Laboratorium 7

Wygenerowano przez Doxygen 1.8.6

Cz, 14 maj 2015 23:21:57

Spis treści

1 Laboratorium 2 **1**

2 Indeks hierarchiczny **1**

2.1 Hierarchia klas 1

3 Indeks klas **1**

3.1 Lista klas 1

4 Indeks plików **2**

4.1 Lista plików 2

5 Dokumentacja klas **2**

5.1 Dokumentacja klasy DataFrame 3

5.1.1 Opis szczegółowy 3

5.1.2 Dokumentacja konstruktora i destruktora 3

5.1.3 Dokumentacja funkcji składowych 4

5.1.4 Dokumentacja atrybutów składowych 4

5.2 Dokumentacja klasy MultiplyByTwo 5

5.2.1 Opis szczegółowy 5

5.2.2 Dokumentacja konstruktora i destruktora 5

5.2.3 Dokumentacja funkcji składowych 6

5.3 Dokumentacja klasy MyBenchmark 6

5.3.1 Opis szczegółowy 7

5.3.2 Dokumentacja konstruktora i destruktora 7

5.3.3 Dokumentacja funkcji składowych 7

5.3.4 Dokumentacja atrybutów składowych 8

5.4 Dokumentacja klasy MyList 8

5.4.1 Opis szczegółowy 9

5.4.2 Dokumentacja konstruktora i destruktora 9

5.4.3 Dokumentacja funkcji składowych 9

5.4.4 Dokumentacja atrybutów składowych 11

5.5 Dokumentacja klasy MyList::MyListElement 11

5.5.1 Opis szczegółowy 11

5.5.2 Dokumentacja konstruktora i destruktora 12

5.5.3 Dokumentacja atrybutów składowych 12

5.6 Dokumentacja klasy MyQueue 12

5.6.1 Opis szczegółowy 13

5.6.2 Dokumentacja funkcji składowych 13

5.7 Dokumentacja klasy MyStack 13

5.7.1 Opis szczegółowy 13

5.7.2 Dokumentacja funkcji składowych . . . 14
5.8 Dokumentacja klasy NumberGenerator . . . 14
5.8.1 Opis szczegółowy . . . 14
5.8.2 Dokumentacja konstruktora i destruktora . . . 15
5.8.3 Dokumentacja funkcji składowych . . . 15
5.9 Dokumentacja klasy StackOnArray . . . 15
5.9.1 Opis szczegółowy . . . 16
5.9.2 Dokumentacja konstruktora i destruktora . . . 16
5.9.3 Dokumentacja funkcji składowych . . . 16
5.9.4 Dokumentacja atrybutów składowych . . . 17

6 Dokumentacja plików **17**
6.1 Dokumentacja pliku dataframe.cpp . . . 17
6.2 dataframe.cpp . . . 18
6.3 Dokumentacja pliku dataframe.h . . . 18
6.4 dataframe.h . . . 18
6.5 Dokumentacja pliku main.cpp . . . 19
6.5.1 Dokumentacja funkcji . . . 19
6.6 main.cpp . . . 21
6.7 Dokumentacja pliku multiplybytwo.cpp . . . 22
6.8 multiplybytwo.cpp . . . 22
6.9 Dokumentacja pliku multiplybytwo.h . . . 22
6.10 multiplybytwo.h . . . 23
6.11 Dokumentacja pliku mybenchmark.cpp . . . 23
6.12 mybenchmark.cpp . . . 23
6.13 Dokumentacja pliku mybenchmark.h . . . 24
6.14 mybenchmark.h . . . 24
6.15 Dokumentacja pliku mylist.cpp . . . 24
6.16 mylist.cpp . . . 24
6.17 Dokumentacja pliku mylist.h . . . 25
6.18 mylist.h . . . 25
6.19 Dokumentacja pliku myqueue.h . . . 26
6.20 myqueue.h . . . 26
6.21 Dokumentacja pliku mystack.h . . . 27
6.22 mystack.h . . . 27
6.23 Dokumentacja pliku numbergenerator.h . . . 27
6.24 numbergenerator.h . . . 27
6.25 Dokumentacja pliku stackonarray.cpp . . . 28
6.26 stackonarray.cpp . . . 28
6.27 Dokumentacja pliku stackonarray.h . . . 29

6.28 stackonarray.h 29
6.29 Dokumentacja pliku strona-glowna.dox 29

1 Indeks hierarchiczny

1.1 Hierarchia klas

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

List< MyListElementType > **??**

MyList< MyListElementType > **8**

List< Observer * > **??**

MyList< Observer * > **8**

ListElement< MyListElementType > **??**

MyListElement< MyListElementType > **??**

ListElement< Observer * > **??**

MyListElement< Observer * > **??**

ListSaver< MyListElementType > **??**

MyBenchmark **6**

MyBenchmarkObserver **??**

NumberGenerator **14**

Observable **??**

ObservableHeapSorter< MyListElementType > **??**

ObservableMergeSorter< MyListElementType > **??**

ObservableQuickSorter< MyListElementType > **??**

Observer **??**

MyBenchmarkObserver **??**

Sorter< MyListElementType > **??**

HeapSorter< MyListElementType > **??**

ObservableHeapSorter< MyListElementType > **??**

MergeSorter< MyListElementType > **??**

ObservableMergeSorter< MyListElementType > **??**

QuickSorter< MyListElementType > **??**

ObservableQuickSorter< MyListElementType > **??**

2 Indeks klas

2.1 Lista klas

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

HeapSorter< MyListElementType >
Klasa sluzaca do obslugi sortowania przez kopcowanie ??

List< MyListElementType > ??

ListElement< MyListElementType >
Klasa ma byc uzywana jako abstrakcyjna do implementacji pojedynczego elemenu listy ??

ListSaver< MyListElementType > ??

MergeSorter< MyListElementType >
Klasa sluzaca do obslugi sortowania przez Scalanie ??

MyBenchmark
Klasa bazowa/interface do testowania algorytmu 6

MyBenchmarkObserver
Mybenchmark obserwator Używana jako obserwator klasa sprawdzajaca odpowiednie objekty ??

MyList< MyListElementType >
Lista dwukierunkowa 8

MyListElement< MyListElementType >
Klasa 'malych struktur' gdzie jest numer i wskaznik do nas elementu ??

NumberGenerator
Klasa generujaca losowe liczby 14

Observable
Klasa abstrakcyjna- bazowa dla objektow do obserowania ??

ObservableHeapSorter< MyListElementType >
Klasa sluzaca do obslugi sortowania przez kopcowanie z dodaniem obserwatora ??

ObservableMergeSorter< MyListElementType >
Klasa sluzaca do obslugi sortowania przez Scalanie z dodaniem obserwatora ??

ObservableQuickSorter< MyListElementType >
Klasa sluzaca do obslugi sortowania przez Sortowanie szybkie z dodaniem obserwatora ??

Observer
Obserwator ??

QuickSorter< MyListElementType >
Klasa sluzaca do obslugi sortowania przez Scalanie ??

Sorter< MyListElementType >
Interfejs kazdego sortowania ??

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

filestreamer.h ??

heapsorter.h ??

list.h ??

listelement.h ??

listsaver.h ??

main.cpp 21

mergesorter.h ??

mybenchmark.cpp 23

mybenchmark.h 24

mylist.h 25

mylistelement.h ??

numbergenerator.h 27

observable.h ??

observableheapsorter.h ??

observablemergesorter.h ??

observablequicksorter.h ??

observer.h ??

quicksorter.h ??

sorter.h ??

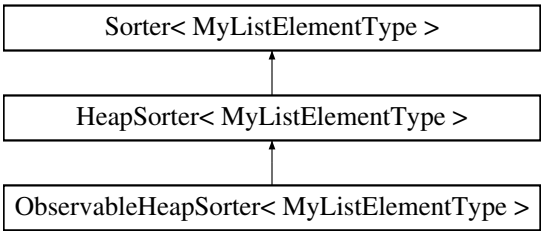
4 Dokumentacja klas

4.1 Dokumentacja szablonu klasy HeapSorter< MyListElementType >

Klasa sluzaca do obslugi sortowania przez kopcowanie.

`#include <heapsorter.h>`

Diagram dziedziczenia dla HeapSorter< MyListElementType >



Metody publiczne

- HeapSorter (List< MyListElementType > &myList)

  Konstruktor.
- virtual ~HeapSorter ()
- List< MyListElementType > & sort ()

  Sortuje przez kopcowanie.

Atrybuty publiczne

- List< MyListElementType > & list

  Skopiowana lista do przeprowadzania sortowania.

4.1.1 Opis szczegółowy

template<class MyListElementType>class HeapSorter< MyListElementType >

Definicja w linii 17 pliku heapsorter.h.

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 template<class MyListElementType > HeapSorter< MyListElementType >::HeapSorter (List< MyListElementType > & *myList*) [inline]

Parametry

&myList	lista, która konstruktor kopiuje aby nie naruszać podanej przez uzytkownika

Definicja w linii 26 pliku heapsorter.h.

Odwołuje się do HeapSorter< MyListElementType >::list.

```
00027         :list(myList.createObjectFromAbstractReference())
00028
00029         {
00030                 this->list.cloneFrom(myList);
00031                 /*this->sizeOfList = myList.sizeOfList;
00032                 this->firstElement = myList.firstElement;
00033                 this->lastElement = myList.lastElement;
00034                 this->iterator=myList.iterator;
00035                 this->isIteratorAfterPop = myList.isIteratorAfterPop;*/
00036         }
```

4.1.2.2 template<class MyListElementType > virtual HeapSorter< MyListElementType >::~HeapSorter ( ) [inline], [virtual]

Definicja w linii 38 pliku heapsorter.h.

```
00038 {};
```

4.1.3 Dokumentacja funkcji składowych

4.1.3.1 template<class MyListElementType > List<MyListElementType>& HeapSorter< MyListElementType >::sort ( ) [inline], [virtual]

Implementuje Sorter< MyListElementType >.

Reimplementowana w ObservableHeapSorter< MyListElementType >.

Definicja w linii 42 pliku heapsorter.h.

Odwołuje się do HeapSorter< MyListElementType >::list.

Odwołania w ObservableHeapSorter< MyListElementType >::sort().

```
00043         {
00044                 int n = this->list.size();
00045             int parent = n/2, index, child, tmp; /* heap indexes */
00046             /* czekam az sie posortuje */
00047             while (1) {
00048                 if (parent > 0)
00049                 {
00050                     tmp = (this->list)[--parent].content;  /* kobie kopie do tmp */
00051                 }
00052                 else {
00053                     n--;
00054                     if (n == 0)
00055                     {
00056                         return this->list; /* Zwraca posortowane */
00057                     }
00058                     tmp = this->list[n].content;
00059                     this->list[n].content = this->list[0].content;
00060                 }
00061                 index = parent;
00062                 child = index * 2 + 1;
00063                 while (child < n) {
00064                     if (child + 1 < n  &&  this->list[child + 1].content > this->
      list[child].content) {
00065                         child++;
00066                     }
00067                     if (this->list[child].content > tmp) {
00068                         this->list[index].content = this->list[child].content;
00069                         index = child;
00070                         child = index * 2 + 1;
00071                     } else {
00072                         break;
00073                     }
00074                 }
00075                 this->list[index].content = tmp;
00076             }
00077             return this->list;
00078         }
```

4.1.4 Dokumentacja atrybutów składowych

4.1.4.1 template<class MyListElementType > List<MyListElementType>& HeapSorter< MyListElementType >::list

Definicja w linii 21 pliku heapsorter.h.

Odwołania w HeapSorter< MyListElementType >::HeapSorter(), main(), ObservableHeapSorter< MyListElementType >::sort() i HeapSorter< MyListElementType >::sort().

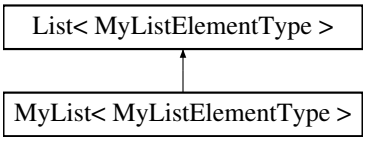
Dokumentacja dla tej klasy została wygenerowana z pliku:

- heapsorter.h

4.2 Dokumentacja szablonu klasy List< MyListElementType >

`#include <list.h>`

Diagram dziedziczenia dla List< MyListElementType >



Metody publiczne

- virtual int & size ()=0

Pobiera rozmiar listy.

- virtual ListElement < MyListElementType > pop_back ()=0

  Zwraca ostatni element z listy.
- virtual ListElement < MyListElementType > pop_front ()=0

  Zwraca pierwszy element z listy.
- virtual void printList ()=0

  Wyswietla liste.
- virtual void push_back (MyListElementType arg)=0
- virtual void push_front (MyListElementType arg)=0

  Wsadza MyListElementType do listy na poczatek.
- virtual MyListElement < MyListElementType > & operator[] (int numberOfElement)=0

  Wsadza MyListElementType do listy na koniec.
- virtual void insertAfter (MyListElement< MyListElementType > arg, int iteratorID)=0

  Wsadza MyListElementType po elemencie.
- virtual MyListElementType & show_front ()=0

  Pokazue pierwszy element na liscie.
- virtual MyListElementType & show_back ()=0

  Pokazue ostatni element na liscie.
- virtual void cloneFrom (List< MyListElementType > &patternList)

  Klonuje listy przydzielajac dla nowej nową pamięć dla każdego z jej elementu.
- virtual List< MyListElementType > & createObjectFromAbstractReference ()=0

  Wzorzec projektowy - fabryki abstrakcyjnej.
- virtual void free ()

  Zwalnia zasoby listy.
- virtual ~List ()

4.2.1 Opis szczegółowy

template<class MyListElementType>class List< MyListElementType >

Interface dla klasy przedstawiających listy

Definicja w linii 17 pliku list.h.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 template<class MyListElementType> virtual List< MyListElementType >::~List ( ) [inline], [virtual]

Definicja w linii 84 pliku list.h.

```
00084 {};
```

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 template<class MyListElementType> virtual void List< MyListElementType >::cloneFrom (List< MyListElementType > & *patternList*) [inline], [virtual]

Definicja w linii 69 pliku list.h.

Odwołania w QuickSorter< MyListElementType >::QuickSorter().

```
00070         {
00071                 // release memory from main list
00072                 while(this->size()) pop_back();
00073                 for(int i=0; i<patternList.size(); i++)
00074                         this->push_back(patternList[i].content);
00075         }
```

4.2.3.2 template<class MyListElementType> virtual List<MyListElementType>& List< MyListElementType >::createObjectFromAbstractReference ( ) [pure virtual]

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.3 template<class MyListElementType> virtual void List< MyListElementType >::free ( ) [inline], [virtual]

Definicja w linii 83 pliku list.h.

