Lab 2

1.0

Generated by Doxygen 1.8.9.1

Thu Mar 19 2015 05:23:33

Contents

Chapter 1

Todo List

Member Benchmark::generateRaport (long double nextTime, int size)

FILE OVERWRITING, NEED TO IMPLEMENT NEW NAMES

Member InputFiles::InputFiles ()

EXCEPTIONS HANDLING

Member Tree::Tree (InputFiles &file, Benchmark &TreeTest)

memory problems, maybe new deconstructor?

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Benchmark Class Reference

`#include <benchmark_frm.h>`

Public Member Functions

- Benchmark ()
- Benchmark (const Benchmark &copy)
- void show_testTimes_v ()

  show filled vector with test times
- void test (InputFiles files)
- void test (std::fstream &file, Node *root)
- void test (int fileSize, std::fstream &openFile, Queue *newQueue)
- void test (std::fstream &openFile, List *newList)
- void test (int fileSize, std::fstream &openFile, Stack *newStack)

Private Member Functions

- void generateRaport (long double nextTime, int size)
- void generateRaport (long double avgTime, int size, std::string fileName, data_type type)
- long double getAvr (std::vector< long double >times)

  Measures the average duration from 10 samples.
- void measureTime (int *dataTable, int dataSize)

  Measures the duration of the work of assignment function.

Private Attributes

- std::vector< long double > testTimes

  A container for calculated times.

Friends

- class List
- class Queue
- class Stack
- class Tree

4.1.1 Detailed Description

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

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Benchmark::Benchmark ( )

```
15                      {
16     ;
17 }
```

4.1.2.2 Benchmark::Benchmark (const Benchmark & *copy*)

```
11                                          {
12     testTimes = copy.testTimes;
13 }
```

4.1.3 Member Function Documentation

4.1.3.1 void Benchmark::generateRaport (long double *nextTime*, int *size*) [private]

Todo FILE OVERWRITING, NEED TO IMPLEMENT NEW NAMES

```
101                                                         {
103     std::ofstream raportFile;
104     std::string stringNextTime = std::to_string(nextTime);
105
106     // .xls as excel file format
107     raportFile.open("test.xls", std::ios::in | std::ios::app);
108         assert(raportFile.is_open() && ("I can't open file."));
109     // need to change '.' on ',' due to excel string format
110     boost::algorithm::replace_first(stringNextTime, ".", ",");
111
112     raportFile << size << "\n" << stringNextTime << "\n";
113     raportFile.close();
114 }
```

4.1.3.2 void Benchmark::generateRaport (long double *avgTime*, int *size*, std::string *fileName*, data_type *type*) [private]

```
116                                                                                                                 {
117     std::ofstream raportFile;
118     std::string stringNextTime = std::to_string(avgTime);
119
120     if (type == BIN_TREE){
121         // .xls as excel file format
122         raportFile.open("test_btree.xls", std::ios::in | std::ios::app);
123         assert(raportFile.is_open() && ("I can't open file."));
124         // need to change '.' on ',' due to excel string format
125         boost::algorithm::replace_first(stringNextTime, ".", ",");
126
127         raportFile << size << "\n" << stringNextTime << "\n";
128         raportFile.close();
129
130         std::cout << "(BinaryTree) For file " << "> " << fileName << " <" << " (size: " << size << ")" << "
    average test time (10 times) was: " << avgTime << std::endl;
131     }
132     else if (type == QUEUE){
133         // .xls as excel file format
134         raportFile.open("test_queue.xls", std::ios::in | std::ios::app);
135         assert(raportFile.is_open() && ("I can't open file."));
136         // need to change '.' on ',' due to excel string format
137         boost::algorithm::replace_first(stringNextTime, ".", ",");
138
139         raportFile << size << "\n" << stringNextTime << "\n";
140         raportFile.close();
141
```

```
142         std::cout << "(Queue) For file " << "> " << fileName << " <" << " (size: " << size << ")" << "
        average test time (10 times) was: " << avgTime << std::endl;
143     }
144     else if (type == LIST){
145         // .xls as excel file format
146         raportFile.open("test_list.xls", std::ios::in | std::ios::app);
147             assert(raportFile.is_open() && ("I can't open file."));
148         // need to change '.' on ',' due to excel string format
149         boost::algorithm::replace_first(stringNextTime, ".", ",");
150 
151         raportFile << size << "\n" << stringNextTime << "\n";
152         raportFile.close();
153 
154         std::cout << "(List) For file " << "> " << fileName << " <" << " (size: " << size << ")" << "
        average test time (10 times) was: " << avgTime << std::endl;
155     }
156     else if (type == STACK){
157         // .xls as excel file format
158         raportFile.open("test_stack.xls", std::ios::in | std::ios::app);
159         assert(raportFile.is_open() && ("I can't open file."));
160         // need to change '.' on ',' due to excel string format
161         boost::algorithm::replace_first(stringNextTime, ".", ",");
162 
163         raportFile << size << "\n" << stringNextTime << "\n";
164         raportFile.close();
165 
166         std::cout << "(Stack) For file " << "> " << fileName << " <" << " (size: " << size << ")" << "
        average test time (10 times) was: " << avgTime << std::endl;
167     }
168 }
```

