Laboratorium 8

Wygenerowano przez Doxygen 1.8.6

Pt, 22 maj 2015 23:59:09

Spis treści

1 Laboratorium 2

Aplikacja umozliwia uzytkownikowi na przeprowadzenia algorytmu mnozenia

przez dwa na dowolnej liczbie elementow.

Najważniejsze cechy

Możliwość włączenia opcji benchmarkującej służącej do sprawdzenia

ile czasu wykonywal sie dany algorytm lub seria tego samego algorytmu

Argumenty wywołania

```
-n liczba       Ilość liczb do odczytania/przerobienia przez algorytm
-t liczba       Włącza opcje benchmarkującą dla seri powtorzen
-o tekst        Wprowadza nazwe pliku do zapisu
-i tekst        Wprowadza nazwe pliku do odczytu
-g              Generuje n liczb i zapisuje je do pliku (po wygenerowaniu konczy program)
```

2 Indeks hierarchiczny

2.1 Hierarchia klas

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

3 Indeks klas

3.1 Lista klas

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

4 Indeks plików

4.1 Lista plików

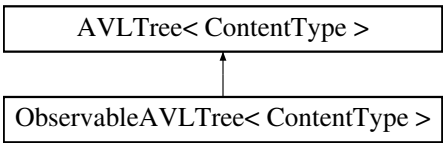
Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

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)

  Wyszukuje element wg wartości klucza.
- AVLTreeNode< ContentType > * findMaxKeyNode (AVLTreeNode< ContentType > *tmpNode)

  Zwraca węzeł z minimalnym kluczem.
- AVLTreeNode< ContentType > * findAtherNodeMatch (AVLTreeNode< ContentType > *nodeComperator)

  Zwraca węzeł poprzednika.
- AVLTreeNode< ContentType > * remove (AVLTreeNode< ContentType > *x)

  Usuwa element x ze struktury AVL. Zwraca usunięty węzeł
- void print ()
- void recurringPrint (AVLTreeNode< ContentType > *x)
- void print (AVLTreeNode< ContentType > *x)

Atrybuty publiczne

- AVLTreeNode< ContentType > * rootNode

  korzeń drzewa

5.1.1 Opis szczegółowy

template<class ContentType>class AVLTree< ContentType >

Definicja w linii 24 pliku avltree.h.

5.1.2 Dokumentacja konstruktora i destruktora

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

Definicja w linii 33 pliku avltree.h.

```
00034     {
00035       rootNode = NULL;
00036     }
```

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

Definicja w linii 37 pliku avltree.h.

```
00038     {
00039       while(rootNode)
00040       {
00041           delete(remove(rootNode));
00042       }
00043     }
```

5.1.3 Dokumentacja funkcji składowych

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

Parametry

key	klucz do wyszukania

Definicja w linii 250 pliku avltree.h.

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

Odwołania w main().

```
00251     {
00252       AVLTreeNode<ContentType> * tmpNode = rootNode;
00253
00254       while((tmpNode) && (tmpNode->key != key))
00255       {
00256           if(key < tmpNode->key)          tmpNode = tmpNode->leftNode;
00257           else                            tmpNode = tmpNode->rightNode;
00258       }
00259
00260       return tmpNode;
00261     }
```

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

Definicja w linii 271 pliku avltree.h.

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

```
00272     {
```

```
00273       if(nodeComperator->leftNode) return findMaxKeyNode(nodeComperator->
      leftNode);
00274
00275       AVLTreeNode<ContentType> * y;
00276
00277       do
00278       {
00279         y = nodeComperator;
00280         nodeComperator = nodeComperator->parentNode;
00281       } while(nodeComperator && (nodeComperator->rightNode != y));
00282
00283       return nodeComperator;
00284     }
```

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

Definicja w linii 264 pliku avltree.h.

Odwołuje się do AVLTreeNode< ContentType >::rightNode.

```
00265     {
00266       while(tmpNode->rightNode) tmpNode = tmpNode->rightNode;
00267       return tmpNode;
00268     }
```

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

Definicja w linii 49 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() i main().

```
00050         {
00051         AVLTreeNode<ContentType>* newNode = new
      AVLTreeNode<ContentType>(newKey);
00052                 //AVLTreeNode<ContentType>  *newNode2 = new AVLTreeNode<ContentType>();
00053           AVLTreeNode<ContentType> * searchingNode =
      rootNode,       // Wskaznik do przeszukania drzewa i znalezienia tego samego klucza
00054                           * parentForNewNode = NULL,// parentForNewNode
00055                           * grandpaNode;
00056
00057           while(searchingNode)
00058           {
00059             if(searchingNode->key == newNode->key) // sprawdzam czy taki klucz juz istnieje
00060             {
00061               //delete n; // skoro istnieje to po co taki TODO: do zmiany
00062
00063               //cout    <<"Taki klucz juz istnieje !\n";
00064               return ;
00065             }
00066             parentForNewNode = searchingNode;
00067             if(newNode->key < searchingNode->key) searchingNode= searchingNode->
      leftNode; // przechodze w lewo czesc drzewa
00068             else searchingNode = searchingNode->rightNode;                          //
       przechodze w prawa czesc drzewa
00069           }
00070
00071
00072           // jezeli to jest pierwszy element to wpisuje go to root'a drzewa
00073           if(!(newNode->parentNode = parentForNewNode))
00074           {
00075             rootNode = newNode;
00076             return ;
00077           }
00078           // wybieram strone gałęzi na której ma byc element
00079           if(newNode->key < parentForNewNode->key) parentForNewNode->leftNode = newNode;
00080           else parentForNewNode->rightNode = newNode;
00081
00082           //sprawdzam czy potrzebne są rotacje, jak nie to koniec ;-)
00083           if(parentForNewNode->balanceFactor)
00084           {
00085             parentForNewNode->balanceFactor = 0;
00086             return ;
00087           }
```

```
00088
00089
00090           //parentForNewNode->balanceFactor = (parentForNewNode->leftNode == newNode) ? 1 : -1;
00091           if(parentForNewNode->leftNode == newNode) parentForNewNode->balanceFactor= 1;
00092           else  parentForNewNode->balanceFactor = -1;
00093           grandpaNode = parentForNewNode->parentNode;
00094
00095           // usatawiam balanceFactors na 1 przed dodaniem
00096           // nowej gałęzi oraz wyznaczam grandpaForNewNode od ktorego zaczynam rotacje
00097           while(grandpaNode)
00098           {
00099             if(grandpaNode->balanceFactor) break; // gdy byly juz wczesniej ustawione to przerwij
00100
00101             if(grandpaNode->leftNode == parentForNewNode) grandpaNode->balanceFactor = 1;
00102             else grandpaNode->balanceFactor = -1;
00103             parentForNewNode = grandpaNode; grandpaNode = grandpaNode->parentNode;
00104           }
00105
00106           // jesli do konca byly zbalansowane to przerwij
00107           if(!grandpaNode) return ;
00108
00109           //rotacje na podstawie balanceFactors
00110           if(grandpaNode->balanceFactor == 1)
00111           {
00112             if(grandpaNode->rightNode == parentForNewNode)
00113             {
00114               grandpaNode->balanceFactor = 0;
00115               return ;
00116             }
00117             if(parentForNewNode->balanceFactor == -1) rotationLR(grandpaNode); //Rotacja podwójna
      w lewo-prawo
00118             else rotationLL(grandpaNode); // Rotacja pojedyncza w prawo
00119             return ;
00120           }
00121           else
00122           {
00123             if(grandpaNode->leftNode == parentForNewNode)
00124             {
00125               grandpaNode->balanceFactor = 0;
00126               return ;
00127             }
00128             if(parentForNewNode->balanceFactor == 1) rotationRL(grandpaNode); //Rotacja podwójna
      w lewo-prawo
00129             else rotationRR(grandpaNode); //Rotacja pojedyncza w lewo
00130             return ;
00131           }
00132         }
```