Odwołania w main().

```
00083 { while(size()) pop_back(); }
```

4.2.3.4 template<class MyListElementType> virtual void List< MyListElementType >::insertAfter (MyListElement< MyListElementType > *arg*, int *iteratorID*) [pure virtual]

Parametry

arg	Element do wsadzenia
iteratorID	id elementu do wsadzenia

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.5 template<class MyListElementType> virtual MyListElement<MyListElementType>& List< MyListElementType >::operator[] (int *numberOfElement*) [pure virtual]

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.6 template<class MyListElementType> virtual ListElement<MyListElementType> List< MyListElementType >::pop_back ( ) [pure virtual]

Zwraca

ostatni element z listy

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

Odwołania w List< Observer * >::cloneFrom() i List< Observer * >::free().

4.2.3.7 template<class MyListElementType> virtual ListElement<MyListElementType> List< MyListElementType >::pop_front ( ) [pure virtual]

Zwraca

pierwszy element z listy

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.8 template<class MyListElementType> virtual void List< MyListElementType >::printList ( ) [pure virtual]

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.9 template<class MyListElementType> virtual void List< MyListElementType >::push_back (MyListElementType *arg*) [pure virtual]

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

Odwołania w List< Observer * >::cloneFrom().

4.2.3.10 template<class MyListElementType> virtual void List< MyListElementType >::push_front (MyListElementType *arg*) [pure virtual]

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.11 template<class MyListElementType> virtual MyListElementType& List< MyListElementType >::show_back ( ) [pure virtual]

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.12 template<class MyListElementType> virtual MyListElementType& List< MyListElementType >::show_front ( ) [pure virtual]

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

4.2.3.13 template<class MyListElementType> virtual int& List< MyListElementType >::size ( ) [pure virtual]

Zwraca

Rozmiar listy

Implementowany w MyList< MyListElementType > i MyList< Observer * >.

Odwołania w List< Observer * >::cloneFrom(), List< Observer * >::free() i MyList< Observer * >::MyList().

Dokumentacja dla tej klasy została wygenerowana z pliku:

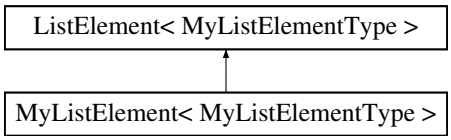
- list.h

4.3 Dokumentacja szablonu klasy ListElement< MyListElementType >

klasa ma byc uzywana jako abstrakcyjna do implementacji pojedynczego elemenu listy

`#include <listelement.h>`

Diagram dziedziczenia dla ListElement< MyListElementType >



Metody publiczne

- virtual ~ListElement ()

Atrybuty publiczne

- MyListElementType content
  Zawartość pojedynczego elelemntu tablicy.

4.3.1 Opis szczegółowy

template<class MyListElementType>class ListElement< MyListElementType >

Definicja w linii 14 pliku listelement.h.

4.3.2 Dokumentacja konstruktora i destruktora

4.3.2.1 template<class MyListElementType> virtual ListElement< MyListElementType >::~ListElement ( ) [inline], [virtual]

Definicja w linii 21 pliku listelement.h.

```
00021 {};
```

4.3.3 Dokumentacja atrybutów składowych

4.3.3.1 template<class MyListElementType> MyListElementType ListElement< MyListElementType >::content

Definicja w linii 19 pliku listelement.h.

Odwołania w MyList< Observer * >::insertAfter(), MyListElement< Observer * >::MyListElement(), MyList< Observer * >::printList() i MyListElement< Observer * >::set().

Dokumentacja dla tej klasy została wygenerowana z pliku:

- listelement.h

4.4 Dokumentacja szablonu klasy ListSaver< MyListElementType >

`#include <listsaver.h>`

Metody prywatne

- ListSaver (MyList< MyListElementType > &listArgument)

  Konstruktor pobierajacy referencje do listy do zapisu.
- void saveToFile (std::string nazwaPliku)

  Zapisuje liste do pliku.

Atrybuty prywatne

- List< MyListElementType > & list

  Klasa pozwalająca na zapis Listy do pliku.

4.4.1 Opis szczegółowy

template<class MyListElementType>class ListSaver< MyListElementType >

Definicja w linii 15 pliku listsaver.h.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 template<class MyListElementType > ListSaver< MyListElementType >::ListSaver (MyList< MyListElementType > & *listArgument*) [inline], [private]

Parametry

listArgument	lista do zapisu

Definicja w linii 24 pliku listsaver.h.

```
00024                                                   :
00025                 list(listArgument)
00026         {}
```

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 template<class MyListElementType > void ListSaver< MyListElementType >::saveToFile (std::string *nazwaPliku*) [inline], [private]

Zwraca

Zwraca 0 gdy zapisywanie powiodlo sie

Definicja w linii 32 pliku listsaver.h.

Odwołuje się do ListSaver< MyListElementType >::list.

```
00033         {
00034                 std::ofstream streamToFile;
00035                 streamToFile.open (nazwaPliku.c_str(), std::ofstream::out);
00036                 for(int i=0; i<list.size() ; i++)
00037                         streamToFile << '{'<<list[i].content<<"} ";
00038                 streamToFile.close();
00039         }
```

4.4.4 Dokumentacja atrybutów składowych

4.4.4.1 template<class MyListElementType > List<MyListElementType>& ListSaver< MyListElementType >::list [private]

Definicja w linii 19 pliku listsaver.h.

Odwołania w ListSaver< MyListElementType >::saveToFile().

Dokumentacja dla tej klasy została wygenerowana z pliku:

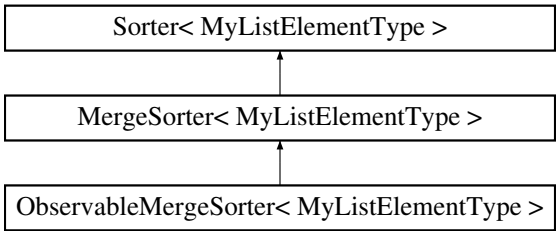
- listsaver.h

4.5 Dokumentacja szablonu klasy MergeSorter< MyListElementType >

Klasa sluzaca do obslugi sortowania przez Scalanie.

`#include <mergesorter.h>`

Diagram dziedziczenia dla MergeSorter< MyListElementType >



Metody publiczne

- MergeSorter (MyList< MyListElementType > &listArg)

  Konstruktor.
- virtual ~MergeSorter ()
- MyList< MyListElementType > merge (MyList< MyListElementType > left, MyList< MyListElementType > right)

  Scalanie list.
- MyList< MyListElementType > mergeSort (MyList< MyListElementType > m)

  Sortuje liste przez scalanie.
- List< MyListElementType > & sort ()

  Sortuje przez scalanie.

Atrybuty publiczne

- MyList< MyListElementType > & list

  Skopiowana lista do przeprowadzania sortowania.

4.5.1 Opis szczegółowy

template<class MyListElementType>class MergeSorter< MyListElementType >

Definicja w linii 16 pliku mergesorter.h.

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 template<class MyListElementType > MergeSorter< MyListElementType >::MergeSorter (MyList< MyListElementType > & *listArg*) [inline]

Parametry

listArg	lista, która konstruktor kopiuje aby nie naruszać podanej przez uzytkownika

Definicja w linii 25 pliku mergesorter.h.

```
00026         :list(listArg)      {}
```

4.5.2.2 template<class MyListElementType > virtual MergeSorter< MyListElementType >::~MergeSorter ( ) [inline], [virtual]

Definicja w linii 28 pliku mergesorter.h.

```
00028 {}
```

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 template<class MyListElementType > MyList<MyListElementType> MergeSorter< MyListElementType >::merge (MyList< MyListElementType > *left,* MyList< MyListElementType > *right*) [inline]

Parametry

left	lewa lista do scalania
right	prawa lista do scalania

Zwraca

zwraca posotrowana liste

Definicja w linii 36 pliku mergesorter.h.

Odwołuje się do MyList< MyListElementType >::pop_front(), MyList< MyListElementType >::push_back(), MyList< MyListElementType >::show_front() i MyList< MyListElementType >::size().

Odwołania w MergeSorter< MyListElementType >::mergeSort().

```
00037         {
00038                 MyList<MyListElementType> result;
00039                 //Gdy jest jeszcze cos do sortowania
00040                 while (left.size() > 0 || right.size() > 0)
00041                 {
00042                         // Jak oba to zamieniamy
00043                         if (left.size() > 0 && right.size() > 0)
00044                         {
00045                                 // Sprawdzam czy zamieniac
00046                                 if (left.show_front() <= right.
      show_front())
00047                                         {
00048                                                 result.push_back(left.
      show_front()); left.pop_front();
00049                                         }
00050                                 else
00051                                 {
00052                                         result.push_back(right.
      show_front()); right.pop_front();
00053                                 }
00054                         }
00055                         // pojedyncze listy (nieparzyse)
00056                         else if (left.size() > 0)
00057                         {
00058                                 for (int i = 0; i < left.size(); i++) result.
      push_back(left[i].content); break;
00059                         }
00060                         // pojedyncze listy (nieparzyse- taka sama sytuacja jak wyzej)
00061                         else if ((int)right.size() > 0)
00062                         {
00063                                 for (int i = 0; i < (int)right.size(); i++) result.
      push_back(right[i].content); break;
00064                         }
00065                 }
00066                 return result;
00067         }
```

4.5.3.2 template<class MyListElementType > MyList<MyListElementType> MergeSorter< MyListElementType >::mergeSort (MyList< MyListElementType > *m*) [inline]

Parametry

m	Lista do posotrowania

Zwraca

zwraca posotrowana liste

Definicja w linii 73 pliku mergesorter.h.

Odwołuje się do MergeSorter< MyListElementType >::merge(), MyList< MyListElementType >::push_back() i MyList< MyListElementType >::size().

Odwołania w MergeSorter< MyListElementType >::sort().

```
00074         {
00075                 if (m.size() <= 1) return m; // gdy juz nic nie ma do sotrowania
00076                 MyList<MyListElementType> left, right, result;
```

```
00077                 int middle = (m.size()+ 1) / 2; // anty-nieparzyscie
00078                 for (int i = 0; i < middle; i++)
00079                         {
00080                                 left.push_back(m[i].content);
00081                         }
00082                 for (int i = middle; i < m.size(); i++)
00083                         {
00084                                 right.push_back(m[i].content);
00085                         }
00086                 left = mergeSort(left);
00087                 right = mergeSort(right);
00088                 result = merge(left, right);
00089                 return result;
00090         }
```

4.5.3.3 template<class MyListElementType > List<MyListElementType>& MergeSorter< MyListElementType >::sort ( ) [inline], [virtual]

Implementuje Sorter< MyListElementType >.

Reimplementowana w ObservableMergeSorter< MyListElementType >.

Definicja w linii 95 pliku mergesorter.h.

Odwołuje się do MergeSorter< MyListElementType >::list i MergeSorter< MyListElementType >::mergeSort().

Odwołania w ObservableMergeSorter< MyListElementType >::sort().

```
00096         {
00097                 this->list=mergeSort(this->list);
00098                 return this->list;
00099         }
```

4.5.4 Dokumentacja atrybutów składowych

4.5.4.1 template<class MyListElementType > MyList<MyListElementType>& MergeSorter< MyListElementType >::list

Definicja w linii 20 pliku mergesorter.h.

Odwołania w main(), ObservableMergeSorter< MyListElementType >::sort() i MergeSorter< MyListElementType >::sort().

Dokumentacja dla tej klasy została wygenerowana z pliku:

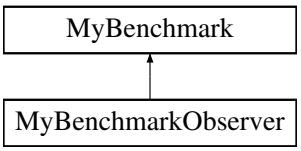
- mergesorter.h

4.6 Dokumentacja klasy MyBenchmark

Klasa bazowa/interface do testowania algorytmu.

`#include <mybenchmark.h>`

Diagram dziedziczenia dla MyBenchmark



Metody publiczne

- MyBenchmark ()
- void timerStart ()

włączam stoper

- double timerStop ()

  wyłączam stoper

- virtual ~MyBenchmark ()

  Usuwam obiekt test biorąc pod uwage jego prawdziwy typ.

Atrybuty publiczne

- double timerValue

  Czas stopera.

4.6.1 Opis szczegółowy

Używana jako interface dla wszystkich algorytmow aby testowac czas wykonywanego algorymtu.

Definicja w linii 20 pliku mybenchmark.h.

4.6.2 Dokumentacja konstruktora i destruktora

4.6.2.1 MyBenchmark::MyBenchmark ( ) [inline]

Definicja w linii 27 pliku mybenchmark.h.

Odwołuje się do timerValue.

```
00028         {
00029                 timerValue = 0;
00030         }
```

4.6.2.2 virtual MyBenchmark::~MyBenchmark ( ) [inline], [virtual]

Definicja w linii 44 pliku mybenchmark.h.

```
00044 {};
```

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 void MyBenchmark::timerStart ( )

Definicja w linii 12 pliku mybenchmark.cpp.

Odwołuje się do timerValue.

Odwołania w MyBenchmarkObserver::receivedStartUpdate().

```
00013 {
00014         timerValue = (( (double)clock() ) /CLOCKS_PER_SEC);
00015 }
```

4.6.3.2 double MyBenchmark::timerStop ( )

Zwraca

Dlugosc dzialania stopera

Definicja w linii 17 pliku mybenchmark.cpp.

Odwołuje się do timerValue.

```
00018 {
00019         return (( (double)clock() ) /CLOCKS_PER_SEC) - timerValue;
00020 }
```

4.6.4 Dokumentacja atrybutów składowych

4.6.4.1 double MyBenchmark::timerValue

Definicja w linii 25 pliku mybenchmark.h.

Odwołania w MyBenchmarkObserver::getTimerValue(), MyBenchmark(), timerStart() i timerStop().

Dokumentacja dla tej klasy została wygenerowana z plików:

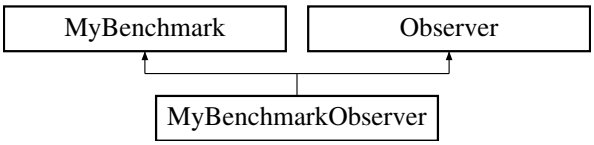
- mybenchmark.h
- mybenchmark.cpp

4.7 Dokumentacja klasy MyBenchmarkObserver

Mybenchmark obserwator Używana jako obserwator klasa sprawdzajaca odpowiednie objekty.