4.1.3.3 Benchmark::getAvr (std::vector< long double > *times*) [private]

Measures the average duration from 10 samples.

Parameters

times	A container with times from tests.

```
170                                                     {
171     long double avrg = 0.0;
172 
173     //add 10 values, than count average
174     for (int i = 0; i < (signed)times.size(); i++){
175         avrg += times[i];
176     }
177 
178     avrg /= (long double)times.size();
179     return avrg;
180 }
```

4.1.3.4 Benchmark::measureTime (int * *dataTable*, int *dataSize*) [private]

Measures the duration of the work of assignment function.

Version for dynamic table

Parameters

dataTable	A container with random integers from earlier made files.
dataSize	A size of the file.

```
182                                                     {
183     // container for counted working times
184     std::vector<long double> estimateTimes;
185 
186     for (int j = 0; j < 10; j++){
187         // Here starts the timer
188         boost::timer::cpu_timer startTime;
189         for (int i = 0; i < dataSize; i++){
190             dataTable[i] *= 2;
191         }
192         // Here it ends
```

```
193         boost::timer::cpu_times endTime = startTime.elapsed();
194         // add new time to the vector
195         estimateTimes.push_back(static_cast<long double>(endTime.wall * SEC));
196     }
197     // for better display
198     std::cout.fixed;
199     long double DurTime = getAvr(estimateTimes);
200     std::cout << "Time (average, 10 samples) for " << dataSize << " elements: " << DurTime << " sec"<<
    std::endl;
201     generateRaport(DurTime, dataSize);
202 }
```

4.1.3.5 Benchmark::show_testTimes_v ( )

show filled vector with test times

```
204                                 {
205     for (int i = 0; i < (signed)this->testTimes.size(); i++){
206         std::cout << testTimes[i] << std::endl;
207     }
208 }
```

4.1.3.6 void Benchmark::test (InputFiles *files*)

```
19                                      {
20     // temp memory container
21     int* tabForData = NULL;
22     int tempValue = 0;
23     std::fstream newFile;
24
25     for (int i = 0; i < files.return_number_files() -
    FIRST_ARGUMENT; i++){
26         // Opening file + making new table with content
27         tabForData = new int[files.return_file_size(i)];
28         newFile.open((files.return_file_name(i) + ".txt"), std::ios::in);
29
30         // Checking if file is opened correctly
31         assert(newFile.is_open() && ("I can't open file."));
32
33         for (int j = 0; j < files.return_file_size(i); j++){
34             newFile >> tempValue;
35             tabForData[j] = tempValue;
36         }
37         newFile.close();
38
39         // Testing time here
40         measureTime(tabForData, files.return_file_size(i));
41         delete[] tabForData;
42     }
43 }
```

4.1.3.7 void Benchmark::test (std::fstream & *file*, Node * *root*)

```
46                                                  {
47     // temporary data container
48     int tempData;
49
50     //here starts timer
51     boost::timer::cpu_timer startTime;
52     while (file >> tempData)
53         root->add(file, tempData);
54     //here ends
55     boost::timer::cpu_times endTime = startTime.elapsed();
56
57     //put new time to the vector
58     this->testTimes.push_back(static_cast<long double>(endTime.wall *
    SEC));
59 }
```

4.1.3.8 void Benchmark::test (int *fileSize*, std::fstream & *openFile*, Queue * *newQueue*)

```
62                                                                          {
63
64     //here starts timer
65     boost::timer::cpu_timer startTime;
66     newQueue->add(openFile, fileSize);
67     //here ends
68     boost::timer::cpu_times endTime = startTime.elapsed();
69
70     //put new time to the vector
71     this->testTimes.push_back(static_cast<long double>(endTime.wall *
       SEC));
72 }
```