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

Definicja w linii 378 pliku avltree.h.

```
00379     {
00380         this->recurringPrint(this->rootNode);
00381     }
```

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

Definicja w linii 389 pliku avltree.h.

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

```
00390     {
00391       cout << x->key << " : bf = " << setw(2) << x->balanceFactor;
00392       cout << endl;
00393       if(x->leftNode)  print(x->leftNode);
00394       if(x->rightNode) print(x->rightNode);
00395     }
```

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

Definicja w linii 382 pliku avltree.h.

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

```
00383     {
00384       cout<< "klucz:"<< x->key <<"\n";
00385       if(x->leftNode)  recurringPrint(x->leftNode);
00386       if(x->rightNode) recurringPrint(x->rightNode);
00387     }
```

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

Definicja w linii 289 pliku avltree.h.

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

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

```
00353               t = (y->leftNode == z) ? y->rightNode : y->
      leftNode;
00354
00355               //Wykonujemy odpowiednią rotację pojedynczą
00356               if(!(t->balanceFactor)) {
00357                 if(y->balanceFactor == 1) rotationLL(y); else
      rotationRR(y);
00358                 break;
00359               }
00360               //Wykonujemy odpowiednią rotację pojedynczą
00361               else if(y->balanceFactor == t->balanceFactor)
00362               {
00363                 if(y->balanceFactor == 1) rotationLL(y); else
      rotationRR(y);
00364                 z = t; y = t->parentNode;
00365               }
00366               //Wykonujemy rotację podwójną
00367               else
00368               {
00369                 if(y->balanceFactor == 1) rotationLR(y); else
      rotationRL(y);
00370                 z = y->parentNode; y = z->parentNode;
00371               }
00372             }
00373           }
00374         }
00375       }
00376       return x;
00377     }
```

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

Definicja w linii 166 pliku avltree.h.

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

```
00167     {
00168       AVLTreeNode<ContentType> * B = A->leftNode, * P = A->
      parentNode;
00169
00170       A->leftNode = B->rightNode;
00171       if(A->leftNode) A->leftNode->parentNode = A;
00172       B->rightNode = A;
00173       B->parentNode = P;
00174       A->parentNode = B;
00175       if(P)
00176       {
00177         if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00178       }
00179       else rootNode = B;
00180
00181       if(B->balanceFactor == 1)
00182       {
00183         A->balanceFactor = B->balanceFactor = 0;
00184       }
00185       else
00186       {
00187         A->balanceFactor = 1; B->balanceFactor = -1;
00188       }
00189
00190       return B;
00191     }
```

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

Definicja w linii 222 pliku avltree.h.

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

```
00223     {
00224       AVLTreeNode<ContentType> * B = A->leftNode, * C = B->
      rightNode, * P = A->parentNode;
00225
00226       B->rightNode = C->leftNode;
```

```
00227       if(B->rightNode) B->rightNode->parentNode = B;
00228       A->leftNode = C->rightNode;
00229       if(A->leftNode) A->leftNode->parentNode = A;
00230       C->rightNode = A;
00231       C->leftNode = B;
00232       A->parentNode = B->parentNode = C;
00233       C->parentNode = P;
00234       if(P)
00235       {
00236         if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00237       }
00238       else rootNode = C;
00239
00240       A->balanceFactor = (C->balanceFactor ==  1) ? -1 : 0;
00241       B->balanceFactor = (C->balanceFactor == -1) ?  1 : 0;
00242       C->balanceFactor = 0;
00243
00244       return C;
00245     }
```

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

Definicja w linii 195 pliku avltree.h.

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

```
00196     {
00197       AVLTreeNode<ContentType> * B = A->rightNode, * C = B->
      leftNode, * P = A->parentNode;
00198
00199       B->leftNode = C->rightNode;
00200       if(B->leftNode) B->leftNode->parentNode = B;
00201       A->rightNode = C->leftNode;
00202       if(A->rightNode) A->rightNode->parentNode = A;
00203       C->leftNode = A;
00204       C->rightNode = B;
00205       A->parentNode = B->parentNode = C;
00206       C->parentNode = P;
00207       if(P)
00208       {
00209         if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00210       }
00211       else rootNode = C;
00212
00213       A->balanceFactor = (C->balanceFactor == -1) ?  1 : 0;
00214       B->balanceFactor = (C->balanceFactor ==  1) ? -1 : 0;
00215       C->balanceFactor = 0;
00216
00217       return C;
00218     }
```

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

Definicja w linii 137 pliku avltree.h.