`#include <mybenchmark.h>`

Diagram dziedziczenia dla MyBenchmarkObserver



Metody publiczne

- MyBenchmarkObserver ()
- double getTimerValue ()

  pobiera czas trwania algorytmu
- void receivedStartUpdate ()

  Odbiera powiadomienie o rozpoczeciu dzialania algorytmu.
- void receivedStopUpdate ()

  Odbiera powiadomienie o zakonczeniu dzialania algorytmu.
- virtual ~MyBenchmarkObserver ()

Dodatkowe Dziedziczone Składowe

4.7.1 Opis szczegółowy

Definicja w linii 52 pliku mybenchmark.h.

4.7.2 Dokumentacja konstruktora i destruktora

4.7.2.1 MyBenchmarkObserver::MyBenchmarkObserver ( ) [inline]

Definicja w linii 55 pliku mybenchmark.h.

```
00055 {};
```

4.7.2.2 virtual MyBenchmarkObserver::~MyBenchmarkObserver ( ) [inline], [virtual]

Definicja w linii 73 pliku mybenchmark.h.

```
00073 {};
```

4.7.3 Dokumentacja funkcji składowych

4.7.3.1 double MyBenchmarkObserver::getTimerValue ( ) [inline], [virtual]

Zwraca

czas trwania algorytmu

Implementuje Observer.

Definicja w linii 60 pliku mybenchmark.h.

Odwołuje się do MyBenchmark::timerValue.

```
00060 {return this->timerValue;}
```

4.7.3.2 void MyBenchmarkObserver::receivedStartUpdate ( ) [inline], [virtual]

Implementuje Observer.

Definicja w linii 64 pliku mybenchmark.h.

Odwołuje się do MyBenchmark::timerStart().

```
00064                                   {
00065                 timerStart();
00066         }
```

4.7.3.3 void MyBenchmarkObserver::receivedStopUpdate ( ) [inline], [virtual]

Implementuje Observer.

Definicja w linii 70 pliku mybenchmark.h.

```
00070                                   {
00071                 // std::cout<<"\nCzas wykonywania operacji: "<<timerStop();
00072         }
```

Dokumentacja dla tej klasy została wygenerowana z pliku:

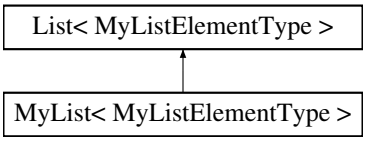
- mybenchmark.h

4.8 Dokumentacja szablonu klasy MyList< MyListElementType >

Lista dwukierunkowa.

`#include <mylist.h>`

Diagram dziedziczenia dla MyList< MyListElementType >



Metody publiczne

- MyList ()

  Konstruktor listy.

- MyList (List< MyListElementType > &list)
- virtual ~MyList ()
- int & size ()

  Zwraca ilosc elementow listy.
- ListElement< MyListElementType > pop_back ()

  Zwraca element ostatni w liscie.
- ListElement< MyListElementType > pop_front ()

  Zwraca element pierwszy w liscie.
- void push_back (MyListElementType arg)

  Wklada element na ostatnie miejsce na liscie.
- void push_front (MyListElementType arg)

  Wklada element na pierwsze miejsce na liscie.
- MyListElementType & show_front ()

  Pokazuje element po poczatku listy.
- MyListElementType & show_back ()

  Pokazuje element po koncu listy.
- void printList ()

  Wyswietla elementy listy.
- MyListElement
  < MyListElementType > & operator[] (int numberOfElement)

  Pobiera element z listy.
- void insertAfter (MyListElement< MyListElementType > arg, int iteratorID)

  Wsadza element po obiekcie iteratora.
- MyList< MyListElementType > & operator= (const MyList< MyListElementType > &pattern)
- List< MyListElementType > & createObjectFromAbstractReference ()

  Wzorzec projektowy - fabryki abstrakcyjnej.

Atrybuty publiczne

- int sizeOfList

  liczba elementow listy
- MyListElement
  < MyListElementType > * firstElement

  wskaznik do 'malej struktury' ktora jest pierwsza na liscie
- MyListElement
  < MyListElementType > * lastElement

  wskaznik do 'malej struktury' ktora jest ostatnia na liscie
- MyListElement
  < MyListElementType > * iterator
- int iteratorElementId
- int isIteratorAfterPop

4.8.1 Opis szczegółowy

template<class MyListElementType>class MyList< MyListElementType >

Klasa przedstawia liste dwukierunkową dynamiczna

Definicja w linii 23 pliku mylist.h.

4.8.2 Dokumentacja konstruktora i destruktora

4.8.2.1 template<class MyListElementType> MyList< MyListElementType >::MyList ( ) [inline]

Definicja w linii 39 pliku mylist.h.

```
00040         {
00041                 firstElement = lastElement = new
      MyListElement<MyListElementType>;
00042                 sizeOfList = 0;
00043                 iteratorElementId =0;
00044                 iterator=NULL;
00045                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00046         }
```

4.8.2.2 template<class MyListElementType> MyList< MyListElementType >::MyList (List< MyListElementType > & *list*) [inline]

Definicja w linii 48 pliku mylist.h.

```
00049         {
00050                 firstElement = lastElement = new
      MyListElement<MyListElementType>;
00051                 sizeOfList = 0;
00052                 iteratorElementId =0;
00053                 iterator=NULL;
00054                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00055                 for(int i=0; i<list.size(); i++)
00056                 {
00057                         this->push_back(list[i]);
00058                 }
00059         }
```

4.8.2.3 template<class MyListElementType> virtual MyList< MyListElementType >::~MyList ( ) [inline], [virtual]

Definicja w linii 60 pliku mylist.h.

```
00060 {};
```

4.8.3 Dokumentacja funkcji składowych

4.8.3.1 template<class MyListElementType> List<MyListElementType>& MyList< MyListElementType >::createObjectFromAbstractReference ( ) [inline], [virtual]

Implementuje List< MyListElementType >.

Definicja w linii 267 pliku mylist.h.

```
00268         {
00269                 return *new MyList<MyListElementType>;
00270         }
```

4.8.3.2 template<class MyListElementType> void MyList< MyListElementType >::insertAfter (MyListElement< MyListElementType > *arg,* int *iteratorID*) [inline], [virtual]

Implementuje List< MyListElementType >.

Definicja w linii 212 pliku mylist.h.

```
00213         {
00214                 if(iteratorID==0 && this->sizeOfList==0)  {push_front(arg.
      content); return;}
00215                 if(iteratorID==this->sizeOfList-1)  {push_back(arg.
```

```
      content); return;}
00216                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00217                 MyListElement<MyListElementType> &tmpThis=(*this)[
      iteratorID], &tmpNext=(*this)[iteratorID+1];
00218                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00219                 newMyListElement -> nextElement = tmpThis.nextElement;
00220                 newMyListElement -> previousElement = &tmpThis;
00221                 tmpThis.nextElement = newMyListElement;
00222                 tmpNext.previousElement = newMyListElement;
00223                 isIteratorAfterPop=1;
00224         }
```

4.8.3.3 template<class MyListElementType> MyList<MyListElementType>& MyList< MyListElementType >::operator= (const MyList< MyListElementType > & *pattern*) [inline]

Definicja w linii 234 pliku mylist.h.

```
00235         {
00236                 //std::cerr<<" @@@";
00237                 this->sizeOfList = pattern.sizeOfList;
00238                 this->firstElement = pattern.firstElement;
00239                 this->lastElement = pattern.lastElement;
00240                 this->iterator=pattern.iterator;
00241                 this->isIteratorAfterPop = pattern.
      isIteratorAfterPop;
00242                 return *this;
00243         }
```

4.8.3.4 template<class MyListElementType> MyListElement<MyListElementType>& MyList< MyListElementType >::operator[] (int *numberOfElement*) [inline], [virtual]

Zwraca

Zwraca 0 gdy zapisywanie powiodlo sie

Implementuje List< MyListElementType >.

Definicja w linii 171 pliku mylist.h.

```
00172         {
00173                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00174                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00175                         {
00176                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
       !";
00177                                 return *iterator;
00178                         }
00179                 if(isIteratorAfterPop)
00180                         {
00181                                 iteratorElementId=0;  // czyli iterator byl zpopowany
00182                                 iterator = firstElement;
00183                                 isIteratorAfterPop=0;
00184                         }
00185                 //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00186                 if((numberOfElement <= iteratorElementId-numberOfElement) &&(
      iteratorElementId-numberOfElement>=0))
00187                 {
00188                         //std::cerr<<"\nJestem w if_1";
00189                         iterator = (this->firstElement);
00190                         iteratorElementId = 0;
00191                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00192                                 iterator = (iterator->nextElement);
00193                 }
00194                 else if(numberOfElement > iteratorElementId)
00195                 {
00196                         //std::cerr<<"\nJestem w if_2";
00197                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00198                                                 iterator = (iterator->nextElement);
00199                 }
00200                 else if( numberOfElement < iteratorElementId)
00201                 {
00202                         //std::cerr<<"\nJestem w if_3";
00203                         for (; iteratorElementId> numberOfElement ;
      iteratorElementId--)
```

```
00204                                                   iterator = (iterator->previousElement);
00205                 }
00206                 return *iterator;
00207         }
```

4.8.3.5 template<class MyListElementType> ListElement<MyListElementType> MyList< MyListElementType >::pop_back ( ) [inline], [virtual]

Zwraca

Zwraca element ostatni w liscie

Implementuje List< MyListElementType >.

Definicja w linii 84 pliku mylist.h.

```
00085         {
00086                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new
      MyListElement<MyListElementType>)); }
00087                 MyListElement<MyListElementType> tmpNumber = *(this ->
      lastElement);
00088                 MyListElement<MyListElementType> *originMyListElement =
      this -> lastElement;
00089                 this -> lastElement = this -> lastElement -> previousElement;
00090                 delete originMyListElement;
00091                 isIteratorAfterPop=1;
00092                 return tmpNumber;
00093         }
```

4.8.3.6 template<class MyListElementType> ListElement<MyListElementType> MyList< MyListElementType >::pop_front ( ) [inline], [virtual]

Zwraca

Zwraca element pierwszy w liscie

Implementuje List< MyListElementType >.

Definicja w linii 98 pliku mylist.h.

Odwołania w MergeSorter< MyListElementType >::merge().

```
00099         {
00100                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new
      MyListElement<MyListElementType>())); }
00101                 MyListElement<MyListElementType> tmpNumber = *(this ->
      firstElement);
00102                 MyListElement<MyListElementType> *originMyListElement =
      this -> firstElement;
00103                 this -> firstElement = this -> firstElement -> nextElement;
00104
00105                 delete originMyListElement;
00106                 isIteratorAfterPop=1;
00107                 return tmpNumber;
00108         }
```

4.8.3.7 template<class MyListElementType> void MyList< MyListElementType >::printList ( ) [inline], [virtual]

Implementuje List< MyListElementType >.

Definicja w linii 156 pliku mylist.h.

```
00157         {
00158                 MyListElement<MyListElementType> *elem = (this->
      firstElement);
00159                 std::cout<<"\nWyswietlam liste (size:"<<this->sizeOfList<<"): ";
00160                 for(int i=0; i< this->sizeOfList; i++)
00161                 {
00162                         std::cout<<" "<<elem->content;
00163                         elem = elem->nextElement;
00164                 }
00165         }
```

4.8.3.8 template<class MyListElementType> void MyList< MyListElementType >::push_back (MyListElementType *arg*) [inline], [virtual]

Implementuje List< MyListElementType >.

Definicja w linii 112 pliku mylist.h.

Odwołania w Observable::add(), NumberGenerator::generateNumbers(), MyList< Observer * >::insertAfter(), MergeSorter< MyListElementType >::merge(), MergeSorter< MyListElementType >::mergeSort() i MyList< Observer * >::MyList().

```
00113         {
00114                 //std::cerr<<"\n(push_back): arg.content="<<arg.content;
00115                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00116                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00117                 //newMyListElement -> nextElement = 0;
00118                 newMyListElement -> previousElement = this -> lastElement;
00119                 this -> lastElement -> nextElement = newMyListElement;
00120                 this->lastElement = newMyListElement;
00121         }
```

4.8.3.9 template<class MyListElementType> void MyList< MyListElementType >::push_front (MyListElementType *arg*) [inline], [virtual]

Implementuje List< MyListElementType >.

Definicja w linii 125 pliku mylist.h.

Odwołania w MyList< Observer * >::insertAfter().

```
00126         {
00127                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00128                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00129                 //newMyListElement -> previousElement =  0;
00130                 newMyListElement -> nextElement = this -> firstElement;
00131                 this -> firstElement -> previousElement = newMyListElement;
00132                 this->firstElement = newMyListElement;
00133                 ++iteratorElementId;
00134         }
```

4.8.3.10 template<class MyListElementType> MyListElementType& MyList< MyListElementType >::show_back ( ) [inline], [virtual]

Zwraca

zwraca kopie tego elementu

Implementuje List< MyListElementType >.

Definicja w linii 147 pliku mylist.h.

```
00148         {
00149                 return lastElement->content;
00150         }
```

4.8.3.11 template<class MyListElementType> MyListElementType& MyList< MyListElementType >::show_front ( ) [inline], [virtual]

Zwraca

zwraca kopie tego elementu

Implementuje List< MyListElementType >.

Definicja w linii 139 pliku mylist.h.

Odwołania w MergeSorter< MyListElementType >::merge().

```
00140         {
00141                 return firstElement->content;
00142         }
```

4.8.3.12 template<class MyListElementType> int& MyList< MyListElementType >::size ( ) [inline], [virtual]

Zwraca

ilosc elementow tablicy

Implementuje List< MyListElementType >.

Definicja w linii 66 pliku mylist.h.

Odwołania w MergeSorter< MyListElementType >::merge(), MergeSorter< MyListElementType >::mergeSort(), Observable::sendStartUpdateToObservers() i Observable::sendStopUpdateToObservers().

```
00067         {
00068                 return sizeOfList;
00069         }
```

4.8.4 Dokumentacja atrybutów składowych

4.8.4.1 template<class MyListElementType> MyListElement<MyListElementType>* MyList< MyListElementType >::firstElement

Definicja w linii 31 pliku mylist.h.

Odwołania w MyList< Observer * >::insertAfter(), MyList< Observer * >::MyList(), MyList< Observer * >::operator=(), MyList< Observer * >::operator[](), MyList< Observer * >::pop_front(), MyList< Observer * >::printList(), MyList< Observer * >::push_back(), MyList< Observer * >::push_front() i MyList< Observer * >::show_front().

4.8.4.2 template<class MyListElementType> int MyList< MyListElementType >::isIteratorAfterPop

Definicja w linii 36 pliku mylist.h.