4.1.3.9 void Benchmark::test (std::fstream & *openFile*, List * *newList*)

```
75                                                      {
76
77     //here starts timer
78     boost::timer::cpu_timer startTime;
79     newList->add(openFile);
80     //here ends
81     boost::timer::cpu_times endTime = startTime.elapsed();
82
83     //put new time to the vector
84     this->testTimes.push_back(static_cast<long double>(endTime.wall *
       SEC));
85 }
```

4.1.3.10 void Benchmark::test (int *fileSize*, std::fstream & *openFile*, Stack * *newStack*)

```
88                                                                          {
89
90     //here starts timer
91     boost::timer::cpu_timer startTime;
92     newStack->add(openFile, fileSize);
93     //here ends
94     boost::timer::cpu_times endTime = startTime.elapsed();
95
96     //put new time to the vector
97     this->testTimes.push_back(static_cast<long double>(endTime.wall *
       SEC));
98 }
```

4.1.4 Friends And Related Function Documentation

4.1.4.1 friend class List [friend]

4.1.4.2 friend class Queue [friend]

4.1.4.3 friend class Stack [friend]

4.1.4.4 friend class Tree [friend]

4.1.5 Member Data Documentation

4.1.5.1 Benchmark::testTimes [private]

A container for calculated times.

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/benchmark_frm.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/benchmark_frm.cpp

4.2 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.

4.2.1 Detailed Description

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

4.2.2 Constructor & Destructor Documentation

4.2.2.1 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

```
7                       {
9      // When there are no arguments from command prompt:
10      filesNumber = UNDEF_VALUE;
11      std::string TempName = std::tmpnam(nullptr);
12      filesNamesTab.push_back(TempName);
13      filesSizes.push_back(UNDEF_VALUE);
14  }
```

4.2.2.2 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

```
16                                                      {
17      filesNumber = filNr;
18      filesSizes = filSiz;
19      // Create new names for files
20      std::string TempName;
21
22      for (int i = 1; i < filesNumber; i++){
23          // Generate new unique name for file
24          TempName = std::tmpnam(nullptr);
25          // Delete all prohibit char from string
26          boost::algorithm::erase_all(TempName, "/");
27          boost::algorithm::erase_all(TempName, "\\");
28          // Put name into names container
29          filesNamesTab.push_back(TempName);
30      }
32      std::ofstream newFile;
33      for (int i = 1; i < filesNumber; i++){
34          newFile.open(filesNamesTab[i - PROGRAM_NAME] + ".txt");
36              assert(newFile.is_open() && "I can't open this file.");
37          newFile.close();
38      }
39  }
```

4.2.3 Member Function Documentation

4.2.3.1 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

```
53                                              {
54      int seedGen = time(NULL);
56
61      std::mt19937 randomNumbr(seedGen);
62
64
69      //std::cout << std::numeric_limits<int>::max() << std::endl;
70      std::uniform_int_distribution<>newDistr;
71
72      std::ofstream NewFile;
73      for (int i = 1; i < filesNumber; i++){
74          NewFile.open((filesNamesTab[i - PROGRAM_NAME] + ".txt"), std::ios::in);
76              assert(NewFile.is_open() && ("I can't open file."));
77          //Generate random int data
78          for (int j = 0; j < filesSizes[i - FIRST_ARGUMENT]; j++){
79              NewFile << newDistr(randomNumbr) << "\n";
80          }
```

```
81
82         NewFile.close();
83     }
84 }
```

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

Return names of files (only for read purpose)

Parameters

Nmbr	Number of the file.

```
70                                                  {
71             return filesNamesTab[Nmbr];
72         }
```

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

Return sizes of files (only for read purpose)

Parameters

Nmbr	Number of the file.

```
78                                                {
79             return filesSizes[Nmbr];
80         }
```

4.2.3.4 InputFiles::return_number_files ( ) [inline]

Return number of files.

```
85                                       {
86             return filesNumber;
87         }
```

4.2.3.5 InputFiles::show_info ( )

Showes info obout files.

