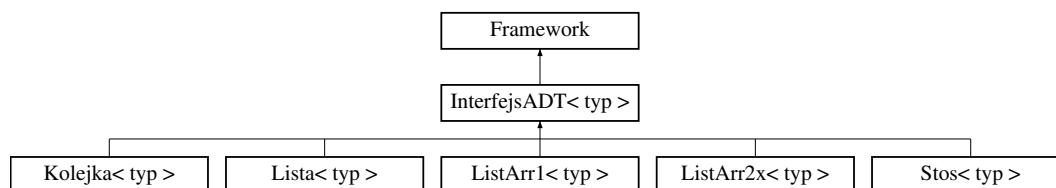
- [Stos.hh](#)

4.5 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.

4.5.1 Opis szczegółowy

Modeluje interfejs do programów wykonywanych w ramach kursu.

Definicja w linii 24 pliku Framework.hh.

4.5.2 Dokumentacja funkcji składowych

4.5.2.1 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 [Lista<typ>](#), [Kolejka<typ>](#), [Stos<typ>](#), [ListArr2x<typ>](#), [ListArr1<typ>](#) i [InterfejsADT<typ>](#).

4.5.2.2 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	<i>nazwaPliku</i>	- nazwa pliku z danymi
in	<i>n</i>	- ilość danych do wczytania

Implementowany w [Lista<typ>](#), [Kolejka<typ>](#), [ListArr2x<typ>](#), [Stos<typ>](#), [ListArr1<typ>](#) i [InterfejsADT<typ>](#).

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

Zwalnia pamięć zajmowaną przez obiekty wykorzystane do testów

Implementowany w [ListArr2x<typ>](#), [ListArr1<typ>](#), [Kolejka<typ>](#), [Lista<typ>](#), [Stos<typ>](#) i [InterfejsADT<typ>](#).

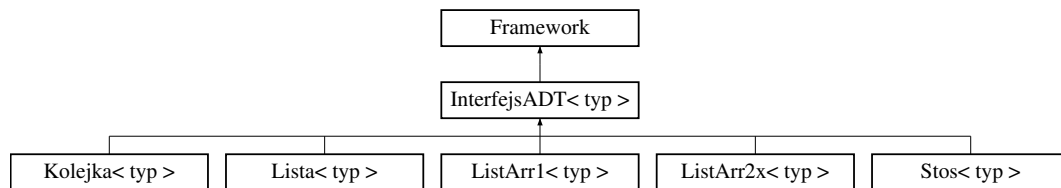
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [Framework.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 typ **pop** (const unsigned int pole)=0
Pobiera element.
- virtual unsigned int **size** () const =0
Liczność elementów.
- 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 typ InterfejsADT< typ >::pop (const unsigned int pole) [pure virtual]`

Pobiera element z typu danych

Parametry

in	<i>pole</i>	- !!!DOSTEPNE TYLKO DLA LISTY!!! nr pola z ktore pobiera element
----	-------------	--

Zwracane wartości

<i>zwraca</i>	wartość danego elementu
---------------	-------------------------

Implementowany w `List< typ >`, `Kolejka< typ >`, `Stos< typ >`, `ListArr2x< typ >` i `ListArr1< 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	<i>dana</i>	- element który chcemy dorzucić do naszego typu
in	<i>pole</i>	- !!!DOSTĘPNE TYLKO DLA LISTY!!! nr pola na które chcemy dodać element

Implementowany w [Kolejka< typ >](#), [Lista< typ >](#), [Stos< typ >](#), [ListArr1< typ >](#) i [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

<i>zwraca</i>	ilość przechowywanych elementów
---------------	---------------------------------

Implementowany w [Lista< typ >](#), [Kolejka< typ >](#), [Stos< typ >](#), [ListArr2x< typ >](#) i [ListArr1< 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	<i>k</i>	- ilość elementów dla których mają zostać wykonane obliczenia.
----	----------	--

Implementuje [Framework](#).

Implementowany w [Lista< typ >](#), [Kolejka< typ >](#), [Stos< typ >](#), [ListArr2x< typ >](#) i [ListArr1< 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	<i>nazwaPliku</i>	- nazwa pliku z danymi
in	<i>n</i>	- ilość danych do wczytania

Implementuje [Framework](#).

Implementowany w [Lista< typ >](#), [Kolejka< typ >](#), [ListArr2x< typ >](#), [Stos< typ >](#) i [ListArr1< 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 >](#), [ListArr1< typ >](#), [Kolejka< typ >](#), [Lista< typ >](#) i [Stos< typ >](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

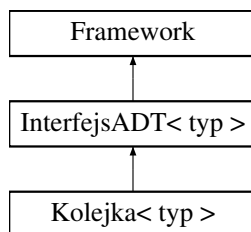
- [InterfejsADT.hh](#)

4.7 Dokumentacja szablonu klasy [Kolejka< typ >](#)

Modeluje pojęcie Kolejki.

```
#include <Kolejka.hh>
```

Diagram dziedziczenia dla [Kolejka< typ >](#)



Komponenty

- struct [Element](#)
Modeluje jeden element Kolejki.

Metody publiczne

- [Kolejka](#) ()
Konstruktor pustej Kolejki.
- void [Zwolnij](#) ()
Destruktor Kolejki.
- void [push](#) (const typ dana, const unsigned int pole=0)
Dodaje daną do Kolejki.
- void [pop](#) (const unsigned int pole=0)
Usuwa element z Kolejki.
- unsigned int [size](#) () const
Sprawdza rozmiar Kolejki.
- void [WczytajDane](#) (const char *nazwaPliku, unsigned int n)
Wczytuje dane z pliku.
- void [Start](#) (const unsigned int k)
Proces obliczeniowy.