Odwołania w MyList< Observer * >::insertAfter(), MyList< Observer * >::MyList(), MyList< Observer * >::operator=(), MyList< Observer * >::operator[](), MyList< Observer * >::pop_back() i MyList< Observer * >::pop_front().

4.8.4.3 template<class MyListElementType> MyListElement<MyListElementType>* MyList< MyListElementType >::iterator

Definicja w linii 34 pliku mylist.h.

Odwołania w MyList< Observer * >::MyList(), MyList< Observer * >::operator=() i MyList< Observer * >::operator[]().

4.8.4.4 template<class MyListElementType> int MyList< MyListElementType >::iteratorElementId

Definicja w linii 35 pliku mylist.h.

Odwołania w MyList< Observer * >::MyList(), MyList< Observer * >::operator[]() i MyList< Observer * >::push_front().

4.8.4.5 template<class MyListElementType> MyListElement<MyListElementType>* MyList< MyListElementType >::lastElement

Definicja w linii 33 pliku mylist.h.

Odwołania w MyList< Observer * >::insertAfter(), MyList< Observer * >::MyList(), MyList< Observer * >::operator=(), MyList< Observer * >::pop_back(), MyList< Observer * >::push_back(), MyList< Observer * >::push_front() i MyList< Observer * >::show_back().

4.8.4.6 template<class MyListElementType> int MyList< MyListElementType >::sizeOfList

Definicja w linii 27 pliku mylist.h.

Odwołania w MyList< Observer * >::insertAfter(), MyList< Observer * >::MyList(), MyList< Observer * >::operator=(), MyList< Observer * >::operator[](), MyList< Observer * >::pop_back(), MyList< Observer * >::pop_front(), MyList< Observer * >::printList(), MyList< Observer * >::push_back(), MyList< Observer * >::push_front() i MyList< Observer * >::size().

Dokumentacja dla tej klasy została wygenerowana z pliku:

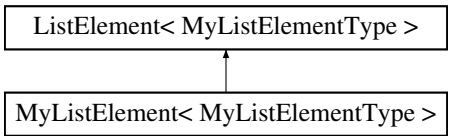
- mylist.h

4.9 Dokumentacja szablonu klasy MyListElement< MyListElementType >

Klasa 'malych struktur' gdzie jest numer i wskaznik do nas elementu.

`#include <mylistelement.h>`

Diagram dziedziczenia dla MyListElement< MyListElementType >



Metody publiczne

- MyListElement ()

  Konstruktor wewnetrznej klasy 'malych struktur'.
- MyListElement (MyListElementType arg)

  Konstruktor wewnetrznej klasy 'malych struktur'.
- MyListElement (const MyListElement &myListElement)

  Konstruktor kopiujacy wewnetrznej klasy 'malych struktur'.
- void set (MyListElementType arg)

  Ustawia liczbe oraz klucz slowanika dla elementu.

Atrybuty publiczne

- MyListElement * nextElement

  Liczba przechowywana.
- MyListElement * previousElement

  wskaznik do poprzedniej 'malej struktury' w liscie

4.9.1 Opis szczegółowy

template<class MyListElementType>class MyListElement< MyListElementType >

Definicja w linii 16 pliku mylistelement.h.

4.9.2 Dokumentacja konstruktora i destruktora

4.9.2.1 template<class MyListElementType> MyListElement< MyListElementType >::MyListElement ( ) [inline]

Definicja w linii 28 pliku mylistelement.h.

```
00029         {
00030                 this -> nextElement =0;
00031                 this -> previousElement =0;
00032         }
```

4.9.2.2 template<class MyListElementType> MyListElement< MyListElementType >::MyListElement (MyListElementType *arg*) [inline]

Parametry

arg	liczba do zapisania w kolejnym elemencie listy

Definicja w linii 37 pliku mylistelement.h.

```
00038         {
00039                 this -> content = arg;
00040                 this -> nextElement =0;
00041                 this -> previousElement =0;
00042                 //std::cerr<<"\n(konstruktor MyListElement): content="<<arg;
00043         }
```

4.9.2.3 template<class MyListElementType> MyListElement< MyListElementType >::MyListElement (const MyListElement< MyListElementType > & *myListElement*) [inline]

Parametry

myListElement	Element o przekopiowania

Definicja w linii 48 pliku mylistelement.h.

```
00049         {
00050                 //this->number = myListElement.number;
00051                 //this->nazwa = myListElement.nazwa;
00052                 this->content = myListElement.content;
00053                 this->nextElement = myListElement.nextElement;
00054                 this->previousElement = myListElement.
      previousElement;
00055                 //std::cerr<<"\n(konstruktor kopiujacy MyListElement): content="<<content;
00056         }
```

4.9.3 Dokumentacja funkcji składowych

4.9.3.1 template<class MyListElementType> void MyListElement< MyListElementType >::set (MyListElementType *arg*) [inline]

Parametry

arg	Liczba do zapisania

Definicja w linii 60 pliku mylistelement.h.

```
00061         {
00062                 this -> content = arg;
00063                 //this -> nazwa = str;
00064         }
```

4.9.4 Dokumentacja atrybutów składowych

4.9.4.1 template<class MyListElementType> MyListElement* MyListElement< MyListElementType >::nextElement

wskaznik do nastepnej 'malej struktury' w liscie

Definicja w linii 21 pliku mylistelement.h.

Odwołania w MyList< Observer * >::insertAfter(), MyListElement< Observer * >::MyListElement() i MyList< Observer * >::printList().

4.9.4.2 template<class MyListElementType> MyListElement* MyListElement< MyListElementType >::previousElement

Definicja w linii 23 pliku mylistelement.h.

Odwołania w MyList< Observer * >::insertAfter() i MyListElement< Observer * >::MyListElement().

Dokumentacja dla tej klasy została wygenerowana z pliku:

- mylistelement.h

4.10 Dokumentacja klasy NumberGenerator

Klasa generujaca losowe liczby.

`#include <numbergenerator.h>`

Statyczne metody publiczne

- template<typename MyListElementType >
  static MyList< MyListElementType > generateNumbers (int range, int quantity)

  Generuje losowe liczby.
- static std::string * generateStrings (int ileStringow)

  Generuje losowe stringi.

4.10.1 Opis szczegółowy

Klasa generujaca losowe liczby na podstawie czasu maszyny na ktorym jest uruchomiona Wszystkie funkcje zapisu pliku dziedziczy z klasy DataFrame

Definicja w linii 27 pliku numbergenerator.h.

4.10.2 Dokumentacja funkcji składowych

4.10.2.1 template<typename MyListElementType > static MyList<MyListElementType> NumberGenerator::generateNumbers (int *range*, int *quantity*) [inline], [static]

Definicja w linii 33 pliku numbergenerator.h.

Odwołuje się do MyList< MyListElementType >::push_back().

```
00034 {
00035         MyList<MyListElementType> &myList = *new
      MyList<MyListElementType>();
00036         time_t randomTime = clock();
00037
00038         for(int i=0; i<quantity ; i++)
00039         {
00040                 srand (randomTime = clock());
00041                 myList.push_back(rand()%range);
00042                 randomTime = clock();
00043         }
00044         return myList;
00045 }
```

4.10.2.2 static std::string* NumberGenerator::generateStrings (int *ileStringow*) [static]

Parametry

ileStringow	Ilosc stringow do stworzenia Generuje losowe stringi na podstawie czasu maszyny

Dokumentacja dla tej klasy została wygenerowana z pliku:

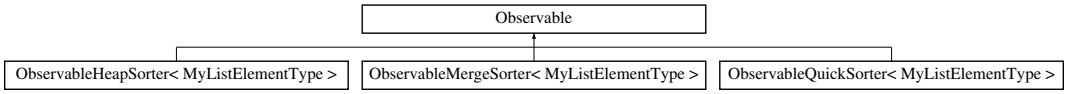
- numbergenerator.h

4.11 Dokumentacja klasy Observable

Klasa abstrakcyjna- bazowa dla objektow do obserowania.

`#include <observable.h>`

Diagram dziedziczenia dla Observable



Metody publiczne

- void add (Observer ∗obserwator)

  Dodaje sie jako obiekt do obserowania dla danego obserwatora.
- void sendStartUpdateToObservers ()

  Wysyla powiadomienie do obserwatorow o rozpoczeciu algorytmu.
- void sendStopUpdateToObservers ()

  Wysyla powiadomienie do obserwatorow o zakonczeniu algorytmu.
- virtual ∼Observable ()

Atrybuty publiczne

- MyList< Observer ∗ > observaters

  Lista obserwatorow.

4.11.1 Opis szczegółowy

Definicja w linii 16 pliku observable.h.

4.11.2 Dokumentacja konstruktora i destruktora

4.11.2.1 virtual Observable::∼Observable ( ) [inline], [virtual]

Definicja w linii 44 pliku observable.h.

```
00044 {}
```

4.11.3 Dokumentacja funkcji składowych

4.11.3.1 void Observable::add (Observer ∗ *obserwator*) [inline]

Definicja w linii 23 pliku observable.h.

Odwołuje się do observaters i MyList< MyListElementType >::push_back().

Odwołania w main().

```
00023                                   {
00024         observaters.push_back(obserwator);
00025     }
```

4.11.3.2 void Observable::sendStartUpdateToObservers ( ) [inline]

Definicja w linii 29 pliku observable.h.

Odwołuje się do observaters i MyList< MyListElementType >::size().

Odwołania w ObservableHeapSorter< MyListElementType >::sort(), ObservableQuickSorter< MyListElementType >::sort() i ObservableMergeSorter< MyListElementType >::sort().

```
00029                                       {
00030         for(int i=0; i<observaters.size(); i++)
00031         {
00032                 //std::cout<<"Wysylam start update";
00033                 observaters[i].content->receivedStartUpdate();
00034         }
00035     }
```

4.11.3.3 void Observable::sendStopUpdateToObservers ( ) [inline]

Definicja w linii 39 pliku observable.h.

Odwołuje się do observaters i MyList< MyListElementType >::size().

Odwołania w ObservableHeapSorter< MyListElementType >::sort(), ObservableQuickSorter< MyListElementType >::sort() i ObservableMergeSorter< MyListElementType >::sort().

```
00039                                       {
00040         for(int i=0; i<observaters.size(); i++)
00041                 observaters[i].content->receivedStopUpdate();
00042     }
```

4.11.4 Dokumentacja atrybutów składowych

4.11.4.1 MyList<Observer*> Observable::observaters

Definicja w linii 19 pliku observable.h.

Odwołania w add(), main(), sendStartUpdateToObservers() i sendStopUpdateToObservers().

Dokumentacja dla tej klasy została wygenerowana z pliku:

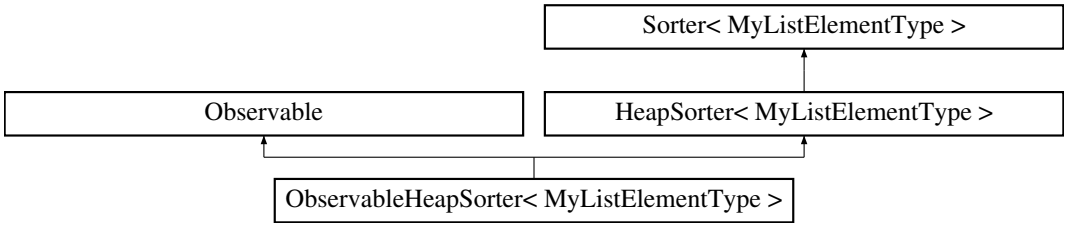
- observable.h

4.12 Dokumentacja szablonu klasy ObservableHeapSorter< MyListElementType >

Klasa sluzaca do obslugi sortowania przez kopcowanie z dodaniem obserwatora.

`#include <observableheapsorter.h>`

Diagram dziedziczenia dla ObservableHeapSorter< MyListElementType >



Metody publiczne

- ObservableHeapSorter (List< MyListElementType > &myList)
- List< MyListElementType > & sort ()

  sortuje przez kopcowanie
- virtual ~ObservableHeapSorter ()

Dodatkowe Dziedziczone Składowe

4.12.1 Opis szczegółowy

template<class MyListElementType>class ObservableHeapSorter< MyListElementType >

Definicja w linii 18 pliku observableheapsorter.h.

4.12.2 Dokumentacja konstruktora i destruktora

4.12.2.1 template<class MyListElementType> ObservableHeapSorter< MyListElementType >::ObservableHeapSorter (List< MyListElementType > & *myList*) [inline]

Definicja w linii 21 pliku observableheapsorter.h.

```
00021                                                       :
00022                 HeapSorter<MyListElementType>::HeapSorter(myList){
      }
```

4.12.2.2 template<class MyListElementType> virtual ObservableHeapSorter< MyListElementType >::~ObservableHeapSorter ( ) [inline], [virtual]

Definicja w linii 33 pliku observableheapsorter.h.

```
00033 {};
```

4.12.3 Dokumentacja funkcji składowych

4.12.3.1 template<class MyListElementType> List<MyListElementType>& ObservableHeapSorter< MyListElementType >::sort ( ) [inline], [virtual]

Reimplementowana z HeapSorter< MyListElementType >.

Definicja w linii 26 pliku observableheapsorter.h.

Odwołuje się do HeapSorter< MyListElementType >::list, Observable::sendStartUpdateToObservers(), Observable::sendStopUpdateToObservers() i HeapSorter< MyListElementType >::sort().

Odwołania w main().

```
00027         {
00028                 sendStartUpdateToObservers();
00029                 HeapSorter<MyListElementType>::sort();
00030                 sendStopUpdateToObservers();
00031                 return this->list;
00032         }
```

Dokumentacja dla tej klasy została wygenerowana z pliku:

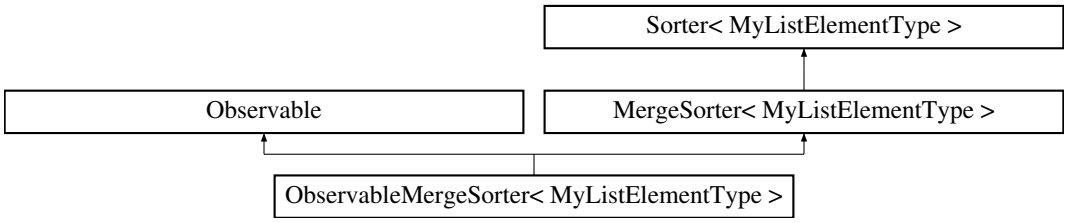
- observableheapsorter.h

4.13 Dokumentacja szablonu klasy ObservableMergeSorter< MyListElementType >

Klasa sluzaca do obslugi sortowania przez Scalanie z dodaniem obserwatora.