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

```
41                            {
42     std::cout << "--------" << std::endl;
43     std::cout << filesNumber - FIRST_ARGUMENT << std::endl;
44     for (int i = 0; i < (signed)filesNamesTab.size(); i++){
45         std::cout << filesNamesTab[i] << std::endl;
46     }
47     for (int i = 0; i < (signed)filesSizes.size(); i++){
48         std::cout << filesSizes[i] << std::endl;
49     }
50     std::cout << "--------" << std::endl;
51 }
```

4.2.4 Member Data Documentation

4.2.4.1 std::vector<std::string> InputFiles::filesNamesTab [private]

Container for generated file names.

4.2.4.2 InputFiles::filesNumber [private]

Number of generated files.

4.2.4.3 InputFiles::filesSizes [private]

Container for file sizes.

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/inputfile_txt.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/inputfile_txt.cpp

4.3 List Class Reference

`#include <list.h>`

Public Member Functions

- void add (std::fstream &file)
- List (InputFiles &file, Benchmark &listTest)

  A constructor.
- ~List ()

  A deconstructor.

Private Attributes

- LNode * headPtr
- LNode * tailPtr
- LNode * tempPtr

4.3.1 Detailed Description

An implementation of linked list.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 List::List (InputFiles & *file,* Benchmark & *listTest*)

A constructor.

```
7                                                   {
8      // open file for data in root
9      std::fstream openFile;
10      // take name of this file
11      std::string fileName;
12      // file size
13      int fileSize;
14      // average time from test
15      long double avrgTime;
16
17      for (int i = 0; i < file.return_number_files() -
       FIRST_ARGUMENT; i++){
18          fileName = file.return_file_name(i);
19          fileSize = file.return_file_size(i);
20          for (int i = 0; i < 10; i++){
21              openFile.open(fileName + ".txt");
```

```
22              // Check if file is opened correctly
23              assert(openFile.is_open() && "I can't open this file.");
24
25              listTest.test(openFile, this);
26
27              //initial again for next test
28              this->~List();
29              this->headPtr = nullptr;
30              this->tailPtr = nullptr;
31              this->tempPtr = nullptr;
32              openFile.close();
33          }
34          //generate raport
35          avrgTime = listTest.getAvr(listTest.testTimes);
36          listTest.generateRaport(avrgTime, fileSize, fileName, LIST);
37      }
38  }
```

4.3.2.2 List::~List ( )

A deconstructor.

```
41              {
42      headPtr = NULL;
43      tailPtr = NULL;
44      tempPtr = NULL;
45
46      delete headPtr;
47      delete tailPtr;
48      delete tempPtr;
49  }
```

4.3.3 Member Function Documentation

4.3.3.1 void List::add (std::fstream & *file*)

```
51                                  {
52      //container for temporary data
53      int tempData;
54
55      //create first node of the list
56      file >> tempData;
57      this->tailPtr = new LNode(tempData);
58      this->headPtr = this->tailPtr;
59      this->tempPtr = this->tailPtr;
60
61      while (file >> tempData){
62          this->tailPtr = new LNode(tempData);
63          this->tempPtr->nextNode = this->tailPtr;
64          this->tempPtr = this->tempPtr->nextNode;
65      }
66  }
```

4.3.4 Member Data Documentation

4.3.4.1 List::headPtr [private]

First element from the list pointer.

4.3.4.2 List::tailPtr [private]

Last element pointer

4.3.4.3 List::tempPtr [private]

Temporary pointer, help for adding new nodes.

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list.cpp

4.4 LNode Class Reference

`#include <list_node.h>`

Public Member Functions

- LNode (int data)

  A constructor.
- ∼LNode ()

  A deconstructor.

Private Attributes

- LNode ∗ nextNode
- int nodeData

Friends

- class List

4.4.1 Detailed Description

An implementation for nodes to the linked list

4.4.2 Constructor & Destructor Documentation

4.4.2.1 LNode::LNode (int *data*)

A constructor.

```
8                       {
9      this->nodeData = data;
10      this->nextNode = nullptr;
11  }
```

4.4.2.2 LNode::∼LNode ( )

A deconstructor.

```
14              {
15      this->nextNode = nullptr;
16      delete this->nextNode;
17  }
```

4.4.3 Friends And Related Function Documentation

4.4.3.1 friend class List [friend]

4.4.4 Member Data Documentation

4.4.4.1 LNode::nextNode [private]

Pointer to the next node in linked list.

4.4.4.2 LNode::nodeData [private]

Container for data inside node.