Atrybuty prywatne

- [Element](#) * [Poczatek](#)
Wskaźnik na pierwszy element Kolejki.
- [Element](#) * [Koniec](#)
Wskaźnik na ostatni element Kolejki.
- unsigned int [Rozmiar](#)
Aktualny rozmiar Kolejki.

4.7.1 Opis szczegółowy

```
template<class typ>class Kolejka< typ >
```

Modeluje pojęcie Kolejki zadeklarowanego w szablonie typu Uwaga! Kolejkę indeksujemy od 0.

Definicja w linii 25 pliku Kolejka.hh.

4.7.2 Dokumentacja konstruktora i destruktora

4.7.2.1 `template<class typ > Kolejka< typ >::Kolejka () [inline]`

Konstruktor bezargumentowy pustej Kolejki tworzy obiekt z wskaźnikiem początek pokazującym na NULL.

Definicja w linii 100 pliku Kolejka.hh.

4.7.3 Dokumentacja funkcji składowych

4.7.3.1 `template<class typ > void Kolejka< typ >::pop (const unsigned int pole = 0) [inline],[virtual]`

Usuwa pierwszy element z Kolejki UWAGA! Nie zmieniać drugiego argumentu wywołania, bądź ustawić 0!

Parametry

in	<i>pole</i>	- numer elementu w Kolejce który wyrzucimy, domyślnie 0, zmiana podczas wywołania nie ma wpływu na działanie metody;
----	-------------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 173 pliku Kolejka.hh.

4.7.3.2 `template<class typ > void Kolejka< typ >::push (const typ dana, const unsigned int pole = 0) [inline],[virtual]`

Dodaje daną podaną jako pierwszy argument wywołania na koniec Kolejki Uwaga! nie zmieniać drugiego argumentu wywołania!

Parametry

in	<i>dana</i>	- dana którą chcemy dodać do Kolejki
in	<i>pole</i>	- numer miejsca gdzie zostanie dodany element - domyślnie koniec kolejki, zmiana argumentu podczas wywołania nie wpływa na działanie metody.

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 146 pliku Kolejka.hh.

4.7.3.3 `template<class typ > unsigned int Kolejka< typ >::size () const [inline],[virtual]`

Sprawdza ile aktualnie elementów znajduje się w Kolejce

Zwracane wartości

<i>zwraca</i>	ilość elementów znajdujących się aktualnie w Kolejce
---------------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 194 pliku Kolejka.hh.

4.7.3.4 `template<class typ > void Kolejka< typ >::Start (const unsigned int k) [inline],[virtual]`

Wykonuje proces obliczeniowy, którego czas wykonania jest mierzony na potrzeby laboratoriów PAMSI W tym wypadku tworzy Kolejke k elementową wypełnioną stałą liczbą '3'.

Parametry

in	<i>k</i>	- ilość danych dla których ma zostać przeprowadzona procedura obliczeniowa
----	----------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 220 pliku Kolejka.hh.

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

Wczytuje dane zamieszczone w pliku do Kolejki. Każdą nową daną umieszcza na końcu Kolejki.

Parametry

in	<i>nazwaPliku</i>	- nazwa pliku z danymi
in	<i>n</i>	- ilość danych do wczytania

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 206 pliku Kolejka.hh.

4.7.3.6 `template<class typ> void Kolejka< typ >::Zwolnij () [inline], [virtual]`

Zwalnia zaalokowana przez Kolejke pamiec

Zwalnia pamięć

Zwalnia pamięć zajmowaną przez Kolejke

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 124 pliku Kolejka.hh.

4.7.4 Dokumentacja atrybutów składowych

4.7.4.1 `template<class typ> Element* Kolejka< typ >::Koniec [private]`

Wskaźnik na ostatni element kolejki zwiększający szybkość dodawania danych na końcu

Definicja w linii 81 pliku Kolejka.hh.

4.7.4.2 `template<class typ> Element* Kolejka< typ >::Poczatek [private]`

Wskaźnik na pierwszy element Kolejki

Definicja w linii 72 pliku Kolejka.hh.

4.7.4.3 `template<class typ> unsigned int Kolejka< typ >::Rozmiar [private]`

Przechowuje aktualną ilość Elementów znajdujących się w Kolejce

Definicja w linii 88 pliku Kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

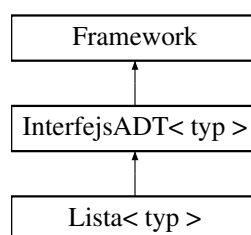
- [Kolejka.hh](#)

4.8 Dokumentacja szablonu klasy Lista< typ >

Modeluje pojęcie listy.

```
#include <Lista.hh>
```

Diagram dziedziczenia dla Lista< typ >



Komponenty

- struct [Element](#)
Modeluje jeden element Listy.

Metody publiczne

- [Lista](#) ()
Konstruktor puste listy.
- void [Zwolnij](#) ()
Destruktor listy.
- void [push](#) (const typ dana, const unsigned int pole)
Dodaje daną do Listy.
- typ [pop](#) (const unsigned int pole)
Usuwa element z Listy.
- unsigned int [size](#) () const
Sprawdza rozmiar Listy.
- void [WczytajDane](#) (const char *nazwaPliku, unsigned int n=0)
Wczytuje dane z pliku.
- typ [operator\[\]](#) (const size_t pole) const
Wyciąga wartość elementu Listy.
- void [Start](#) (const unsigned int k)
Proces obliczeniowy.

