

Laboratorium 8

3.0

Wygenerowano przez Doxygen 1.8.6

Cz, 21 maj 2015 04:36:06

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 destruktor	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 destruktor	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 Laboratorium 8

Aplikacja umożliwia użytkownikowi na przeprowadzenia algorytmu mnożenia przez dwa na dowolnej liczbie elementów.

Najważniejsze cechy

Możliwość włączenia opcji benchmarkującej służącej do sprawdzenia ile czasu wykonywał się dany algorytm lub seria tego samego algorytmu

Argumenty wywołania

```
-n liczba      Ilość liczb do odczytania/przerobienia przez algorytm
-t liczba      Włącza opcję benchmarkującą dla serii powtorzeń
-o tekst       Wprowadza nazwę pliku do zapisu
-i tekst       Wprowadza nazwę pliku do odczytu
-g             Generuje n liczb i zapisuje je do pliku (po wygenerowaniu kończy program)
```

2 Indeks hierarchiczny

2.1 Hierarchia klas

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

AVLTree< ContentType >	??
ObservableAVLTree< ContentType >	??
AVLTreeNode< ContentType >	??
List< MyListElementType >	??
MyList< MyListElementType >	8
MyQueue	12
MyStack	13
List< Observer * >	??
MyList< Observer * >	8
ListElement< MyListElementType >	??
MyListElement< MyListElementType >	??
ListElement< Observer * >	??
MyListElement< Observer * >	??
ListSaver< MyListElementType >	??
MyBenchmark	6
MyBenchmarkObserver	??

NumberGenerator	14
Observable	??
ObservableAVLTree< ContentType >	??
ObservableHeapSorter< MyListElementType >	??
ObservableMergeSorter< MyListElementType >	??
ObservableQuickSorter< MyListElementType >	??
Observer	??
MyBenchmarkObserver	??
Sorter< MyListElementType >	??
HeapSorter< MyListElementType >	??
ObservableHeapSorter< MyListElementType >	??
MergeSorter< MyListElementType >	??
ObservableMergeSorter< MyListElementType >	??
QuickSorter< MyListElementType >	??
ObservableQuickSorter< MyListElementType >	??

3 Indeks klas

3.1 Lista klas

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

AVLTree< ContentType >	??
AVLTreeNode< ContentType >	??
HeapSorter< MyListElementType >	??
List< MyListElementType >	??
ListElement< MyListElementType >	??
ListSaver< MyListElementType >	??
MergeSorter< MyListElementType >	??
MyBenchmark	
Klasa bazowa/interface do testowania algorytmu	6
MyBenchmarkObserver	??
MyList< MyListElementType >	
Lista dwukierunkowa	8
MyListElement< MyListElementType >	
Klasa 'małych struktur' gdzie jest numer i wskaźnik do nas elementu	??

MyQueue	
Klasa reprezentuje kolejkę	12
MyStack	
Klasa reprezentuje stos	13
NumberGenerator	
Klasa generująca losowe liczby	14
Observable	??
ObservableAVLTree< ContentType >	??
ObservableHeapSorter< MyListElementType >	??
ObservableMergeSorter< MyListElementType >	??
ObservableQuickSorter< MyListElementType >	??
Observer	??
QuickSorter< MyListElementType >	??
Sorter< MyListElementType >	??

4 Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

avltree.h	??
avltreeelement.h	??
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	??
myqueue.h	26
mystack.h	27

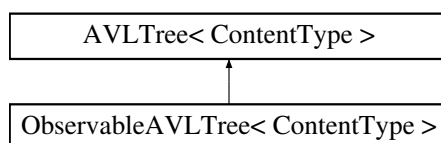
numbergenerator.h	27
observable.h	??
observableavltree.h	??
observableheapsorter.h	??
observablemergesorter.h	??
observablequicksorter.h	??
observer.h	??
quicksorter.h	??
sorter.h	??

5 Dokumentacja klas

5.1 Dokumentacja szablonu klasy AVLTree< ContentType >

```
#include <avltree.h>
```

Diagram dziedziczenia dla AVLTree< ContentType >



Metody publiczne

- [AVLTree](#) ()
- [~AVLTree](#) ()
- void [insert](#) (int &newKey)
- [AVLTreeNode](#)< ContentType > * [rotationRR](#) ([AVLTreeNode](#)< ContentType > *A)
- [AVLTreeNode](#)< ContentType > * [rotationLL](#) ([AVLTreeNode](#)< ContentType > *A)
- [AVLTreeNode](#)< ContentType > * [rotationRL](#) ([AVLTreeNode](#)< ContentType > *A)
- [AVLTreeNode](#)< ContentType > * [rotationLR](#) ([AVLTreeNode](#)< ContentType > *A)
- [AVLTreeNode](#)< ContentType > * [find](#) (int key)
- [AVLTreeNode](#)< ContentType > * [findMaxKeyNode](#) ([AVLTreeNode](#)< ContentType > *tmpNode)
- [AVLTreeNode](#)< ContentType > * [findAtherNodeMatch](#) ([AVLTreeNode](#)< ContentType > *nodeComperator)
- [AVLTreeNode](#)< ContentType > * [remove](#) ([AVLTreeNode](#)< ContentType > *x)
Usuwa element x ze struktury AVL. Zwraca usunięty węzeł
- void [print](#) ()
- void [recurringPrint](#) ([AVLTreeNode](#)< ContentType > *x)
- void [walk](#) ([AVLTreeNode](#)< ContentType > *x)

Atrybuty publiczne

- [AVLTreeNode](#)< ContentType > * [rootNode](#)

5.1.1 Opis szczegółowy

```
template<class ContentType>class AVLTree< ContentType >
```

Definicja w linii 21 pliku [avltree.h](#).

5.1.2 Dokumentacja konstruktora i destruktora

```
5.1.2.1 template<class ContentType > AVLTree< ContentType >::AVLTree ( ) [inline]
```

Definicja w linii 29 pliku [avltree.h](#).

```
00030     {
00031         rootNode = NULL;
00032     }
```

```
5.1.2.2 template<class ContentType > AVLTree< ContentType >::~~AVLTree ( ) [inline]
```

Definicja w linii 33 pliku [avltree.h](#).

```
00034     {
00035         while(rootNode)
00036         {
00037             delete(remove(rootNode));
00038         }
00039     }
```

5.1.3 Dokumentacja funkcji składowych

```
5.1.3.1 template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::find ( int key )
[inline]
```

Definicja w linii 229 pliku [avltree.h](#).

Odwołuje się do [AVLTreeNode< ContentType >::key](#), [AVLTreeNode< ContentType >::leftNode](#) i [AVLTreeNode< ContentType >::rightNode](#).

```
00230     {
00231         AVLTreeNode<ContentType> * tmpNode = rootNode;
00232
00233         while((tmpNode) && (tmpNode->key != key))
00234         {
00235             if(key < tmpNode->key) tmpNode = tmpNode->leftNode;
00236             else tmpNode = tmpNode->rightNode;
00237         }
00238
00239         return tmpNode;
00240     }
```

```
5.1.3.2 template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::findAtherNodeMatch (
AVLTreeNode< ContentType > * nodeComperator ) [inline]
```

Definicja w linii 253 pliku [avltree.h](#).

Odwołuje się do [AVLTreeNode< ContentType >::leftNode](#), [AVLTreeNode< ContentType >::parentNode](#) i [AVLTreeNode< ContentType >::rightNode](#).

```
00254     {
00255         if(nodeComperator->leftNode) return findMaxKeyNode(nodeComperator->
leftNode);
00256
00257         AVLTreeNode<ContentType> * y;
00258
00259         do
00260         {
00261             y = nodeComperator;
```



```

00262         nodeComperator = nodeComperator->parentNode;
00263     } while (nodeComperator && (nodeComperator->rightNode != y));
00264
00265     return nodeComperator;
00266 }

```

5.1.3.3 `template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::findMaxKeyNode (AVLTreeNode< ContentType > * tmpNode) [inline]`

Definicja w linii 244 pliku `avltree.h`.

Odwołuje się do `AVLTreeNode< ContentType >::rightNode`.

```

00245     {
00246         while (tmpNode->rightNode) tmpNode = tmpNode->rightNode;
00247         return tmpNode;
00248     }

```

5.1.3.4 `template<class ContentType > void AVLTree< ContentType >::insert (int & newKey) [inline]`

Definicja w linii 40 pliku `avltree.h`.

Odwołuje się do `AVLTreeNode< ContentType >::balanceFactor`, `AVLTreeNode< ContentType >::key`, `AVLTreeNode< ContentType >::leftNode`, `AVLTreeNode< ContentType >::parentNode` i `AVLTreeNode< ContentType >::rightNode`.

Odwołania w `ObservableAVLTree< ContentType >::insert()`.

```

00041     {
00042         AVLTreeNode<ContentType>* newNode = new
AVLTreeNode<ContentType>(newKey);
00043         //AVLTreeNode<ContentType> *newNode2 = new AVLTreeNode<ContentType>();
00044         AVLTreeNode<ContentType> * searchingNode =
rootNode, // Wskaznik do przeszukania drzewa i znalezienia tego samego klucza
00045             * parentForNewNode = NULL, // parentForNewNode
00046             * grandpaNode;
00047
00048         while (searchingNode)
00049         {
00050             if (searchingNode->key == newNode->key) // sprawdzam czy taki klucz juz istnieje
00051             {
00052                 //delete n; // skoro istnieje to po co taki TODO: do zmiany
00053
00054                 //cout <<"Taki klucz juz istnieje !\n";
00055                 return ;
00056             }
00057             parentForNewNode = searchingNode;
00058             if (newNode->key < searchingNode->key) searchingNode = searchingNode->
leftNode; // przechodze w lewo czesc drzewa
00059             else searchingNode = searchingNode->rightNode; //
przechodze w prawa czesc drzewa
00060         }
00061
00062         // jezeli to jest pierwszy element to wpisuje go to root'a drzewa
00063         if (!(newNode->parentNode = parentForNewNode))
00064         {
00065             rootNode = newNode;
00066             return ;
00067         }
00068         // wybieram strone gałęzi na której ma byc element
00069         if (newNode->key < parentForNewNode->key) parentForNewNode->leftNode = newNode;
00070         else parentForNewNode->rightNode = newNode;
00071
00072         //sprawdzam czy potrzebne są rotacje, jak nie to koniec ;- )
00073         if (parentForNewNode->balanceFactor)
00074         {
00075             parentForNewNode->balanceFactor = 0;
00076             return ;
00077         }
00078
00079
00080
00081         //parentForNewNode->balanceFactor = (parentForNewNode->leftNode == newNode) ? 1 : -1;
00082         if (parentForNewNode->leftNode == newNode) parentForNewNode->balanceFactor = 1;
00083         else parentForNewNode->balanceFactor = -1;
00084         grandpaNode = parentForNewNode->parentNode;
00085
00086         // usatwiam balanceFactors na 1 przed dodaniem

```

```

00087         // nowej gałęci oraz wyznaczam grandpaForNewNode od ktorego zaczynam rotacje
00088         while(grandpaNode)
00089         {
00090             if(grandpaNode->balanceFactor) break; // gdy byly juz wzczesniej ustawione to przerwij
00091
00092             if(grandpaNode->leftNode == parentForNewNode) grandpaNode->balanceFactor = 1;
00093             else grandpaNode->balanceFactor = -1;
00094             parentForNewNode = grandpaNode; grandpaNode = grandpaNode->parentNode;
00095         }
00096
00097         // jesli do konca byly zbalansowane to przerwij
00098         if(!grandpaNode) return ;
00099
00100         //rotacje na podstawie balanceFactors
00101         if(grandpaNode->balanceFactor == 1)
00102         {
00103             if(grandpaNode->rightNode == parentForNewNode)
00104             {
00105                 grandpaNode->balanceFactor = 0;
00106                 return ;
00107             }
00108             if(parentForNewNode->balanceFactor == -1) rotationLR(grandpaNode); //Rotacja podwójna
00109             w lewo-prawo
00110             else rotationLL(grandpaNode); // Rotacja pojedyncza w prawo
00111             return ;
00112         }
00113         else
00114         {
00115             if(grandpaNode->leftNode == parentForNewNode)
00116             {
00117                 grandpaNode->balanceFactor = 0;
00118                 return ;
00119             }
00120             if(parentForNewNode->balanceFactor == 1) rotationRL(grandpaNode); //Rotacja podwójna
00121             w lewo-prawo
00122             else rotationRR(grandpaNode); //Rotacja pojedyncza w lewo
00123             return ;
00124         }
00125     }

```

5.1.3.5 template<class ContentType > void AVLTree< ContentType >::print () [inline]

Definicja w linii 360 pliku [avltree.h](#).

```

00361     {
00362         this->recurringPrint(this->rootNode);
00363     }

```

5.1.3.6 template<class ContentType > void AVLTree< ContentType >::recurringPrint (AVLTreeNode< ContentType > * x) [inline]

Definicja w linii 364 pliku [avltree.h](#).

Odwołuje się do [AVLTreeNode< ContentType >::key](#), [AVLTreeNode< ContentType >::leftNode](#) i [AVLTreeNode< ContentType >::rightNode](#).

```

00365     {
00366         cout<< "klucz:"<< x->key <<"\n";
00367         if(x->leftNode) recurringPrint(x->leftNode);
00368         if(x->rightNode) recurringPrint(x->rightNode);
00369     }

```

5.1.3.7 template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::remove (AVLTreeNode< ContentType > * x) [inline]

Definicja w linii 271 pliku [avltree.h](#).

Odwołuje się do [AVLTreeNode< ContentType >::balanceFactor](#), [AVLTreeNode< ContentType >::leftNode](#), [AVLTreeNode< ContentType >::parentNode](#) i [AVLTreeNode< ContentType >::rightNode](#).

```

00272     {
00273         AVLTreeNode<ContentType> * t, * y, * z;
00274         bool nest;
00275     }

```

```

00276 // Jeśli węzeł x posiada dwójkę dzieci, lewego i prawego potomka:
00277 if((x->leftNode) && (x->rightNode))
00278 {
00279     y = remove(findAtherNodeMatch(x));
00280     //rekurencyjnie usuwamy y za pomocą tego samego algorytmu
00281     nest = false;
00282 }
00283 //Jeśli węzeł x posiada tylko jedno dziecko lub nie posiada wcale dzieci:
00284 else {
00285     if(x->leftNode) {
00286         y = x->leftNode;
00287         x->leftNode = NULL;
00288     }
00289     else {
00290         y = x->rightNode; x->rightNode = NULL;
00291     }
00292     x->balanceFactor = 0;
00293     nest = true;
00294 }
00295
00296 if(y) {
00297     y->parentNode = x->parentNode;
00298     if(x->leftNode)
00299     {
00300         y->leftNode = x->leftNode;
00301         y->leftNode->parentNode = y;
00302     }
00303     if(x->rightNode)
00304     {
00305         y->rightNode = x->rightNode;
00306         y->rightNode->parentNode = y;
00307     }
00308     y->balanceFactor = x->balanceFactor;
00309 }
00310
00311 if(x->parentNode) {
00312     if(x->parentNode->leftNode == x) x->parentNode->
leftNode = y; else x->parentNode->rightNode = y;
00313 }
00314 else rootNode = y;
00315
00316 if(nest) {
00317     z = y;
00318     y = x->parentNode;
00319     while(y)
00320     {
00321         // węzeł y był w stanie równowagi przed usunięciem węzła x w jednym z jego poddrzew.
00322         if(!(y->balanceFactor)) {
00323             y->balanceFactor = (y->leftNode == z) ? -1 : 1;
00324             break;
00325         }
00326         else {
00327             //skrócone zostało cięższe poddrzewo
00328             if ((y->balanceFactor == 1) &&
00329                 (y->leftNode == z)) || ((y->balanceFactor == -1) &&
00330                 (y->rightNode == z)) {
00331                 y->balanceFactor = 0;
00332                 z = y; y = y->parentNode;
00333             }
00334             else {
00335                 t = (y->leftNode == z) ? y->rightNode : y->
leftNode;
00336
00337                 //Wykonujemy odpowiednią rotację pojedynczą
00338                 if(!(t->balanceFactor)) {
00339                     if(y->balanceFactor == 1) rotationLL(y); else
rotationRR(y);
00340                     break;
00341                 }
00342                 //Wykonujemy odpowiednią rotację pojedynczą
00343                 else if(y->balanceFactor == t->balanceFactor)
00344                 {
00345                     if(y->balanceFactor == 1) rotationLL(y); else
rotationRR(y);
00346                     z = t; y = t->parentNode;
00347                 }
00348                 //Wykonujemy rotację podwójną
00349                 else
00350                 {
00351                     if(y->balanceFactor == 1) rotationLR(y); else
rotationRL(y);
00352                     z = y->parentNode; y = z->parentNode;
00353                 }
00354             }
00355         }
00356     }
00357 }

```

```
00358     return x;
00359 }
```

5.1.3.8 template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::rotationLL (AVLTreeNode< ContentType > * A) [inline]

Definicja w linii 151 pliku [avltree.h](#).