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list_node.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list_node.cpp

4.5 Node Class Reference

`#include <tree_node.h>`

Public Member Functions

- void add (std::fstream &openFile, int tempData)
  adding new branch nodes to the parent node
- const int get_data_container ()
  return value of date from node
- bool is_emptyf ()
  return state of node
- Node (int data)
  root constructor
- Node (int data, Node *parent_ptr)
  child constructor
- Node * return_left ()
  return pointer to the left branch of node
- Node * return_right ()
  return pointer to the right branch of node
- ~Node ()
  deconstructor

Private Attributes

- int data_container
  Container for random integer data.
- bool is_empty
- Node * left
  link with left branch node from parent
- Node * parent
  link with parent node from tree
- Node * right
  link with right branch node from parent

4.5.1 Detailed Description

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Node::Node (int *data*)

root constructor

```
8                   {
9      data_container = data;
10      parent = NULL;
11      left = NULL;
12      right = NULL;
13      is_empty = false;
14  }
```

4.5.2.2 Node::Node (int *data,* Node * *parent_ptr*)

child constructor

```
17                                      {
18      data_container = data;
19      parent = parent_ptr;
20      left = NULL;
21      right = NULL;
22      is_empty = false;
23  }
```

4.5.2.3 Node::~Node ( )

deconstructor

```
26              {
27      delete this->left;
28      delete this->right;
29  }
```

4.5.3 Member Function Documentation

4.5.3.1 Node::add (std::fstream & *openFile,* int *tempData*)

adding new branch nodes to the parent node

!! This is a recursive function.!! End point: when file is ended. Open file with random int data, check if there is open left/right branch node from parent: Algorithm: add(openFile,dataFromFile){ if rootData is bigger or equal dataFromFile if thereIsLeftNode add(openFile,datafromFile) // opening new node else create leftNode and add data if rootData is smaller than dataFromFile if thereisRightNode add(openFile,datafromFile) // opening new node else create rightNode and add data

Parameters

openFile	actually opened file with random int data
tempData	container for data from file

```
31                                              {
32      // algorithm described in header file
33      if (this->data_container >= tempData){
34          if (this->left == NULL){
35              this->left = new Node(tempData, this->parent);
36          }
37          else{
38              this->left->add(openFile, tempData);
39          }
```

```
40      }
41      else{
42          if (this->right == NULL){
43              this->right = new Node(tempData, this->parent);
44          }
45          else{
46              this->right->add(openFile, tempData);
47          }
48      }
49  }
```

4.5.3.2 Node::get_data_container ( ) [inline]

return value of date from node

```
40                                          {
41              return data_container;
42          }
```

4.5.3.3 Node::is_emptyf ( ) [inline]

return state of node

true - empty node false - not empty node

```
64                          {
65              return is_empty;
66          }
```

4.5.3.4 Node::return_left ( ) [inline]

return pointer to the left branch of node

```
47                                  {
48              return left;
49          }
```

4.5.3.5 Node::return_right ( ) [inline]

return pointer to the right branch of node

```
54                                   {
55              return right;
56          }
```

4.5.4 Member Data Documentation

4.5.4.1 Node::data_container [private]

Container for random integer data.

4.5.4.2 bool Node::is_empty [private]

4.5.4.3 Node::left [private]

link with left branch node from parent

Data smaller than parent data.

4.5.4.4 Node::parent [private]

link with parent node from tree

4.5.4.5 Node::right [private]

link with right branch node from parent

Data bigger than parent data.

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree_node.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree_node.cpp

4.6 Queue Class Reference

`#include <queue.h>`

Public Member Functions

- void add (std::fstream &openFile, int fileSize)

  Adding new element to the rear part of queue.
- Queue (InputFiles &file, Benchmark &queueTest)

  A constructor.
- ~Queue ()

  A deconstructor.

