Lab3

Generated by Doxygen 1.8.9.1

Thu Mar 26 2015 00:11:03

Contents

Chapter 1

LAB 3 - TESTOWANIE STRUKTUR DANYCH - 209186

Obsługa następujących struktur danych:

1. Stos
    - push: wrzucenie danych na góre stosu, gdy brak miejsca, dodaje jedno miejsce
    - push_prc: wrzucenie danych na góre stosu, gdy brak miejsca, podwaja stos
    - pop: ściągnięcie + zwrócenie danych z góry stosu
    - size: rozmiar stosu

    Obsługuje wszystkie typu danych (szablony).

2.Lista

- push: wrzucenie danych na koniec listy
- push_front: wrzucenie danych na początek listy
- pop: usunięcie + zwrócenie elementu z końca listy
- pop_front: usunięcie + zwrócenie elementu z początku listy
- size: rozmiar listy

Obsługuje wszystkie typu danych (szablony).

3.Kolejka

- push: wrzucenie danych na początek kolejki
- pop: usunięcie danych z końca kolejki
- size: rozmiar kolejki

Kolejka automatycznie zwiększa liczbę miejsc, jeśli pełna i wszystko zapełnione.

Kolejka automatycznie dodaje element na wolne miejsce, jeśli jest pełna i jest coś wolnego.

Kolejka nie usuwa miejsca po zwróceniu elementu, wsadza tam 0, a miejsce obsługuje kolejne dane.

Obsługuje wszystkie typu danych (szablony).

4.Drzewo binarne

- push: wrzucenie elementu do drzewa
- pop: usuwanie węzła z argumentem podanym w wyłowaniu funkcji (aktualnie usuwa poprawnie tylko węzły bez dzieci)
- size: ilość węzłów drzewie (romiar drzewa)
- show_tree: pokazanie zawartości pierwszych 3 poziomów drzewa
- is_empty: sprawdzenie czy drzewo jest puste

Obsługuje wszystkie typu danych (szablony).

Chapter 2

Todo List

Member InputFiles::InputFiles ()
EXCEPTIONS HANDLING

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Benchmark< T > Class Template Reference

`#include <benchmark_frm.h>`

Public Member Functions

- void raport (InputFiles &files)
  Create a .xls file (excel) with file sizes and test times.
- void test (T data_structure, InputFiles &files)
  Main testing function.

Private Member Functions

- long double average (std::vector< long double >times)
  get average time from testing (10 probes)

Private Attributes

- std::vector< long double > avrTestTimes
  A container for calculated times.

5.1.1 Detailed Description

template<class T>class Benchmark< T >

Making a framework for testing inserted data structure. Using time to estimate computational complexity.

5.1.2 Member Function Documentation

5.1.2.1 template<class T > long double Benchmark< T >::average (std::vector< long double > *times*) [private]

get average time from testing (10 probes)

5.1.2.2 template<class T > void Benchmark< T >::raport (InputFiles & *files*)

Create a .xls file (excel) with file sizes and test times.

5.1.2.3 template<class T > void Benchmark< T >::test (T *data_structure,* InputFiles & *files*)

Main testing function.

Parameters

files	random generated files with integers
data_structure	tested structure

5.1.3 Member Data Documentation

5.1.3.1 template<class T > Benchmark< T >::avrTestTimes [private]

A container for calculated times.

The documentation for this class was generated from the following files:

- inc/benchmark_frm.h
- src/benchmark_frm.cpp

5.2 InputFiles Class Reference

`#include <inputfile_txt.h>`

Public Member Functions

- void generate_random_int_data ()

  Puts random int data into files.
- InputFiles ()

  A default constructor.
- InputFiles (int filNr, std::vector< int >filSiz)

  A constructor.
- const std::string return_file_name (int Nmbr)

  Return names of files (only for read purpose)
- const int return_file_size (int Nmbr)

  Return sizes of files (only for read purpose)
- const int return_number_files ()

  Return number of files.
- void show_info ()

  Showes info obout files.

Private Attributes

- std::vector< std::string > filesNamesTab

  Container for generated file names.
- int filesNumber

  Number of generated files.
- std::vector< int > filesSizes

  Container for file sizes.

5.2.1 Detailed Description

Making an object which contain text files with generated random integer numbers.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 InputFiles::InputFiles ( )

A default constructor.

Adding number of files(UNDEF_VALUE = 1); Generating file name; Adding size of file (UNDEF_VALUE = 1);

Just in case, when program starts without any parameters.