Atrybuty prywatne

- [Element](#) * [Początek](#)
Wskaźnik na pierwszy element Listy.
- [Element](#) * [Koniec](#)
Wskaźnik na ostatni element listy.
- unsigned int [Rozmiar](#)
Aktualny rozmiar Listy.

4.8.1 Opis szczegółowy

```
template<class typ>class Lista< typ >
```

Modeluje pojęcie listy zadeklarowanego w szablonie typu Uwaga! Listę indeksujemy od 0.

Definicja w linii 24 pliku Lista.hh.

4.8.2 Dokumentacja konstruktora i destruktoru

```
4.8.2.1 template<class typ> Lista< typ >::Lista ( ) [inline]
```

Konstruktor bezargumentowy pustej listy tworzy obiekt z wskaźnikiem początek pokazującym na NULL.

Definicja w linii 98 pliku Lista.hh.

4.8.3 Dokumentacja funkcji składowych

```
4.8.3.1 template<class typ> typ Lista< typ >::operator[] ( const size_t pole ) const [inline]
```

Wyluskuje wartość danego elementu z Listy

Parametry

in	<i>pole</i>	- "indeks" z którego chcemy pobrać wartość indeksujemy od 0!
----	-------------	--

Zwracane wartości

-	zwraca wartość elementu z danego pola lub '-1' w przypadku błędu
---	--

Definicja w linii 284 pliku Lista.hh.

4.8.3.2 `template<class typ> typ Lista< typ >::pop (const unsigned int pole) [inline],[virtual]`

Usuwa interesujący nas element z Listy. Jeżeli chcesz usunąć pierwszy element wywołaj pole nr '0'. Dla ostatniego elementu wywołaj pole nr '[Lista.size\(\)-1](#)'.

Parametry

in	<i>pole</i>	- numer elementu Listy z którego chcemy pobrać daną
----	-------------	---

Zwracane wartości

<i>zwraca</i>	wartość danego elementu listy lub '-1' w przypadku błędu
---------------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 190 pliku Lista.hh.

4.8.3.3 `template<class typ> void Lista< typ >::push (const typ dana, const unsigned int pole) [inline],[virtual]`

Dodaje daną podaną jako pierwszy argument wywołania na określone drugim argumentem miejsce w Liście

Parametry

in	<i>dana</i>	- daną którą chcemy dodać do listy
in	<i>pole</i>	- numer elementu listy na który chcemy dodać daną (szieze() jeżeli na koniec)

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 142 pliku Lista.hh.

4.8.3.4 `template<class typ> unsigned int Lista< typ >::size () const [inline],[virtual]`

Sprawdza ile aktualnie elementów znajdują się na Liście

Zwracane wartości

<i>zwraca</i>	ilość elementów znajdujących się aktualnie na liście
---------------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 240 pliku Lista.hh.

4.8.3.5 `template<class typ> void Lista< typ >::Start (const unsigned int k) [inline],[virtual]`

Wykonuje proces obliczeniowy, którego czas wykonania jest mierzony na potrzeby laboratoriów PAMSI W tym wypadku tworzy Listę k elementową wypełnioną stałą liczbą '3'.

Parametry

in	<i>k</i>	- ilość danych dla których ma zostać przeprowadzona procedura obliczeniowa
----	----------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 306 pliku Lista.hh.

```
4.8.3.6  template<class typ> void Lista< typ >::WczytajDane ( const char * nazwaPliku, unsigned int n = 0 )  
        [inline],[virtual]
```

Wczytuje dane zamieszczone w pliku do Listy. Każdą nową daną umieszcza na końcu listy.

Parametry

in	<i>nazwaPliku</i>	- nazwa pliku z danymi
in	<i>n</i>	- ilość danych do wczytania (domyślnie 0 - wszystkie dane z pliku, zmiana wartości nie ma wpływu na działanie metody w aktualnej wersji)

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 254 pliku Lista.hh.

4.8.3.7 `template<class typ> void Lista< typ >::Zwolnij () [inline], [virtual]`

Zwalnia zaalokowana przez listę pamięć

Zwalnia pamięć

Zwalnia pamięć zajmowaną przez listę

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 122 pliku Lista.hh.

4.8.4 Dokumentacja atrybutów składowych

4.8.4.1 `template<class typ> Element* Lista< typ >::Koniec [private]`

Wskaźnik na ostatni element listy

Definicja w linii 79 pliku Lista.hh.

4.8.4.2 `template<class typ> Element* Lista< typ >::Poczatek [private]`

Wskaźnik na pierwszy element Listy

Definicja w linii 71 pliku Lista.hh.

4.8.4.3 `template<class typ> unsigned int Lista< typ >::Rozmiar [private]`

Przechowuje aktualną ilość Elementów znajdujących się na Liście

Definicja w linii 86 pliku Lista.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

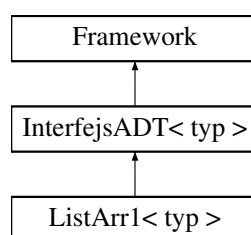
- [Lista.hh](#)

4.9 Dokumentacja szablonu klasy ListArr1< typ >

Modeluje pojęcie Listy (array)

```
#include <ListArr1.hh>
```

Diagram dziedziczenia dla ListArr1< typ >



Metody publiczne

- `ListArr1 ()`
Konstruktor bezargumentowy.
- `void push (const typ dana, const unsigned int pole)`
Dodaje element do ListyArr1.
- `typ pop (const unsigned int pole)`
Pobiera element z ListyArr1.
- `unsigned int size () const`
Wielkość listy.
- `void Start (const unsigned int k)`
Metoda testująca czas.
- `void WczytajDane (const char *nazwaPliku, unsigned int n)`
Wczytuje dane z pliku.
- `void Zwolnij ()`
Zwalnia pamięć

Atrybuty prywatne

- `typ * tab`
Wskaźnik na dynamiczną tablicę
- `unsigned int RozmiarT`
Rozmiar tablicy.
- `unsigned int RozmiarL`
Rozmiar Listy.