Odwołuje się do [AVLTreeNode< ContentType >::balanceFactor](#), [AVLTreeNode< ContentType >::leftNode](#), [AVLTreeNode< ContentType >::parentNode](#) i [AVLTreeNode< ContentType >::rightNode](#).

```
00152 {
00153     AVLTreeNode<ContentType> * B = A->leftNode, * P = A->
parentNode;
00154
00155     A->leftNode = B->rightNode;
00156     if(A->leftNode) A->leftNode->parentNode = A;
00157     B->rightNode = A;
00158     B->parentNode = P;
00159     A->parentNode = B;
00160     if(P)
00161     {
00162         if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00163     }
00164     else rootNode = B;
00165
00166     if(B->balanceFactor == 1)
00167     {
00168         A->balanceFactor = B->balanceFactor = 0;
00169     }
00170     else
00171     {
00172         A->balanceFactor = 1; B->balanceFactor = -1;
00173     }
00174
00175     return B;
00176 }
```

5.1.3.9 template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::rotationLR (AVLTreeNode< ContentType > * A) [inline]

Definicja w linii 201 pliku [avltree.h](#).

Odwołuje się do [AVLTreeNode< ContentType >::balanceFactor](#), [AVLTreeNode< ContentType >::leftNode](#), [AVLTreeNode< ContentType >::parentNode](#) i [AVLTreeNode< ContentType >::rightNode](#).

```
00202 {
00203     AVLTreeNode<ContentType> * B = A->leftNode, * C = B->
rightNode, * P = A->parentNode;
00204
00205     B->rightNode = C->leftNode;
00206     if(B->rightNode) B->rightNode->parentNode = B;
00207     A->leftNode = C->rightNode;
00208     if(A->leftNode) A->leftNode->parentNode = A;
00209     C->rightNode = A;
00210     C->leftNode = B;
00211     A->parentNode = B->parentNode = C;
00212     C->parentNode = P;
00213     if(P)
00214     {
00215         if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00216     }
00217     else rootNode = C;
00218
00219     A->balanceFactor = (C->balanceFactor == 1) ? -1 : 0;
00220     B->balanceFactor = (C->balanceFactor == -1) ? 1 : 0;
00221     C->balanceFactor = 0;
00222
00223     return C;
00224 }
```

5.1.3.10 template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::rotationRL (AVLTreeNode< ContentType > * A) [inline]

Definicja w linii 177 pliku [avltree.h](#).

Odwołuje się do `AVLTreeNode< ContentType >::balanceFactor`, `AVLTreeNode< ContentType >::leftNode`, `AVLTreeNode< ContentType >::parentNode` i `AVLTreeNode< ContentType >::rightNode`.

```

00178     {
00179         AVLTreeNode<ContentType> * B = A->rightNode, * C = B->
leftNode, * P = A->parentNode;
00180
00181         B->leftNode = C->rightNode;
00182         if(B->leftNode) B->leftNode->parentNode = B;
00183         A->rightNode = C->leftNode;
00184         if(A->rightNode) A->rightNode->parentNode = A;
00185         C->leftNode = A;
00186         C->rightNode = B;
00187         A->parentNode = B->parentNode = C;
00188         C->parentNode = P;
00189         if(P)
00190         {
00191             if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00192         }
00193         else rootNode = C;
00194
00195         A->balanceFactor = (C->balanceFactor == -1) ? 1 : 0;
00196         B->balanceFactor = (C->balanceFactor == 1) ? -1 : 0;
00197         C->balanceFactor = 0;
00198
00199         return C;
00200     }

```

5.1.3.11 `template<class ContentType> AVLTreeNode<ContentType>* AVLTree< ContentType >::rotationRR (AVLTreeNode< ContentType> * A) [inline]`

Definicja w linii 125 pliku `avltree.h`.

Odwołuje się do `AVLTreeNode< ContentType >::balanceFactor`, `AVLTreeNode< ContentType >::leftNode`, `AVLTreeNode< ContentType >::parentNode` i `AVLTreeNode< ContentType >::rightNode`.

```

00126     {
00127         AVLTreeNode<ContentType> * B = A->rightNode, * P = A->
parentNode;
00128
00129         A->rightNode = B->leftNode;
00130         if(A->rightNode) A->rightNode->parentNode = A;
00131         B->leftNode = A;
00132         B->parentNode = P;
00133         A->parentNode = B;
00134         if(P)
00135         {
00136             if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00137         }
00138         else rootNode = B;
00139
00140         if(B->balanceFactor == -1)
00141         {
00142             A->balanceFactor = B->balanceFactor = 0;
00143         }
00144         else
00145         {
00146             A->balanceFactor = -1; B->balanceFactor = 1;
00147         }
00148         return B;
00149     }

```

5.1.3.12 `template<class ContentType> void AVLTree< ContentType>::walk (AVLTreeNode< ContentType> * x) [inline]`

Definicja w linii 370 pliku `avltree.h`.

Odwołuje się do `AVLTreeNode< ContentType >::balanceFactor`, `AVLTreeNode< ContentType >::key`, `AVLTreeNode< ContentType >::leftNode`, `AVLTreeNode< ContentType >::parentNode` i `AVLTreeNode< ContentType >::rightNode`.

```

00371     {
00372         cout << x->key << " : bf = " << setw(2) << x->balanceFactor << " : Left-> ";
00373         if(x->leftNode) cout << setw(3) << x->leftNode->key;
00374         else cout << "NIL";
00375         cout << ", Right-> ";

```

```

00376         if(x->rightNode) cout << setw(3) << x->rightNode->key;
00377     else      cout << "NIL";
00378     cout << " : p -> ";
00379     if(x->parentNode) cout << setw(3) << x->parentNode->
key;
00380     else      cout << "NIL";
00381     cout << endl;
00382     if(x->leftNode)  walk(x->leftNode);
00383     if(x->rightNode) walk(x->rightNode);
00384 }

```

5.1.4 Dokumentacja atrybutów składowych

5.1.4.1 template<class ContentType > AVLTreeNode<ContentType>* AVLTree< ContentType >::rootNode

Definicja w linii 25 pliku [avltree.h](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [avltree.h](#)

5.2 Dokumentacja szablonu klasy AVLTreeNode< ContentType >

```
#include <avltreeelement.h>
```

Metody publiczne

- [AVLTreeNode\(\)](#)
- [AVLTreeNode\(int newKey\)](#)

Atrybuty publiczne

- [AVLTreeNode * parentNode](#)
- [AVLTreeNode * leftNode](#)
- [AVLTreeNode * rightNode](#)
- int [key](#)
- int [balanceFactor](#)
- [ContentType content](#)

5.2.1 Opis szczegółowy

```
template<class ContentType>class AVLTreeNode< ContentType >
```

Definicja w linii 14 pliku [avltreeelement.h](#).

5.2.2 Dokumentacja konstruktora i destruktoru

5.2.2.1 template<class ContentType> AVLTreeNode< ContentType >::AVLTreeNode() [inline]

Definicja w linii 27 pliku [avltreeelement.h](#).

Odwołuje się do [AVLTreeNode< ContentType >::balanceFactor](#), [AVLTreeNode< ContentType >::key](#), [AVLTreeNode< ContentType >::leftNode](#), [AVLTreeNode< ContentType >::parentNode](#) i [AVLTreeNode< ContentType >::rightNode](#).

```

00028 {
00029     parentNode = leftNode = rightNode = NULL;
00030     key = balanceFactor = 0;
00031 }

```

5.2.2.2 `template<class ContentType> AVLTreeNode< ContentType >::AVLTreeNode (int newKey) [inline]`

Definicja w linii 32 pliku `avltreeelement.h`.

Odwołuje się do `AVLTreeNode< ContentType >::balanceFactor`, `AVLTreeNode< ContentType >::key`, `AVLTreeNode< ContentType >::leftNode`, `AVLTreeNode< ContentType >::parentNode` i `AVLTreeNode< ContentType >::rightNode`.

```
00033     {
00034         parentNode = leftNode = rightNode = NULL;
00035         balanceFactor = 0;
00036         key = newKey;
00037     }
```

5.2.3 Dokumentacja atrybutów składowych

5.2.3.1 `template<class ContentType> int AVLTreeNode< ContentType >::balanceFactor`

Definicja w linii 24 pliku `avltreeelement.h`.

Odwołania w `AVLTreeNode< ContentType >::AVLTreeNode()`, `AVLTree< ContentType >::insert()`, `AVLTree< ContentType >::remove()`, `AVLTree< ContentType >::rotationLL()`, `AVLTree< ContentType >::rotationLR()`, `AVLTree< ContentType >::rotationRL()`, `AVLTree< ContentType >::rotationRR()` i `AVLTree< ContentType >::walk()`.

5.2.3.2 `template<class ContentType> ContentType AVLTreeNode< ContentType >::content`

Definicja w linii 25 pliku `avltreeelement.h`.

5.2.3.3 `template<class ContentType> int AVLTreeNode< ContentType >::key`

Definicja w linii 24 pliku `avltreeelement.h`.

Odwołania w `AVLTreeNode< ContentType >::AVLTreeNode()`, `AVLTree< ContentType >::find()`, `AVLTree< ContentType >::insert()`, `AVLTree< ContentType >::recurringPrint()` i `AVLTree< ContentType >::walk()`.

5.2.3.4 `template<class ContentType> AVLTreeNode * AVLTreeNode< ContentType >::leftNode`

Definicja w linii 19 pliku `avltreeelement.h`.

Odwołania w `AVLTreeNode< ContentType >::AVLTreeNode()`, `AVLTree< ContentType >::find()`, `AVLTree< ContentType >::findAtherNodeMatch()`, `AVLTree< ContentType >::insert()`, `AVLTree< ContentType >::recurringPrint()`, `AVLTree< ContentType >::remove()`, `AVLTree< ContentType >::rotationLL()`, `AVLTree< ContentType >::rotationLR()`, `AVLTree< ContentType >::rotationRL()`, `AVLTree< ContentType >::rotationRR()` i `AVLTree< ContentType >::walk()`.

5.2.3.5 `template<class ContentType> AVLTreeNode* AVLTreeNode< ContentType >::parentNode`

Definicja w linii 19 pliku `avltreeelement.h`.

Odwołania w `AVLTreeNode< ContentType >::AVLTreeNode()`, `AVLTree< ContentType >::findAtherNodeMatch()`, `AVLTree< ContentType >::insert()`, `AVLTree< ContentType >::remove()`, `AVLTree< ContentType >::rotationLL()`, `AVLTree< ContentType >::rotationLR()`, `AVLTree< ContentType >::rotationRL()`, `AVLTree< ContentType >::rotationRR()` i `AVLTree< ContentType >::walk()`.

5.2.3.6 `template<class ContentType> AVLTreeNode * AVLTreeNode< ContentType >::rightNode`

Definicja w linii 19 pliku `avltreeelement.h`.

Odwołania w `AVLTreeNode< ContentType >::AVLTreeNode()`, `AVLTree< ContentType >::find()`, `AVLTree< ContentType >::findAtherNodeMatch()`, `AVLTree< ContentType >::findMaxKeyNode()`, `AVLTree< ContentType >::insert()`, `AVLTree< ContentType >::recurringPrint()`, `AVLTree< ContentType >::remove()`, `AVLTree< ContentType >::rotationLL()`, `AVLTree< ContentType >::rotationLR()`, `AVLTree< ContentType >::rotationRL()`, `AVLTree< ContentType >::rotationRR()` i `AVLTree< ContentType >::walk()`.

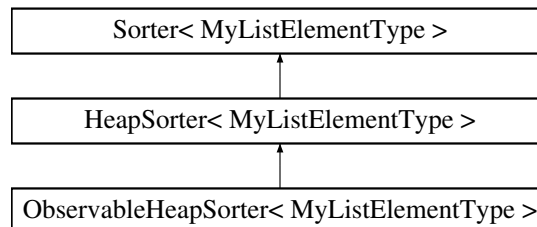
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [avltreeelement.h](#)

5.3 Dokumentacja szablonu klasy HeapSorter< MyListElementType >

```
#include <heapsorter.h>
```

Diagram dziedziczenia dla HeapSorter< MyListElementType >



Metody publiczne

- [HeapSorter](#) ([List](#)< [MyListElementType](#) > &myList)
- virtual [~HeapSorter](#) ()
- [List](#)< [MyListElementType](#) > & [sort](#) ()

Atrybuty publiczne

- [List](#)< [MyListElementType](#) > & [list](#)

5.3.1 Opis szczegółowy

```
template<class MyListElementType>class HeapSorter< MyListElementType >
```

Definicja w linii 16 pliku [heapsorter.h](#).

5.3.2 Dokumentacja konstruktora i destruktor

5.3.2.1 `template<class MyListElementType > HeapSorter< MyListElementType >::HeapSorter (List< MyListElementType > & myList) [inline]`

Definicja w linii 21 pliku [heapsorter.h](#).

Odwołuje się do [HeapSorter< MyListElementType >::list](#).

```

00022         :list(myList.createObjectFromAbstractReference())
00023
00024     {
00025         this->list.cloneFrom(myList);
00026         /*this->sizeOfList = myList.sizeOfList;
00027         this->firstElement = myList.firstElement;
00028         this->lastElement = myList.lastElement;
00029         this->iterator=myList.iterator;
00030         this->isIteratorAfterPop = myList.isIteratorAfterPop;*/
00031     }
  
```

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

Definicja w linii 33 pliku [heapsorter.h](#).

```
00033 {};
```


5.3.3 Dokumentacja funkcji składowych

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

Implementuje `Sorter< MyListElementType >`.

Reimplementowana w `ObservableHeapSorter< MyListElementType >`.

Definicja w linii 35 pliku `heapsorter.h`.

Odwołuje się do `HeapSorter< MyListElementType >::list`.