Todo EXCEPTIONS HANDLING

5.2.2.2 InputFiles::InputFiles (int *filNr,* std::vector< int > *filSiz*)

A constructor.

Adding number of files; Generating files names; Adding sizes of files; Parameters inherit from list of arguments from command prompt

Parameters

filNr	number of files
filSiz	sizes of files

Open files with new names

Check if file is opened correctly

5.2.3 Member Function Documentation

5.2.3.1 void InputFiles::generate_random_int_data ( )

Puts random int data into files.

Generating random integers data (size from filesSizes vector) and putting them into files (names from filesNames↩ Tab) < Seed for Mersenne Twister 19937 generator

Mersenne Twister 19937 generator

More info about this generator: http://pl.wikipedia.org/wiki/Mersenne_Twister

Uniform distribution random number

Max number: uncomment next line More info about this distribution: http://pl.wikipedia.org/wiki/↩ Rozk%C5%82ad_jednostajny

Check if file is opened correctly

5.2.3.2 InputFiles::return_file_name (int *Nmbr*) [inline]

Return names of files (only for read purpose)

Parameters

Nmbr	Number of the file.

5.2.3.3 InputFiles::return_file_size (int *Nmbr*) [inline]

Return sizes of files (only for read purpose)

Parameters

Nmbr	Number of the file.

5.2.3.4 InputFiles::return_number_files ( ) [inline]

Return number of files.

5.2.3.5 void InputFiles::show_info ( )

Showes info obout files.

Display: number of files, names of files, sizes of files

5.2.4 Member Data Documentation

5.2.4.1 InputFiles::filesNamesTab [private]

Container for generated file names.

5.2.4.2 InputFiles::filesNumber [private]

Number of generated files.

5.2.4.3 InputFiles::filesSizes [private]

Container for file sizes.

The documentation for this class was generated from the following files:

- inc/inputfile_txt.h
- src/inputfile_txt.cpp

5.3 List< T > Class Template Reference

`#include <list.h>`

Public Member Functions

- List ()
  A default constructor.
- T pop ()
  return data from last element of the list
- T pop_front ()

return data from first element of the list

- void push (T data)

  Add new node to the END of the list.

- void push_front (T data)

  Add new node to the BEGINING of the list.

- const Node< T > * return_head () const

  return first node

- int size () const

  return size of the list

- ~List ()

  A destructor.

Private Member Functions

- void delete_list ()

  delete all nodes from list

Private Attributes

- Node< T > * headPtr
- int **sizeLst**
- Node< T > * tailPtr
- Node< T > * tempPtr

Friends

- std::ostream & **operator**<< (std::ostream &out, const List &list)

5.3.1 Detailed Description

template<class T>class List< T >

An implementation of linked list.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 template<class T > List< T >::List ( ) [inline]

A default constructor.

Create empty list.

5.3.2.2 template<class T > List< T >::~List ( ) [inline]

A destructor.

Null for all pointers Delete memory.

5.3.3 Member Function Documentation

5.3.3.1 template<class T > void List< T >::delete_list ( ) [private]

delete all nodes from list

5.3.3.2 template<class T > T List< T >::pop ( )

return data from last element of the list

5.3.3.3 template<class T > T List< T >::pop_front ( )

return data from first element of the list

5.3.3.4 template<class T > void List< T >::push (T *data*)

Add new node to the END of the list.

Parameters

data	inserted data into node.

5.3.3.5 template<class T > void List< T >::push_front (T *data*)

Add new node to the BEGINING of the list.

Parameters

data	inserted data into node

5.3.3.6 template<class T > List< T >::return_head ( ) const [inline]

return first node

5.3.3.7 template<class T > List< T >::size ( ) const [inline]

return size of the list

Size of the list

5.3.4 Member Data Documentation

5.3.4.1 template<class T > List< T >::headPtr [private]

First element from the list pointer.

5.3.4.2 template<class T > List< T >::tailPtr [private]

Pointer for last element

5.3.4.3 template<class T > List< T >::tempPtr [private]

Temporary pointer, help for adding new nodes.

The documentation for this class was generated from the following files:

- inc/list.h
- src/list.cpp

5.4 LNode Class Reference

`#include <list_node.h>`

5.4.1 Detailed Description

t An implementation for nodes to the linked list

The documentation for this class was generated from the following file:

- inc/list_node.h

5.5 Node< T > Class Template Reference

`#include <tree_node.h>`

Public Member Functions

- const T get_data () const
  return data from selected node
- const Node< T > * **get_nextNode** () const
- **Node** (T data)
- ~Node ()
  A destructor.