4.9.1 Opis szczegółowy

```
template<class typ>class ListArr1< typ >
```

Modeluje pojęcie Listy opartej na dynamicznej tablicy. Dodając elementy zwiększa tablicę o 1.

Definicja w linii 20 pliku ListArr1.hh.

4.9.2 Dokumentacja konstruktora i destruktoru

```
4.9.2.1 template<class typ> ListArr1< typ >::ListArr1 ( ) [inline]
```

Konstruktor alokujący tablicę jednoelementową z której będzie tworzona lista

Definicja w linii 55 pliku ListArr1.hh.

4.9.3 Dokumentacja funkcji składowych

```
4.9.3.1 template<class typ> typ ListArr1< typ >::pop ( const unsigned int pole ) [inline],[virtual]
```

Pobiera element z Listy Arr1 usuwając go z niej i zmniejszając rozmiar.

param[in] - pole - nr pola z którego chcemy pobrać element

retval - zwraca wartosc pobranej danej lub '-1' w przypadku bledu

Implementuje `InterfejsADT< typ >`.

Definicja w linii 104 pliku ListArr1.hh.

4.9.3.2 `template<class typ> void ListArr1< typ >::push (const typ dana, const unsigned int pole) [inline],
[virtual]`

Dodaje nowy element do ListyArr1

Parametry

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

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 72 pliku ListArr1.hh.

4.9.3.3 `template<class typ> unsigned int ListArr1< typ >::size () const [inline],[virtual]`

Informuje o ilości elementów znajdujących się na LiścieArr1

Zwracane wartości

-	zwraca liczbę elementów ListyArr1
---	-----------------------------------

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 137 pliku ListArr1.hh.

4.9.3.4 `template<class typ> void ListArr1< typ >::Start (const unsigned int k) [inline],[virtual]`

Metoda testująca czas wczytania n elementów na ListęArr1

Parametry

in	<i>k</i>	- ilość elementów do wczytania
----	----------	--------------------------------

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 147 pliku ListArr1.hh.

4.9.3.5 `template<class typ> void ListArr1< typ >::WczytajDane (const char * nazwaPliku, unsigned int n) [inline],[virtual]`

Wczytuje dane z pliku do [ListArr1](#)

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 161 pliku ListArr1.hh.

4.9.3.6 `template<class typ> void ListArr1< typ >::Zwolnij () [inline],[virtual]`

Zwalnia pamięć zaalokowaną przez [ListArr1](#)

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 169 pliku ListArr1.hh.

4.9.4 Dokumentacja atrybutów składowych

4.9.4.1 `template<class typ> unsigned int ListArr1< typ >::RozmiarL [private]`

Aktualny rozmiar ListyArr1

Definicja w linii 44 pliku ListArr1.hh.

4.9.4.2 `template<class typ> unsigned int ListArr1< typ >::RozmiarT [private]`

Aktualny rozmiar tablicy.

Definicja w linii 36 pliku ListArr1.hh.

4.9.4.3 `template<class typ> typ* ListArr1< typ >::tab [private]`

Wskaźnik na dynamiczną tablicę tworzącą ListęArr1

Definicja w linii 28 pliku ListArr1.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

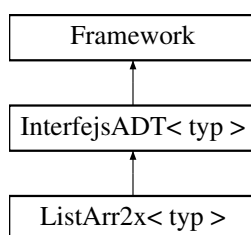
- [ListArr1.hh](#)

4.10 Dokumentacja szablonu klasy ListArr2x< typ >

Modeluje pojęcie Listy (array)

```
#include <ListArr2x.hh>
```

Diagram dziedziczenia dla ListArr2x< typ >



Metody publiczne

- [ListArr2x \(\)](#)
Konstruktor bezargumentowy.
- void [push](#) (const typ dana, const unsigned int pole)
Dodaje element do ListyArr1.
- typ [pop](#) (const unsigned int pole)
Pobiera element z ListyArr1.
- unsigned int [size](#) () const
Wielkość listy.
- void [Start](#) (const unsigned int k)
Metoda testująca czas.
- void [WczytajDane](#) (const char *nazwaPliku, unsigned int n)
Wczytuje dane z pliku.
- void [Zwolnij](#) ()
Zwalnia pamięć

Atrybuty prywatne

- typ * [tab](#)
Wskaźnik na dynamiczną tablicę
- unsigned int [RozmiarT](#)
Rozmiar tablicy.
- unsigned int [RozmiarL](#)
Rozmiar Listy.

4.10.1 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 20 pliku ListArr2x.hh.

4.10.2 Dokumentacja konstruktora i destruktor

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

Konstruktor alokujący tablicę jednoelementową z której będzie tworzona lista

Definicja w linii 55 pliku ListArr2x.hh.