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

```
00138     {
00139       AVLTreeNode<ContentType> * B = A->rightNode, * P = A->
      parentNode;
00140
00141       A->rightNode = B->leftNode;
00142       if(A->rightNode) A->rightNode->parentNode = A;
00143       B->leftNode = A;
00144       B->parentNode = P;
00145       A->parentNode = B;
00146       if(P)
00147       {
00148         if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00149       }
00150       else rootNode = B;
00151
00152       if(B->balanceFactor == -1)
00153       {
```

```
00154         A->balanceFactor = B->balanceFactor = 0;
00155       }
00156       else
00157       {
00158         A->balanceFactor = -1; B->balanceFactor = 1;
00159       }
00160       return B;
00161     }
```

5.1.4 Dokumentacja atrybutów składowych

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

Definicja w linii 29 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 destruktora

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 >::print(), AVLTree< ContentType >::remove(), AVLTree< ContentType >::rotationLL(), AVLTree< ContentType >::rotationLR(), AVLTree< ContentType >::rotationRL() i AVLTree< ContentType >::rotationRR().

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 >::print() i AVLTree< ContentType >::recurringPrint().

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 >::print(), AVLTree< ContentType >::recurringPrint(), AVLTree< ContentType >::remove(), AVLTree< ContentType >::rotationLL(), AVLTree< ContentType >::rotationLR(), AVLTree< ContentType >::rotationRL() i AVLTree< ContentType >::rotationRR().

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() i AVLTree< ContentType >::rotationRR().

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 >::print(), AVLTree< ContentType >::recurringPrint(), AVLTree< ContentType >::remove(), AVLTree< ContentType >::rotationLL(), AVLTree< ContentType >::rotationLR(), AVLTree< ContentType >::rotationRL() i AVLTree< ContentType >::rotationRR().

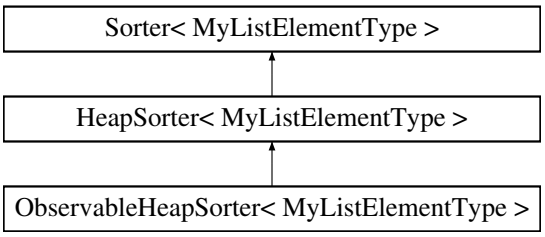
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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().

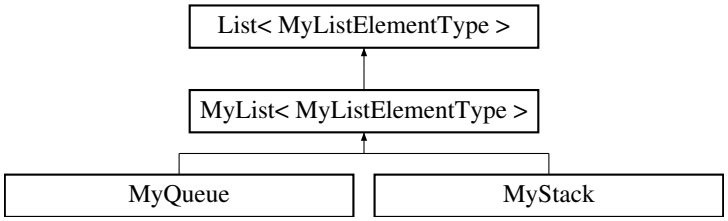
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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.

Odwołania 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 * >.

Odwołania 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 * >.

Odwołania 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().

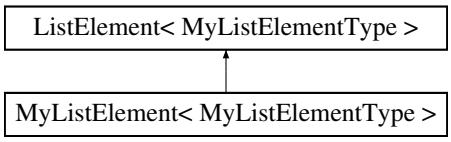
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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 powiodlo sie

Definicja w linii 27 pliku listsaver.h.

Odwołuje 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.

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

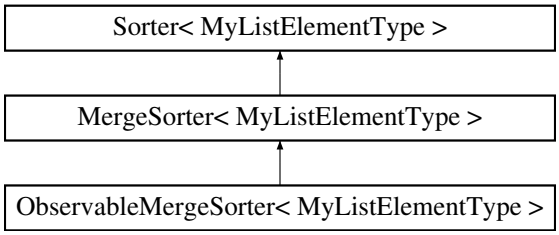
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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                         }
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         }
```

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

Parametry

m	Lista do posotrowania

Zwraca

zwraca posotrowana liste

Definicja w linii 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:

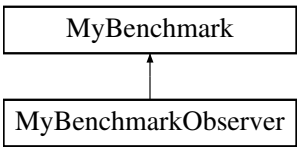
- mergesorter.h

5.8 Dokumentacja klasy MyBenchmark

Klasa bazowa/interface do testowania algorytmu.

`#include <mybenchmark.h>`

Diagram dziedziczenia dla MyBenchmark



Metody publiczne

- MyBenchmark ()
- void newL ()
- void tab ()
- void timerStart ()

  włączam stoper
- double timerStop ()

  wyłączam stoper
- double timerStopAndSaveToFile ()
- virtual ~MyBenchmark ()

  Usuwam obiekt test biorąc pod uwage jego prawdziwy typ.

Atrybuty publiczne

- double timerValue

  Interface metody algorytmu glownego.
- std::ofstream streamToFile
- double timerValueStatic

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 destruktora

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::newL ( )

Definicja w linii 33 pliku mybenchmark.cpp.

Odwołuje się do streamToFile.

Odwołania w main().

```
00034 {
00035         streamToFile<<"\n";
00036
00037 }
```

5.8.3.2 void MyBenchmark::tab ( )

Definicja w linii 28 pliku mybenchmark.cpp.

Odwołuje się do streamToFile.

Odwołania w main().

```
00029 {
00030         streamToFile<<" ";
00031
00032 }
```

5.8.3.3 void MyBenchmark::timerStart ( )

Definicja w linii 12 pliku mybenchmark.cpp.

Odwołuje się do timerValueStatic.

Odwołania w main() i MyBenchmarkObserver::receivedStartUpdate().

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

5.8.3.4 double MyBenchmark::timerStop ( )

Zwraca

Dlugosc dzialania stopera

Definicja w linii 17 pliku mybenchmark.cpp.

Odwołuje się do timerValueStatic.

Odwołania w main(), MyBenchmarkObserver::receivedStopUpdate(), MyBenchmarkObserver::receivedStopUpdateAndSaveToFile() i timerStopAndSaveToFile().

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

5.8.3.5 double MyBenchmark::timerStopAndSaveToFile ( )

Definicja w linii 22 pliku mybenchmark.cpp.

Odwołuje się do streamToFile i timerStop().

Odwołania w main().