Private Attributes

- Node< T > * nextNode
- T nodeData

Friends

- template<class T >
  class **List**

5.5.1 Detailed Description

template<class T>class Node< T >

Binary tree nodes implementation.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 template<class T> Node< T >::~Node ( ) [inline]

A destructor.

5.5.3 Member Function Documentation

5.5.3.1 template<class T> Node< T >::get_data ( ) const [inline]

return data from selected node

return pointer to the next node.

5.5.4 Member Data Documentation

5.5.4.1 template<class T> Node< T >::nextNode [private]

Pointer to the next node in linked list.

5.5.4.2 template<class T> Node< T >::nodeData [private]

Container for data inside node.

The documentation for this class was generated from the following file:

- inc/list_node.h

5.6 NodeB< T > Class Template Reference

Public Member Functions

- T **get_data** ()
- const bool is_empty ()
  return state of node
- NodeB ()
  A constructor.
- **NodeB** (T data)
- NodeB< T > * return_left ()
  return pointer to the left branch of node
- NodeB< T > * return_parent ()
  return pointer to the parent of node
- NodeB< T > * return_right ()
  return pointer to the right branch of node
- ~NodeB ()
  A destructor.

Private Attributes

- T data
  Container for random integer data.

- bool empty
- NodeB< T > * left

  link with left branch node from parent
- NodeB< T > * parent

  link with parent node from tree
- NodeB< T > * right

  link with right branch node from parent

Friends

- template<class T >
  class **Tree**

5.6.1 Constructor & Destructor Documentation

5.6.1.1 template<class T> NodeB< T >::NodeB ( ) [inline]

A constructor.

Create empty node. Every pointer is NULL. root data = 0;

Create empty NodeB. Every pointer is NULL.

Parameters

data	inserted data from argument

5.6.1.2 template<class T> NodeB< T >::~NodeB ( ) [inline]

A destructor.

Free all memory. Deleting this pointer, so it's dangerous.

5.6.2 Member Function Documentation

5.6.2.1 template<class T> NodeB< T >::is_empty ( ) [inline]

return state of node

true - empty NodeB false - not empty NodeB

5.6.2.2 template<class T> NodeB< T >::return_left ( ) [inline]

return pointer to the left branch of node

5.6.2.3 template<class T> NodeB< T >::return_parent ( ) [inline]

return pointer to the parent of node

5.6.2.4 template<class T> NodeB< T >::return_right ( ) [inline]

return pointer to the right branch of node

5.6.3 Member Data Documentation

5.6.3.1 template<class T> NodeB< T >::data [private]

Container for random integer data.

5.6.3.2 template<class T> NodeB< T >::empty [private]

Flag for empty(true)/nonempty node(false)

5.6.3.3 template<class T> NodeB< T >::left [private]

link with left branch node from parent

Data smaller than parent data.

5.6.3.4 template<class T> NodeB< T >::parent [private]

link with parent node from tree

5.6.3.5 template<class T> NodeB< T >::right [private]

link with right branch node from parent

Data bigger than parent data.

The documentation for this class was generated from the following file:

- inc/tree_node.h

5.7 Queue< T > Class Template Reference

`#include <queue.h>`

Public Member Functions

- T **pop** ()
- void push (T data)

  add new data to the queue
- **Queue** (int queueSize)
- const int size ()

  return size of the queue

Private Attributes

- int addCount

  Counter of elements added to the vector.
- int head
- std::vector< T > queue
- int **sizeQue**
- int tail

Friends

- std::ostream & **operator**<< (std::ostream &out, const Queue &queue)

5.7.1 Detailed Description

template<class T>class Queue< T >

An implementation of queue based on dynamically created table. Info based on http://www.↩ studytonight.com/data-structures/queue-data-structure

5.7.2 Member Function Documentation

5.7.2.1 template<class T > void Queue< T >::push (T *data*)

add new data to the queue

When queue is full + all filled -> add new place at the end When queue is full + some free space -> put at first free space new element When queue is not full -> put at first free space new element When queue is empty, and you want to pop something -> assert is on When queue is fill, and you want to pop -> pop very first element. Just like normal queue.