4.10.3 Dokumentacja funkcji składowych

```
4.10.3.1 template<class typ> typ 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 którego chcemy pobrać element (indeksowane od 0)

retval - zwraca wartość pobranej danej lub '-1' w przypadku błędu

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 122 pliku ListArr2x.hh.

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

Dodaje nowy element do ListyArr1

Parametry

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

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 72 pliku ListArr2x.hh.

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

Informuje o ilości elementów znajdujących się na LiścieArr1

Zwracane wartości

-	zwraca liczbę elementów ListyArr1
---	-----------------------------------

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 172 pliku ListArr2x.hh.

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

Metoda testująca czas wczytania n elementów na ListęArr1

Parametry

<code>in</code>	<code>k</code>	- ilość elementów do wczytania
-----------------	----------------	--------------------------------

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 182 pliku ListArr2x.hh.

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

Wczytuje dane z pliku do [ListArr1](#)

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 196 pliku ListArr2x.hh.

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

Zwalnia pamięć zaalokowaną przez [ListArr1](#)

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 204 pliku ListArr2x.hh.

4.10.4 Dokumentacja atrybutów składowych

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

Aktualny rozmiar ListyArr2x

Definicja w linii 44 pliku ListArr2x.hh.

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

Aktualny rozmiar tablicy.

Definicja w linii 36 pliku ListArr2x.hh.

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

Wskaźnik na dynamiczną tablicę tworzącą ListęArr2x

Definicja w linii 28 pliku ListArr2x.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [ListArr2x.hh](#)

4.11 Dokumentacja klasy Statystyka

Modeluje pojęcie statystyki.

```
#include <Statystyka.hh>
```

Metody publiczne

- [Statystyka](#) (const unsigned int iloscProb, unsigned int *proby)
Konstruktor z dwoma parametrami.
- [~Statystyka](#) ()

Destruktor - zwalnia pamięć

- double & `operator[]` (unsigned int i)
Indeksuje tablicę czasową
- void `ZapiszStaty` (std::string nazwaPliku)
Zapisuje statystykę do pliku.

Atrybuty prywatne

- unsigned int `IleProb`
Ilość prób.
- unsigned int * `Proba`
Tablica z rozmiarami prób.
- double * `Czas`
Średni czas wykonania danej próby.

4.11.1 Opis szczegółowy

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

Definicja w linii 22 pliku Statystyka.hh.

4.11.2 Dokumentacja konstruktora i destruktora

4.11.2.1 Statystyka::Statystyka (const unsigned int *iloscProb*, unsigned int * *proby*)

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

Parametry

in	<i>iloscProb</i>	- liczba prob w ksperymentcie
in	<i>proby</i>	- tablica z licznosciami prób.

Definicja w linii 14 pliku Statystyka.cpp.

4.11.2.2 Statystyka::~Statystyka () [inline]

Zwalnia pamięć zaalokowaną na dynamiczne tablice przechowujące statystykę.

Definicja w linii 68 pliku Statystyka.hh.

4.11.3 Dokumentacja funkcji składowych

4.11.3.1 double& Statystyka::operator[] (unsigned int *i*) [inline]

Zwraca referencję do i-tego indeksu tablicy czasowej.

Parametry

in	<i>i</i>	- indeks tablicy czasowej
----	----------	---------------------------

Zwracane wartości

<i>Czas[i]</i>	referencja do wybranego indeksu
----------------	---------------------------------

Definicja w linii 80 pliku Statystyka.hh.

4.11.3.2 void Statystyka::ZapiszStaty (std::string nazwaPliku)

Zapisuje statystykę do pliku o nazwie "statystyka.dat". Pierwsza linia pliku to wielkości prób druga to średnie czasy wykonania podane w ms;

Definicja w linii 22 pliku Statystyka.cpp.

4.11.4 Dokumentacja atrybutów składowych

4.11.4.1 double* Statystyka::Czas [private]

wskaźnik na tablicę ze średnimi czasami wykonania kolejnych prób.

Definicja w linii 46 pliku Statystyka.hh.

4.11.4.2 unsigned int Statystyka::IleProb [private]

Ilość prób do utworzenia statystyki

Definicja w linii 30 pliku Statystyka.hh.

4.11.4.3 unsigned int* Statystyka::Proba [private]

Wskaźnik na tablicę zawierającą wielkości danych prób.

Definicja w linii 38 pliku Statystyka.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

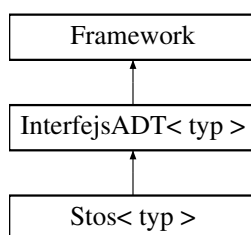
- [Statystyka.hh](#)
- [Statystyka.cpp](#)

4.12 Dokumentacja szablonu klasy Stos< typ >

Modeluje pojęcie Stosu.