```
00023 {
00024         double x= timerStop();
00025                         streamToFile<<x;
00026                         return x;
00027 }
```

5.8.4 Dokumentacja atrybutów składowych

5.8.4.1 std::ofstream MyBenchmark::streamToFile

Definicja w linii 41 pliku mybenchmark.h.

Odwołania w MyBenchmark(), newL(), MyBenchmarkObserver::receivedStopUpdateAndSaveToFile(), tab(), timerStopAndSaveToFile() i MyBenchmarkObserver::∼MyBenchmarkObserver().

5.8.4.2 double MyBenchmark::timerValue

Metoda abstrakcyjna, ktora jest interfacem do implementacji przez glowny algorytm. To znaczy, ze kazdy algorytm ma byc uruchamiany tą funkcjaCzas stopera

Definicja w linii 40 pliku mybenchmark.h.

Odwołania w MyBenchmarkObserver::getTimerValue(), MyBenchmark() i MyBenchmarkObserver::receivedStopUpdateAndSaveToFile().

5.8.4.3 double MyBenchmark::timerValueStatic

Definicja w linii 42 pliku mybenchmark.h.

Odwołania w timerStart() i timerStop().

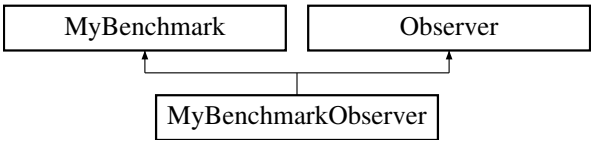
Dokumentacja dla tej klasy została wygenerowana z plików:

- 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 destruktora

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:

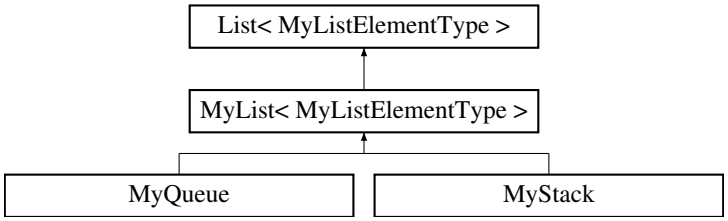
- mybenchmark.h

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

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

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

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

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

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

  Pokazuje element po koncu listy.
- void printList ()

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

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

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

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 powiodlo sie

Implementuje List< MyListElementType >.

Definicja w linii 171 pliku mylist.h.

```
00172         {
00173                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00174                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00175                         {
00176                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
      !";
00177                                 return *iterator;
00178                         }
00179                 if(isIteratorAfterPop)
00180                         {
00181                                 iteratorElementId=0;  // czyli iterator byl zpopowany
00182                                 iterator = firstElement;
00183                                 isIteratorAfterPop=0;
```

```
00184                           }
00185                   //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00186                   if((numberOfElement <= iteratorElementId-numberOfElement) &&(
      iteratorElementId-numberOfElement>=0))
00187                   {
00188                           //std::cerr<<"\nJestem w if_1";
00189                           iterator = (this->firstElement);
00190                           iteratorElementId = 0;
00191                           for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00192                                   iterator = (iterator->nextElement);
00193                   }
00194                   else if(numberOfElement > iteratorElementId)
00195                   {
00196                           //std::cerr<<"\nJestem w if_2";
00197                           for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00198                                                   iterator = (iterator->nextElement);
00199                   }
00200                   else if( numberOfElement < iteratorElementId)
00201                   {
00202                           //std::cerr<<"\nJestem w if_3";
00203                           for (; iteratorElementId> numberOfElement ;
      iteratorElementId--)
00204                                                   iterator = (iterator->previousElement);
00205                   }
00206                   return *iterator;
00207           }
```

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

Zwraca

Zwraca element ostatni w liscie

Implementuje List< MyListElementType >.

Definicja w linii 84 pliku mylist.h.

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 liscie

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
00105                 delete originMyListElement;
00106                 isIteratorAfterPop=1;
00107                 return tmpNumber;
00108         }
```

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 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:

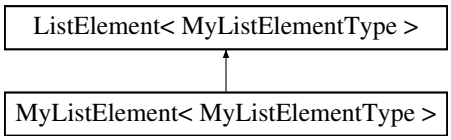
- mylist.h

5.11 Dokumentacja szablonu klasy MyListElement< MyListElementType >

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

`#include <mylistelement.h>`

Diagram dziedziczenia dla MyListElement< MyListElementType >



Metody publiczne

- MyListElement ()

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

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

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

Ustawia liczbe oraz klucz slowanika dla elementu.

Atrybuty publiczne

- MyListElement * nextElement

  Liczba przechowywana.
- MyListElement * previousElement

  wskaznik do poprzedniej 'malej struktury' w liscie

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

arg	liczba do zapisania w kolejnym elemencie listy
str	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

myListElement	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

arg	Liczba do zapisania
str	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 nastepnej 'malej struktury' w liscie

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:

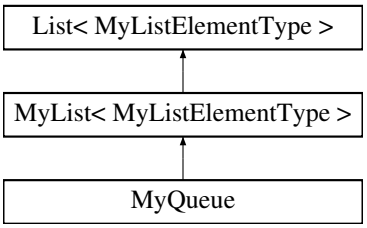
- mylistelement.h

5.12 Dokumentacja klasy MyQueue

Klasa reprezentuje kolejke.

`#include <myqueue.h>`

Diagram dziedziczenia dla MyQueue



Metody publiczne

- void push (int arg)
- int pop ()

  Wyciaga 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:

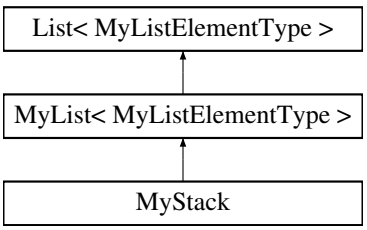
- myqueue.h

5.13 Dokumentacja klasy MyStack

Klasa reprezentuje stos.