5.7.2.2 template<class T > Queue< T >::size ( ) [inline]

return size of the queue

5.7.3 Member Data Documentation

5.7.3.1 template<class T > Queue< T >::addCount [private]

Counter of elements added to the vector.

Testing purpose.

5.7.3.2 template<class T > Queue< T >::head [private]

Last element from the queue

5.7.3.3 template<class T > A constructor with fixed size of the Queue< T >::queue [private]

Parameters

queueSize	fixed size of the queue;

5.7.3.4 template<class T > Queue< T >::tail [private]

A free place inside the queue

The documentation for this class was generated from the following files:

- inc/queue.h
- src/queue.cpp

5.8 Stack< T > Class Template Reference

`#include <stack.h>`

Public Member Functions

- T **pop** ()
- void push (T data)

  Puts one object on the stack.
- void **push_prc** (T data)
- int size ()

  return size of the stack
- Stack ()

  Create stack memory, assign size of the stack.
- **Stack** (int stackSize)

Private Attributes

- int addCount

  Counter of elements added to the vector.
- int sizeStc

  size of the stack
- std::vector< T > stack

  Container for stack date.

Friends

- std::ostream & **operator**<< (std::ostream &out, const Stack &stack)

5.8.1 Detailed Description

template<class T>class Stack< T >

Own implementation of stack. As simple as possible.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 template<class T > Stack< T >::Stack ( ) [inline]

Create stack memory, assign size of the stack.

Parameters

stackSize	size of the new stack

5.8.3 Member Function Documentation

5.8.3.1 template<class T > void Stack< T >::push (T *data*)

Puts one object on the stack.

When stack is overload, automatticaly add one more place. (Used vector abillity to adding new memory space in this case)

Parameters

data	object inserted to the stack

When stack is overload, automatticaly add percent of old place as a added new memory. After hitting fullness, multiply size of the vector twice.

Parameters

data	object inserted to the stack

5.8.3.2 template<class T > Stack< T >::size ( ) [inline]

return size of the stack

5.8.4 Member Data Documentation

5.8.4.1 template<class T > Stack< T >::addCount [private]

Counter of elements added to the vector.

Used in push_prc, if addCount == size, dupcilate vector. Testing purpose.

5.8.4.2 template<class T > Stack< T >::sizeStc [private]

size of the stack

5.8.4.3 template<class T > Stack< T >::stack [private]

Container for stack date.

The documentation for this class was generated from the following files:

- inc/stack.h
- src/stack.cpp

5.9 Tree< T > Class Template Reference

`#include <tree.h>`

Public Member Functions

- const bool is_empty ()
  return info if the tree is empty
- T pop (T data)
  delete node with data from argument, than return data from this node
- void push (T data)
  add new node to the tree
- void show_tree ()
  show 3 levels from tree, starting from root node.
- const int size ()
  return size of the tree
- Tree ()

A constructor (default)

- **Tree** (T data)

Private Attributes

- bool empty
- NodeB< T > * root

  Root node of the tree.
- int sizeTre

  Quantity of nodes from the tree (with root node)

5.9.1 Detailed Description

template<class T>class Tree< T >

Own binary tree class, based on information from book: "Data structure and algorithms in C++"- Goodrich Time tested in constructor.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 template<class T > Tree< T >::Tree ( ) [inline]

A constructor (default)

A constructor.

Create tree with only root node (init with 0 in this type constructor)

Create tree with only root node (init with data from argument)

Parameters

data	new data inserted to the node

5.9.3 Member Function Documentation

5.9.3.1 template<class T > Tree< T >::is_empty ( ) [inline]

return info if the tree is empty

5.9.3.2 template<class T > T Tree< T >::pop (T *data*)

delete node with data from argument, than return data from this node

Find data inside the tree than delete node with it, reorganize the tree. If pop from tree with size = 1, return value from root node, than set it to 0.

Parameters

data	element with data will be deleted

5.9.3.3 template<class T > void Tree< T >::push (T *data*)

add new node to the tree

Parameters

data	new data inserted to the node

5.9.3.4 template<class T > void Tree< T >::show_tree ( )

show 3 levels from tree, starting from root node.

5.9.3.5 template<class T > Tree< T >::size ( ) [inline]

return size of the tree

5.9.4 Member Data Documentation

5.9.4.1 template<class T > Tree< T >::empty [private]

True - tree has only default root node with 0; False - tree has something inside

if there is only init root node with something from user, still false

5.9.4.2 template<class T > Tree< T >::root [private]

Root node of the tree.