```
#include <Stos.hh>
```

Diagram dziedziczenia dla Stos< typ >



Komponenty

- struct [Element](#)
Modeluje jeden element Stosu.

Metody publiczne

- [Stos](#) ()
Konstruktor pustego Stosu.
- void [Zwolnij](#) ()

Destruktor Stosu.

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

Dodaje daną do Listy.

- void **pop** (const unsigned int pole=0)

Usuwa element ze Stosu.

- unsigned int **size** () const

Sprawdza rozmiar Stosu.

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

Wczytuje dane z pliku.

- void **Start** (const unsigned int k)

Proces obliczeniowy.

Atrybuty prywatne

- **Element** * **Początek**

Wskaźnik na pierwszy element Stosu.

- unsigned int **Rozmiar**

Aktualny rozmiar Stosu.

4.12.1 Opis szczegółowy

`template<class typ>class Stos< typ >`

Modeluje pojęcie Stosu.

Definicja w linii 22 pliku Stos.hh.

4.12.2 Dokumentacja konstruktora i destruktora

4.12.2.1 `template<class typ > Stos< typ >::Stos () [inline]`

Konstruktor bezargumentowy pustego Stosu tworzy obiekt z wskaźnikiem początek pokazującym na NULL.

Definicja w linii 88 pliku Stos.hh.

4.12.3 Dokumentacja funkcji składowych

4.12.3.1 `template<class typ > void Stos< typ >::pop (const unsigned int pole = 0) [inline],[virtual]`

Usuwa 'górny' element Stosu

Parametry

<code>in</code>	<code>pole</code>	- numer elementu Listy z którego chcemy pobrać daną
-----------------	-------------------	---

Implementuje **InterfejsADT< typ >**.

Definicja w linii 151 pliku Stos.hh.

4.12.3.2 `template<class typ > void Stos< typ >::push (const typ dana, const unsigned int pole = 0) [inline],[virtual]`

Dodaje daną podaną jako argument wywołania

Parametry

in	<i>dana</i>	- dana którą chcemy dodać do Stosu
in	<i>pole</i>	- numer elementu Stosu na który chcemy dodać daną, domyślnie - 0, zmiana argumentu wywołania nie ma wpływu na działanie metody

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 132 pliku Stos.hh.

4.12.3.3 `template<class typ > unsigned int Stos< typ >::size () const [inline],[virtual]`

Sprawdza ile aktualnie elementów znajduje się na Stosie

Zwracane wartości

<i>zwraca</i>	ilość elementów znajdujących się aktualnie na Stosie
---------------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 177 pliku Stos.hh.

4.12.3.4 `template<class typ > void Stos< typ >::Start (const unsigned int k) [inline],[virtual]`

Wykonuje proces obliczeniowy, którego czas wykonania jest mierzony na potrzeby laboratoriów PAMSI W tym wypakdu tworzy [Stos](#) k elementowy wypełniony stałą liczbą '3'.

Parametry

in	<i>k</i>	- ilość danych dla których ma zostać przeprowadzona procedura obliczeniowa
----	----------	--

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 203 pliku Stos.hh.

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

Wczytuje dane zamieszczone w pliku do Stosu. Każdą nową daną umieszcza na 'górze' Stosu.

Parametry

in	<i>nazwaPliku</i>	- nazwa pliku z danymi
in	<i>n</i>	- ilość danych do wczytania

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 189 pliku Stos.hh.

4.12.3.6 `template<class typ > void Stos< typ >::Zwolnij () [inline],[virtual]`

Zwalnia zaalokowana przez [Stos](#) pamięć

Zwalnia pamięć

Zwalnia pamięć zajmowaną przez [Stos](#)

Implementuje [InterfejsADT< typ >](#).

Definicja w linii 112 pliku Stos.hh.

4.12.4 Dokumentacja atrybutów składowych

4.12.4.1 `template<class typ > Element* Stos< typ >::Poczatek [private]`

Wskaźnik na pierwszy element Stosu

Definicja w linii 68 pliku Stos.hh.

4.12.4.2 `template<class typ > unsigned int Stos< typ >::Rozmiar [private]`

Przechowuje aktualną ilość Elementów znajdujących się na Stosie

Definicja w linii 76 pliku Stos.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [Stos.hh](#)

5 Dokumentacja plików

5.1 Dokumentacja pliku Benchmark.hh

Definicja klasy [Benchmark](#).

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

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 InterfejsADT.hh

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

Komponenty

- class [InterfejsADT< typ >](#)

5.4 Dokumentacja pliku Kolejka.hh

Definicja klasy [Kolejka](#).

```
#include "InterfejsADT.hh"
#include "Pliki.hh"
#include <ctime>
```

Komponenty

- class [Kolejka< typ >](#)
Modeluje pojęcie Kolejki.
- struct [Kolejka< typ >::Element](#)
Modeluje jeden element Kolejki.

5.4.1 Opis szczegółowy

Plik zawiera definicję klasy [Kolejka](#) ujętej w szablon typu przechowujących zmiennych więc zawiera też definicję metod klasy.

Definicja w pliku [Kolejka.hh](#).

5.5 Dokumentacja pliku Lista.hh

Definicja klasy [Lista](#).

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

Komponenty

- class [Lista< typ >](#)
Modeluje pojęcie listy.
- struct [Lista< typ >::Element](#)
Modeluje jeden element Listy.

5.5.1 Opis szczegółowy

Plik zawiera definicję klasy lista ujętej w szablon typu przechowujących zmiennych więc zawiera też definicję metod klasy.

Definicja w pliku [Lista.hh](#).

5.6 Dokumentacja pliku ListArr1.hh

Definicja klasy ListaArr1.

```
#include "InterfejsADT.hh"
```

Komponenty

- class [ListArr1< typ >](#)
Modeluje pojęcie Listy (array)

5.6.1 Opis szczegółowy

Plik zawiera definicję klasy ListaArr1 ujętej w szablon typu wraz z jej składowymi metodami.

Definicja w pliku [ListArr1.hh](#).

5.7 Dokumentacja pliku ListArr2x.hh

Definicja klasy [ListArr1](#).

```
#include "InterfejsADT.hh"
```

Komponenty

- class [ListArr2x< typ >](#)
Modeluje pojęcie Listy (array)

5.7.1 Opis szczegółowy

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

Definicja w pliku [ListArr2x.hh](#).

5.8 Dokumentacja pliku main.cpp

Moduł główny programu.