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

```

00036     {
00037         int n = this->list.size();
00038         int parent = n/2, index, child, tmp; /* heap indexes */
00039         /* czekam az sie posortuje */
00040         while (1) {
00041             if (parent > 0)
00042             {
00043                 tmp = (this->list)[--parent].content; /* kobie kopie do tmp */
00044             }
00045             else {
00046                 n--;
00047                 if (n == 0)
00048                 {
00049                     return this->list; /* Zwraca posortowane */
00050                 }
00051                 tmp = this->list[n].content;
00052                 this->list[n].content = this->list[0].content;
00053             }
00054             index = parent;
00055             child = index * 2 + 1;
00056             while (child < n) {
00057                 if (child + 1 < n && this->list[child + 1].content > this->
list[child].content) {
00058                     child++;
00059                 }
00060                 if (this->list[child].content > tmp) {
00061                     this->list[index].content = this->list[child].content;
00062                     index = child;
00063                     child = index * 2 + 1;
00064                 } else {
00065                     break;
00066                 }
00067             }
00068             this->list[index].content = tmp;
00069         }
00070         return this->list;
00071     }

```

5.3.4 Dokumentacja atrybutów składowych

5.3.4.1 `template<class MyListElementType > List<MyListElementType>& HeapSorter< MyListElementType >::list`

Definicja w linii 19 pliku `heapsorter.h`.

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

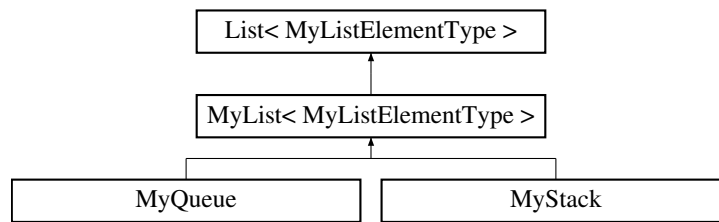
Dokumentacja dla tej klasy została wygenerowana z pliku:

- `heapsorter.h`

5.4 Dokumentacja szablonu klasy `List< MyListElementType >`

```
#include <list.h>
```

Diagram dziedziczenia dla `List< MyListElementType >`



Metody publiczne

- virtual int & [size](#) ()=0
- virtual [ListElement](#)
 < MyListElementType > [pop_back](#) ()=0
- virtual [ListElement](#)
 < MyListElementType > [pop_front](#) ()=0
- virtual void [printList](#) ()=0
- virtual void [push_back](#) (MyListElementType arg)=0
- virtual void [push_front](#) (MyListElementType arg)=0
- virtual [MyListElement](#)
 < MyListElementType > & [operator\[\]](#) (int numberOfElement)=0
- virtual void [insertAfter](#) ([MyListElement](#)< MyListElementType > arg, int iteratorID)=0
- virtual MyListElementType & [show_front](#) ()=0
- virtual MyListElementType & [show_back](#) ()=0
- virtual void [cloneFrom](#) (List< MyListElementType > &patternList)
- virtual List< MyListElementType > & [createObjectFromAbstractReference](#) ()=0
- virtual void [free](#) ()
- virtual ~List ()

5.4.1 Opis szczegółowy

template<class MyListElementType>class List< MyListElementType >

Definicja w linii 15 pliku [list.h](#).

5.4.2 Dokumentacja konstruktora i destruktoru

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

Definicja w linii 41 pliku [list.h](#).

```
00041 {};
```

5.4.3 Dokumentacja funkcji składowych

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

Definicja w linii 31 pliku [list.h](#).

Odwwołania w [QuickSorter< MyListElementType >::QuickSorter\(\)](#).

```

00032     {
00033         // release memory from main list
00034         while(this->size()) pop_back();
00035         for(int i=0; i<patternList.size(); i++)
00036             this->push_back(patternList[i].content);
00037     }
  
```

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

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

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

Definicja w linii 40 pliku `list.h`.

```
00040 { while(size()) pop_back(); }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

Odwolania w `List< Observer * >::cloneFrom()`.

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

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

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

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

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

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

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

Implementowany w [MyList< MyListElementType >](#) i [MyList< Observer * >](#).

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

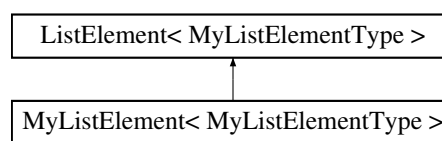
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [list.h](#)

5.5 Dokumentacja szablonu klasy ListElement< MyListElementType >

`#include <listelement.h>`

Diagram dziedziczenia dla ListElement< MyListElementType >



Atrybuty publiczne

- MyListElementType [content](#)

5.5.1 Opis szczegółowy

`template<class MyListElementType> class ListElement< MyListElementType >`

Definicja w linii 12 pliku [listelement.h](#).

5.5.2 Dokumentacja atrybutów składowych

5.5.2.1 `template<class MyListElementType> MyListElementType ListElement< MyListElementType >::content`

Definicja w linii 15 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](#)

5.6 Dokumentacja szablonu klasy ListSaver< MyListElementType >

`#include <listsaver.h>`

Metody prywatne

- [ListSaver](#) ([MyList](#)< MyListElementType > &listArgument)
- int [saveToFile](#) (std::string nazwaPliku)

Zapisuje liste do pliku.

Atrybuty prywatne

- `List< MyListElementType > & list`

5.6.1 Opis szczegółowy

`template<class MyListElementType>class ListSaver< MyListElementType >`

Definicja w linii 15 pliku `listsaver.h`.

5.6.2 Dokumentacja konstruktora i destruktor

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

Definicja w linii 19 pliku `listsaver.h`.

```
00019                                     :
00020                                     list(listArgument)
00021     {}
```

5.6.3 Dokumentacja funkcji składowych

5.6.3.1 `template<class MyListElementType > int ListSaver< MyListElementType >::saveToFile (std::string nazwaPliku) [inline],[private]`

Zwraca

Zwraca 0 gdy zapisywanie powiodło się

Definicja w linii 27 pliku `listsaver.h`.

Odwwołuje się do `ListSaver< MyListElementType >::list`.

```
00028     {
00029         std::ofstream streamToFile;
00030         streamToFile.open (nazwaPliku.c_str(), std::ofstream::out);
00031         for(int i=0; i<list.size() ; i++)
00032             streamToFile << '{'<<list[i].content<<" } ";
00033
00034         return 0;
00035     }
```

5.6.4 Dokumentacja atrybutów składowych

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

Definicja w linii 17 pliku `listsaver.h`.

Odwwołania w `ListSaver< MyListElementType >::saveToFile()`.

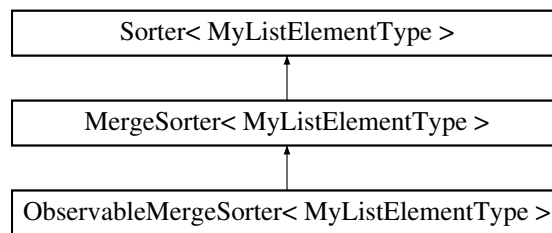
Dokumentacja dla tej klasy została wygenerowana z pliku:

- `listsaver.h`

5.7 Dokumentacja szablonu klasy MergeSorter< MyListElementType >

```
#include <mergesorter.h>
```

Diagram dziedziczenia dla MergeSorter< MyListElementType >



Metody publiczne

- [MergeSorter](#) ([MyList](#)< [MyListElementType](#) > &listArg)
- virtual [~MergeSorter](#) ()
- [MyList](#)< [MyListElementType](#) > [merge](#) ([MyList](#)< [MyListElementType](#) > left, [MyList](#)< [MyListElementType](#) > right)
- [MyList](#)< [MyListElementType](#) > [mergeSort](#) ([MyList](#)< [MyListElementType](#) > m)
Sortuje liste przez scalanie.
- [List](#)< [MyListElementType](#) > & [sort](#) ()

Atrybuty publiczne

- [MyList](#)< [MyListElementType](#) > & [list](#)

5.7.1 Opis szczegółowy

template<class [MyListElementType](#)>class MergeSorter< [MyListElementType](#) >

Definicja w linii 15 pliku [mergesorter.h](#).

5.7.2 Dokumentacja konstruktora i destruktor

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

Definicja w linii 20 pliku [mergesorter.h](#).

```
00021         :list (listArg)        {}
```

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

Definicja w linii 23 pliku [mergesorter.h](#).

```
00023 {}
```

5.7.3 Dokumentacja funkcji składowych

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

Definicja w linii 26 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()`.

```

00027     {
00028         MyList<MyListElementType> result;
00029         //Gdy jest jeszcze cos do sortowania
00030         while (left.size() > 0 || right.size() > 0)
00031         {
00032             // Jak oba to zamieniamy
00033             if (left.size() > 0 && right.size() > 0)
00034             {
00035                 // Sprawdzam czy zamieniac
00036                 if (left.show_front() <= right.
show_front())
00037                     {
00038                         result.push_back(left.
show_front()); left.pop_front();
00039                     }
00040                     else
00041                     {
00042                         result.push_back(right.
show_front()); right.pop_front();
00043                     }
00044                 // pojedyncze listy (nieparzyse)
00045                 else if (left.size() > 0)
00046                 {
00047                     for (int i = 0; i < left.size(); i++) result.
push_back(left[i].content); break;
00049                 }
00050                 // pojedyncze listy (nieparzyse- taka sama sytuacja jak wyzej)
00051                 else if ((int)right.size() > 0)
00052                 {
00053                     for (int i = 0; i < (int)right.size(); i++) result.
push_back(right[i].content); break;
00054                 }
00055             }
00056             return result;
00057         }

```

5.7.3.2 `template<class MyListElementType > MyList<MyListElementType> MergeSorter< MyListElementType >::mergeSort (MyList< MyListElementType > m) [inline]`

Parametry

<i>m</i>	Lista do posotrowania
----------	-----------------------

Zwraca

zwraca posotrowana liste

Definicja w linii 63 pliku `mergesorter.h`.

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

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

```

00064     {
00065         if (m.size() <= 1) return m; // gdy juz nic nie ma do sotrowania
00066         MyList<MyListElementType> left, right, result;
00067         int middle = (m.size()+ 1) / 2; // anty-nieparzyscie
00068         for (int i = 0; i < middle; i++)
00069         {
00070             left.push_back(m[i].content);
00071         }
00072         for (int i = middle; i < m.size(); i++)
00073         {
00074             right.push_back(m[i].content);
00075         }
00076         left = mergeSort(left);
00077         right = mergeSort(right);
00078         result = merge(left, right);
00079         return result;
00080     }

```

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

Implementuje `Sorter< MyListElementType >`.

Reimplementowana w `ObservableMergeSorter< MyListElementType >`.

Definicja w linii 83 pliku `mergesorter.h`.

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

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

```
00084         {
00085             this->list=mergeSort(this->list);
00086             return this->list;
00087         }
```

5.7.4 Dokumentacja atrybutów składowych

5.7.4.1 `template<class MyListElementType > MyList<MyListElementType>& MergeSorter< MyListElementType >::list`

Definicja w linii 18 pliku `mergesorter.h`.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

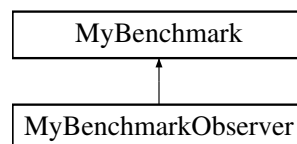
- [mergesorter.h](#)

5.8 Dokumentacja klasy MyBenchmark

Klasa bazowa/interface do testowania algorytmu.

```
#include <mybenchmark.h>
```

Diagram dziedziczenia dla MyBenchmark



Metody publiczne

- `MyBenchmark ()`
- `virtual ~MyBenchmark ()`

Usuwa obiekt test biorąc pod uwagę jego prawdziwy typ.

Statyczne metody publiczne

- `static void timerStart ()`
włączam stoper
- `static double timerStop ()`
wyłączam stoper

Atrybuty publiczne

- double `timerValue`
Interface metody algorytmu glownego.
- `std::ofstream` `streamToFile`

Statyczne atrybuty publiczne

- static double `timerValueStatic` =0

5.8.1 Opis szczegółowy

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

Definicja w linii 23 pliku `mybenchmark.h`.

5.8.2 Dokumentacja konstruktora i destruktor

5.8.2.1 `MyBenchmark::MyBenchmark () [inline]`

Definicja w linii 43 pliku `mybenchmark.h`.

Odwołuje się do `streamToFile` i `timerValue`.

```
00044     {
00045         timerValue = 0;
00046         streamToFile.open ("log.txt", std::ofstream::out | std::ofstream::trunc);
00047         streamToFile.close();
00048         streamToFile.open ("log.txt", std::ofstream::app);
00049         streamToFile << std::fixed;
00050     }
```

5.8.2.2 `virtual MyBenchmark::~~MyBenchmark () [inline],[virtual]`

Definicja w linii 66 pliku `mybenchmark.h`.

```
00066 {};
```

5.8.3 Dokumentacja funkcji składowych

5.8.3.1 `void MyBenchmark::timerStart () [static]`

Definicja w linii 12 pliku `mybenchmark.cpp`.

Odwołuje się do `timerValueStatic`.

Odwołania w `MyBenchmarkObserver::receivedStartUpdate()`.

```
00013 {
00014     MyBenchmark::timerValueStatic = (( (double)clock() ) /CLOCKS_PER_SEC);
00015 }
```

5.8.3.2 `double MyBenchmark::timerStop () [static]`

Zwraca

Długość działania stopera

Definicja w linii 17 pliku [mybenchmark.cpp](#).

Odwołuje się do [timerValueStatic](#).

Odwołania w [MyBenchmarkObserver::receivedStopUpdate\(\)](#) i [MyBenchmarkObserver::receivedStopUpdateAndSaveToFile\(\)](#).

```
00018 {
00019     return (( (double)clock() ) /CLOCKS_PER_SEC) -
    MyBenchmark::timerValueStatic;
00020 }
```

5.8.4 Dokumentacja atrybutów składowych**5.8.4.1 std::ofstream MyBenchmark::streamToFile**

Definicja w linii 41 pliku [mybenchmark.h](#).

Odwołania w [MyBenchmark\(\)](#), [MyBenchmarkObserver::receivedStopUpdateAndSaveToFile\(\)](#) i [MyBenchmarkObserver::~MyBenchmarkObserver\(\)](#).

5.8.4.2 double MyBenchmark::timerValue

Metoda abstrakcyjna, która jest interfacem do implementacji przez główny algorytm. To znaczy, że każdy algorytm ma być uruchamiany tą funkcją Czas stopera

Definicja w linii 40 pliku [mybenchmark.h](#).

Odwołania w [MyBenchmarkObserver::getTimerValue\(\)](#), [MyBenchmark\(\)](#) i [MyBenchmarkObserver::receivedStopUpdateAndSaveToFile\(\)](#).

5.8.4.3 double MyBenchmark::timerValueStatic =0 [static]

Definicja w linii 42 pliku [mybenchmark.h](#).

Odwołania w [timerStart\(\)](#) i [timerStop\(\)](#).

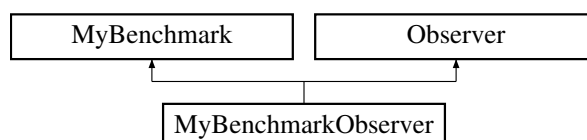
Dokumentacja dla tej klasy została wygenerowana z plików:

- [mybenchmark.h](#)
- [mybenchmark.cpp](#)

5.9 Dokumentacja klasy MyBenchmarkObserver

```
#include <mybenchmark.h>
```

Diagram dziedziczenia dla MyBenchmarkObserver

**Metody publiczne**

- [MyBenchmarkObserver \(\)](#)

- double `getTimerValue ()`
- void `receivedStartUpdate ()`
- void `receivedStopUpdate ()`
- void `receivedStopUpdateAndSaveToFile ()`
- virtual `~MyBenchmarkObserver ()`

Dodatkowe Dziedziczone Składowe

5.9.1 Opis szczegółowy

Definicja w linii 72 pliku `mybenchmark.h`.

5.9.2 Dokumentacja konstruktora i destruktoru

5.9.2.1 `MyBenchmarkObserver::MyBenchmarkObserver () [inline]`

Definicja w linii 75 pliku `mybenchmark.h`.