`#include <mystack.h>`

Diagram dziedziczenia dla MyStack



Metody publiczne

- void push (int arg)
- int pop ()

  Wyciaga element ze stosu.

Dodatkowe Dziedziczone Składowe

5.13.1 Opis szczegółowy

Stos, którego index po pushu pokazuje na miejsce nastepne(nastepne 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 generujaca losowe liczby.

`#include <numbergenerator.h>`

Statyczne metody publiczne

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

5.14.1 Opis szczegółowy

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

Definicja w linii 27 pliku numbergenerator.h.

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

zakres	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

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

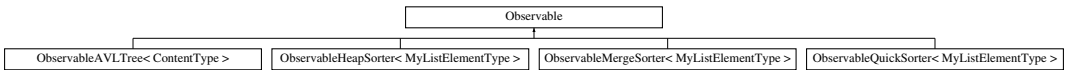
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 * > observaters

5.15.1 Opis szczegółowy

Definicja w linii 14 pliku observable.h.

5.15.2 Dokumentacja konstruktora i destruktora

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 observaters i MyList< MyListElementType >::push_back().

```
00019                             {
00020         observaters.push_back(o);
00021     }
```

5.15.3.2 void Observable::sendStartUpdateToObservers ( ) [inline]

Definicja w linii 23 pliku observable.h.

Odwołuje się do observaters 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<observaters.size(); i++)
00025         {
00026                 //std::cout<<"Wysylam start update";
00027                 observaters[i].content->receivedStartUpdate();
00028         }
00029     }
```

5.15.3.3 void Observable::sendStopUpdateToObservers ( ) [inline]

Definicja w linii 30 pliku observable.h.

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

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

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

5.15.3.4 void Observable::sendStopUpdateToObserversAndSaveToFile ( ) [inline]

Definicja w linii 34 pliku observable.h.

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

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

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

5.15.4 Dokumentacja atrybutów składowych

5.15.4.1 MyList<Observer*> Observable::observaters

Definicja w linii 17 pliku observable.h.

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

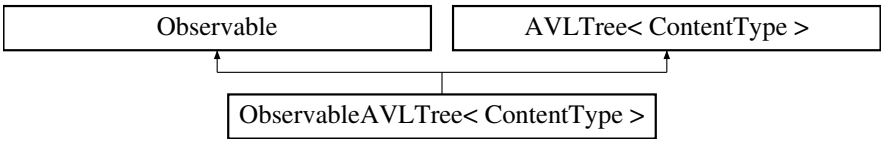
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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         }
```

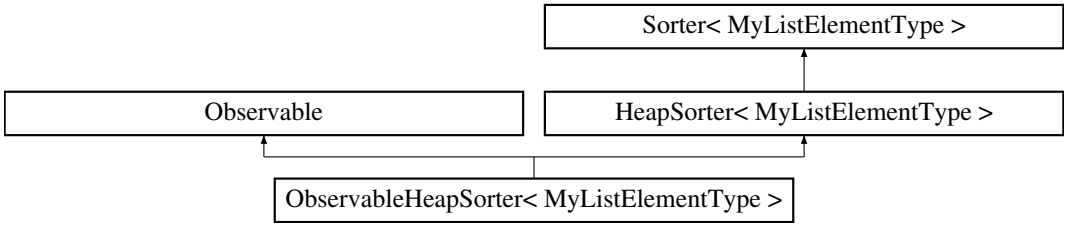
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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         }
```

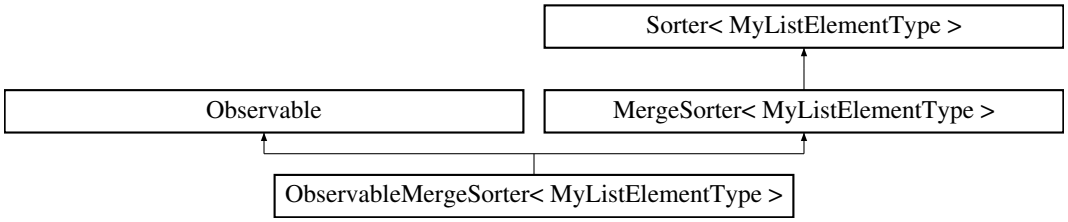
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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.

Odwołuje 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         }
```