```
#include "../inc/Lista.hh"  
#include "../inc/Stos.hh"  
#include "../inc/Kolejka.hh"  
#include "../inc/ListArr1.hh"  
#include "../inc/ListArr2x.hh"  
#include "../inc/Statystyka.hh"  
#include "../inc/Benchmark.hh"
```

Definicje

- #define [ILOSC_POWTORZEN](#) 10
Ilość powtórzeń danej próby.
- #define [ILOSC_PROB](#) 3
Ilość prób.

Funkcje

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

5.8.1 Opis szczegółowy

Program wykonuje serię 10 pomiarów czasu wykonania metody start dla różnych wielkości problemu obliczeniowego, dla każdego zaimplementowanego typu danych - LinkLista, ListaArr1, ListaArr2x. Procedura obliczeniowa polega na utworzeniu 'objektu' przechowującego n danych (stałych liczb). statystykę pomiarów zapisuje do pliku o nazwie "Typ-Daych.dat". gdzie "TypDanych" to odpowiednio [Lista](#), ListaArr1 i ListaArr2x

OBSŁUGA PROGRAMU: Aby wywołać program należy w lini poleceń wywołać jego nazę np: "./a.out"

Definicja w pliku [main.cpp](#).

5.8.2 Dokumentacja definicji

5.8.2.1 #define ILOSC_POWTORZEN 10

Ilość powtórzeń danej próby

Definicja w linii 36 pliku main.cpp.

5.8.2.2 #define ILOSC_PROB 3

Ilość prób = ilość rozmiarów prób

Definicja w linii 44 pliku main.cpp.

5.8.3 Dokumentacja funkcji

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

Definicja w linii 46 pliku main.cpp.

5.9 Dokumentacja pliku Pliki.cpp

Definicje funkcji obsługi plików.

```
#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 czyszcząc jego zawartość
- void [LosujIntDoPliku](#) (const unsigned int n, const unsigned int zakres)
Zapisuje n losowych liczb(int) do pliku.

5.9.1 Opis szczegółowy

Plik zawiera definicje funkcji związanych z obsługą plików.

Definicja w pliku [Pliki.cpp](#).

5.9.2 Dokumentacja funkcji

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

Losuje *n* liczb z zakresu od 1 do podanego przez użytkownika następnie zapisuje wylosowane dane do pliku o nazwie "dane.dat"

Parametry

in	<i>n</i>	- ilość liczb do zapisania
in	<i>zakres</i>	- górny zakres wartości liczb

Definicja w linii 27 pliku Pliki.cpp.

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

Otwiera plik i sprawdza czy otwarcie się powiodło jeżeli nie to koczy program

Parametry

in	<i>nazwaPliku</i>	- nazwa pliku który chcemy otworzyć
in	<i>plik</i>	- strumień powiązany z plikiem

Definicja w linii 11 pliku Pliki.cpp.

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

Otwiera plik i sprawdza czy otwarcie się powiodło jeżeli nie to koczy program

Parametry

in	<i>nazwaPliku</i>	- nazwa pliku który chcemy otworzyć
in	<i>plik</i>	- strumień powiązany z plikiem

Definicja w linii 19 pliku Pliki.cpp.

5.10 Dokumentacja pliku Pliki.hh

Funkcje obsługi plików.

```
#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 czyszcząc jego zawartość
- void [LosujIntDoPliku](#) (const unsigned int *n*, const unsigned int *zakres*)
*Zapisuje *n* losowych liczb(int) do pliku.*

5.10.1 Opis szczegółowy

Plik zawiera deklaracje funkcji związanych z obsługą plików

Definicja w pliku [Pliki.hh](#).

5.10.2 Dokumentacja funkcji

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

Losuje *n* liczb z zakresu od 1 do podanego przez użytkownika następnie zapisuje wylosowane dane do pliku o nazwie "dane.dat"

Parametry

in	<i>n</i>	- ilość liczb do zapisania
in	<i>zakres</i>	- górny zakres wartości liczb

Definicja w linii 27 pliku Pliki.cpp.

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

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

Parametry

in	<i>nazwaPliku</i>	- nazwa pliku ktory chcemy otworzyc
in	<i>plik</i>	- strumien powiazany z plikiem

Definicja w linii 11 pliku Pliki.cpp.

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

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

Parametry

in	<i>nazwaPliku</i>	- nazwa pliku ktory chcemy otworzyc
in	<i>plik</i>	- strumien powiazany z plikiem

Definicja w linii 19 pliku Pliki.cpp.

5.11 Dokumentacja pliku Statystyka.cpp

Zawiera definicję metod klasy [Statystyka](#).

```
#include "../inc/Statystyka.hh"
#include <fstream>
#include <cstdlib>
#include <string>
```

5.11.1 Opis szczegółowy

Plik zawiera definicję metod klasy [Statystyka](#).

Definicja w pliku [Statystyka.cpp](#).

5.12 Dokumentacja pliku Statystyka.hh

Zawiera definicję klasy [Statystyka](#).

```
#include <iostream>
```

Komponenty

- class [Statystyka](#)
Modeluje pojęcie statystyki.

5.12.1 Opis szczegółowy

Zawiera definicję klasy [Statystyka](#)

Definicja w pliku [Statystyka.hh](#).

5.13 Dokumentacja pliku Stos.hh

Zawiera definicję Stosu.