```
00075 {};
```

5.9.2.2 `virtual MyBenchmarkObserver::~~MyBenchmarkObserver () [inline],[virtual]`

Definicja w linii 92 pliku `mybenchmark.h`.

Odwołuje się do `MyBenchmark::streamToFile`.

```
00092 {streamToFile.close();};
```

5.9.3 Dokumentacja funkcji składowych

5.9.3.1 `double MyBenchmarkObserver::getTimerValue () [inline],[virtual]`

Implementuje `Observer`.

Definicja w linii 76 pliku `mybenchmark.h`.

Odwołuje się do `MyBenchmark::timerValue`.

```
00076 {return this->timerValue;}
```

5.9.3.2 `void MyBenchmarkObserver::receivedStartUpdate () [inline],[virtual]`

Implementuje `Observer`.

Definicja w linii 77 pliku `mybenchmark.h`.

Odwołuje się do `MyBenchmark::timerStart()`.

```
00077                                     {
00078                                     //std::cout<<"\nWlaczam stoper...";
00079                                     timerStart();
00080                                     }
```

5.9.3.3 `void MyBenchmarkObserver::receivedStopUpdate () [inline],[virtual]`

Implementuje `Observer`.

Definicja w linii 82 pliku `mybenchmark.h`.

Odwołuje się do `MyBenchmark::timerStop()`.

```

00082         {
00083             std::cout<<"\nCzas wykonywania operacji: "<<timerStop();
00084         }

```

5.9.3.4 void MyBenchmarkObserver::receivedStopUpdateAndSaveToFile () [inline],[virtual]

Implementuje `Observer`.

Definicja w linii 86 pliku `mybenchmark.h`.

Odwołuje się do `MyBenchmark::streamToFile`, `MyBenchmark::timerStop()` i `MyBenchmark::timerValue`.

```

00086         {
00087             timerStop();
00088             streamToFile<<timerValue<<std::endl;
00089         }
00090

```

Dokumentacja dla tej klasy została wygenerowana z pliku:

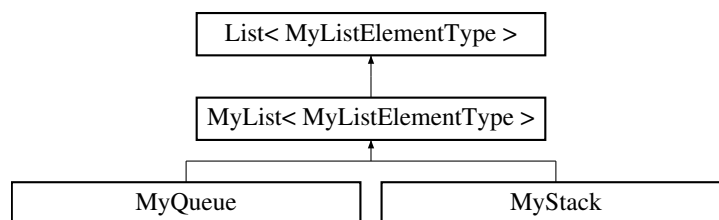
- [mybenchmark.h](#)

5.10 Dokumentacja szablonu klasy `MyList< MyListElementType >`

Lista dwukierunkowa.

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

Diagram dziedziczenia dla `MyList< MyListElementType >`



Metody publiczne

- `MyList insertAfter ()`
- `MyList ()`
Konstruktor listy.
- `MyList (List< MyListElementType > &list)`
- `virtual ~MyList ()`
- `int & size ()`
Zwraca ilość elementów listy.
- `ListElement< MyListElementType > pop_back ()`
Zwraca element ostatni w liście.
- `ListElement< MyListElementType > pop_front ()`
Zwraca element pierwszy w liście.
- `void push_back (MyListElementType arg)`
Wkłada element na ostatnie miejsce na liście.
- `void push_front (MyListElementType arg)`
Wkłada element na pierwsze miejsce na liście.
- `MyListElementType & show_front ()`
Pokazuje element po początku 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 ()`

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`

5.10.1 Opis szczegółowy

```
template<class MyListElementType>class MyList< MyListElementType >
```

Klasa przedstawia liste dwukierunkową dynamiczna

Definicja w linii 23 pliku `mylist.h`.

5.10.2 Dokumentacja konstruktora i destruktora

```
5.10.2.1 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     }
```

5.10.2.2 `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     }
```

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

Definicja w linii 60 pliku `mylist.h`.

```
00060 {};
```

5.10.3 Dokumentacja funkcji składowych

5.10.3.1 `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     }
```

5.10.3.2 `template<class MyListElementType> MyList MyList< MyListElementType >::insertAfter () [inline]`

Definicja w linii 29 pliku `mylist.h`.

```
00029 {return *new MyList<MyListElementType>();}
```

5.10.3.3 `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     }
```

5.10.3.4 `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     }
```

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

Zwraca

Zwraca 0 gdy zapisywanie powiodło się

Implementuje `List< MyListElementType >`.

Definicja w linii 171 pliku `mylist.h`.

```
00172     {
00173         //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00174         if(numberOfElement > (sizeOfList-1)) // jeżeli wyszedłem poza listę
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 był 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     }
```

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

Zwraca

Zwraca element ostatni w liście

Implementuje `List< MyListElementType >`.

Definicja w linii 84 pliku `mylist.h`.

Odwołania w `MyStack::pop()`.

```
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     }
```

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

Zwraca

Zwraca element pierwszy w liście

Implementuje `List< MyListElementType >`.

Definicja w linii 98 pliku `mylist.h`.

Odwołania w `MergeSorter< MyListElementType >::merge()` i `MyQueue::pop()`.

```
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         delete originMyListElement;
00105         isIteratorAfterPop=1;
00106         return tmpNumber;
00107     }
```

5.10.3.8 `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     }
```

5.10.3.9 `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()`, `MyList< Observer * >::MyList()`, `MyQueue::push()` i `MyStack::push()`.

```
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     }
```

5.10.3.10 `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     }
```

5.10.3.11 `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     }
```

5.10.3.12 `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     }
```

5.10.3.13 `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 `main()`, `MergeSorter< MyListElementType >::merge()`, `MergeSorter< MyListElementType >::mergeSort()`, `Observable::sendStartUpdateToObservers()`, `Observable::sendStopUpdateToObservers()` i `Observable::sendStopUpdateToObserversAndSaveToFile()`.

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

5.10.4 Dokumentacja atrybutów składowych

5.10.4.1 `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()`.

5.10.4.2 `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()`.

5.10.4.3 `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[]()`.

5.10.4.4 `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()`.

5.10.4.5 `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()`.

5.10.4.6 `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:

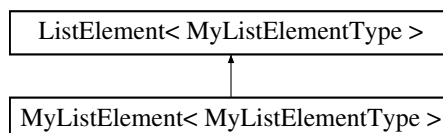
- [mylist.h](#)

5.11 Dokumentacja szablonu klasy `MyListElement< MyListElementType >`

Klasa 'małych struktur' gdzie jest numer i wskaźnik do nas elementu.

```
#include <mylistelement.h>
```

Diagram dziedziczenia dla `MyListElement< MyListElementType >`



Metody publiczne

- `MyListElement ()`
Konstruktor wewnętrznej klasy 'małych struktur'.
- `MyListElement (MyListElementType arg)`
Konstruktor wewnętrznej klasy 'małych struktur'.
- `MyListElement (const MyListElement &myListElement)`
Konstruktor kopiujący wewnętrznej klasy 'małych struktur'.
- `void set (MyListElementType arg)`
Ustawia liczbę oraz klucz słownika dla elementu.

Atrybuty publiczne

- `MyListElement * nextElement`
Liczba przechowywana.
- `MyListElement * previousElement`
wskaźnik do poprzedniej 'małej struktury' w liście

5.11.1 Opis szczegółowy

```
template<class MyListElementType>class MyListElement< MyListElementType >
```

Definicja w linii 16 pliku `mylistelement.h`.

5.11.2 Dokumentacja konstruktora i destruktora

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

Definicja w linii 28 pliku [mylistelement.h](#).

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

5.11.2.2 `template<class MyListElementType> MyListElement< MyListElementType >::MyListElement (`
`MyListElementType arg) [inline]`

Parametry

<i>arg</i>	liczba do zapisania w kolejnym elemencie listy
<i>str</i>	klucz tablicy asocjacyjnej

Definicja w linii 38 pliku [mylistelement.h](#).

```
00039     {
00040         this -> content = arg;
00041         this -> nextElement =0;
00042         this -> previousElement =0;
00043         //std::cerr<<"\n(konstruktor MyListElement): content="<<arg;
00044     }
```

5.11.2.3 `template<class MyListElementType> MyListElement< MyListElementType >::MyListElement (const`
`MyListElement< MyListElementType > & myListElement) [inline]`

Parametry

<i>myListElement</i>	Element o przekopiowania
----------------------	--------------------------

Definicja w linii 49 pliku [mylistelement.h](#).

```
00050     {
00051         //this->number = myListElement.number;
00052         //this->nazwa = myListElement.nazwa;
00053         this->content = myListElement.content;
00054         this->nextElement = myListElement.nextElement;
00055         this->previousElement = myListElement.
previousElement;
00056         //std::cerr<<"\n(konstruktor kopiujacy MyListElement): content="<<content;
00057     }
```

5.11.3 Dokumentacja funkcji składowych

5.11.3.1 `template<class MyListElementType> void MyListElement< MyListElementType >::set (MyListElementType arg)`
`[inline]`

Parametry

<i>arg</i>	Liczba do zapisania
<i>str</i>	String do zapisania

Definicja w linii 63 pliku [mylistelement.h](#).

```
00064     {
00065         this -> content = arg;
00066         //this -> nazwa = str;
00067     }
```

5.11.4 Dokumentacja atrybutów składowych

5.11.4.1 `template<class MyListElementType> MyListElement* MyListElement< MyListElementType >::nextElement`

wskaznik do następnej 'małej struktury' w liście

Definicja w linii 21 pliku [mylistelement.h](#).

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

5.11.4.2 `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:

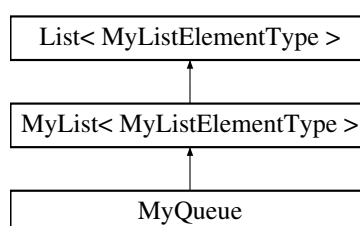
- [mylistelement.h](#)

5.12 Dokumentacja klasy MyQueue

Klasa reprezentuje kolejkę.

```
#include <myqueue.h>
```

Diagram dziedziczenia dla MyQueue



Metody publiczne

- void [push](#) (int arg)
- int [pop](#) ()
Wyciąga element z kolejki.

Dodatkowe Dziedziczone Składowe

5.12.1 Opis szczegółowy

Definicja w linii 16 pliku [myqueue.h](#).

5.12.2 Dokumentacja funkcji składowych

5.12.2.1 `int MyQueue::pop () [inline]`

Definicja w linii 27 pliku [myqueue.h](#).

Odwołuje się do [MyList< MyListElementType >::pop_front\(\)](#).

```

00027         {
00028             return pop_front();
00029     }

```

5.12.2.2 void MyQueue::push (int arg) [inline]

Definicja w linii 23 pliku [myqueue.h](#).

Odwołuje się do [MyList< MyListElementType >::push_back\(\)](#).

```

00023         {
00024             push_back (arg);
00025     }

```

Dokumentacja dla tej klasy została wygenerowana z pliku:

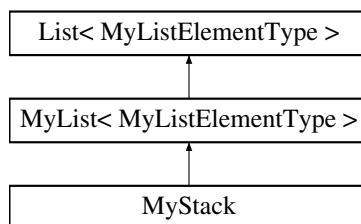
- [myqueue.h](#)

5.13 Dokumentacja klasy MyStack

Klasa reprezentuje stos.

```
#include <mystack.h>
```

Diagram dziedziczenia dla MyStack



Metody publiczne

- void [push](#) (int arg)
- int [pop](#) ()
Wyciąga element ze stosu.

Dodatkowe Dziedziczone Składowe

5.13.1 Opis szczegółowy

Stos, którego index po pushu pokazuje na miejsce następne(następne za tym elementem)

Definicja w linii 18 pliku [mystack.h](#).

5.13.2 Dokumentacja funkcji składowych

5.13.2.1 int MyStack::pop () [inline]

Definicja w linii 29 pliku [mystack.h](#).

Odwołuje się do [MyList< MyListElementType >::pop_back\(\)](#).

```

00029         {
00030             return pop_back();
00031     }

```

5.13.2.2 void MyStack::push (int arg) [inline]

Definicja w linii 25 pliku [mystack.h](#).

Odwołuje się do [MyList< MyListElementType >::push_back\(\)](#).

```
00025 {
00026     push_back (arg);
00027 }
```

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [mystack.h](#)

5.14 Dokumentacja klasy NumberGenerator

Klasa generująca losowe liczby.

```
#include <numbergenerator.h>
```

Statyczne metody publiczne

- `template<typename MyListElementType > static MyList< MyListElementType > generateNumbers (int range, int quantity)`
Generuje losowe liczby na podstawie czasu maszyny.
- `static std::string * generateStrings (int ileStringow)`
Generuje losowe stringi.

5.14.1 Opis szczegółowy

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

Definicja w linii 27 pliku [numbergenerator.h](#).

5.14.2 Dokumentacja funkcji składowych

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

Parametry

<i>zakres</i>	Zakres liczb do wygenerowania
---------------	-------------------------------

Definicja w linii 37 pliku [numbergenerator.h](#).

Odwołuje się do [MyList< MyListElementType >::push_back\(\)](#).

```
00038 {
00039     MyList<MyListElementType> &myList = *new
MyList<MyListElementType> ();
00040     time_t randomTime = clock();
00041
00042     for(int i=0; i<quantity ; i++)
00043     {
00044         srand (randomTime = clock());
00045         myList.push_back (rand()%range);
00046         randomTime = clock();
00047     }
00048     return myList;
00049 }
```

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

Parametry

<i>ileStringow</i>	Ilosc stringow do stworzenia Generuje losowe stringi na podstawie czasu maszyny
--------------------	---

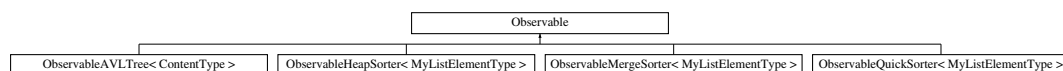
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [numbergenerator.h](#)

5.15 Dokumentacja klasy Observable

```
#include <observable.h>
```

Diagram dziedziczenia dla Observable



Metody publiczne

- void [add](#) ([Observer](#) *o)
- void [sendStartUpdateToObservers](#) ()
- void [sendStopUpdateToObservers](#) ()
- void [sendStopUpdateToObserversAndSaveToFile](#) ()
- virtual [~Observable](#) ()

Atrybuty publiczne

- [MyList< Observer * > observers](#)

5.15.1 Opis szczegółowy

Definicja w linii 14 pliku [observable.h](#).

5.15.2 Dokumentacja konstruktora i destruktor

5.15.2.1 `virtual Observable::~~Observable () [inline],[virtual]`

Definicja w linii 40 pliku [observable.h](#).

```
00040 {}
```

5.15.3 Dokumentacja funkcji składowych

5.15.3.1 `void Observable::add (Observer * o) [inline]`

Definicja w linii 19 pliku [observable.h](#).

Odwołuje się do [observers](#) i [MyList< MyListElementType >::push_back\(\)](#).

```

00019         {
00020             observers.push\_back(o);
00021         }
  
```


5.15.3.2 void Observable::sendStartUpdateToObservers () [inline]

Definicja w linii 23 pliku [observable.h](#).

Odwołuje się do [observers](#) i [MyList< MyListElementType >::size\(\)](#).

Odwołania w [ObservableAVLTree< ContentType >::insert\(\)](#), [ObservableQuickSorter< MyListElementType >::sort\(\)](#), [ObservableMergeSorter< MyListElementType >::sort\(\)](#) i [ObservableHeapSorter< MyListElementType >::sort\(\)](#).