`#include <observablemergesorter.h>`

Diagram dziedziczenia dla ObservableMergeSorter< MyListElementType >



Metody publiczne

- ObservableMergeSorter (MyList< MyListElementType > &myList)
- List< MyListElementType > & sort ()

  sortuje przez scalanie
- virtual ~ObservableMergeSorter ()

Dodatkowe Dziedziczone Składowe

4.13.1 Opis szczegółowy

template<class MyListElementType>class ObservableMergeSorter< MyListElementType >

Definicja w linii 18 pliku observablemergesorter.h.

4.13.2 Dokumentacja konstruktora i destruktora

4.13.2.1 template<class MyListElementType> ObservableMergeSorter< MyListElementType >::ObservableMergeSorter (MyList< MyListElementType > & *myList*) [inline]

Definicja w linii 21 pliku observablemergesorter.h.

```
00021                                                                 :
00022                 MergeSorter<MyListElementType>::MergeSorter(
      myList){}
```

4.13.2.2 template<class MyListElementType> virtual ObservableMergeSorter< MyListElementType >::~ObservableMergeSorter ( ) [inline], [virtual]

Definicja w linii 33 pliku observablemergesorter.h.

```
00033 {};
```

4.13.3 Dokumentacja funkcji składowych

4.13.3.1 template<class MyListElementType> List<MyListElementType>& ObservableMergeSorter< MyListElementType >::sort ( ) [inline], [virtual]

Reimplementowana z MergeSorter< MyListElementType >.

Definicja w linii 26 pliku observablemergesorter.h.

Odwołuje się do MergeSorter< MyListElementType >::list, Observable::sendStartUpdateToObservers(), Observable::sendStopUpdateToObservers() i MergeSorter< MyListElementType >::sort().

Odwołania w main().

```
00027         {
00028                 sendStartUpdateToObservers();
00029                 MergeSorter<MyListElementType>::sort();
00030                 sendStopUpdateToObservers();
00031                 return this->list;
00032         }
```

Dokumentacja dla tej klasy została wygenerowana z pliku:

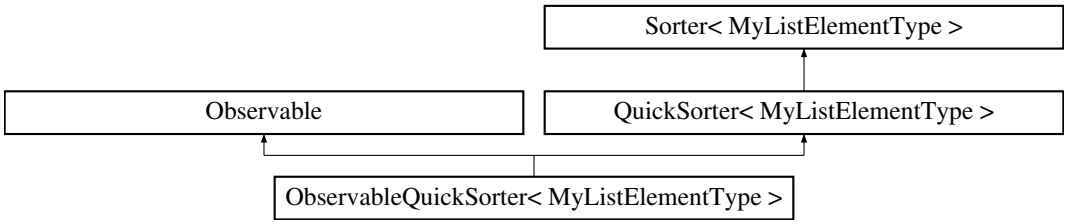
- observablemergesorter.h

4.14 Dokumentacja szablonu klasy ObservableQuickSorter< MyListElementType >

Klasa sluzaca do obslugi sortowania przez Sortowanie szybkie z dodaniem obserwatora.

`#include <observablequicksorter.h>`

Diagram dziedziczenia dla ObservableQuickSorter< MyListElementType >



Metody publiczne

- ObservableQuickSorter (List< MyListElementType > &list)
- List< MyListElementType > & sort ()

  sortuje przez scalanie
- virtual ~ObservableQuickSorter ()

Dodatkowe Dziedziczone Składowe

4.14.1 Opis szczegółowy

template<class MyListElementType>class ObservableQuickSorter< MyListElementType >

Definicja w linii 18 pliku observablequicksorter.h.

4.14.2 Dokumentacja konstruktora i destruktora

4.14.2.1 template<class MyListElementType> ObservableQuickSorter< MyListElementType >::ObservableQuickSorter (List< MyListElementType > & *list*) [inline]

Definicja w linii 21 pliku observablequicksorter.h.

```
00021                                                   :
00022                 QuickSorter<MyListElementType>::QuickSorter(list
      ){}
```

4.14.2.2 template<class MyListElementType> virtual ObservableQuickSorter< MyListElementType >::~ObservableQuickSorter ( ) [inline], [virtual]

Definicja w linii 33 pliku observablequicksorter.h.

```
00033 {};
```

4.14.3 Dokumentacja funkcji składowych

4.14.3.1 template<class MyListElementType> List<MyListElementType>& ObservableQuickSorter< MyListElementType >::sort ( ) [inline], [virtual]

Implementuje Sorter< MyListElementType >.

Definicja w linii 26 pliku observablequicksorter.h.

Odwołuje się do QuickSorter< MyListElementType >::list, Observable::sendStartUpdateToObservers(), Observable::sendStopUpdateToObservers() i QuickSorter< MyListElementType >::sort().

Odwołania w main().

```
00027         {
00028                 sendStartUpdateToObservers();
00029                 QuickSorter<MyListElementType>::sort();
00030                 sendStopUpdateToObservers();
00031                 return this->list;
00032         }
```

Dokumentacja dla tej klasy została wygenerowana z pliku:

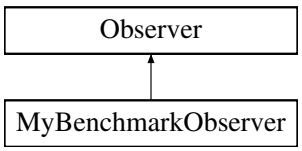
- observablequicksorter.h

4.15 Dokumentacja klasy Observer

obserwator

`#include <observer.h>`

Diagram dziedziczenia dla Observer



Metody publiczne

- virtual double getTimerValue ()=0

  pobiera czas trwania algorytmu
- virtual void receivedStartUpdate ()=0

  Odbiera powiadomienie o rozpoczeciu dzialania algorytmu.
- virtual void receivedStopUpdate ()=0

  Odbiera powiadomienie o zakonczeniu dzialania algorytmu.
- virtual ~Observer ()

4.15.1 Opis szczegółowy

Interfejs obserwatora

Definicja w linii 19 pliku observer.h.

4.15.2 Dokumentacja konstruktora i destruktora

4.15.2.1 virtual Observer::~Observer ( ) [inline], [virtual]

Definicja w linii 33 pliku observer.h.

```
00033 {};
```

4.15.3 Dokumentacja funkcji składowych

4.15.3.1 virtual double Observer::getTimerValue ( ) [pure virtual]

Zwraca

czas trwania algorytmu

Implementowany w MyBenchmarkObserver.

4.15.3.2 virtual void Observer::receivedStartUpdate ( ) [pure virtual]

Implementowany w MyBenchmarkObserver.

4.15.3.3 virtual void Observer::receivedStopUpdate ( ) [pure virtual]

Implementowany w MyBenchmarkObserver.

Dokumentacja dla tej klasy została wygenerowana z pliku:

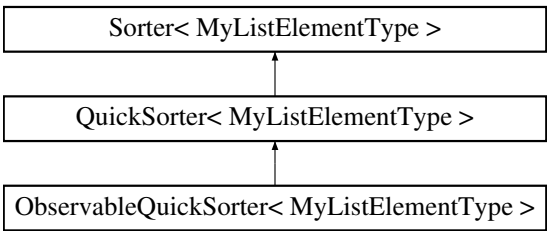
- observer.h

4.16 Dokumentacja szablonu klasy QuickSorter< MyListElementType >

Klasa sluzaca do obslugi sortowania przez Scalanie.

`#include <quicksorter.h>`

Diagram dziedziczenia dla QuickSorter< MyListElementType >



Metody publiczne

- QuickSorter (List< MyListElementType > &list)

  Konstruktor.
- virtual ~QuickSorter ()

- void quicksort (int lewy, int prawy)

  Szuka liczb do porownaia z pivotem.
- List< MyListElementType > & sort ()

  Sortuje przez Sortowanie szybkie.

Atrybuty publiczne

- int enablePivot
- List< MyListElementType > & list

  Skopiowana lista do przeprowadzania sortowania.

4.16.1 Opis szczegółowy

template<class MyListElementType>class QuickSorter< MyListElementType >

Definicja w linii 18 pliku quicksorter.h.

4.16.2 Dokumentacja konstruktora i destruktora

4.16.2.1 template<class MyListElementType > QuickSorter< MyListElementType >::QuickSorter (List< MyListElementType > & *list*) [inline]

Parametry

&list	lista, która konstruktor kopiuje aby nie naruszać podanej przez uzytkownika

Definicja w linii 28 pliku quicksorter.h.

Odwołuje się do List< MyListElementType >::cloneFrom() i QuickSorter< MyListElementType >::enablePivot.

```
00029         :list(list.createObjectFromAbstractReference())
00030         {
00031                 this->list.cloneFrom(list);
00032                 this->enablePivot=1;
00033         }
```

4.16.2.2 template<class MyListElementType > virtual QuickSorter< MyListElementType >::~QuickSorter ( ) [inline], [virtual]

Definicja w linii 35 pliku quicksorter.h.

```
00035 {};
```

4.16.3 Dokumentacja funkcji składowych

4.16.3.1 template<class MyListElementType > void QuickSorter< MyListElementType >::quicksort (int *lewy*, int *prawy*) [inline]

Parametry

lewy	brzeg poszukiwan
prawy	brzeg poszukiwan

Definicja w linii 42 pliku quicksorter.h.

Odwołuje się do QuickSorter< MyListElementType >::enablePivot i QuickSorter< MyListElementType >::list.

Odwołania w QuickSorter< MyListElementType >::sort().

```
00043         {
00044             int pivot=list[(int)(lewy+prawy)/2].content;
00045             int i,j,x;
00046             i=lewy;
00047             j=prawy;
00048             if(enablePivot) pivot=(list[(int)(lewy+prawy)/2].content +
      list[lewy].content + list[prawy].content)/3;
00049             do
00050             {
00051                 while(list[i].content<pivot) {i++; }
00052                 while(list[j].content>pivot) {j--; }
00053                 if(i<=j)
00054                 {
00055                     x=list[i].content;
00056                     list[i].content=list[j].content;
00057                     list[j].content=x;
00058                     i++;
00059                     j--;
00060                 }
00061             }
00062             while(i<=j);
00063             if(j>lewy) quicksort(lewy, j);
00064             if(i<prawy) quicksort(i, prawy);
00065         }
```

4.16.3.2 template<class MyListElementType > List<MyListElementType>& QuickSorter< MyListElementType >::sort ( ) [inline], [virtual]

Implementuje Sorter< MyListElementType >.

Definicja w linii 69 pliku quicksorter.h.

Odwołuje się do QuickSorter< MyListElementType >::list i QuickSorter< MyListElementType >::quicksort().

Odwołania w ObservableQuickSorter< MyListElementType >::sort().

```
00070         {
00071                 //std::cout<<"(QuickSort)";
00072                 quicksort(0, list.size()-1);
00073                 return list;
00074         }
```

4.16.4 Dokumentacja atrybutów składowych

4.16.4.1 template<class MyListElementType > int QuickSorter< MyListElementType >::enablePivot

Definicja w linii 21 pliku quicksorter.h.

Odwołania w QuickSorter< MyListElementType >::quicksort() i QuickSorter< MyListElementType >::QuickSorter().

4.16.4.2 template<class MyListElementType > List<MyListElementType>& QuickSorter< MyListElementType >::list

Definicja w linii 23 pliku quicksorter.h.

Odwołania w main(), QuickSorter< MyListElementType >::quicksort(), ObservableQuickSorter< MyListElementType >::sort() i QuickSorter< MyListElementType >::sort().

Dokumentacja dla tej klasy została wygenerowana z pliku:

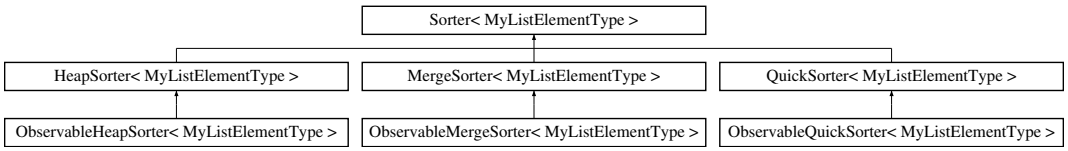
- quicksorter.h

4.17 Dokumentacja szablonu klasy Sorter< MyListElementType >

interfejs kazdego sortowania

`#include <sorter.h>`

Diagram dziedziczenia dla Sorter< MyListElementType >



Metody publiczne

- virtual List< MyListElementType > & sort ()=0
- virtual ~Sorter ()

  Sortuje przez scalanie.

4.17.1 Opis szczegółowy

template<class MyListElementType>class Sorter< MyListElementType >

Definicja w linii 15 pliku sorter.h.

4.17.2 Dokumentacja konstruktora i destruktora

4.17.2.1 template<class MyListElementType > virtual Sorter< MyListElementType >::~Sorter ( ) [inline], [virtual]

Definicja w linii 23 pliku sorter.h.

```
00023 {};
```

4.17.3 Dokumentacja funkcji składowych

4.17.3.1 template<class MyListElementType > virtual List<MyListElementType>& Sorter< MyListElementType >::sort ( ) [pure virtual]

Implementowany w MergeSorter< MyListElementType >, QuickSorter< MyListElementType >, HeapSorter< MyListElementType >, ObservableHeapSorter< MyListElementType >, ObservableMergeSorter< MyListElementType > i ObservableQuickSorter< MyListElementType >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- sorter.h

5 Dokumentacja plików

5.1 Dokumentacja pliku filestreamer.h

`#include <string>`

```
#include <fstream>
#include <iomanip>
```

Funkcje

- void writeStringToFile (std::string fileName, std::string textToSave)
- void writeStringToFile (std::string fileName, double textToSave)
- void writeStringToFile (std::string fileName, int textToSave)
- void clearFile (std::string fileName)

5.1.1 Dokumentacja funkcji

5.1.1.1 void clearFile (std::string *fileName*)

Definicja w linii 41 pliku filestreamer.h.

Odwołania w main().

```
00042 {
00043         std::ofstream streamToFile;
00044         streamToFile.open (fileName.c_str(), std::ofstream::out   | std::ofstream::trunc);
00045         streamToFile.close();
00046 }
```

5.1.1.2 void writeStringToFile (std::string *fileName,* std::string *textToSave*)

Definicja w linii 15 pliku filestreamer.h.