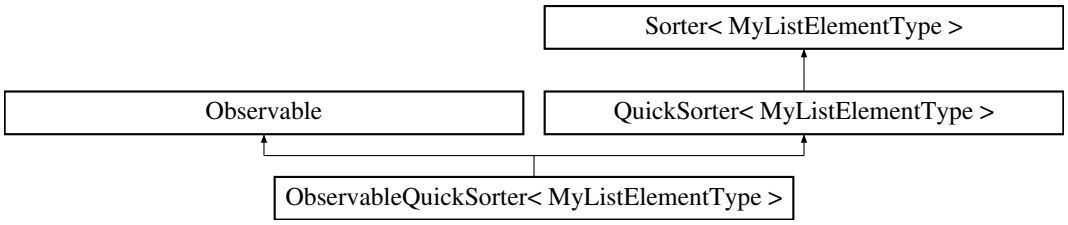
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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         }
```

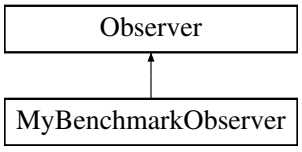
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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.

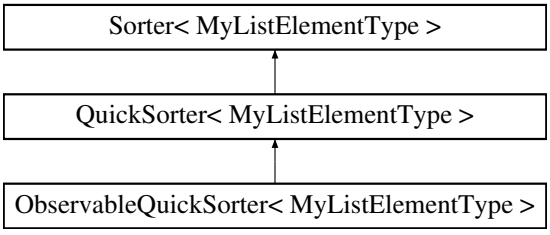
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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().

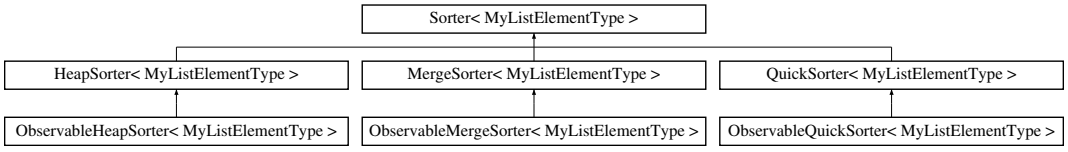
Dokumentacja dla tej klasy została wygenerowana z pliku:

- 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 destruktora

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 AVLTREE_H_
```

```
00009 #define AVLTREE_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 /*
00021  * @brief Definiuje obiekt drzewa AVL
00022  */
00023 template <class ContentType>
00024 class AVLTree
00025 {
00026   public:
00027
00029     AVLTreeNode<ContentType> * rootNode;
00030
00031
00032
00033     AVLTree()
00034     {
00035       rootNode = NULL;
00036     }
00037     ~AVLTree()
00038     {
00039       while(rootNode)
00040       {
00041           delete(remove(rootNode));
00042       }
00043     }
00044
00045     /*
00046      * @brief Wsadza klucz do drzewa
00047      * @param newkey klucz do wsadzenia
00048      */
00049     void insert(int &newKey)
00050         {
00051         AVLTreeNode<ContentType>* newNode = new
      AVLTreeNode<ContentType>(newKey);
00052                 //AVLTreeNode<ContentType>  *newNode2 = new AVLTreeNode<ContentType>();
00053           AVLTreeNode<ContentType> * searchingNode = rootNode,  // Wskaznik do
       przeszukania drzewa i znalezienia tego samego klucza
00054                           * parentForNewNode = NULL,// parentForNewNode
00055                           * grandpaNode;
00056
00057           while(searchingNode)
00058           {
00059             if(searchingNode->key == newNode->key) // sprawdzam czy taki klucz juz istnieje
00060             {
00061               //delete n; // skoro istnieje to po co taki TODO: do zmiany
00062
00063               //cout    <<"Taki klucz juz istnieje !\n";
00064               return ;
00065             }
00066             parentForNewNode = searchingNode;
00067             if(newNode->key < searchingNode->key) searchingNode= searchingNode->
      leftNode; // przechodze w lewo czesc drzewa
00068             else searchingNode = searchingNode->rightNode;                              //
       przechodze w prawa czesc drzewa
00069           }
00070
00071
00072           // jezeli to jest pierwszy element to wpisuje go to root'a drzewa
00073           if(!(newNode->parentNode = parentForNewNode))
00074           {
00075             rootNode = newNode;
00076             return ;
00077           }
00078           // wybieram strone gałęzi na której ma byc element
00079           if(newNode->key < parentForNewNode->key) parentForNewNode->leftNode = newNode;
00080           else parentForNewNode->rightNode = newNode;
00081
00082           //sprawdzam czy potrzebne są rotacje, jak nie to koniec ;-)
00083           if(parentForNewNode->balanceFactor)
00084           {
00085             parentForNewNode->balanceFactor = 0;
00086             return ;
00087           }
00088
00089
00090           //parentForNewNode->balanceFactor = (parentForNewNode->leftNode == newNode) ? 1 : -1;
00091           if(parentForNewNode->leftNode == newNode) parentForNewNode->balanceFactor= 1;
00092           else  parentForNewNode->balanceFactor = -1;
```

```
00093           grandpaNode = parentForNewNode->parentNode;
00094
00095           // usatawiam balanceFactors na 1 przed dodaniem
00096           // nowej gałęzi oraz wyznaczam grandpaForNewNode od ktorego zaczynam rotacje
00097           while(grandpaNode)
00098           {
00099             if(grandpaNode->balanceFactor) break; // gdy byly juz wczesniej ustawione to przerwij
00100
00101             if(grandpaNode->leftNode == parentForNewNode) grandpaNode->balanceFactor = 1;
00102             else grandpaNode->balanceFactor = -1;
00103             parentForNewNode = grandpaNode; grandpaNode = grandpaNode->parentNode;
00104           }
00105
00106           // jesli do konca byly zbalansowane to przerwij
00107           if(!grandpaNode) return ;
00108
00109           //rotacje na podstawie balanceFactors
00110           if(grandpaNode->balanceFactor == 1)
00111           {
00112             if(grandpaNode->rightNode == parentForNewNode)
00113             {
00114               grandpaNode->balanceFactor = 0;
00115               return ;
00116             }
00117             if(parentForNewNode->balanceFactor == -1) rotationLR(grandpaNode); //Rotacja podwójna w
     lewo-prawo
00118             else rotationLL(grandpaNode); // Rotacja pojedyncza w prawo
00119             return ;
00120           }
00121           else
00122           {
00123             if(grandpaNode->leftNode == parentForNewNode)
00124             {
00125               grandpaNode->balanceFactor = 0;
00126               return ;
00127             }
00128             if(parentForNewNode->balanceFactor == 1) rotationRL(grandpaNode); //Rotacja podwójna w
     lewo-prawo
00129             else rotationRR(grandpaNode); //Rotacja pojedyncza w lewo
00130             return ;
00131           }
00132         }
00133
00134     /*
00135      * @brief rotacja pojedyncza w lewo
00136      */
00137     AVLTreeNode<ContentType> * rotationRR(
    AVLTreeNode<ContentType> * A)
00138     {
00139       AVLTreeNode<ContentType> * B = A->rightNode, * P = A->
    parentNode;
00140
00141       A->rightNode = B->leftNode;
00142       if(A->rightNode) A->rightNode->parentNode = A;
00143       B->leftNode = A;
00144       B->parentNode = P;
00145       A->parentNode = B;
00146       if(P)
00147       {
00148         if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00149       }
00150       else rootNode = B;
00151
00152       if(B->balanceFactor == -1)
00153       {
00154         A->balanceFactor = B->balanceFactor = 0;
00155       }
00156       else
00157       {
00158         A->balanceFactor = -1; B->balanceFactor = 1;
00159       }
00160       return B;
00161     }
00162
00163     /*
00164      * @brief rotacja pojedyncza w prawo
00165      */
00166     AVLTreeNode<ContentType> * rotationLL(
    AVLTreeNode<ContentType> * A)
00167     {
00168       AVLTreeNode<ContentType> * B = A->leftNode, * P = A->
    parentNode;
00169
00170       A->leftNode = B->rightNode;
00171       if(A->leftNode) A->leftNode->parentNode = A;
00172       B->rightNode = A;
00173       B->parentNode = P;
```

```
00174       A->parentNode = B;
00175       if(P)
00176       {
00177         if(P->leftNode == A) P->leftNode = B; else P->rightNode = B;
00178       }
00179       else rootNode = B;
00180
00181       if(B->balanceFactor == 1)
00182       {
00183         A->balanceFactor = B->balanceFactor = 0;
00184       }
00185       else
00186       {
00187         A->balanceFactor = 1; B->balanceFactor = -1;
00188       }
00189
00190       return B;
00191     }
00192     /*
00193      * @brief rotacja podwojna lewo prawo
00194      */
00195     AVLTreeNode<ContentType> * rotationRL(
      AVLTreeNode<ContentType> * A)
00196     {
00197       AVLTreeNode<ContentType> * B = A->rightNode, * C = B->
      leftNode, * P = A->parentNode;
00198
00199       B->leftNode = C->rightNode;
00200       if(B->leftNode) B->leftNode->parentNode = B;
00201       A->rightNode = C->leftNode;
00202       if(A->rightNode) A->rightNode->parentNode = A;
00203       C->leftNode = A;
00204       C->rightNode = B;
00205       A->parentNode = B->parentNode = C;
00206       C->parentNode = P;
00207       if(P)
00208       {
00209         if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00210       }
00211       else rootNode = C;
00212
00213       A->balanceFactor = (C->balanceFactor == -1) ?  1 : 0;
00214       B->balanceFactor = (C->balanceFactor ==  1) ? -1 : 0;
00215       C->balanceFactor = 0;
00216
00217       return C;
00218     }
00219     /*
00220      * @brief rotacja podwojna lewo prawo
00221      */
00222     AVLTreeNode<ContentType> * rotationLR(
      AVLTreeNode<ContentType> * A)
00223     {
00224       AVLTreeNode<ContentType> * B = A->leftNode, * C = B->
      rightNode, * P = A->parentNode;
00225