```
00023         {
00024             for(int i=0; i<observers.size(); i++)
00025             {
00026                 //std::cout<<"Wysylam start update";
00027                 observers[i].content->receivedStartUpdate();
00028             }
00029         }
```

5.15.3.3 void Observable::sendStopUpdateToObservers () [inline]

Definicja w linii 30 pliku [observable.h](#).

Odwołuje się do [observers](#) i [MyList< MyListElementType >::size\(\)](#).

Odwołania w [ObservableHeapSorter< MyListElementType >::sort\(\)](#), [ObservableQuickSorter< MyListElementType >::sort\(\)](#) i [ObservableMergeSorter< MyListElementType >::sort\(\)](#).

```
00030         {
00031             for(int i=0; i<observers.size(); i++)
00032                 observers[i].content->receivedStopUpdate();
00033         }
```

5.15.3.4 void Observable::sendStopUpdateToObserversAndSaveToFile () [inline]

Definicja w linii 34 pliku [observable.h](#).

Odwołuje się do [observers](#) i [MyList< MyListElementType >::size\(\)](#).

Odwołania w [ObservableAVLTree< ContentType >::insert\(\)](#).

```
00034         {
00035             for(int i=0; i<observers.size(); i++)
00036                 observers[i].content->receivedStopUpdateAndSaveToFile();
00037         }
```

5.15.4 Dokumentacja atrybutów składowych

5.15.4.1 MyList<Observer*> Observable::observers

Definicja w linii 17 pliku [observable.h](#).

Odwołania w [add\(\)](#), [sendStartUpdateToObservers\(\)](#), [sendStopUpdateToObservers\(\)](#) i [sendStopUpdateToObserversAndSaveToFile\(\)](#).

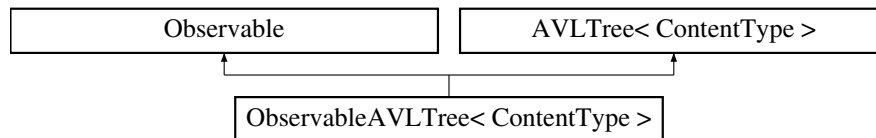
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [observable.h](#)

5.16 Dokumentacja szablonu klasy ObservableAVLTree< ContentType >

```
#include <observableavltree.h>
```

Diagram dziedziczenia dla ObservableAVLTree< ContentType >



Metody publiczne

- void [insert](#) (int newKey)
- [~ObservableAVLTree](#) ()

Dodatkowe Dziedziczone Składowe

5.16.1 Opis szczegółowy

`template<class ContentType>class ObservableAVLTree< ContentType >`

Definicja w linii 16 pliku [observableavltree.h](#).

5.16.2 Dokumentacja konstruktora i destruktor

5.16.2.1 `template<class ContentType > ObservableAVLTree< ContentType >::~~ObservableAVLTree ()`
[inline]

Definicja w linii 26 pliku [observableavltree.h](#).

```
00026 {}
```

5.16.3 Dokumentacja funkcji składowych

5.16.3.1 `template<class ContentType > void ObservableAVLTree< ContentType >::insert (int newKey)` [inline]

Definicja w linii 19 pliku [observableavltree.h](#).

Odwołuje się do [AVLTree< ContentType >::insert\(\)](#), [Observable::sendStartUpdateToObservers\(\)](#) i [Observable::sendStopUpdateToObserversAndSaveToFile\(\)](#).

```

00020     {
00021         sendStartUpdateToObservers ();
00022         AVLTree<ContentType>::insert (newKey);
00023         sendStopUpdateToObserversAndSaveToFile ();
00024     }
  
```

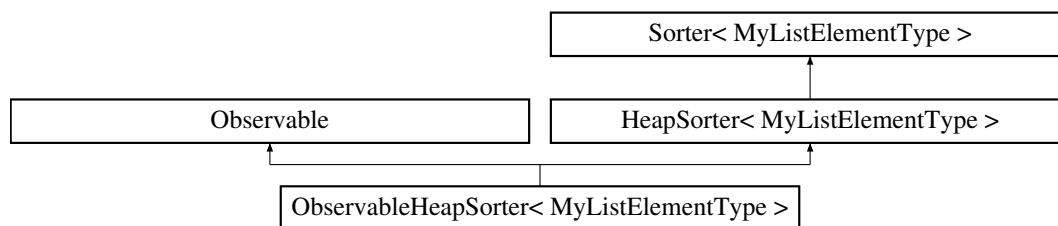
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [observableavltree.h](#)

5.17 Dokumentacja szablonu klasy ObservableHeapSorter< MyListElementType >

```
#include <observableheapsorter.h>
```

Diagram dziedziczenia dla ObservableHeapSorter< MyListElementType >



Metody publiczne

- [ObservableHeapSorter](#) ([List](#)< [MyListElementType](#) > &[myList](#))
- [List](#)< [MyListElementType](#) > & [sort](#) ()
- virtual [~ObservableHeapSorter](#) ()

Dodatkowe Dziedziczone Składowe

5.17.1 Opis szczegółowy

`template<class MyListElementType>class ObservableHeapSorter< MyListElementType >`

Definicja w linii 16 pliku [observableheapsorter.h](#).

5.17.2 Dokumentacja konstruktora i destruktor

5.17.2.1 `template<class MyListElementType > ObservableHeapSorter< MyListElementType >::ObservableHeapSorter (List< MyListElementType > & myList) [inline]`

Definicja w linii 19 pliku [observableheapsorter.h](#).

```

00019                                     :
00020         HeapSorter<MyListElementType>::HeapSorter (myList) {
    }
  
```

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

Definicja w linii 30 pliku [observableheapsorter.h](#).

```

00030 {};
```

5.17.3 Dokumentacja funkcji składowych

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

Reimplementowana z [HeapSorter< MyListElementType >](#).

Definicja w linii 23 pliku [observableheapsorter.h](#).

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

```

00024     {
00025         sendStartUpdateToObservers ();
00026         HeapSorter<MyListElementType>::sort ();
00027         sendStopUpdateToObservers ();
00028         return this->list;
00029     }
  
```

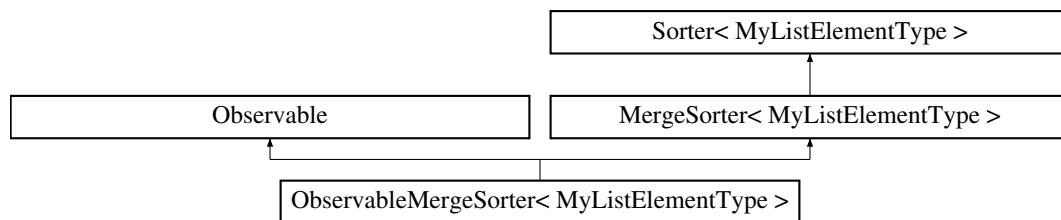
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [observableheapsorter.h](#)

5.18 Dokumentacja szablonu klasy ObservableMergeSorter< MyListElementType >

```
#include <observablemergesorter.h>
```

Diagram dziedziczenia dla ObservableMergeSorter< MyListElementType >



Metody publiczne

- [ObservableMergeSorter](#) ([MyList](#)< MyListElementType > &myList)
- [List](#)< MyListElementType > & [sort](#) ()
- virtual [~ObservableMergeSorter](#) ()

Dodatkowe Dziedziczone Składowe

5.18.1 Opis szczegółowy

```
template<class MyListElementType>class ObservableMergeSorter< MyListElementType >
```

Definicja w linii 16 pliku [observablemergesorter.h](#).

5.18.2 Dokumentacja konstruktora i destruktor

```
5.18.2.1 template<class MyListElementType > ObservableMergeSorter< MyListElementType
>::ObservableMergeSorter ( MyList< MyListElementType > & myList ) [inline]
```

Definicja w linii 19 pliku [observablemergesorter.h](#).

```
00019                                     :
00020         MergeSorter<MyListElementType>::MergeSorter (
        myList) {}
```

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

Definicja w linii 30 pliku [observablemergesorter.h](#).

```
00030 {};
```

5.18.3 Dokumentacja funkcji składowych

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

Reimplementowana z [MergeSorter< MyListElementType >](#).

Definicja w linii 23 pliku [observablemergesorter.h](#).

Odwwołuje się do [MergeSorter< MyListElementType >::list](#), [Observable::sendStartUpdateToObservers\(\)](#), [Observable::sendStopUpdateToObservers\(\)](#) i [MergeSorter< MyListElementType >::sort\(\)](#).

```
00024     {
00025         sendStartUpdateToObservers ();
00026         MergeSorter<MyListElementType>::sort ();
00027         sendStopUpdateToObservers ();
00028         return this->list;
00029     }
```

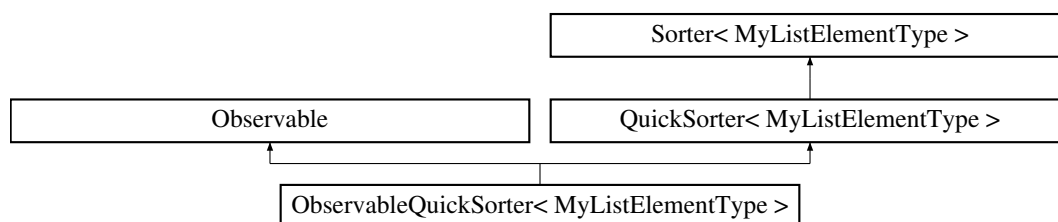
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [observablemergesorter.h](#)

5.19 Dokumentacja szablonu klasy ObservableQuickSorter< MyListElementType >

```
#include <observablequicksorter.h>
```

Diagram dziedziczenia dla ObservableQuickSorter< MyListElementType >



Metody publiczne

- [ObservableQuickSorter](#) (List< MyListElementType > &list)
- [List< MyListElementType > & sort](#) ()
- virtual [~ObservableQuickSorter](#) ()

Dodatkowe Dziedziczone Składowe

5.19.1 Opis szczegółowy

```
template<class MyListElementType>class ObservableQuickSorter< MyListElementType >
```

Definicja w linii 16 pliku [observablequicksorter.h](#).

5.19.2 Dokumentacja konstruktora i destruktor

5.19.2.1 `template<class MyListElementType > ObservableQuickSorter< MyListElementType >::ObservableQuickSorter (List< MyListElementType > & list) [inline]`

Definicja w linii 19 pliku [observablequicksorter.h](#).

```
00019                                     :
00020         QuickSorter<MyListElementType>::QuickSorter(list
    ) {}
```

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

Definicja w linii 30 pliku [observablequicksorter.h](#).

```
00030 {};
```

5.19.3 Dokumentacja funkcji składowych

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

Implementuje [Sorter< MyListElementType >](#).

Definicja w linii 23 pliku [observablequicksorter.h](#).

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

```
00024 {
00025     sendStartUpdateToObservers ();
00026     QuickSorter<MyListElementType>::sort ();
00027     sendStopUpdateToObservers ();
00028     return this->list;
00029 }
```

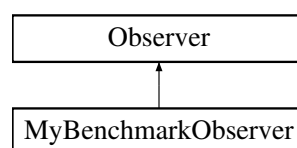
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [observablequicksorter.h](#)

5.20 Dokumentacja klasy Observer

```
#include <observer.h>
```

Diagram dziedziczenia dla Observer



Metody publiczne

- virtual double [getTimerValue](#) ()=0
- virtual void [receivedStartUpdate](#) ()=0
- virtual void [receivedStopUpdate](#) ()=0
- virtual void [receivedStopUpdateAndSaveToFile](#) ()=0
- virtual [~Observer](#) ()

5.20.1 Opis szczegółowy

Definicja w linii 16 pliku [observer.h](#).

5.20.2 Dokumentacja konstruktora i destruktora

5.20.2.1 `virtual Observer::~~Observer () [inline],[virtual]`

Definicja w linii 22 pliku [observer.h](#).

```
00022 {};
```

5.20.3 Dokumentacja funkcji składowych

5.20.3.1 `virtual double Observer::getTimerValue () [pure virtual]`

Implementowany w [MyBenchmarkObserver](#).

5.20.3.2 `virtual void Observer::receivedStartUpdate () [pure virtual]`

Implementowany w [MyBenchmarkObserver](#).

5.20.3.3 `virtual void Observer::receivedStopUpdate () [pure virtual]`

Implementowany w [MyBenchmarkObserver](#).

5.20.3.4 `virtual void Observer::receivedStopUpdateAndSaveToFile () [pure virtual]`

Implementowany w [MyBenchmarkObserver](#).

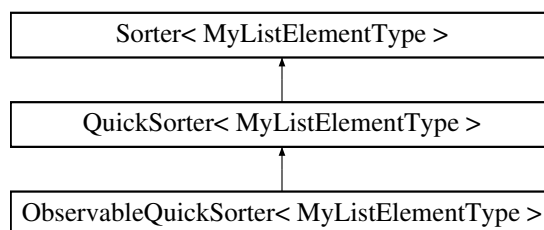
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [observer.h](#)

5.21 Dokumentacja szablonu klasy `QuickSorter< MyListElementType >`

```
#include <quicksorter.h>
```

Diagram dziedziczenia dla `QuickSorter< MyListElementType >`



Metody publiczne

- [QuickSorter](#) ([List< MyListElementType >](#) &[list](#))
- `virtual ~QuickSorter ()`
- `void quicksort (int lewy, int prawy)`
- [List< MyListElementType >](#) & [sort](#) ()

Atrybuty publiczne

- `int enablePivot`
- [List< MyListElementType >](#) & [list](#)

5.21.1 Opis szczegółowy

`template<class MyListElementType>class QuickSorter< MyListElementType >`

Szybkie sortowanie Janka

Definicja w linii 19 pliku [quicksorter.h](#).

5.21.2 Dokumentacja konstruktora i destruktora

5.21.2.1 `template<class MyListElementType > QuickSorter< MyListElementType >::QuickSorter (List< MyListElementType > & list) [inline]`

Definicja w linii 27 pliku [quicksorter.h](#).

Odwołuje się do [List< MyListElementType >::cloneFrom\(\)](#) i [QuickSorter< MyListElementType >::enablePivot](#).

```
00028         :list(list.createObjectFromAbstractReference())
00029     {
00030         this->list.cloneFrom(list);
00031         this->enablePivot=1;
00032     }
```

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

Definicja w linii 34 pliku [quicksorter.h](#).

```
00034 {};
```

5.21.3 Dokumentacja funkcji składowych

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

Definicja w linii 36 pliku [quicksorter.h](#).

Odwołuje się do [QuickSorter< MyListElementType >::enablePivot](#) i [QuickSorter< MyListElementType >::list](#).

Odwołania w [QuickSorter< MyListElementType >::sort\(\)](#).