Odwołania w main().

```
00016 {
00017         std::ofstream streamToFile;
00018         streamToFile.open (fileName.c_str(), std::ofstream::app);
00019         streamToFile << std::fixed;
00020         streamToFile << std::setprecision(5) <<textToSave;
00021         streamToFile.close();
00022 }
```

5.1.1.3 void writeStringToFile (std::string *fileName,* double *textToSave*)

Definicja w linii 23 pliku filestreamer.h.

```
00024 {
00025         std::ofstream streamToFile;
00026         streamToFile.open (fileName.c_str(), std::ofstream::app);
00027         streamToFile << std::fixed;
00028         streamToFile<<std::setprecision(5)  << textToSave;
00029         streamToFile.close();
00030 }
```

5.1.1.4 void writeStringToFile (std::string *fileName,* int *textToSave*)

Definicja w linii 32 pliku filestreamer.h.

```
00033 {
00034         std::ofstream streamToFile;
00035         streamToFile.open (fileName.c_str(), std::ofstream::app);
00036         streamToFile << std::fixed;
00037         streamToFile <<std::setprecision(5) << textToSave;
00038         streamToFile.close();
00039 }
```

5.2 filestreamer.h

```
00001 /*
00002  * filestreamer.h
00003  *
00004  *  Created on: May 14, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef FILESTREAMER_H_
00009 #define FILESTREAMER_H_
00010
00011 #include <string>
00012 #include <fstream>
00013 #include <iomanip>
00014
00015 void writeStringToFile(std::string fileName, std::string textToSave)
00016 {
00017         std::ofstream streamToFile;
00018         streamToFile.open (fileName.c_str(), std::ofstream::app);
00019         streamToFile << std::fixed;
00020         streamToFile << std::setprecision(5) <<textToSave;
00021         streamToFile.close();
00022 }
00023 void writeStringToFile(std::string fileName, double textToSave)
00024 {
00025         std::ofstream streamToFile;
00026         streamToFile.open (fileName.c_str(), std::ofstream::app);
00027         streamToFile << std::fixed;
00028         streamToFile<<std::setprecision(5)  << textToSave;
00029         streamToFile.close();
00030 }
00031
00032 void writeStringToFile(std::string fileName, int textToSave)
00033 {
00034         std::ofstream streamToFile;
00035         streamToFile.open (fileName.c_str(), std::ofstream::app);
00036         streamToFile << std::fixed;
00037         streamToFile <<std::setprecision(5) << textToSave;
00038         streamToFile.close();
00039 }
00040
00041 void clearFile(std::string fileName)
00042 {
00043         std::ofstream streamToFile;
00044         streamToFile.open (fileName.c_str(), std::ofstream::out  | std::ofstream::trunc);
00045         streamToFile.close();
00046 }
00047
00048 #endif /* FILESTREAMER_H_ */
```

5.3 Dokumentacja pliku heapsorter.h

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

Komponenty

- class HeapSorter< MyListElementType >

  Klasa sluzaca do obslugi sortowania przez kopcowanie.

5.4 heapsorter.h

```
00001 /*
00002  * heapsorter.h
00003  *
00004  *  Created on: May 12, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef HEAPSORTER_H_
00009 #define HEAPSORTER_H_
00010
00011
```

```
00012 #include "sorter.h"
00013 #include "list.h"
00014
00016 template <class MyListElementType>
00017 class HeapSorter: public Sorter<MyListElementType>
00018 {
00019 public:
00021         List<MyListElementType> &list;
00022
00026         HeapSorter(List<MyListElementType> &myList)
00027         :list(myList.createObjectFromAbstractReference())
00028
00029         {
00030                 this->list.cloneFrom(myList);
00031                 /*this->sizeOfList = myList.sizeOfList;
00032                 this->firstElement = myList.firstElement;
00033                 this->lastElement = myList.lastElement;
00034                 this->iterator=myList.iterator;
00035                 this->isIteratorAfterPop = myList.isIteratorAfterPop;*/
00036         }
00037
00038         virtual ~HeapSorter(){};
00039
00042         List<MyListElementType> &sort()
00043         {
00044                 int n = this->list.size();
00045             int parent = n/2, index, child, tmp; /* heap indexes */
00046             /* czekam az sie posortuje */
00047             while (1) {
00048                 if (parent > 0)
00049                 {
00050                     tmp = (this->list)[--parent].content;  /* kobie kopie do tmp */
00051                 }
00052                 else {
00053                     n--;
00054                     if (n == 0)
00055                     {
00056                         return this->list; /* Zwraca posortowane */
00057                     }
00058                     tmp = this->list[n].content;
00059                     this->list[n].content = this->list[0].content;
00060                 }
00061                 index = parent;
00062                 child = index * 2 + 1;
00063                 while (child < n) {
00064                     if (child + 1 < n  &&  this->list[child + 1].content > this->
      list[child].content) {
00065                         child++;
00066                     }
00067                     if (this->list[child].content > tmp) {
00068                         this->list[index].content = this->list[child].content;
00069                         index = child;
00070                         child = index * 2 + 1;
00071                     } else {
00072                         break;
00073                     }
00074                 }
00075                 this->list[index].content = tmp;
00076             }
00077             return this->list;
00078         }
00079
00080
00081
00082 };
00083
00084
00085 #endif /* HEAPSORTER_H_ */
```

5.5 Dokumentacja pliku list.h

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

Komponenty

- class List< MyListElementType >

5.6 list.h

```
00001 /*
00002  * list.h
00003  *
00004  *  Created on: May 13, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef LIST_H_
00009 #define LIST_H_
00010
00011 #include "listelement.h"
00012 #include "list.h"
00013
00016 template <class MyListElementType>
00017 class List
00018 {
00019 public:
00023         int virtual &size() = 0;
00024
00025         // Zaczerpnite z wzorca projektowego Budowniczy
00026
00030         ListElement<MyListElementType> virtual
      pop_back()      =       0;
00031
00035         ListElement<MyListElementType> virtual
      pop_front()             =       0;
00036
00039         void virtual printList()       =       0;
00040         void virtual push_back(MyListElementType arg)  =       0;
00041
00044         void virtual push_front(MyListElementType arg)        =       0;
00045
00048         MyListElement<MyListElementType> virtual &
      operator[](int numberOfElement)               =       0;
00049
00054         void virtual insertAfter(MyListElement<MyListElementType>
       arg, int iteratorID)    =       0;
00055
00058         MyListElementType virtual &show_front()       =       0;
00059
00062         MyListElementType virtual &show_back() =       0;
00063
00064
00065         //List<MyListElementType> virtual &operator=(const List<MyListElementType> &pattern)    =       0;
00066
00069         void virtual cloneFrom(List<MyListElementType> &patternList)
00070         {
00071                 // release memory from main list
00072                 while(this->size()) pop_back();
00073                 for(int i=0; i<patternList.size(); i++)
00074                         this->push_back(patternList[i].content);
00075         }
00076
00079         List<MyListElementType> virtual &
      createObjectFromAbstractReference() = 0;
00080
00083         void virtual free(){ while(size()) pop_back(); }
00084         virtual ~List(){};
00085 };
00086
00087
00088
00089 #endif /* LIST_H_ */
```

5.7 Dokumentacja pliku listelement.h

Komponenty

- class ListElement< MyListElementType >

  klasa ma byc uzywana jako abstrakcyjna do implementacji pojedynczego elemenu listy

5.8 listelement.h

```
00001 /*
00002  * listelement.h
00003  *
```

```
00004  *  Created on: May 13, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef LISTELEMENT_H_
00009 #define LISTELEMENT_H_
00010
00013 template <class MyListElementType>
00014 class ListElement
00015 {
00016 public:
00017
00019         MyListElementType content;
00020
00021         virtual ~ListElement(){};
00022
00023 };
00024
00025
00026
00027
00028 #endif /* LISTELEMENT_H_ */
```

5.9 Dokumentacja pliku listsaver.h

```
#include <string>
#include <fstream>
```

Komponenty

- class ListSaver< MyListElementType >

5.10 listsaver.h

```
00001 /*
00002  * ListIO.h
00003  *
00004  *  Created on: May 14, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef LISTSAVER_H_
00009 #define LISTSAVER_H_
00010
00011 #include <string>
00012 #include <fstream>
00013
00014 template <class MyListElementType>
00015 class ListSaver
00016 {
00019         List<MyListElementType> &list;
00020
00024         ListSaver(MyList<MyListElementType> &listArgument):
00025                 list(listArgument)
00026         {}
00027
00032         void saveToFile(std::string nazwaPliku)
00033         {
00034                 std::ofstream streamToFile;
00035                 streamToFile.open (nazwaPliku.c_str(), std::ofstream::out);
00036                 for(int i=0; i<list.size() ; i++)
00037                         streamToFile << '{'<<list[i].content<<"} ";
00038                 streamToFile.close();
00039         }
00040
00041 };
00042
00043
00044
00045
00046
00047 #endif /* LISTSAVER_H_ */
```

5.11 Dokumentacja pliku main.cpp

```
#include <iostream>
#include <unistd.h>
#include "numbergenerator.h"
#include "mylist.h"
#include "mybenchmark.h"
#include "observable.h"
#include "observer.h"
#include "observableheapsorter.h"
#include "observablequicksorter.h"
#include "observablemergesorter.h"
#include "filestreamer.h"
```

Definicje

- #define ILOSC_LICZB_DO_SORTOWANIA 1000

Funkcje

- int main (int argc, char ∗argv[])

5.11.1 Dokumentacja definicji

5.11.1.1 #define ILOSC_LICZB_DO_SORTOWANIA 1000

Definicja w linii 20 pliku main.cpp.

Odwołania w main().

5.11.2 Dokumentacja funkcji

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

Zmienna uzywana przez GETOPT

Definicja w linii 22 pliku main.cpp.

Odwołuje się do Observable::add(), clearFile(), List< MyListElementType >::free(), ILOSC_LICZB_DO_SORTOWANIA, MergeSorter< MyListElementType >::list, HeapSorter< MyListElementType >::list, QuickSorter< MyListElementType >::list, Observable::observaters, ObservableQuickSorter< MyListElementType >::sort(), ObservableMergeSorter< MyListElementType >::sort(), ObservableHeapSorter< MyListElementType >::sort() i writeStringToFile().

```
00023 {
00024         MyList<int> lista;
00025         //int isSetN = 0;
00026         int opt;
00027         while ((opt = getopt(argc, argv, "n:o:i:gx")) != -1) {
00028                 switch(opt){
00029                 case 'n':       // ilosc liczb do przetworzenia
00030                         lista = NumberGenerator::generateNumbers<int>(10000000, atoi(optarg));
00031                         //isSetN = 1;
00032                         break;
00033
00034                 case 'o':
00035                         //podstawoweInfoIO.outputFileName = optarg;
00036                         break;
00037
00038                 case 'i':
00039                         //podstawoweInfoIO.inputFileName=optarg;
00040                         break;
```

```
00041
00042                 case '?':       default:
00043                         std::cout<<"\nPodano zly argument";
00044                         return -1;
00045                 }
00046         }
00047         //if(!isSetN) {std::cerr<<"\nNie podano argumentu: -n X\n"; return -1;}
00048
00049
00050                 std::cout<<"\n -> Prosze czekac trwa sortowanie\n";
00051
00052                 clearFile("log.txt");
00053                 writeStringToFile("log.txt", "Ilosc\t");
00054                 writeStringToFile("log.txt", "HeapS.\t");
00055                 writeStringToFile("log.txt", "QuickS.\t");
00056                 writeStringToFile("log.txt", "MergeS.\n");
00057                 for(int i=ILOSC_LICZB_DO_SORTOWANIA; i<
      ILOSC_LICZB_DO_SORTOWANIA*10; i+=
      ILOSC_LICZB_DO_SORTOWANIA)
00058                 {
00059                         lista.free();
00060                         lista = NumberGenerator::generateNumbers<int>(10000000, i);
00061                 MyBenchmarkObserver *o1 = new
      MyBenchmarkObserver();
00062                 ObservableHeapSorter<int> heapSorter(lista);
00063                 ObservableQuickSorter<int> quickSorter(lista);
00064                 ObservableMergeSorter<int> mergeSorter(lista);
00065                 heapSorter.add(o1);
00066                 quickSorter.add(o1);
00067                 mergeSorter.add(o1);
00068
00069
00070                 writeStringToFile("log.txt", i);
00071                 writeStringToFile("log.txt", "\t");
00072
00073                 heapSorter.sort();
00074                 writeStringToFile("log.txt", heapSorter.observaters[0].content->
      getTimerValue());
00075                 writeStringToFile("log.txt", "\t");
00076                 heapSorter.list.free();
00077
00078                 quickSorter.sort();
00079                 writeStringToFile("log.txt", quickSorter.observaters[0].content->
      getTimerValue());
00080                 writeStringToFile("log.txt", "\t");
00081                 quickSorter.list.free();
00082
00083                 mergeSorter.sort();
00084                 writeStringToFile("log.txt", mergeSorter.observaters[0].content->
      getTimerValue());
00085                 writeStringToFile("log.txt", "\n");
00086                 mergeSorter.list.free();
00087                 }
00088
00089         std::cout<<" -> Sortowanie zakonczone\n";
00090         std::cout<<" -> Zapisano do pliku log.txt\n";
00091         std::cout<<std::endl;
00092         return 0;
00093 }
```