Private Attributes

- int front
- int * queueTab
- int rear

4.6.1 Detailed Description

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

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Queue::Queue (InputFiles & *file,* Benchmark & *queueTest*)

A constructor.

```
10                                                  {
11      // open file for data in root
12      std::fstream openFile;
13      // take name of this file
14      std::string fileName;
15      // file size
16      int fileSize;
17      // average time from test
18      long double avrgTime;
19
```

```
20     for (int i = 0; i < file.return_number_files() -
       FIRST_ARGUMENT; i++){
21         fileName = file.return_file_name(i);
22         fileSize = file.return_file_size(i);
23         for (int i = 0; i < 10; i++){
24             openFile.open(fileName + ".txt");
25             // Check if file is opened correctly
26             assert(openFile.is_open() && "I can't open this file.");
27 
28             this->queueTab = new int[fileSize];
29             this->front = 0;
30             this->rear = 0;
31 
32             queueTest.test(fileSize, openFile, this);
33 
34             //initial again for next test
35             this->~Queue();
36             this->front = 0;
37             this->rear = 0;
38             openFile.close();
39         }
40         //generate raport
41         avrgTime = queueTest.getAvr(queueTest.testTimes);
42         queueTest.generateRaport(avrgTime, fileSize, fileName,
       QUEUE);
43     }
44 }
```

4.6.2.2 Queue::~Queue ( )

A deconstructor.

```
47             {
48     //clear everything
49     queueTab = NULL;
50     delete[] queueTab;
51 }
```

4.6.3 Member Function Documentation

4.6.3.1 Queue::add (std::fstream & *openFile,* int *fileSize*)

Adding new element to the rear part of queue.

Adding new element, on the rear part, move this pointer to the next element.

Parameters

openFile	a pointer to the opened file
fileSize	a size of opened file

```
55                                                 {
56     // temporary data container
57     int tempData;
58     int counter = 0;
59 
60     while (openFile >> tempData){
61         counter++;
62         //check if there is some place in queue
63         if (counter > fileSize){
64             std::cout << " Queue is full! Program is going to be terminated" << std::endl;
65             std::cin.get();
66             exit(true);
67         }
68         else{
69             if (this->rear)
70                 // add element on the rear part of queue
71                 this->queueTab[this->rear] = tempData;
72             // move rear poiner
73             // if it move behind queue, make it on the front.
74             this->rear = (this->rear + 1) % fileSize;
75         }
76     }
77 }
```

4.6.4 Member Data Documentation

4.6.4.1 Queue::front [private]

A head of queue position

4.6.4.2 Queue::queueTab [private]

A dynamic container for elements of queue

4.6.4.3 Queue::rear [private]

A tail of queue position

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/queue.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/queue.cpp

4.7 Stack Class Reference

`#include <stack.h>`

Public Member Functions

- void add (std::fstream &openFile, int fileSize)
- Stack (InputFiles &file, Benchmark &stackTest)

  A constructor.

Private Attributes

- int nrOfElement
- int * stackContainer

4.7.1 Detailed Description

Own implementation of stack. As simple as possible.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Stack::Stack (InputFiles & *file,* Benchmark & *stackTest*)

A constructor.

```
8                                                   {
9      // open file for data in root
10      std::fstream openFile;
11      // take name of this file
12      std::string fileName;
13      // file size
14      int fileSize;
15      // average time from test
16      long double avrgTime;
17
18      for (int i = 0; i < file.return_number_files() -
```

```
    FIRST_ARGUMENT; i++){
19          fileName = file.return_file_name(i);
20          fileSize = file.return_file_size(i);
21          for (int i = 0; i < 10; i++){
22              openFile.open(fileName + ".txt");
23              // Check if file is opened correctly
24              assert(openFile.is_open() && "I can't open this file.");
25
26              this->stackContainer = new int[fileSize];
27              this->nrOfElement = 0;
28              stackTest.test(fileSize, openFile, this);
29
30              //initial again for next test
31              delete this->stackContainer;
32              this->nrOfElement = 0;
33              openFile.close();
34          }
35          //generate raport
36          avrgTime = stackTest.getAvr(stackTest.testTimes);
37          stackTest.generateRaport(avrgTime, fileSize, fileName,
    STACK);
38      }
39  }
```

4.7.3 Member Function Documentation

4.7.3.1 Stack::add (std::fstream & *openFile*, int *fileSize*)

Puts new element on the top of the stack

Parameters

openFile	pointer to the actually opened file with random data
fileSize	size of actually opened file

```
41                                                  {
42      //container for temporary data
43      int tempData;
44
45      //take first element
46      openFile >> tempData;
47      this->stackContainer[this->nrOfElement];
48
49      while (openFile >> tempData){
50          this->nrOfElement += 1;
51          if (this->nrOfElement >= fileSize){
52              std::cout << "STACK OVERFLOW OMG OMG OMG NOSCOPE M8" << std::endl;
53          }
54          else{
55              this->stackContainer[this->nrOfElement] = tempData;
56          }
57      }
58  }
```

4.7.4 Member Data Documentation

4.7.4.1 Stack::nrOfElement [private]

Number of actually element from stack.