```
00037     {
00038         int pivot=list[(int)(lewy+prawy)/2].content;
00039         int i,j,x;
00040         i=lewy;
00041         j=prawy;
00042         if(enablePivot) pivot=(list[(int)(lewy+prawy)/2].content +
list[lewy].content + list[prawy].content)/3;
00043         do
00044         {
00045             while(list[i].content<pivot) {i++; }
00046             while(list[j].content>pivot) {j--; }
00047             if(i<=j)
00048             {
00049                 x=list[i].content;
00050                 list[i].content=list[j].content;
00051                 list[j].content=x;
00052                 i++;
00053                 j--;
00054             }
00055         }
00056         while(i<=j);
00057         if(j>lewy) quicksort(lewy, j);
00058         if(i<prawy) quicksort(i, prawy);
00059     }
```


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

Implementuje `Sorter< MyListElementType >`.

Definicja w linii 61 pliku `quicksorter.h`.

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

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

```
00062     {
00063         //std::cout<<"(QuickSort) ";
00064         quicksort(0, list.size()-1);
00065         return list;
00066     }
```

5.21.4 Dokumentacja atrybutów składowych

5.21.4.1 `template<class MyListElementType > int QuickSorter< MyListElementType >::enablePivot`

Definicja w linii 22 pliku `quicksorter.h`.

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

5.21.4.2 `template<class MyListElementType > List<MyListElementType>& QuickSorter< MyListElementType >::list`

Definicja w linii 23 pliku `quicksorter.h`.

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

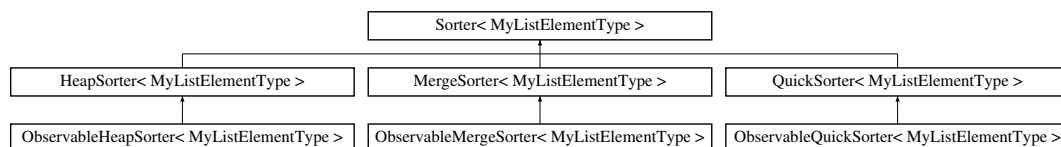
Dokumentacja dla tej klasy została wygenerowana z pliku:

- `quicksorter.h`

5.22 Dokumentacja szablonu klasy `Sorter< MyListElementType >`

```
#include <sorter.h>
```

Diagram dziedziczenia dla `Sorter< MyListElementType >`



Metody publiczne

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

5.22.1 Opis szczegółowy

```
template<class MyListElementType>class Sorter< MyListElementType >
```

Definicja w linii 15 pliku `sorter.h`.

5.22.2 Dokumentacja konstruktora i destruktor

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

Definicja w linii 20 pliku [sorter.h](#).

```
00020 {};
```

5.22.3 Dokumentacja funkcji składowych

5.22.3.1 `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](#)

6 Dokumentacja plików

6.1 Dokumentacja pliku avltree.h

```
#include <iostream>
#include <iomanip>
#include "avltreeelement.h"
```

Komponenty

- class [AVLTree< ContentType >](#)

6.2 avltree.h

```
00001 /*
00002  * avltree.h
00003  *
00004  * Created on: May 20, 2015
00005  * Author: serek8
00006  */
00007
00008 #ifndef AVL_TREE_H_
00009 #define AVL_TREE_H_
00010
00011 #include <iostream>
00012 #include <iomanip>
00013 #include "avltreeelement.h"
00014
00015 using namespace std;
00016
00017
00018 // Definicja klasy obsługującej drzewo AVL
00019 //-----
00020 template <class ContentType>
00021 class AVLTree
00022 {
00023     public:
00024
00025     AVLTreeNode<ContentType> * rootNode; // korzeń drzewa
00026
00027 }
```

```

00028
00029     AVLTree()
00030     {
00031         rootNode = NULL;
00032     }
00033     ~AVLTree()
00034     {
00035         while(rootNode)
00036         {
00037             delete(remove(rootNode));
00038         }
00039     }
00040     void insert(int &newKey)
00041     {
00042         AVLTreeNode<ContentType>* newNode = new
AVLTreeNode<ContentType>(newKey);
00043         //AVLTreeNode<ContentType> *newNode2 = new AVLTreeNode<ContentType>();
00044         AVLTreeNode<ContentType> * searchingNode = rootNode, // Wskaznik do
przeszukiwania drzewa i znalezienia tego samego klucza
00045             * parentForNewNode = NULL, // parentForNewNode
00046             * grandpaNode;
00047
00048         while(searchingNode)
00049         {
00050             if(searchingNode->key == newNode->key) // sprawdzam czy taki klucz juz istnieje
00051             {
00052                 //delete n; // skoro istnieje to po co taki TODO: do zmiany
00053
00054                 //cout <<"Taki klucz juz istnieje !\n";
00055                 return ;
00056             }
00057             parentForNewNode = searchingNode;
00058             if(newNode->key < searchingNode->key) searchingNode= searchingNode->
leftNode; // przechodze w lewo czesc drzewa
00059             else searchingNode = searchingNode->rightNode; //
przechodze w prawa czesc drzewa
00060         }
00061
00062
00063         // jezeli to jest pierwszy element to wpisuje go to root'a drzewa
00064         if(!(newNode->parentNode = parentForNewNode))
00065         {
00066             rootNode = newNode;
00067             return ;
00068         }
00069         // wybieram strone gałęzi na której ma byc element
00070         if(newNode->key < parentForNewNode->key) parentForNewNode->leftNode = newNode;
00071         else parentForNewNode->rightNode = newNode;
00072
00073         //sprawdzam czy potrzebne są rotacje, jak nie to koniec ;- )
00074         if(parentForNewNode->balanceFactor)
00075         {
00076             parentForNewNode->balanceFactor = 0;
00077             return ;
00078         }
00079
00080
00081         //parentForNewNode->balanceFactor = (parentForNewNode->leftNode == newNode) ? 1 : -1;
00082         if(parentForNewNode->leftNode == newNode) parentForNewNode->balanceFactor= 1;
00083         else parentForNewNode->balanceFactor = -1;
00084         grandpaNode = parentForNewNode->parentNode;
00085
00086         // usatwiam balanceFactors na 1 przed dodaniem
00087         // nowej gałęzi oraz wyznaczam grandpaForNewNode od ktorego zaczynam rotacje
00088         while(grandpaNode)
00089         {
00090             if(grandpaNode->balanceFactor) break; // gdy byly juz wczesniej ustawione to przerwij
00091
00092             if(grandpaNode->leftNode == parentForNewNode) grandpaNode->balanceFactor = 1;
00093             else grandpaNode->balanceFactor = -1;
00094             parentForNewNode = grandpaNode; grandpaNode = grandpaNode->parentNode;
00095         }
00096
00097         // jesli do konca byly zbalansowane to przerwij
00098         if(!grandpaNode) return ;
00099
00100         //rotacje na podstawie balanceFactors
00101         if(grandpaNode->balanceFactor == 1)
00102         {
00103             if(grandpaNode->rightNode == parentForNewNode)
00104             {
00105                 grandpaNode->balanceFactor = 0;
00106                 return ;
00107             }
00108             if(parentForNewNode->balanceFactor == -1) rotationLR(grandpaNode); //Rotacja podwójna w
lewo-prawo
00109             else rotationLL(grandpaNode); // Rotacja pojedyncza w prawo

```

```

00110         return ;
00111     }
00112     else
00113     {
00114         if(grandpaNode->leftNode == parentForNewNode)
00115         {
00116             grandpaNode->balanceFactor = 0;
00117             return ;
00118         }
00119         if(parentForNewNode->balanceFactor == 1) rotationRL(grandpaNode); //Rotacja podwójna w
lewo-prawo
00120     else rotationRR(grandpaNode); //Rotacja pojedyncza w lewo
00121     return ;
00122     }
00123 }
00124
00125 AVLTreeNode<ContentType> * rotationRR(
AVLTreeNode<ContentType> * A)
00126 {
00127     AVLTreeNode<ContentType> * B = A->rightNode, * P = A->
parentNode;
00128
00129     A->rightNode = B->leftNode;
00130     if(A->rightNode) A->rightNode->parentNode = A;
00131     B->leftNode = A;
00132     B->parentNode = P;
00133     A->parentNode = B;
00134     if(P)
00135     {
00136         if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00137     }
00138     else rootNode = B;
00139
00140     if(B->balanceFactor == -1)
00141     {
00142         A->balanceFactor = B->balanceFactor = 0;
00143     }
00144     else
00145     {
00146         A->balanceFactor = -1; B->balanceFactor = 1;
00147     }
00148     return B;
00149 }
00150
00151 AVLTreeNode<ContentType> * rotationLL(
AVLTreeNode<ContentType> * A)
00152 {
00153     AVLTreeNode<ContentType> * B = A->leftNode, * P = A->
parentNode;
00154
00155     A->leftNode = B->rightNode;
00156     if(A->leftNode) A->leftNode->parentNode = A;
00157     B->rightNode = A;
00158     B->parentNode = P;
00159     A->parentNode = B;
00160     if(P)
00161     {
00162         if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00163     }
00164     else rootNode = B;
00165
00166     if(B->balanceFactor == 1)
00167     {
00168         A->balanceFactor = B->balanceFactor = 0;
00169     }
00170     else
00171     {
00172         A->balanceFactor = 1; B->balanceFactor = -1;
00173     }
00174
00175     return B;
00176 }
00177 AVLTreeNode<ContentType> * rotationRL(
AVLTreeNode<ContentType> * A)
00178 {
00179     AVLTreeNode<ContentType> * B = A->rightNode, * C = B->
leftNode, * P = A->parentNode;
00180
00181     B->leftNode = C->rightNode;
00182     if(B->leftNode) B->leftNode->parentNode = B;
00183     A->rightNode = C->leftNode;
00184     if(A->rightNode) A->rightNode->parentNode = A;
00185     C->leftNode = A;
00186     C->rightNode = B;
00187     A->parentNode = B->parentNode = C;
00188     C->parentNode = P;
00189     if(P)

```

```

00190     {
00191         if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00192     }
00193     else rootNode = C;
00194
00195     A->balanceFactor = (C->balanceFactor == -1) ? 1 : 0;
00196     B->balanceFactor = (C->balanceFactor == 1) ? -1 : 0;
00197     C->balanceFactor = 0;
00198
00199     return C;
00200 }
00201 AVLTreeNode<ContentType> * rotationLR(
AVLTreeNode<ContentType> * A)
00202 {
00203     AVLTreeNode<ContentType> * B = A->leftNode, * C = B->
rightNode, * P = A->parentNode;
00204
00205     B->rightNode = C->leftNode;
00206     if(B->rightNode) B->rightNode->parentNode = B;
00207     A->leftNode = C->rightNode;
00208     if(A->leftNode) A->leftNode->parentNode = A;
00209     C->rightNode = A;
00210     C->leftNode = B;
00211     A->parentNode = B->parentNode = C;
00212     C->parentNode = P;
00213     if(P)
00214     {
00215         if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00216     }
00217     else rootNode = C;
00218
00219     A->balanceFactor = (C->balanceFactor == 1) ? -1 : 0;
00220     B->balanceFactor = (C->balanceFactor == -1) ? 1 : 0;
00221     C->balanceFactor = 0;
00222
00223     return C;
00224 }
00225
00226 // Wyszukuje element wg wartości klucza
00227 //-----
00228
00229 AVLTreeNode<ContentType> * find(int key)
00230 {
00231     AVLTreeNode<ContentType> * tmpNode = rootNode;
00232
00233     while((tmpNode) && (tmpNode->key != key))
00234     {
00235         if(key < tmpNode->key) tmpNode = tmpNode->leftNode;
00236         else tmpNode = tmpNode->rightNode;
00237     }
00238
00239     return tmpNode;
00240 }
00241 // Zwraca węzeł z minimalnym kluczem
00242 //-----
00243
00244 AVLTreeNode<ContentType> * findMaxKeyNode(
AVLTreeNode<ContentType> * tmpNode)
00245 {
00246     while(tmpNode->rightNode) tmpNode = tmpNode->rightNode;
00247     return tmpNode;
00248 }
00249
00250 // Zwraca węzeł poprzednika
00251 //-----
00252 //
00253 AVLTreeNode<ContentType> * findAtherNodeMatch(
AVLTreeNode<ContentType> * nodeComperator) // lowerValueFrom nodeComperator
00254 {
00255     if(nodeComperator->leftNode) return findMaxKeyNode(nodeComperator->
leftNode);
00256
00257     AVLTreeNode<ContentType> * y;
00258
00259     do
00260     {
00261         y = nodeComperator;
00262         nodeComperator = nodeComperator->parentNode;
00263     } while(nodeComperator && (nodeComperator->rightNode != y));
00264
00265     return nodeComperator;
00266 }
00267
00271 AVLTreeNode<ContentType> * remove(
AVLTreeNode<ContentType> * x)
00272 {
00273     AVLTreeNode<ContentType> * t, * y, * z;

```

```

00274     bool nest;
00275
00276     // Jeśli węzeł x posiada dwójkę dzieci, lewego i prawego potomka:
00277     if((x->leftNode) && (x->rightNode))
00278     {
00279         y = remove(findAtherNodeMatch(x));
00280         //rekurencyjnie usuwamy y za pomocą tego samego algorytmu
00281         nest = false;
00282     }
00283     //Jeśli węzeł x posiada tylko jedno dziecko lub nie posiada wcale dzieci:
00284     else {
00285         if(x->leftNode) {
00286             y = x->leftNode;
00287             x->leftNode = NULL;
00288         }
00289         else {
00290             y = x->rightNode; x->rightNode = NULL;
00291         }
00292         x->balanceFactor = 0;
00293         nest = true;
00294     }
00295
00296     if(y) {
00297         y->parentNode = x->parentNode;
00298         if(x->leftNode)
00299         {
00300             y->leftNode = x->leftNode;
00301             y->leftNode->parentNode = y;
00302         }
00303         if(x->rightNode)
00304         {
00305             y->rightNode = x->rightNode;
00306             y->rightNode->parentNode = y;
00307         }
00308         y->balanceFactor = x->balanceFactor;
00309     }
00310
00311     if(x->parentNode) {
00312         if(x->parentNode->leftNode == x) x->parentNode->leftNode = y; else x->
parentNode->rightNode = y;
00313     }
00314     else rootNode = y;
00315
00316     if(nest) {
00317         z = y;
00318         y = x->parentNode;
00319         while(y)
00320         {
00321             // węzeł y był w stanie równowagi przed usunięciem węzła x w jednym z jego poddrzew.
00322             if(!(y->balanceFactor)) {
00323                 y->balanceFactor = (y->leftNode == z) ? -1 : 1;
00324                 break;
00325             }
00326             else {
00327                 //skrócone zostało cięższe poddrzewo
00328                 if ((y->balanceFactor == 1) &&
00329                     (y->leftNode == z)) || ((y->balanceFactor == -1) &&
00330                     (y->rightNode == z)) {
00331                     y->balanceFactor = 0;
00332                     z = y; y = y->parentNode;
00333                 }
00334                 else {
00335                     t = (y->leftNode == z) ? y->rightNode : y->
leftNode;
00336
00337                     //Wykonujemy odpowiednią rotację pojedynczą
00338                     if(!(t->balanceFactor)) {
00339                         if(y->balanceFactor == 1) rotationLL(y); else rotationRR(y);
00340                         break;
00341                     }
00342                     //Wykonujemy odpowiednią rotację pojedynczą
00343                     else if(y->balanceFactor == t->balanceFactor)
00344                     {
00345                         if(y->balanceFactor == 1) rotationLL(y); else rotationRR(y);
00346                         z = t; y = t->parentNode;
00347                     }
00348                     //Wykonujemy rotację podwójną
00349                     else
00350                     {
00351                         if(y->balanceFactor == 1) rotationLR(y); else rotationRL(y);
00352                         z = y->parentNode; y = z->parentNode;
00353                     }
00354                 }
00355             }
00356         }
00357     }
00358     return x;

```

```

00359     }
00360     void print()
00361     {
00362         this->recurringPrint(this->rootNode);
00363     }
00364     void recurringPrint(AVLTreeNode<ContentType> * x)
00365     {
00366         cout<< "klucz:"<< x->key <<"\n";
00367         if(x->leftNode) recurringPrint(x->leftNode);
00368         if(x->rightNode) recurringPrint(x->rightNode);
00369     }
00370     void walk(AVLTreeNode<ContentType> * x)
00371     {
00372         cout << x->key << " : bf = " << setw(2) << x->balanceFactor << " : Left-> ";
00373         if(x->leftNode) cout << setw(3) << x->leftNode->key;
00374         else cout << "NIL";
00375         cout << ", Right-> ";
00376         if(x->rightNode) cout << setw(3) << x->rightNode->key;
00377         else cout << "NIL";
00378         cout << " : p -> ";
00379         if(x->parentNode) cout << setw(3) << x->parentNode->
key;
00380         else cout << "NIL";
00381         cout << endl;
00382         if(x->leftNode) walk(x->leftNode);
00383         if(x->rightNode) walk(x->rightNode);
00384     }
00385 };
00386
00387
00388
00389
00390 #endif /* AVLTREE_H_ */

```

6.3 Dokumentacja pliku avltreeelement.h

Komponenty

- class [AVLTreeNode< ContentType >](#)

6.4 avltreeelement.h

```

00001 /*
00002  * avltreeelement.h
00003  *
00004  * Created on: May 21, 2015
00005  * Author: serek8
00006  */
00007
00008 #ifndef AVLTREEELEMENT_H_
00009 #define AVLTREEELEMENT_H_
00010
00011 // definicja typu danych reprezentującego węzeł drzewa AVL
00012 //-----
00013 template <class ContentType>
00014 class AVLTreeNode
00015 {
00016 public:
00017     // Gałąź rodzica, lewa podgałąź, prawa podgałąź
00018     AVLTreeNode * parentNode,
00019                 * leftNode,
00020                 * rightNode;
00021
00022     //klucz
00023     int key, balanceFactor;
00024     ContentType content;
00025
00026     AVLTreeNode()
00027     {
00028         parentNode = leftNode = rightNode = NULL;
00029         key = balanceFactor = 0;
00030     }
00031     AVLTreeNode(int newKey)
00032     {
00033         parentNode = leftNode = rightNode = NULL;
00034         balanceFactor = 0;
00035         key = newKey;
00036     }
00037 };
00038 };

```

```
00039
00040
00041
00042
00043 #endif /* AVLTREEELEMENT_H_ */
```

6.5 Dokumentacja pliku filestreamer.h

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

Funkcje

- void [writeString1ToFile](#) (std::string fileName, std::string textToSave)
- void [writeString2ToFile](#) (std::string fileName, double textToSave)
- void [writeString3ToFile](#) (std::string fileName, int textToSave)
- void [clearFile](#) (std::string fileName)

6.5.1 Dokumentacja funkcji

6.5.1.1 void clearFile (std::string fileName)

Definicja w linii 41 pliku [filestreamer.h](#).

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

6.5.1.2 void writeString1ToFile (std::string fileName, std::string textToSave)

Definicja w linii 15 pliku [filestreamer.h](#).