5.12 main.cpp

```
00001 /*
00002  * main.cpp
00003  *
00004  *  Created on: Mar 6, 2015
00005  *      Author: serek8
00006  */
00008 #include <iostream>
00009 #include <unistd.h>
00010 #include "numbergenerator.h"
00011 #include "mylist.h"
00012 #include "mybenchmark.h"
00013 #include "observable.h"
00014 #include "observer.h"
00015 #include "observableheapsorter.h"
00016 #include "observablequicksorter.h"
00017 #include "observablemergesorter.h"
00018 #include "filestreamer.h"
00019
00020 #define ILOSC_LICZB_DO_SORTOWANIA 1000
00021
00022 int main(int argc, char *argv[])
00023 {
```

```
00024         MyList<int> lista;
00025         //int isSetN = 0;
00026         int opt;
00027         while ((opt = getopt(argc, argv, "n:o:i:gx")) != -1) {
00028                 switch(opt){
00029                 case 'n':       // ilosc liczb do przetworzenia
00030                         lista = NumberGenerator::generateNumbers<int>(10000000, atoi(optarg));
00031                         //isSetN = 1;
00032                         break;
00033
00034                 case 'o':
00035                         //podstawoweInfoIO.outputFileName = optarg;
00036                         break;
00037
00038                 case 'i':
00039                         //podstawoweInfoIO.inputFileName=optarg;
00040                         break;
00041
00042                 case '?':       default:
00043                         std::cout<<"\nPodano zly argument";
00044                         return -1;
00045                 }
00046         }
00047         //if(!isSetN) {std::cerr<<"\nNie podano argumentu: -n X\n"; return -1;}
00048
00049
00050                 std::cout<<"\n -> Prosze czekac trwa sortowanie\n";
00051
00052                 clearFile("log.txt");
00053                 writeStringToFile("log.txt", "Ilosc\t");
00054                 writeStringToFile("log.txt", "HeapS.\t");
00055                 writeStringToFile("log.txt", "QuickS.\t");
00056                 writeStringToFile("log.txt", "MergeS.\n");
00057                 for(int i=ILOSC_LICZB_DO_SORTOWANIA; i<
      ILOSC_LICZB_DO_SORTOWANIA*10; i+=
      ILOSC_LICZB_DO_SORTOWANIA)
00058                 {
00059                         lista.free();
00060                         lista = NumberGenerator::generateNumbers<int>(10000000, i);
00061                 MyBenchmarkObserver *o1 = new
      MyBenchmarkObserver();
00062                 ObservableHeapSorter<int> heapSorter(lista);
00063                 ObservableQuickSorter<int> quickSorter(lista);
00064                 ObservableMergeSorter<int> mergeSorter(lista);
00065                 heapSorter.add(o1);
00066                 quickSorter.add(o1);
00067                 mergeSorter.add(o1);
00068
00069
00070                 writeStringToFile("log.txt", i);
00071                 writeStringToFile("log.txt", "\t");
00072
00073                 heapSorter.sort();
00074                 writeStringToFile("log.txt", heapSorter.
      observaters[0].content->getTimerValue());
00075                 writeStringToFile("log.txt", "\t");
00076                 heapSorter.list.free();
00077
00078                 quickSorter.sort();
00079                 writeStringToFile("log.txt", quickSorter.
      observaters[0].content->getTimerValue());
00080                 writeStringToFile("log.txt", "\t");
00081                 quickSorter.list.free();
00082
00083                 mergeSorter.sort();
00084                 writeStringToFile("log.txt", mergeSorter.
      observaters[0].content->getTimerValue());
00085                 writeStringToFile("log.txt", "\n");
00086                 mergeSorter.list.free();
00087                 }
00088
00089         std::cout<<" -> Sortowanie zakonczone\n";
00090         std::cout<<" -> Zapisano do pliku log.txt\n";
00091         std::cout<<std::endl;
00092         return 0;
00093 }
```

5.13 Dokumentacja pliku mergesorter.h

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

Komponenty

- class MergeSorter< MyListElementType >

  Klasa sluzaca do obslugi sortowania przez Scalanie.

5.14 mergesorter.h

```
00001 /*
00002  * mergesort.h
00003  *
00004  *  Created on: May 11, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef MERGESORT_H_
00009 #define MERGESORT_H_
00010
00011 #include "sorter.h"
00012 #include "list.h"
00013
00015 template <class MyListElementType>
00016 class MergeSorter: public Sorter<MyListElementType> {
00017 public:
00018
00020         MyList<MyListElementType> &list;
00021
00025         MergeSorter(MyList<MyListElementType> &listArg)
00026         :list(listArg)      {}
00027
00028         virtual ~MergeSorter(){}
00029
00036         MyList<MyListElementType> merge(
      MyList<MyListElementType> left,
      MyList<MyListElementType> right)
00037         {
00038                 MyList<MyListElementType> result;
00039                 //Gdy jest jeszcze cos do sortowania
00040                 while (left.size() > 0 || right.size() > 0)
00041                 {
00042                         // Jak oba to zamieniamy
00043                         if (left.size() > 0 && right.size() > 0)
00044                         {
00045                                 // Sprawdzam czy zamieniac
00046                                 if (left.show_front() <= right.
      show_front())
00047                                         {
00048                                                 result.push_back(left.
      show_front()); left.pop_front();
00049                                         }
00050                                 else
00051                                 {
00052                                         result.push_back(right.
      show_front()); right.pop_front();
00053                                 }
00054                         }
00055                         // pojedyncze listy (nieparzyse)
00056                         else if (left.size() > 0)
00057                         {
00058                                 for (int i = 0; i < left.size(); i++) result.
      push_back(left[i].content); break;
00059                         }
00060                         // pojedyncze listy (nieparzyse- taka sama sytuacja jak wyzej)
00061                         else if ((int)right.size() > 0)
00062                         {
00063                                 for (int i = 0; i < (int)right.size(); i++) result.
      push_back(right[i].content); break;
00064                         }
00065                 }
00066                 return result;
00067         }
00073         MyList<MyListElementType> mergeSort(
      MyList<MyListElementType> m)
00074         {
00075                 if (m.size() <= 1) return m; // gdy juz nic nie ma do sotrowania
00076                 MyList<MyListElementType> left, right, result;
00077                 int middle = (m.size()+ 1) / 2; // anty-nieparzyscie
00078                 for (int i = 0; i < middle; i++)
00079                         {
00080                                 left.push_back(m[i].content);
00081                         }
00082                 for (int i = middle; i < m.size(); i++)
00083                         {
00084                                 right.push_back(m[i].content);
```

```
00085                         }
00086                 left = mergeSort(left);
00087                 right = mergeSort(right);
00088                 result = merge(left, right);
00089                 return result;
00090         }
00091
00092
00095         List<MyListElementType> &sort()
00096         {
00097                 this->list=mergeSort(this->list);
00098                 return this->list;
00099         }
00100
00101 };
00102
00103 #endif /* MERGESORT_H_ */
```

5.15 Dokumentacja pliku mybenchmark.cpp

```
#include "mybenchmark.h"
```

5.16 mybenchmark.cpp

```
00001 /*
00002  * mybenchmark.cpp
00003  *
00004  *  Created on: Mar 6, 2015
00005  *      Author: serek8
00006  */
00009 #include "mybenchmark.h"
00010
00011
00012 void MyBenchmark :: timerStart()
00013 {
00014         timerValue = (( (double)clock() ) /CLOCKS_PER_SEC);
00015 }
00016
00017 double MyBenchmark :: timerStop()
00018 {
00019         return (( (double)clock() ) /CLOCKS_PER_SEC) - timerValue;
00020 }
```

5.17 Dokumentacja pliku mybenchmark.h

```
#include <ctime>
#include "observer.h"
#include <iostream>
```

Komponenty

- class MyBenchmark

  Klasa bazowa/interface do testowania algorytmu.
- class MyBenchmarkObserver

  Mybenchmark obserwator Używana jako obserwator klasa sprawdzajaca odpowiednie objekty.

5.18 mybenchmark.h

```
00001 /*
00002  * mybenchmark.h
00003  *
00004  *  Created on: Mar 6, 2015
00005  *      Author: serek8
00006  */
00008 #ifndef MYBENCHMARK_H_
```

```
00009 #define MYBENCHMARK_H_
00010
00011 #include <ctime>
00012 #include "observer.h"
00013 #include <iostream>
00020 class MyBenchmark
00021 {
00022 public:
00023
00025         double timerValue;
00026
00027         MyBenchmark()
00028         {
00029                 timerValue = 0;
00030         }
00031
00033         void timerStart();
00034
00039         double timerStop();
00040
00044         virtual ~MyBenchmark() {};
00045         //using DataFrame::operator=;
00046 };
00047
00052 class MyBenchmarkObserver : public MyBenchmark, public
      Observer
00053 {
00054 public:
00055         MyBenchmarkObserver(){};
00056
00060         double getTimerValue() {return this->timerValue;}
00061
00064         void receivedStartUpdate () {
00065                 timerStart();
00066         }
00067
00070         void receivedStopUpdate () {
00071                 // std::cout<<"\nCzas wykonywania operacji: "<<timerStop();
00072         }
00073         virtual ~MyBenchmarkObserver(){};
00074
00075 };
00076
00077
00078
00079 #endif /* MYBENCHMARK_H_ */
```

5.19 Dokumentacja pliku mylist.h

```
#include <iostream>
#include <string>
#include "mylistelement.h"
#include "observer.h"
#include "list.h"
#include "listelement.h"
```

Komponenty

- class MyList< MyListElementType >

  Lista dwukierunkowa.

5.20 mylist.h

```
00001 /*
00002  * mylist.h
00003  *
00004  *  Created on: Mar 12, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef MYLIST_H_
00009 #define MYLIST_H_
00010
```

```
00011 #include <iostream>
00012 #include <string>
00013 #include "mylistelement.h"
00014 #include "observer.h"
00015 #include "list.h"
00016 #include "listelement.h"
00022 template <class MyListElementType>
00023 class MyList : public List<MyListElementType>{
00024
00025 public:
00027         int sizeOfList;
00028
00029
00031         MyListElement<MyListElementType> *
      firstElement;
00033         MyListElement<MyListElementType> *
      lastElement;
00034         MyListElement<MyListElementType> *
      iterator;
00035         int iteratorElementId; // nie ruszac !
00036         int isIteratorAfterPop;
00038
00039         MyList()
00040         {
00041                 firstElement = lastElement = new
      MyListElement<MyListElementType>;
00042                 sizeOfList = 0;
00043                 iteratorElementId =0;
00044                 iterator=NULL;
00045                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00046         }
00047
00048         MyList(List<MyListElementType> &list)
00049         {
00050                 firstElement = lastElement = new
      MyListElement<MyListElementType>;
00051                 sizeOfList = 0;
00052                 iteratorElementId =0;
00053                 iterator=NULL;
00054                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00055                 for(int i=0; i<list.size(); i++)
00056                 {
00057                         this->push_back(list[i]);
00058                 }
00059         }
00060         virtual ~MyList(){};
00061
00066         int &size()
00067         {
00068                 return sizeOfList;
00069         }
00074         /*MyListElement<MyListElementType> &pop_back()
00075         {
00076                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new MyListElement<MyListElementType>)); }
00077                 MyListElement<MyListElementType> tmpNumber = *(this -> lastElement);
00078                 MyListElement<MyListElementType> *originMyListElement = this -> lastElement;
00079                 this -> lastElement = this -> lastElement -> previousElement;
00080                 delete originMyListElement;
00081                 isIteratorAfterPop=1;
00082                 return tmpNumber;
00083         }*/
00084         ListElement<MyListElementType> pop_back()
00085         {
00086                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new
      MyListElement<MyListElementType>)); }
00087                 MyListElement<MyListElementType> tmpNumber = *(this ->
      lastElement);
00088                 MyListElement<MyListElementType> *originMyListElement =
      this -> lastElement;
00089                 this -> lastElement = this -> lastElement -> previousElement;
00090                 delete originMyListElement;
00091                 isIteratorAfterPop=1;
00092                 return tmpNumber;
00093         }
00098         ListElement<MyListElementType> pop_front()
00099         {
00100                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new
      MyListElement<MyListElementType>())); }
00101                 MyListElement<MyListElementType> tmpNumber = *(this ->
      firstElement);
00102                 MyListElement<MyListElementType> *originMyListElement =
      this -> firstElement;
00103                 this -> firstElement = this -> firstElement -> nextElement;
00104
00105                 delete originMyListElement;
```

```
00106                 isIteratorAfterPop=1;
00107                 return tmpNumber;
00108         }
00112         void push_back(MyListElementType arg)
00113         {
00114                 //std::cerr<<"\n(push_back): arg.content="<<arg.content;
00115                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00116                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00117                 //newMyListElement -> nextElement = 0;
00118                 newMyListElement -> previousElement = this -> lastElement;
00119                 this -> lastElement -> nextElement = newMyListElement;
00120                 this->lastElement = newMyListElement;
00121         }
00125         void push_front(MyListElementType arg)
00126         {
00127                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00128                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00129                 //newMyListElement -> previousElement =  0;
00130                 newMyListElement -> nextElement = this -> firstElement;
00131                 this -> firstElement -> previousElement = newMyListElement;
00132                 this->firstElement = newMyListElement;
00133                 ++iteratorElementId;
00134         }
00139         MyListElementType &show_front()
00140         {
00141                 return firstElement->content;
00142         }
00147         MyListElementType &show_back()
00148         {
00149                 return lastElement->content;
00150         }
00151
00152
00156         void  printList()
00157         {
00158                 MyListElement<MyListElementType> *elem = (this->
      firstElement);
00159                 std::cout<<"\nWyswietlam liste (size:"<<this->sizeOfList<<"): ";
00160                 for(int i=0; i< this->sizeOfList; i++)
00161                 {
00162                         std::cout<<" "<<elem->content;
00163                         elem = elem->nextElement;
00164                 }
00165         }
00166
00171         MyListElement<MyListElementType> &
      operator[](int numberOfElement)
00172         {
00173                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00174                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00175                         {
00176                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
       !";
00177                                 return *iterator;
00178                         }
00179                 if(isIteratorAfterPop)
00180                         {
00181                                 iteratorElementId=0;  // czyli iterator byl zpopowany
00182                                 iterator = firstElement;
00183                                 isIteratorAfterPop=0;
00184                         }
00185                 //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00186                 if((numberOfElement <= iteratorElementId-numberOfElement) &&(
      iteratorElementId-numberOfElement>=0))
00187                 {
00188                         //std::cerr<<"\nJestem w if_1";
00189                         iterator = (this->firstElement);
00190                         iteratorElementId = 0;
00191                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00192                                 iterator = (iterator->nextElement);
00193                 }
00194                 else if(numberOfElement > iteratorElementId)
00195                 {
00196                         //std::cerr<<"\nJestem w if_2";
00197                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00198                                                 iterator = (iterator->nextElement);
00199                 }
00200                 else if( numberOfElement < iteratorElementId)
00201                 {
00202                         //std::cerr<<"\nJestem w if_3";
00203                         for (; iteratorElementId> numberOfElement ;
```

```
    iteratorElementId--)
00204                                                   iterator = (iterator->previousElement);
00205                 }
00206                 return *iterator;
00207         }
00208
00212         void insertAfter(MyListElement<MyListElementType> arg,
    int iteratorID)
00213         {
00214                 if(iteratorID==0 && this->sizeOfList==0)  {push_front(arg.
    content); return;}
00215                 if(iteratorID==this->sizeOfList-1)  {push_back(arg.
    content); return;}
00216                 MyListElement<MyListElementType> *newMyListElement = new
    MyListElement<MyListElementType>(arg);
00217                 MyListElement<MyListElementType> &tmpThis=(*this)[
    iteratorID], &tmpNext=(*this)[iteratorID+1];
00218                 if(!sizeOfList++) {firstElement =
    lastElement = newMyListElement;}
00219                 newMyListElement -> nextElement = tmpThis.nextElement;
00220                 newMyListElement -> previousElement = &tmpThis;
00221                 tmpThis.nextElement = newMyListElement;
00222                 tmpNext.previousElement = newMyListElement;
00223                 isIteratorAfterPop=1;
00224         }
00225
00226
00227         //MyListElement operator[](int numberOfElement);
00228         //virtual MyList<MyListElementType> sort()
00229         //{
00230         //      std::cerr<<"\nError: Sortowanie z klasy MyList !!!";
00231         //      //return m;
00232         //}
00233
00234         MyList<MyListElementType> &operator=(const
    MyList<MyListElementType> &pattern)
00235         {
00236                 //std::cerr<<" @@@";
00237                 this->sizeOfList = pattern.sizeOfList;
00238                 this->firstElement = pattern.firstElement;
00239                 this->lastElement = pattern.lastElement;
00240                 this->iterator=pattern.iterator;
00241                 this->isIteratorAfterPop = pattern.
    isIteratorAfterPop;
00242                 return *this;
00243         }
00244 //      List<MyListElementType> &operator=(const List<MyListElementType> &pattern)
00245 //      {
00246 //              std::cerr<<" ###";
00252 //              //this->cloneFrom(pattern);
00253 //              return *this;
00254 //      }
00255
00256 /*      void cloneFrom(MyList<MyListElementType> patternList)
00257         {
00258                 MyList<MyListElementType> &clonedList = *new MyList<MyListElementType>;
00259                 // release memory from main list
00260                 while(this->size()) pop_back();
00261                 for(int i=0; i<patternList.size(); i++)
00262                         clonedList.push_back(patternList[i]);
00263                 *this = clonedList;
00264         }
00265 */
00266
00267         List<MyListElementType> &
    createObjectFromAbstractReference(/*MyList<MyListElementType>
     abstractPattern*/)
00268         {
00269                 return *new MyList<MyListElementType>;
00270         }
00271
00272
00273
00274 };
00276
00278
00279
00280
00281 /*class MyListObserved : public MyList, public Observed
00282 {
00283 public:
00284         void mergeSort(MyList m)
00285         {
00286         MyList::mergeSort(m);
00287         powiadom();
00288
00289         }
```

```
00290         MyListObserved(){};
00291         ~MyListObserved(){};
00292
00293
00294 };*/
00295
00296 #endif /* MYLIST_H_ */
```