4.7.4.2 Stack::stackContainer [private]

Dynamically created table for date from stack.

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/stack.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/stack.cpp

4.8 Tree Class Reference

`#include <tree.h>`

Public Member Functions

- void show_tree ()

  show 3 levels from tree, starting from root node.
- Tree (InputFiles &file, Benchmark &TreeTest)

  constructor
- ~Tree ()

  deconstructor

Private Attributes

- Node * root

  Root node of the tree.

4.8.1 Detailed Description

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Tree::Tree (InputFiles & *file,* Benchmark & *TreeTest*)

constructor

Open file with random integers (range: 0 - 99999), create root node, than add new branch nodes depends on data from file.

Todo memory problems, maybe new deconstructor?

```
8                                                   {
11      // open file for data in root
12      std::fstream openFile;
13      // container for first data
14      int rootData;
15      // take name of this file
16      std::string fileName;
17      // file size
18      int fileSize;
19      // average time from test
20      long double avrgTime;
21
22      for (int i = 0; i < file.return_number_files() -
      FIRST_ARGUMENT; i++){
23          fileName = file.return_file_name(i);
24          fileSize = file.return_file_size(i);
25          for (int i = 0; i < 10; i++){
26              openFile.open(fileName + ".txt");
27              // Check if file is opened correctly
28              assert(openFile.is_open() && "I can't open this file.");
29
30              // create root node, put there first value from file
31              openFile >> rootData;
32              root = new Node(rootData);
33
34              //start test here
35              treeTest.test(openFile, root);
36
37              //initial again for next test
38              root->~Node();
39              root = NULL;
40              rootData = 0;
41              openFile.close();
42          }
43          avrgTime = treeTest.getAvr(treeTest.testTimes);
```

```
44          treeTest.generateRaport(avrgTime, fileSize, fileName, BIN_TREE);
45      }
46  }
```

4.8.2.2 Tree::~Tree ( )

deconstructor

Empty all memory space for tree

```
50              {
51      delete root;
52  }
```

4.8.3 Member Function Documentation

4.8.3.1 Tree::show_tree ( )

show 3 levels from tree, starting from root node.

```
55  {
56      //root 1lvl
57      std::cout << "root: " << root->get_data_container() << std::endl;
58      //2lvl
59      if (root->return_left() != NULL)
60          std::cout << "root(L): " << root->return_left()->
        get_data_container() << "\t";
61      if (root->return_right() != NULL)
62          std::cout << "root(R): " << root->return_right()->
        get_data_container() << std::endl;
63      //3lvl
64      if (root->return_left()->return_left() != NULL)
65          std::cout << "root(LL2): " << root->return_left()->
        return_left()->get_data_container() << "\t";
66      if (root->return_left()->return_right() != NULL)
67          std::cout << "root(LR2): " << root->return_left()->
        return_right()->get_data_container() << "\t";
68      if (root->return_right()->return_left() != NULL)
69          std::cout << "root(RL2): " << root->return_right()->
        return_left()->get_data_container() << "\t";
70      if (root->return_right()->return_right() != NULL)
71          std::cout << "root(RR2): " << root->return_right()->
        return_right()->get_data_container() << std::endl;
72
73  }
```

4.8.4 Member Data Documentation

4.8.4.1 Tree::root [private]

Root node of the tree.

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

- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree.h
- C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree.cpp

Chapter 5

File Documentation

5.1 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/benchmark_frm.cpp File Reference

Source code for Benchmark class.

```
#include "benchmark_frm.h"
#include "queue.h"
#include "list.h"
#include "stack.h"
```

5.1.1 Detailed Description

Source code for Benchmark class.

5.2 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/benchmark_frm.h File Reference

A Benchmark class.

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

Classes

- class Benchmark

Enumerations

- enum data_type { BIN_TREE = 1, QUEUE = 2, LIST = 3, STACK = 4 }

Variables

- const long double SEC = 0.000000001

5.2.1 Detailed Description

A Benchmark class.