```
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 }
```

6.5.1.3 void writeString2ToFile (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 }
```

6.5.1.4 void writeString3ToFile (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 }

```

6.6 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 writeString1ToFile(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 writeString2ToFile(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 writeString3ToFile(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_ */

```

6.7 Dokumentacja pliku heapsorter.h

```

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

```

Komponenty

- class `HeapSorter< MyListElementType >`

6.8 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
00015 template <class MyListElementType>
00016 class HeapSorter: public Sorter<MyListElementType>
00017 {
00018 public:
00019     List<MyListElementType> &list;
00020
00021     HeapSorter(List<MyListElementType> &myList)
00022     :list(myList.createObjectFromAbstractReference())
00023     {
00024
00025         this->list.cloneFrom(myList);
00026         /*this->sizeOfList = myList.sizeOfList;
00027         this->firstElement = myList.firstElement;
00028         this->lastElement = myList.lastElement;
00029         this->iterator=myList.iterator;
00030         this->isIteratorAfterPop = myList.isIteratorAfterPop;*/
00031     }
00032
00033     virtual ~HeapSorter(){};
00034
00035     List<MyListElementType> &sort()
00036     {
00037         int n = this->list.size();
00038         int parent = n/2, index, child, tmp; /* heap indexes */
00039         /* czekam az sie posortuje */
00040         while (1) {
00041             if (parent > 0)
00042             {
00043                 tmp = (this->list)[--parent].content; /* kobie kopie do tmp */
00044             }
00045             else {
00046                 n--;
00047                 if (n == 0)
00048                 {
00049                     return this->list; /* Zwraca posortowane */
00050                 }
00051                 tmp = this->list[n].content;
00052                 this->list[n].content = this->list[0].content;
00053             }
00054             index = parent;
00055             child = index * 2 + 1;
00056             while (child < n) {
00057                 if (child + 1 < n && this->list[child + 1].content > this->
list[child].content) {
00058                     child++;
00059                 }
00060                 if (this->list[child].content > tmp) {
00061                     this->list[index].content = this->list[child].content;
00062                     index = child;
00063                     child = index * 2 + 1;
00064                 } else {
00065                     break;
00066                 }
00067             }
00068             this->list[index].content = tmp;
00069         }
00070         return this->list;
00071     }
00072
00073
00074
00075 };
00076
00077
00078 #endif /* HEAPSORTER_H_ */

```

6.9 Dokumentacja pliku list.h

```

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

```

Komponenty

- class [List< MyListElementType >](#)

6.10 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
00014 template <class MyListElementType>
00015 class List
00016 {
00017 public:
00018
00019     int virtual &size() = 0;
00020     ListElement<MyListElementType> virtual
00021     pop_back() = 0;
00022     ListElement<MyListElementType> virtual
00023     pop_front() = 0;
00024     void virtual printList() = 0;
00025     void virtual push_back(MyListElementType arg) = 0;
00026     void virtual push_front(MyListElementType arg) = 0;
00027     MyListElement<MyListElementType> virtual &
00028     operator[] (int numberOfElement) = 0;
00029     void virtual insertAfter(MyListElement<MyListElementType>
00030     arg, int iteratorID) = 0;
00031     MyListElementType virtual &show_front() = 0;
00032     MyListElementType virtual &show_back() = 0;
00033     //List<MyListElementType> virtual &operator=(const List<MyListElementType> &pattern) = 0;
00034     void virtual cloneFrom(List<MyListElementType> &patternList)
00035     {
00036         // release memory from main list
00037         while(this->size()) pop_back();
00038         for(int i=0; i<patternList.size(); i++)
00039             this->push_back(patternList[i].content);
00040     }
00041     List<MyListElementType> virtual &
00042     createObjectFromAbstractReference() = 0;
00043     void virtual free(){ while(size()) pop_back(); }
00044     virtual ~List(){};
00045 };
00046 #endif /* LIST_H_ */

```

6.11 Dokumentacja pliku listelement.h

Komponenty

- class [ListElement< MyListElementType >](#)

6.12 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

```

```

00011 template <class MyListElementType>
00012 class ListElement
00013 {
00014 public:
00015     MyListElementType content;
00016
00017 };
00018
00019
00020
00021
00022 #endif /* LISTELEMENT_H_ */

```

6.13 Dokumentacja pliku listsaver.h

```

#include <string>
#include <fstream>

```

Komponenty

- class `ListSaver< MyListElementType >`

6.14 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 {
00017     List<MyListElementType> &list;
00018
00019     ListSaver(MyList<MyListElementType> &listArgument):
00020         list(listArgument)
00021     {}
00022
00027     int saveToFile(std::string nazwaPliku)
00028     {
00029         std::ofstream streamToFile;
00030         streamToFile.open (nazwaPliku.c_str(), std::ofstream::out);
00031         for(int i=0; i<list.size() ; i++)
00032             streamToFile << '{'<<list[i].content<<" ";
00033
00034         return 0;
00035     }
00036
00037 };
00038
00039
00040
00041
00042
00043 #endif /* LISTSAVER_H_ */

```

6.15 Dokumentacja pliku main.cpp

```
#include <iostream>
#include <unistd.h>
#include "numbergenerator.h"
#include "observableavltree.h"
#include "mybenchmark.h"
#include "mylist.h"
```

Definicje

- `#define ILE_WYRAZOW_DO_WSTAWIENIA 100`

Funkcje

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

6.15.1 Dokumentacja definicji

6.15.1.1 `#define ILE_WYRAZOW_DO_WSTAWIENIA 100`

Definicja w linii 15 pliku `main.cpp`.

Odwołania w `main()`.

6.15.2 Dokumentacja funkcji

6.15.2.1 `int main (int argc, char * argv[])`

Definicja w linii 17 pliku `main.cpp`.

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

```
00018 {
00019     MyList<int> lista;
00020
00021
00022     lista = NumberGenerator::generateNumbers<int>(100,
00023     ILE_WYRAZOW_DO_WSTAWIENIA);
00024     ObservableAVLTree<int> tree;
00025     MyBenchmarkObserver *ol = new MyBenchmarkObserver();
00026     tree.add(ol);
00027
00028     for(int i=0; i<lista.size(); i++)
00029     {
00030         tree.insert(lista[i].content);
00031     }
00032     std::cout<<std::endl;
00033     return 0;
00034 }
```

6.16 main.cpp

```
00001 /*
00002  * main.cpp
00003  *
00004  * Created on: Mar 6, 2015
00005  * Author: serek8
00006  */
00007
00008 #include <iostream>
00009 #include <unistd.h>
00010 #include "numbergenerator.h"
00011 #include "observableavltree.h"
00012 #include "mybenchmark.h"
```

```

00013 #include "mylist.h"
00014
00015 #define ILE_WYRAZOW_DO_WSTAWIENIA 100
00016
00017 int main(int argc, char *argv[])
00018 {
00019     MyList<int> lista;
00020
00021
00022     lista = NumberGenerator::generateNumbers<int>(100,
00023     ILE_WYRAZOW_DO_WSTAWIENIA);
00023     ObservableAVLTree<int> tree;
00024     MyBenchmarkObserver *ol = new MyBenchmarkObserver();
00025     tree.add(ol);
00026
00027     for(int i=0; i<lista.size(); i++)
00028     {
00029         tree.insert(lista[i].content);
00030     }
00031     std::cout<<std::endl;
00032     return 0;
00033 }

```

6.17 Dokumentacja pliku mergesorter.h

```

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

```

Komponenty

- class [MergeSorter< MyListElementType >](#)

6.18 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
00014 template <class MyListElementType>
00015 class MergeSorter: public Sorter<MyListElementType> {
00016 public:
00017
00018     MyList<MyListElementType> &list;
00019
00020     MergeSorter(MyList<MyListElementType> &listArg)
00021     :list(listArg) {}
00022
00023     virtual ~MergeSorter() {}
00024
00025
00026     MyList<MyListElementType> merge(
00027     MyList<MyListElementType> left,
00028     MyList<MyListElementType> right)
00029     {
00030         MyList<MyListElementType> result;
00031         //Gdy jest jeszcze cos do sortowania
00032         while (left.size() > 0 || right.size() > 0)
00033         {
00034             // Jak oba to zamieniamy
00035             if (left.size() > 0 && right.size() > 0)
00036             {
00037                 // Sprawdzam czy zamieniac
00038                 if (left.show_front() <= right.
00039                 show_front())
00040                 {
00041                     result.push_back(left.
00042                     show_front()); left.pop_front();
00043                 }
00044                 else
00045                 {
00046                     result.push_back(right.
00047                     show_front()); right.pop_front();
00048                 }
00049             }
00050             else if (left.size() > 0)
00051             {
00052                 result.push_back(left.
00053                 show_front()); left.pop_front();
00054             }
00055             else if (right.size() > 0)
00056             {
00057                 result.push_back(right.
00058                 show_front()); right.pop_front();
00059             }
00060         }
00061         return result;
00062     }
00063 };

```

```

00039         }
00040         else
00041         {
00042             result.push_back(right.
show_front()); right.pop_front();
00043         }
00044     }
00045     // pojedyncze listy (nieparzyse)
00046     else if (left.size() > 0)
00047     {
00048         for (int i = 0; i < left.size(); i++) result.
push_back(left[i].content); break;
00049     }
00050     // pojedyncze listy (nieparzyse- taka sama sytuacja jak wyzej)
00051     else if ((int)right.size() > 0)
00052     {
00053         for (int i = 0; i < (int)right.size(); i++) result.
push_back(right[i].content); break;
00054     }
00055     }
00056     return result;
00057 }
00063 MyList<MyListElementType> mergeSort (
MyList<MyListElementType> m)
00064 {
00065     if (m.size() <= 1) return m; // gdy juz nic nie ma do sortowania
00066     MyList<MyListElementType> left, right, result;
00067     int middle = (m.size()+ 1) / 2; // anty-nieparzyscie
00068     for (int i = 0; i < middle; i++)
00069     {
00070         left.push_back(m[i].content);
00071     }
00072     for (int i = middle; i < m.size(); i++)
00073     {
00074         right.push_back(m[i].content);
00075     }
00076     left = mergeSort(left);
00077     right = mergeSort(right);
00078     result = merge(left, right);
00079     return result;
00080 }
00081
00082
00083 List<MyListElementType> &sort ()
00084 {
00085     this->list=mergeSort(this->list);
00086     return this->list;
00087 }
00088
00089 };
00090
00091 #endif /* MERGESORT_H_ */

```

6.19 Dokumentacja pliku mybenchmark.cpp

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

6.20 mybenchmark.cpp

```

00001 /*
00002  * mybenchmark.cpp
00003  *
00004  * Created on: Mar 6, 2015
00005  * Author: serek8
00006  */
00009 #include "mybenchmark.h"
00010
00011 double MyBenchmark::timerValueStatic=0;
00012 void MyBenchmark :: timerStart ()
00013 {
00014     MyBenchmark::timerValueStatic = (( (double)clock() ) /CLOCKS_PER_SEC);
00015 }
00016
00017 double MyBenchmark :: timerStop()
00018 {
00019     return (( (double)clock() ) /CLOCKS_PER_SEC) -
MyBenchmark::timerValueStatic;
00020 }

```

6.21 Dokumentacja pliku mybenchmark.h

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

Komponenty

- class [MyBenchmark](#)
Klasa bazowa/interface do testowania algorytmu.
- class [MyBenchmarkObserver](#)

6.22 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>
00014 #include <fstream>
00015 // #include "filestreamer.h"
00023 class MyBenchmark
00024 {
00025
00037 public:
00038
00040     double timerValue;
00041     std::ofstream streamToFile;
00042     static double timerValueStatic;
00043     MyBenchmark()
00044     {
00045         timerValue = 0;
00046         streamToFile.open ("log.txt", std::ofstream::out | std::ofstream::trunc);
00047         streamToFile.close();
00048         streamToFile.open ("log.txt", std::ofstream::app);
00049         streamToFile << std::fixed;
00050     }
00051
00052
00053
00055     static void timerStart();
00056
00061     static double timerStop();
00062
00066     virtual ~MyBenchmark() {};
00067     //using DataFrame::operator=;
00068
00069 };
00070
00071
00072 class MyBenchmarkObserver : public MyBenchmark, public
00073     Observer
00074 {
00075 public:
00076     MyBenchmarkObserver(){};
00077     double getTimerValue() {return this->timerValue;}
00078     void receivedStartUpdate () {
00079         //std::cout<<"\nWlaczam stoper...";
00080         timerStart();
00081     }
00082     void receivedStopUpdate () {
00083         std::cout<<"\nCzas wykonywania operacji: "<<timerStop();
00084     }
00085
00086     void receivedStopUpdateAndSaveToFile () {
00087         timerStop();
00088         streamToFile<<timerValue<<std::endl;
```



```

00089
00090     }
00091
00092     virtual ~MyBenchmarkObserver() {streamToFile.close();};
00093
00094 };
00095
00096
00097
00098 #endif /* MYBENCHMARK_H_ */

```

6.23 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.