5.21 Dokumentacja pliku mylistelement.h

```
#include "mylist.h"
#include "listelement.h"
```

Komponenty

- class MyListElement< MyListElementType >

  Klasa 'malych struktur' gdzie jest numer i wskaznik do nas elementu.

5.22 mylistelement.h

```
00001 /*
00002  * mylistelement.h
00003  *
00004  *  Created on: May 11, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef MYLISTELEMENT_H_
00009 #define MYLISTELEMENT_H_
00010
00011 #include "mylist.h"
00012 #include "listelement.h"
00013
00015 template <class MyListElementType>
00016 class MyListElement : public ListElement<MyListElementType>{
00018 public:
00019         //MyListElementType content;
00021         MyListElement *nextElement;
00023         MyListElement *previousElement;
00024 public:
00028         MyListElement()
00029         {
00030                 this -> nextElement =0;
00031                 this -> previousElement =0;
00032         }
00037         MyListElement(MyListElementType arg)
00038         {
00039                 this -> content = arg;
00040                 this -> nextElement =0;
00041                 this -> previousElement =0;
00042                 //std::cerr<<"\n(konstruktor MyListElement): content="<<arg;
00043         }
00048         MyListElement(const MyListElement &myListElement)
00049         {
00050                 //this->number = myListElement.number;
00051                 //this->nazwa = myListElement.nazwa;
00052                 this->content = myListElement.content;
00053                 this->nextElement = myListElement.nextElement;
00054                 this->previousElement = myListElement.
      previousElement;
00055                 //std::cerr<<"\n(konstruktor kopiujacy MyListElement): content="<<content;
00056         }
00060         void set(MyListElementType arg)
00061         {
00062                 this -> content = arg;
00063                 //this -> nazwa = str;
00064         }
00065         //friend class MyList;
00066 };
00067 #endif /* MYLISTELEMENT_H_ */
```

5.23 Dokumentacja pliku numbergenerator.h

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include "mylist.h"
#include <string>
```

Komponenty

- class NumberGenerator

  Klasa generujaca losowe liczby.

Definicje

- #define MAX_HEX_ASCII_KOD 127
- #define ROZMIAR_STRINGU 20

5.23.1 Dokumentacja definicji

5.23.1.1 #define MAX_HEX_ASCII_KOD 127

Definicja w linii 17 pliku numbergenerator.h.

5.23.1.2 #define ROZMIAR_STRINGU 20

Definicja w linii 18 pliku numbergenerator.h.

5.24 numbergenerator.h

```
00001 /*
00002  * numbergenerator.h
00003  *
00004  *  Created on: Mar 11, 2015
00005  *      Author: serek8
00006  */
00008 #ifndef NUMBERGENERATOR_H_
00009 #define NUMBERGENERATOR_H_
00010
00011 #include <stdlib.h>     /* srand, rand */
00012 #include <time.h>       /* time */
00013 #include <iostream>
00014 #include "mylist.h"
00015 #include <string>
00016
00017 #define MAX_HEX_ASCII_KOD 127
00018 #define ROZMIAR_STRINGU 20
00019
00027 class NumberGenerator
00028 {
00029 public:
00032 template <typename MyListElementType>
00033 MyList<MyListElementType> static generateNumbers(int range, int
      quantity)
00034 {
00035         MyList<MyListElementType> &myList = *new
      MyList<MyListElementType>();
00036         time_t randomTime = clock();
00037
00038         for(int i=0; i<quantity ; i++)
00039         {
00040                 srand (randomTime = clock());
00041                 myList.push_back(rand()%range);
00042                 randomTime = clock();
00043         }
00044         return myList;
00045 }
```

```
00046
00053 static std::string *generateStrings(int ileStringow);
00054
00055
00056
00057 //using DataFrame::operator=;
00058
00059 };
00060
00061 #endif /* NUMBERGENERATOR_H_ */
```

5.25 Dokumentacja pliku observable.h

```
#include <iostream>
#include "mylist.h"
```

Komponenty

- class Observable

  Klasa abstrakcyjna- bazowa dla objektow do obserowania.

5.26 observable.h

```
00001 /*
00002  * observable.h
00003  *
00004  *  Created on: May 14, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef OBSERVABLE_H_
00009 #define OBSERVABLE_H_
00010
00011 #include <iostream>
00012 #include "mylist.h"
00013
00016 class Observable {
00017 public:
00019         MyList<Observer*> observaters;
00020
00023     void add(Observer *obserwator) {
00024         observaters.push_back(obserwator);
00025     }
00026
00029     void sendStartUpdateToObservers () {
00030         for(int i=0; i<observaters.size(); i++)
00031         {
00032                 //std::cout<<"Wysylam start update";
00033                 observaters[i].content->receivedStartUpdate();
00034         }
00035     }
00036
00039     void sendStopUpdateToObservers () {
00040         for(int i=0; i<observaters.size(); i++)
00041                 observaters[i].content->receivedStopUpdate();
00042     }
00043
00044         virtual ~Observable(){}
00045
00046
00047
00048 };
00049
00050 #endif /* OBSERVABLE_H_ */
```

5.27 Dokumentacja pliku observableheapsorter.h

```
#include "observable.h"
#include "heapsorter.h"
```

Komponenty

- class ObservableHeapSorter< MyListElementType >

  Klasa sluzaca do obslugi sortowania przez kopcowanie z dodaniem obserwatora.

5.28 observableheapsorter.h

```
00001 /*
00002  * observableheapsorter.h
00003  *
00004  *  Created on: May 14, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef OBSERVABLEHEAPSORTER_H_
00009 #define OBSERVABLEHEAPSORTER_H_
00010
00011
00012 #include "observable.h"
00013 #include "heapsorter.h"
00014
00017 template <class MyListElementType>
00018 class ObservableHeapSorter : public Observable, public
      HeapSorter<MyListElementType>
00019 {
00020 public:
00021         ObservableHeapSorter(List<MyListElementType> &myList):
00022                 HeapSorter<MyListElementType>::HeapSorter(myList){}
00023
00026         List<MyListElementType> &sort()
00027         {
00028                 sendStartUpdateToObservers();
00029                 HeapSorter<MyListElementType>::sort();
00030                 sendStopUpdateToObservers();
00031                 return this->list;
00032         }
00033         virtual ~ObservableHeapSorter(){};
00034
00035
00036 };
00037
00038
00039 #endif /* OBSERVABLEHEAPSORTER_H_ */
```

5.29 Dokumentacja pliku observablemergesorter.h

```
#include "observable.h"
#include "mergesorter.h"
```

Komponenty

- class ObservableMergeSorter< MyListElementType >

  Klasa sluzaca do obslugi sortowania przez Scalanie z dodaniem obserwatora.

5.30 observablemergesorter.h

```
00001 /*
00002  * observablemergesorter.h
00003  *
00004  *  Created on: May 14, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef OBSERVABLEMERGESORTER_H_
00009 #define OBSERVABLEMERGESORTER_H_
00010
00011
00012 #include "observable.h"
00013 #include "mergesorter.h"
00014
```

```
00017 template <class MyListElementType>
00018 class ObservableMergeSorter : public Observable, public
      MergeSorter<MyListElementType>
00019 {
00020 public:
00021         ObservableMergeSorter(MyList<MyListElementType> &
      myList):
00022                 MergeSorter<MyListElementType>::MergeSorter(myList){}
00023
00026         List<MyListElementType> &sort()
00027         {
00028                 sendStartUpdateToObservers();
00029                 MergeSorter<MyListElementType>::sort();
00030                 sendStopUpdateToObservers();
00031                 return this->list;
00032         }
00033         virtual ~ObservableMergeSorter(){};
00034
00035
00036 };
00037
00038
00039 #endif /* OBSERVABLEMERGESORTER_H_ */
```

5.31 Dokumentacja pliku observablequicksorter.h

```
#include "observable.h"
#include "quicksorter.h"
```

Komponenty

- class ObservableQuickSorter< MyListElementType >

  Klasa sluzaca do obslugi sortowania przez Sortowanie szybkie z dodaniem obserwatora.

5.32 observablequicksorter.h

```
00001 /*
00002  * observablequicksort.h
00003  *
00004  *  Created on: May 14, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef OBSERVABLEQUICKSORTER_H_
00009 #define OBSERVABLEQUICKSORTER_H_
00010
00011
00012 #include "observable.h"
00013 #include "quicksorter.h"
00014
00017 template <class MyListElementType>
00018 class ObservableQuickSorter : public Observable, public
      QuickSorter<MyListElementType>
00019 {
00020 public:
00021         ObservableQuickSorter(List<MyListElementType> &
      list):
00022                 QuickSorter<MyListElementType>::QuickSorter(list){}
00023
00026         List<MyListElementType> &sort()
00027         {
00028                 sendStartUpdateToObservers();
00029                 QuickSorter<MyListElementType>::sort();
00030                 sendStopUpdateToObservers();
00031                 return this->list;
00032         }
00033         virtual ~ObservableQuickSorter(){};
00034
00035
00036 };
00037
00038
00039 #endif /* OBSERVABLEQUICKSORTER_H_ */
```

5.33 Dokumentacja pliku observer.h

Komponenty

- class Observer

  obserwator

5.34 observer.h

```
00001 /*
00002  * observer.h
00003  *
00004  *  Created on: Apr 30, 2015
00005  *      Author: serek8
00006  */
00007
00008
00009
00010 #ifndef OBSERVER_H_
00011 #define OBSERVER_H_
00012
00013
00014
00019 class Observer {
00020 public:
00024         virtual double getTimerValue() = 0;
00025
00028     virtual void receivedStartUpdate() = 0;
00029
00032     virtual void receivedStopUpdate() = 0;
00033         virtual ~Observer(){};
00034 };
00035
00036
00037
00038
00039
00040
00041
00042
00043 #endif /* OBSERVER_H_ */
```

5.35 Dokumentacja pliku quicksorter.h

```
#include "sorter.h"
#include "list.h"
#include <iostream>
```

Komponenty

- class QuickSorter< MyListElementType >

  Klasa sluzaca do obslugi sortowania przez Scalanie.

5.36 quicksorter.h

```
00001 /*
00002  * quicksort.h
00003  *
00004  *  Created on: May 12, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef QUICKSORT_H_
00009 #define QUICKSORT_H_
00010
00011 #include "sorter.h"
00012 #include "list.h"
00013 #include <iostream>
00014
```

```
00015
00017 template <class MyListElementType>
00018 class QuickSorter : public Sorter<MyListElementType>
00019 {
00020 public:
00021         int enablePivot;
00023         List<MyListElementType> &list;
00024
00028         QuickSorter(List<MyListElementType> &
      list)
00029         :list(list.createObjectFromAbstractReference())
00030         {
00031                 this->list.cloneFrom(list);
00032                 this->enablePivot=1;
00033         }
00034
00035         virtual ~QuickSorter(){};
00036
00042         void quicksort(int lewy, int prawy)
00043         {
00044             int pivot=list[(int)(lewy+prawy)/2].content;
00045             int i,j,x;
00046             i=lewy;
00047             j=prawy;
00048             if(enablePivot) pivot=(list[(int)(lewy+prawy)/2].content +
      list[lewy].content + list[prawy].content)/3;
00049             do
00050             {
00051                 while(list[i].content<pivot) {i++; }
00052                 while(list[j].content>pivot) {j--; }
00053                 if(i<=j)
00054                 {
00055                     x=list[i].content;
00056                     list[i].content=list[j].content;
00057                     list[j].content=x;
00058                     i++;
00059                     j--;
00060                 }
00061             }
00062             while(i<=j);
00063             if(j>lewy) quicksort(lewy, j);
00064             if(i<prawy) quicksort(i, prawy);
00065         }
00066
00069         List<MyListElementType> &sort()
00070         {
00071                 //std::cout<<"(QuickSort)";
00072                 quicksort(0, list.size()-1);
00073                 return list;
00074         }
00075 };
00076
00077
00078
00079 #endif /* QUICKSORT_H_ */
```

5.37 Dokumentacja pliku sorter.h

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

Komponenty

- class Sorter< MyListElementType >

  interfejs kazdego sortowania

5.38 sorter.h

```
00001 /*
00002  * Sorter.h
00003  *
00004  *  Created on: May 13, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef SORTER_H_
```

```
00009 #define SORTER_H_
00010
00011 #include "list.h"
00012
00014 template <class MyListElementType>
00015 class Sorter
00016 {
00017 public:
00018
00019         virtual List<MyListElementType> &sort() = 0;
00020
00023         virtual ~Sorter(){};
00024 };
00025
00026
00027 #endif /* SORTER_H_ */
```