5.2.2 Enumeration Type Documentation

5.2.2.1 enum data_type

enum type to describe provided data types: binary tree(1), queue(2), linked list(3), stack(4)

Enumerator

BIN_TREE

QUEUE

LIST

STACK

```
29              {
30     BIN_TREE = 1,
31     QUEUE = 2,
32     LIST = 3,
33     STACK = 4
34 };
```

5.2.3 Variable Documentation

5.2.3.1 const long double SEC = 0.000000001

5.3 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/inputfile_txt.cpp File Reference

Source code for InputFile class.

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

5.3.1 Detailed Description

Source code for InputFile class.

5.4 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/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 ∗/.

5.4.1 Detailed Description

A InputFile class.

5.4.2 Variable Documentation

5.4.2.1 const int FIRST_ARGUMENT = 1

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

5.4.2.2 const int PROGRAM_NAME = 1

The same as FIRST_ARGUMENT ∗/.

5.4.2.3 const int UNDEF_VALUE = 1

A value for undefined arguments ∗/.

5.5 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list.cpp File Reference

A source code for list data structure.

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

5.5.1 Detailed Description

A source code for list data structure.

5.6 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list.h File Reference

Own implementation of linked list.

```
#include "benchmark_frm.h"
#include "inputfile_txt.h"
#include "list_node.h"
```

Classes

- class List

5.6.1 Detailed Description

Own implementation of linked list.

5.7 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list_node.cpp File Reference

A source code for LNode class.

```
#include "list_node.h"
```

5.7.1 Detailed Description

A source code for LNode class.

5.8 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/list_node.h File Reference

Nodes for list.

Classes

- class LNode

Macros

- #define LIST_NODE_H

5.8.1 Detailed Description

Nodes for list.

5.8.2 Macro Definition Documentation

5.8.2.1 #define LIST_NODE_H

5.9 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/main.cpp File Reference

A master file.

```
#include "tree.h"
#include "tree_node.h"
#include "queue.h"
#include "list.h"
#include "stack.h"
#include "inputfile_txt.h"
#include "benchmark_frm.h"
```

Functions

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

5.9.1 Detailed Description

A master file.

5.9.2 Function Documentation

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

```
15                                  {
16
17      // contains only one size, implementation from Lab1
18      std::vector<int>fileSize;
19      // First argument is a name of the program so i = 1
20      for (int i = 1; i < argc; i++)
21          fileSize.push_back(atoi(argv[i]));
22
23      InputFiles openFile(argc, fileSize);
24      openFile.generate_random_int_data();
25
26      Benchmark treeTest;
27      Tree newTree(openFile,treeTest);
28
29      Benchmark queueTest(treeTest);
30      Queue newQueue(openFile, queueTest);
31
32      Benchmark listTest(treeTest);
33      List newList(openFile, listTest);
34
35      Benchmark stackTest(treeTest);
36      Stack newStack(openFile, stackTest);
37
38      std::cin.clear();
39      std::cin.get();
40      return 0;
41  }
```

5.10 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/queue.cpp File Reference

```
#include "inputfile_txt.h"
#include "benchmark_frm.h"
#include "queue.h"
```

5.11 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/queue.h File Reference

Own implementation of queue.

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

Classes

- class Queue

5.11.1 Detailed Description

Own implementation of queue.

5.12 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/stack.cpp File Reference

A stack class source code.

```
#include "stack.h"
```

5.12.1 Detailed Description

A stack class source code.

5.13 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/stack.h File Reference

Own implementation of stack data structure.

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

Classes

- class Stack

5.13.1 Detailed Description

Own implementation of stack data structure.

5.14 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree.cpp File Reference

A source file for Tree class.

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

5.14.1 Detailed Description

A source file for Tree class.

5.15 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree.h File Reference

Own implementation of tree data structure.

```
#include "tree_node.h"
#include "inputfile_txt.h"
#include "benchmark_frm.h"
```

Classes

- class Tree

Variables

- const int FIRST_FILE = 0

5.15.1 Detailed Description

Own implementation of tree data structure.

5.15.2 Variable Documentation

5.15.2.1 FIRST_FILE = 0

const int for first file index

5.16 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree_node.cpp File Reference

A source file for Node class.

```
#include "tree_node.h"
```

5.16.1 Detailed Description

A source file for Node class.

5.17 C:/Users/Daniel/Desktop/myStuff/209185/Lab2/src/tree_node.h File Reference

Own implementation of tree data structure (nodes of tree)

```
#include <vector>
#include <fstream>
```

Classes

- class Node

5.17.1 Detailed Description

Own implementation of tree data structure (nodes of tree)

Index