6.24 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     MyList insertAfter(){return *new
MyList<MyListElementType>();}
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
00062     int &size()
00063     {
00064         return sizeOfList;
00065     }
00066
00067     /*MyListElement<MyListElementType> &pop_back()
00068     {
00069         if(!(sizeOfList--)) { sizeOfList=0; return *(new MyListElement<MyListElementType>); }
00070         MyListElement<MyListElementType> tmpNumber = *(this -> lastElement);
00071         MyListElement<MyListElementType> *originMyListElement = this -> lastElement;
00072         this -> lastElement = this -> lastElement -> previousElement;
00073         delete originMyListElement;
00074         isIteratorAfterPop=1;
00075         return tmpNumber;
00076     }*/
00077     ListElement<MyListElementType> pop_back()
00078     {
00079         if(!(sizeOfList--)) { sizeOfList=0; return *(new
MyListElement<MyListElementType>); }
00080         MyListElement<MyListElementType> tmpNumber = *(this ->
lastElement);
00081         MyListElement<MyListElementType> *originMyListElement =
this -> lastElement;
00082         this -> lastElement = this -> lastElement -> previousElement;
00083         delete originMyListElement;
00084         isIteratorAfterPop=1;
00085         return tmpNumber;
00086     }
00087     ListElement<MyListElementType> pop_front()
00088     {
00089         if(!(sizeOfList--)) { sizeOfList=0; return *(new
MyListElement<MyListElementType>()); }
00090         MyListElement<MyListElementType> tmpNumber = *(this ->
firstElement);
00091         MyListElement<MyListElementType> *originMyListElement =
this -> firstElement;
00092         this -> firstElement = this -> firstElement -> nextElement;
00093         delete originMyListElement;
00094         isIteratorAfterPop=1;
00095         return tmpNumber;
00096     }
00097     void push_back(MyListElementType arg)
00098     {
00099         //std::cerr<<"\n(push_back): arg.content="<<arg.content;
MyListElement<MyListElementType> *newMyListElement = new
MyListElement<MyListElementType>(arg);
00100         if(!sizeOfList++) {firstElement =
lastElement = newMyListElement;
00101         //newMyListElement -> nextElement = 0;
00102         newMyListElement -> previousElement = this -> lastElement;
00103         this -> lastElement -> nextElement = newMyListElement;
00104         this->lastElement = newMyListElement;
00105     }
00106     void push_front(MyListElementType arg)
00107     {
00108         MyListElement<MyListElementType> *newMyListElement = new
MyListElement<MyListElementType>(arg);
00109         if(!sizeOfList++) {firstElement =
lastElement = newMyListElement;
00110         //newMyListElement -> previousElement = 0;
00111         newMyListElement -> nextElement = this -> firstElement;
00112         this -> firstElement -> previousElement = newMyListElement;
00113         this->firstElement = newMyListElement;
00114         ++iteratorElementId;
00115     }
00116     MyListElementType &show_front()
00117     {
00118         return firstElement->content;
00119     }
00120     MyListElementType &show_back()
00121     {
00122         return lastElement->content;
00123     }

```

```

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

```

```

00235     MyList<MyListElementType> &pattern)
00236     {
00237         //std::cerr<<" @@@";
00238         this->sizeOfList = pattern.sizeOfList;
00239         this->firstElement = pattern.firstElement;
00240         this->lastElement = pattern.lastElement;
00241         this->iterator=pattern.iterator;
00242         this->isIteratorAfterPop = pattern.
00243         isIteratorAfterPop;
00244         return *this;
00245     }
00246     List<MyListElementType> &operator=(const List<MyListElementType> &pattern)
00247     {
00248         std::cerr<<" ###";
00249         //this->cloneFrom(pattern);
00250         return *this;
00251     }
00252     void cloneFrom(MyList<MyListElementType> patternList)
00253     {
00254         MyList<MyListElementType> &clonedList = *new MyList<MyListElementType>;
00255         // release memory from main list
00256         while(this->size()) pop_back();
00257         for(int i=0; i<patternList.size(); i++)
00258             clonedList.push_back(patternList[i]);
00259         *this = clonedList;
00260     }
00261     /*
00262     List<MyListElementType> &
00263     createObjectFromAbstractReference (/*MyList<MyListElementType>
00264     abstractPattern*/)
00265     {
00266         return *new MyList<MyListElementType>;
00267     }
00268 };
00269
00270
00271
00272
00273
00274
00275
00276
00277
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     MyListObserved(){};
00290     ~MyListObserved(){};
00291 };*/
00292
00293
00294
00295
00296 #endif /* MYLIST_H_ */

```

6.25 Dokumentacja pliku mylistelement.h

```

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

```

Komponenty

- class `MyListElement< MyListElementType >`

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

6.26 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     }
00038     MyListElement(MyListElementType arg)
00039     {
00040         this -> content = arg;
00041         this -> nextElement =0;
00042         this -> previousElement =0;
00043         //std::cerr<<"\n(konstruktor MyListElement): content="<<arg;
00044     }
00049     MyListElement(const MyListElement &myListElement)
00050     {
00051         //this->number = myListElement.number;
00052         //this->nazwa = myListElement.nazwa;
00053         this->content = myListElement.content;
00054         this->nextElement = myListElement.nextElement;
00055         this->previousElement = myListElement.
00056         previousElement;
00057         //std::cerr<<"\n(konstruktor kopiujacy MyListElement): content="<<content;
00058     }
00063     void set(MyListElementType arg)
00064     {
00065         this -> content = arg;
00066         //this -> nazwa = str;
00067     }
00068     //friend class MyList;
00069 };
00070 #endif /* MYLISTELEMENT_H_ */

```

6.27 Dokumentacja pliku myqueue.h

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

Komponenty

- class **MyQueue**

Klasa reprezentuje kolejkę.

6.28 myqueue.h

```

00001 /*
00002  * myqueue.h
00003  *
00004  *   Created on: Mar 16, 2015
00005  *   Author: serek8
00006  */
00007
00008 #ifndef MYQUEUE_H_
00009 #define MYQUEUE_H_
00010 #include "mylist.h"
00011
00016 class MyQueue : public MyList
00017 {
00018 public:
00019     /*
00020     * @brief Dodaje element do kolejki

```

```

00021      * @param arg Liczba dodawana do kolejki
00022      */
00023      void push(int arg) {
00024          push_back(arg);
00025      }
00027      int pop() {
00028          return pop_front();
00029      }
00030 };
00031
00032 #endif /* MYQUEUE_H_ */

```

6.29 Dokumentacja pliku mystack.h

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

Komponenty

- class [MyStack](#)
Klasa reprezentuje stos.

6.30 mystack.h

```

00001 /*
00002  * mystack.h
00003  *
00004  * Created on: Mar 16, 2015
00005  * Author: serek8
00006  */
00007
00008 #ifndef MYSTACK_H_
00009 #define MYSTACK_H_
00010
00011 #include "mylist.h"
00012
00018 class MyStack : public MyList
00019 {
00020 public:
00021     /*
00022     * @brief Dodaje element do kolejki
00023     * @param arg Liczba dodawana do stosu
00024     */
00025     void push(int arg) {
00026         push_back(arg);
00027     }
00029     int pop() {
00030         return pop_back();
00031     }
00032 };
00033
00034 #endif /* MYSTACK_H_ */

```

6.31 Dokumentacja pliku numbergenerator.h

```

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

```

Komponenty

- class [NumberGenerator](#)
Klasa generująca losowe liczby.

Definicje

- `#define MAX_HEX_ASCII_KOD 127`
- `#define ROZMIAR_STRINGU 20`

6.31.1 Dokumentacja definicji

6.31.1.1 `#define MAX_HEX_ASCII_KOD 127`

Definicja w linii 17 pliku `numbergenerator.h`.

6.31.1.2 `#define ROZMIAR_STRINGU 20`

Definicja w linii 18 pliku `numbergenerator.h`.

6.32 `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:
00036 template <typename MyListElementType>
00037 MyList<MyListElementType> static generateNumbers(int range, int
00038 quantity)
00039 {
00039     MyList<MyListElementType> &myList = *new
00040     MyList<MyListElementType>();
00041     time_t randomTime = clock();
00042     for(int i=0; i<quantity ; i++)
00043     {
00044         srand (randomTime = clock());
00045         myList.push_back(rand()%range);
00046         randomTime = clock();
00047     }
00048     return myList;
00049 }
00050
00057 static std::string *generateStrings(int ileStringow);
00058
00059
00060
00061 //using DataFrame::operator=;
00062 };
00063 };
00064
00065 #endif /* NUMBERGENERATOR_H_ */
```

6.33 Dokumentacja pliku `observable.h`

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

Komponenty

- class [Observable](#)

6.34 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
00014 class Observable {
00015 public:
00016     //MyList<int> observerswww;
00017     MyList<Observer*> observers;
00018
00019     void add(Observer *o) {
00020         observers.push_back(o);
00021     }
00022
00023     void sendStartUpdateToObservers () {
00024         for(int i=0; i<observers.size(); i++)
00025         {
00026             //std::cout<<"Wysylam start update";
00027             observers[i].content->receivedStartUpdate();
00028         }
00029     }
00030     void sendStopUpdateToObservers () {
00031         for(int i=0; i<observers.size(); i++)
00032             observers[i].content->receivedStopUpdate();
00033     }
00034     void sendStopUpdateToObserversAndSaveToFile () {
00035         for(int i=0; i<observers.size(); i++)
00036             observers[i].content->receivedStopUpdateAndSaveToFile();
00037     }
00038
00039     virtual ~Observable() {}
00040
00041 };
00042
00043 #endif /* OBSERVABLE_H_ */

```

6.35 Dokumentacja pliku observableavltree.h

```

#include "observable.h"
#include "avltree.h"

```

Komponenty

- class [ObservableAVLTree< ContentType >](#)

6.36 observableavltree.h

```

00001 /*
00002  * observableavltree.h
00003  *
00004  * Created on: May 21, 2015
00005  * Author: serek8
00006  */
00007
00008 #ifndef OBSERVABLEAVLTREE_H_

```



```

00009 #define OBSERVABLEAVLTREE_H_
00010
00011 #include "observable.h"
00012 #include "avltree.h"
00013
00014
00015 template <class ContentType>
00016 class ObservableAVLTree : public Observable, public
    AVLTree<ContentType>
00017 {
00018 public:
00019     void insert(int newKey)
00020     {
00021         sendStartUpdateToObservers();
00022         AVLTree<ContentType>::insert(newKey);
00023         sendStopUpdateToObserversAndSaveToFile();
00024     }
00025
00026     ~ObservableAVLTree() {}
00027 };
00028
00029
00030
00031 #endif /* OBSERVABLEAVLTREE_H_ */

```

6.37 Dokumentacja pliku observableheapsorter.h

```

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

```

Komponenty

- class `ObservableHeapSorter< MyListElementType >`

6.38 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
00015 template <class MyListElementType>
00016 class ObservableHeapSorter : public Observable, public
    HeapSorter<MyListElementType>
00017 {
00018 public:
00019     ObservableHeapSorter(List<MyListElementType> &myList):
        HeapSorter<MyListElementType>::HeapSorter(myList) {}
00020
00021
00022
00023     List<MyListElementType> &sort ()
00024     {
00025         sendStartUpdateToObservers();
00026         HeapSorter<MyListElementType>::sort();
00027         sendStopUpdateToObservers();
00028         return this->list;
00029     }
00030     virtual ~ObservableHeapSorter() {};
00031
00032
00033 };
00034
00035
00036 #endif /* OBSERVABLEHEAPSORTER_H_ */

```

6.39 Dokumentacja pliku observablemergesorter.h

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

Komponenty

- class `ObservableMergeSorter< MyListElementType >`

6.40 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
00015 template <class MyListElementType>
00016 class ObservableMergeSorter : public Observable, public
MergeSorter<MyListElementType>
00017 {
00018 public:
00019     ObservableMergeSorter(MyList<MyListElementType> &
myList):
00020         MergeSorter<MyListElementType>::MergeSorter(myList){}
00021
00022
00023     List<MyListElementType> &sort()
00024     {
00025         sendStartUpdateToObservers();
00026         MergeSorter<MyListElementType>::sort();
00027         sendStopUpdateToObservers();
00028         return this->list;
00029     }
00030     virtual ~ObservableMergeSorter(){};
00031
00032
00033 };
00034
00035
00036 #endif /* OBSERVABLEMERGESORTER_H_ */
```

6.41 Dokumentacja pliku observablequicksorter.h

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

Komponenty

- class `ObservableQuickSorter< MyListElementType >`

6.42 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
00015 template <class MyListElementType>
00016 class ObservableQuickSorter : public Observable, public
    QuickSorter<MyListElementType>
00017 {
00018 public:
00019     ObservableQuickSorter(List<MyListElementType> &
        list):
00020         QuickSorter<MyListElementType>::QuickSorter(list){}
00021
00022
00023     List<MyListElementType> &sort()
00024     {
00025         sendStartUpdateToObservers();
00026         QuickSorter<MyListElementType>::sort();
00027         sendStopUpdateToObservers();
00028         return this->list;
00029     }
00030     virtual ~ObservableQuickSorter(){};
00031
00032
00033 };
00034
00035
00036 #endif /* OBSERVABLEQUICKSORTER_H_ */

```

6.43 Dokumentacja pliku observer.h

Komponenty

- class `Observer`

6.44 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
00015
00016 class Observer {
00017 public:
00018     virtual double getTimerValue() = 0;
00019     virtual void receivedStartUpdate() = 0;
00020     virtual void receivedStopUpdate() = 0;
00021     virtual void receivedStopUpdateAndSaveToFile() = 0;
00022     virtual ~Observer(){};
00023 };
00024
00025
00026
00027
00028
00029
00030
00031
00032 #endif /* OBSERVER_H_ */

```

6.45 Dokumentacja pliku quicksorter.h

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

Komponenty

- class `QuickSorter< MyListElementType >`

6.46 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 template <class MyListElementType>
00015 class QuickSorter : public Sorter<MyListElementType>
00016 {
00017 public:
00018     int enablePivot;
00019     List<MyListElementType> &list;
00020
00021     QuickSorter(List<MyListElementType> &
00022 list)
00023 :list(list.createObjectFromAbstractReference())
00024 {
00025     this->list.cloneFrom(list);
00026     this->enablePivot=1;
00027 }
00028
00029 virtual ~QuickSorter(){};
00030 //void quicksort(int lewy, int prawy, int enablePivot)
00031 void quicksort(int lewy, int prawy)
00032 {
00033     int pivot=list[(int) (lewy+prawy)/2].content;
00034     int i,j,x;
00035     i=lewy;
00036     j=prawy;
00037     if(enablePivot) pivot=(list[(int) (lewy+prawy)/2].content +
00038 list[lewy].content + list[prawy].content)/3;
00039     do
00040     {
00041         while(list[i].content<pivot) {i++; }
00042         while(list[j].content>pivot) {j--; }
00043         if(i<=j)
00044         {
00045             x=list[i].content;
00046             list[i].content=list[j].content;
00047             list[j].content=x;
00048             i++;
00049             j--;
00050         }
00051     } while(i<=j);
00052     if(j>lewy) quicksort(lewy, j);
00053     if(i<prawy) quicksort(i, prawy);
00054 }
00055
00056 List<MyListElementType> &sort ()
00057 {
00058     //std::cout<<"(QuickSort) ";
00059     quicksort(0, list.size()-1);
00060     return list;
00061 }
00062 };
00063
```

```
00069
00070
00071 #endif /* QUICKSORT_H_ */
```

6.47 Dokumentacja pliku sorter.h

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

Komponenty

- class `Sorter< MyListElementType >`

6.48 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
00013
00014 template <class MyListElementType>
00015 class Sorter
00016 {
00017 public:
00018
00019     virtual List<MyListElementType> &sort() = 0;
00020     virtual ~Sorter(){};
00021 };
00022
00023
00024 #endif /* SORTER_H_ */
```

6.49 Dokumentacja pliku strona-glowna.dox