00226       B->rightNode = C->leftNode;
00227       if(B->rightNode) B->rightNode->parentNode = B;
00228       A->leftNode = C->rightNode;
00229       if(A->leftNode) A->leftNode->parentNode = A;
00230       C->rightNode = A;
00231       C->leftNode = B;
00232       A->parentNode = B->parentNode = C;
00233       C->parentNode = P;
00234       if(P)
00235       {
00236         if(P->leftNode == A) P->leftNode = C; else P->rightNode = C;
00237       }
00238       else rootNode = C;
00239
00240       A->balanceFactor = (C->balanceFactor ==  1) ? -1 : 0;
00241       B->balanceFactor = (C->balanceFactor == -1) ?  1 : 0;
00242       C->balanceFactor = 0;
00243
00244       return C;
00245     }
00246
00249
00250     AVLTreeNode<ContentType> * find(int key)
00251     {
00252       AVLTreeNode<ContentType> * tmpNode = rootNode;
00253
00254       while((tmpNode) && (tmpNode->key != key))
00255       {
00256           if(key < tmpNode->key)         tmpNode = tmpNode->leftNode;
00257           else                           tmpNode = tmpNode->rightNode;
00258       }
```

```
00259
00260       return tmpNode;
00261     }
00263
00264     AVLTreeNode<ContentType> * findMaxKeyNode(
      AVLTreeNode<ContentType> * tmpNode)
00265     {
00266       while(tmpNode->rightNode) tmpNode = tmpNode->rightNode;
00267       return tmpNode;
00268     }
00269
00271     AVLTreeNode<ContentType> * findAtherNodeMatch(
      AVLTreeNode<ContentType> * nodeComperator)
00272     {
00273       if(nodeComperator->leftNode) return findMaxKeyNode(nodeComperator->
      leftNode);
00274
00275       AVLTreeNode<ContentType> * y;
00276
00277       do
00278       {
00279         y = nodeComperator;
00280         nodeComperator = nodeComperator->parentNode;
00281       } while(nodeComperator && (nodeComperator->rightNode != y));
00282
00283       return nodeComperator;
00284     }
00285
00289     AVLTreeNode<ContentType> * remove(
      AVLTreeNode<ContentType> * x)
00290     {
00291       AVLTreeNode<ContentType> * t, * y, * z;
00292       bool nest;
00293
00294       // Jeśli węzeł x posiada dwójkę dzieci, lewego i prawego potomka:
00295       if((x->leftNode) && (x->rightNode))
00296       {
00297         y = remove(findAtherNodeMatch(x));
00298         //rekurencyjnie usuwamy y za pomocą tego samego algorytmu
00299         nest = false;
00300       }
00301       //Jeśli węzeł x posiada tylko jedno dziecko lub nie posiada wcale dzieci:
00302       else  {
00303         if(x->leftNode) {
00304           y = x->leftNode;
00305           x->leftNode = NULL;
00306         }
00307         else {
00308           y = x->rightNode; x->rightNode = NULL;
00309         }
00310         x->balanceFactor = 0;
00311         nest = true;
00312       }
00313
00314       if(y)  {
00315         y->parentNode = x->parentNode;
00316         if(x->leftNode)
00317                 {
00318                         y->leftNode  = x->leftNode;
00319                         y->leftNode->parentNode  = y;
00320                 }
00321         if(x->rightNode)
00322                 {
00323                         y->rightNode = x->rightNode;
00324                         y->rightNode->parentNode = y;
00325                 }
00326         y->balanceFactor = x->balanceFactor;
00327       }
00328
00329       if(x->parentNode)   {
00330         if(x->parentNode->leftNode == x) x->parentNode->leftNode = y; else x->
      parentNode->rightNode = y;
00331       }
00332       else rootNode = y;
00333
00334       if(nest)  {
00335         z = y;
00336         y = x->parentNode;
00337         while(y)
00338         {
00339                 // węzeł y był w stanie równowagi przed usunięciem węzła x w jednym z jego poddrzew.
00340           if(!(y->balanceFactor)) {
00341             y->balanceFactor = (y->leftNode == z) ? -1 : 1;
00342             break;
00343           }
00344           else {
00345                 //skrócone zostało cięższe poddrzewo
```

```
00346             if          (((y->balanceFactor ==  1) &&
00347                          (y->leftNode  == z)) || ((y->balanceFactor == -1) &&
00348                          (y->rightNode == z))) {
00349               y->balanceFactor = 0;
00350               z = y; y = y->parentNode;
00351             }
00352             else {
00353               t = (y->leftNode == z) ? y->rightNode : y->
    leftNode;
00354
00355               //Wykonujemy odpowiednią rotację pojedynczą
00356               if(!(t->balanceFactor)) {
00357                 if(y->balanceFactor == 1) rotationLL(y); else rotationRR(y);
00358                 break;
00359               }
00360               //Wykonujemy odpowiednią rotację pojedynczą
00361               else if(y->balanceFactor == t->balanceFactor)
00362               {
00363                 if(y->balanceFactor == 1) rotationLL(y); else rotationRR(y);
00364                 z = t; y = t->parentNode;
00365               }
00366               //Wykonujemy rotację podwójną
00367               else
00368               {
00369                 if(y->balanceFactor == 1) rotationLR(y); else rotationRL(y);
00370                 z = y->parentNode; y = z->parentNode;
00371               }
00372             }
00373           }
00374         }
00375       }
00376       return x;
00377     }
00378     void      print()
00379     {
00380         this->recurringPrint(this->rootNode);
00381     }
00382     void recurringPrint(AVLTreeNode<ContentType> * x)
00383     {
00384       cout<< "klucz:"<< x->key <<"\n";
00385       if(x->leftNode)  recurringPrint(x->leftNode);
00386       if(x->rightNode) recurringPrint(x->rightNode);
00387     }
00388
00389     void print(AVLTreeNode<ContentType> * x)
00390     {
00391       cout << x->key << " : bf = " << setw(2) << x->balanceFactor;
00392       cout << endl;
00393       if(x->leftNode)  print(x->leftNode);
00394       if(x->rightNode) print(x->rightNode);
00395     }
00396 };
00397
00398
00399
00400
00401 #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
00018         // Gałąź rodzica, lewa podgałąć, prawa podgałąź
00019         AVLTreeNode * parentNode,
00020                                 * leftNode,
00021                                 * rightNode;
00022
00023   //klucz
00024   int key, balanceFactor;
00025   ContentType content;
00026
00027    AVLTreeNode()
00028   {
00029           parentNode = leftNode = rightNode = NULL;
00030           key = balanceFactor = 0;
00031   }
00032    AVLTreeNode(int newKey)
00033    {
00034           parentNode = leftNode = rightNode = NULL;
00035           balanceFactor = 0;
00036                key = newKey;
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
      pop_back()      =       0;
00021         ListElement<MyListElementType> virtual
      pop_front()             =       0;
00022         void virtual printList()        =       0;
00023         void virtual push_back(MyListElementType arg)   =       0;
00024         void virtual push_front(MyListElementType arg)          =       0;
00025         MyListElement<MyListElementType> virtual &
      operator[](int numberOfElement)                 =       0;
00026         void virtual insertAfter(MyListElement<MyListElementType>
       arg, int iteratorID)    =       0;
00027         MyListElementType virtual &show_front()         =       0;
00028         MyListElementType virtual &show_back() =        0;
00029         //List<MyListElementType> virtual &operator=(const List<MyListElementType> &pattern)    =       0;
00030
00031         void virtual cloneFrom(List<MyListElementType> &patternList)
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         }
00038         List<MyListElementType> virtual &
      createObjectFromAbstractReference() = 0;
00039
00040         void virtual free(){ while(size()) pop_back(); }
00041         virtual ~List(){};
00042 };
00043
00044
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 1000000000

Funkcje

- int main (int argc, char ∗argv[])

6.15.1 Dokumentacja definicji

6.15.1.1 #define ILE_WYRAZOW_DO_WSTAWIENIA 1000000000

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 18 pliku main.cpp.

Odwołuje się do AVLTree< ContentType >::find(), ILE_WYRAZOW_DO_WSTAWIENIA, AVLTree< ContentType >::insert(), MyBenchmark::newL(), MyBenchmark::tab(), MyBenchmark::timerStart(), MyBenchmark::timerStop() i MyBenchmark::timerStopAndSaveToFile().