```
#include "InterfejsADT.hh"
```

Komponenty

- class [Stos< typ >](#)
Modeluje pojęcie Stosu.
- struct [Stos< typ >::Element](#)
Modeluje jeden element Stosu.

5.13.1 Opis szczegółowy

Plik zawiera definicję klasy [Stos](#), oraz definicję jej metod, gdyż klasa ujęta jest w szablonie.

Definicja w pliku [Stos.hh](#).

Skorowidz

~Statystyka
Statystyka, 25

Benchmark
Benchmark, 4
IleDanych, 4
IlePowtorzen, 4
IleProb, 4
stat, 4
Test, 4

Benchmark< typ >, 3
Benchmark.hh, 29

Czas
Statystyka, 26

Element
Kolejka::Element, 5
Lista::Element, 6
Stos::Element, 7

Framework, 7
Start, 8
WczytajDane, 8
Zwolnij, 8
Framework.hh, 29

ILOSC_POWTORZEN
main.cpp, 32

ILOSC_PROB
main.cpp, 32

IleDanych
Benchmark, 4

IlePowtorzen
Benchmark, 4

IleProb
Benchmark, 4
Statystyka, 26

InterfejsADT
pop, 9
push, 9
size, 10
Start, 10
WczytajDane, 10
Zwolnij, 10

InterfejsADT< typ >, 9
InterfejsADT.hh, 30

Kolejka
Kolejka, 12
Koniec, 14
Poczatek, 14
pop, 12
push, 12
Rozmiar, 14
size, 12
Start, 12

WczytajDane, 12
Zwolnij, 14

Kolejka< typ >, 10
Kolejka< typ >::Element, 5
Kolejka.hh, 30
Kolejka::Element
Element, 5
nastepny, 5
wartosc, 5

Koniec
Kolejka, 14
Lista, 18

ListArr1
ListArr1, 19
ListArr1, 19
pop, 19
push, 19
RozmiarL, 21
RozmiarT, 21
size, 21
Start, 21
tab, 21
WczytajDane, 21
Zwolnij, 21

ListArr1< typ >, 18
ListArr1.hh, 31

ListArr2x
ListArr2x, 23
ListArr2x, 23
pop, 23
push, 23
RozmiarL, 24
RozmiarT, 24
size, 23
Start, 23
tab, 24
WczytajDane, 24
Zwolnij, 24

ListArr2x< typ >, 22
ListArr2x.hh, 31

Lista
Koniec, 18
Lista, 15
Poczatek, 18
pop, 16
push, 16
Rozmiar, 18
size, 16
Start, 16
WczytajDane, 16
Zwolnij, 18

Lista< typ >, 14
Lista< typ >::Element, 5
Lista.hh, 30

Lista::Element
 Element, [6](#)
 nastepny, [6](#)
 wartosc, [6](#)
 LosujIntDoPliku
 Pliki.cpp, [33](#)
 Pliki.hh, [34](#)

 main
 main.cpp, [32](#)
 main.cpp, [31](#)
 ILOSC_POWTORZEN, [32](#)
 ILOSC_PROB, [32](#)
 main, [32](#)

 nastepny
 Kolejka::Element, [5](#)
 Lista::Element, [6](#)
 Stos::Element, [7](#)

 OtworzPlikIn
 Pliki.cpp, [33](#)
 Pliki.hh, [34](#)
 OtworzPlikOut
 Pliki.cpp, [33](#)
 Pliki.hh, [34](#)

 Pliki.cpp, [32](#)
 LosujIntDoPliku, [33](#)
 OtworzPlikIn, [33](#)
 OtworzPlikOut, [33](#)
 Pliki.hh, [33](#)
 LosujIntDoPliku, [34](#)
 OtworzPlikIn, [34](#)
 OtworzPlikOut, [34](#)

 Poczatek
 Kolejka, [14](#)
 Lista, [18](#)
 Stos, [28](#)

 pop
 InterfejsADT, [9](#)
 Kolejka, [12](#)
 Lista, [16](#)
 ListArr1, [19](#)
 ListArr2x, [23](#)
 Stos, [27](#)

 Proba
 Statystyka, [26](#)

 push
 InterfejsADT, [9](#)
 Kolejka, [12](#)
 Lista, [16](#)
 ListArr1, [19](#)
 ListArr2x, [23](#)
 Stos, [27](#)

 Rozmiar
 Kolejka, [14](#)
 Lista, [18](#)

 Stos, [29](#)
 RozmiarL
 ListArr1, [21](#)
 ListArr2x, [24](#)
 RozmiarT
 ListArr1, [21](#)
 ListArr2x, [24](#)

 size
 InterfejsADT, [10](#)
 Kolejka, [12](#)
 Lista, [16](#)
 ListArr1, [21](#)
 ListArr2x, [23](#)
 Stos, [28](#)

 Start
 Framework, [8](#)
 InterfejsADT, [10](#)
 Kolejka, [12](#)
 Lista, [16](#)
 ListArr1, [21](#)
 ListArr2x, [23](#)
 Stos, [28](#)

 stat
 Benchmark, [4](#)
 Statystyka, [24](#)
 ~Statystyka, [25](#)
 Czas, [26](#)
 IleProb, [26](#)
 Proba, [26](#)
 Statystyka, [25](#)
 ZapiszStaty, [25](#)
 Statystyka.cpp, [34](#)
 Statystyka.hh, [34](#)
 Stos
 Poczatek, [28](#)
 pop, [27](#)
 push, [27](#)
 Rozmiar, [29](#)
 size, [28](#)
 Start, [28](#)
 Stos, [27](#)
 WczytajDane, [28](#)
 Zwolnij, [28](#)
 Stos< typ >, [26](#)
 Stos< typ >::Element, [6](#)
 Stos.hh, [35](#)
 Stos::Element
 Element, [7](#)
 nastepny, [7](#)
 wartosc, [7](#)

 tab
 ListArr1, [21](#)
 ListArr2x, [24](#)

 Test
 Benchmark, [4](#)

 wartosc

- Kolejka::Element, [5](#)
- Lista::Element, [6](#)
- Stos::Element, [7](#)
- WczytajDane
 - Framework, [8](#)
 - InterfejsADT, [10](#)
 - Kolejka, [12](#)
 - Lista, [16](#)
 - ListArr1, [21](#)
 - ListArr2x, [24](#)
 - Stos, [28](#)
- ZapiszStaty
 - Statystyka, [25](#)
- Zwolnij
 - Framework, [8](#)
 - InterfejsADT, [10](#)
 - Kolejka, [14](#)
 - Lista, [18](#)
 - ListArr1, [21](#)
 - ListArr2x, [24](#)
 - Stos, [28](#)