5.9.4.3 template<class T > Tree< T >::sizeTre [private]

Quantity of nodes from the tree (with root node)

if root node has default value = 0 <- sizeTre = 0

The documentation for this class was generated from the following files:

- inc/tree.h
- src/tree.cpp

Chapter 6

File Documentation

6.1 inc/benchmark_frm.h File Reference

A Benchmark class.

```
#include <vector>
#include <cassert>
#include <boost\timer\timer.hpp>
#include <boost\chrono\duration.hpp>
#include <boost\algorithm\string\replace.hpp>
#include "inputfile_txt.h"
```

Classes

- class Benchmark< T >

Variables

- const long double SEC = 0.000000001

  For multiplying.

6.1.1 Detailed Description

A Benchmark class.

6.1.2 Variable Documentation

6.1.2.1 SEC = 0.000000001

For multiplying.

6.2 inc/inputfile_txt.h File Reference

A InputFile class.

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <cstdio>
#include <cassert>
#include <ctime>
#include <random>
#include <boost/algorithm/string/erase.hpp>
```

Classes

- class InputFiles

Variables

- const int FIRST_ARGUMENT = 1

  A const value for representing first argument from command prompt (name of the program) ∗/.
- const int PROGRAM_NAME = 1

  The same as FIRST_ARGUMENT ∗/.
- const int UNDEF_VALUE = 1

  A value for undefined arguments ∗/.

6.2.1 Detailed Description

A InputFile class.

6.2.2 Variable Documentation

6.2.2.1 const int FIRST_ARGUMENT = 1

A const value for representing first argument from command prompt (name of the program) ∗/.

6.2.2.2 const int PROGRAM_NAME = 1

The same as FIRST_ARGUMENT ∗/.

6.2.2.3 const int UNDEF_VALUE = 1

A value for undefined arguments ∗/.

6.3 inc/list.h File Reference

Own implementation of linked list.

```
#include "list_node.h"
#include <iostream>
#include <cassert>
```

Classes

- class List< T >

6.3.1 Detailed Description

Own implementation of linked list.

6.4 inc/list_node.h File Reference

Nodes for the list.

Classes

- class Node< T >

6.4.1 Detailed Description

Nodes for the list.

6.5 inc/queue.h File Reference

Own implementation of queue.

```
#include <vector>
#include <cassert>
```

Classes

- class Queue< T >

6.5.1 Detailed Description

Own implementation of queue.

6.6 inc/stack.h File Reference

Own implementation of stack data structure.

```
#include <vector>
#include <cassert>
```

Classes

- class Stack< T >

6.6.1 Detailed Description

Own implementation of stack data structure.

6.7 inc/tree.h File Reference

Own implementation of tree data structure.

```
#include <cassert>
#include "tree_node.h"
```

Classes

- class Tree< T >

6.7.1 Detailed Description

Own implementation of tree data structure.

6.8 inc/tree_node.h File Reference

Own implementation of tree data structure (Node of tree)

Classes

- class NodeB< T >

6.8.1 Detailed Description

Own implementation of tree data structure (Node of tree)

6.9 src/benchmark_frm.cpp File Reference

Source code for Benchmark class.

```
#include "benchmark_frm.h"
```

6.9.1 Detailed Description

Source code for Benchmark class.

6.10 src/inputfile_txt.cpp File Reference

Source code for InputFile class.

```
#include "inputfile_txt.h"
```

6.10.1 Detailed Description

Source code for InputFile class.

6.11 src/list.cpp File Reference

A source code for list data structure.

```
#include "list.h"
```

6.11.1 Detailed Description

A source code for list data structure.

6.12 src/main.cpp File Reference

A master file.

```
#include <iostream>
#include <typeinfo>
#include "benchmark_frm.h"
#include "inputfile_txt.h"
#include "stack.h"
#include "list.h"
#include "queue.h"
#include "tree.h"
#include "benchmark_frm.cpp"
#include "list.cpp"
#include "stack.cpp"
#include "queue.cpp"
#include "tree.cpp"
```

Functions

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

6.12.1 Detailed Description

A master file.

6.13 src/stack.cpp File Reference

A stack class source code.

```
#include <stack.h>
```

6.13.1 Detailed Description

A stack class source code.

6.14 src/tree.cpp File Reference

A source file for Tree class.

```
#include "tree.h"
```

6.14.1 Detailed Description

A source file for Tree class.

Index