```
00019 {
00020
00021         MyList<int> lista;
00022         MyBenchmark benchmark;
00023         benchmark.timerStart();
00024         cout<<"Generuje "<<ILE_WYRAZOW_DO_WSTAWIENIA<<" losowych liczb. Prosze o
      cierpliwosc... \n";
00025         lista = NumberGenerator::generateNumbers<int>(100000, ILE_WYRAZOW_DO_WSTAWIENIA/10);
00026         cout<<"Wygenerowalem losowe liczby w "<<benchmark.timerStop()<<" sekund\n";
00027         AVLTree<int> tree;
00028         int x=0;
00029
00031         for(int j=10; j<ILE_WYRAZOW_DO_WSTAWIENIA; j=j*10)
00032                 {
00033
00034                 benchmark.timerStart();
00035                         for(int i=0; i<j-x; i++)
```

```
00036                           {
00037                                   tree.insert(lista[i].content);
00038                           }
00039                           cout<<j<<" "<<benchmark.timerStopAndSaveToFile()<<" ";
00040                           benchmark.tab();
00041                           //szukanie
00042                   benchmark.timerStart();
00043                           for(int i=0; i<j-x; i++)
00044                           {
00045                                   tree.find(lista[i].content);
00046                           }
00047                           cout<<benchmark.timerStopAndSaveToFile()<<endl;
00048                           benchmark.newL();
00049                           //szukanie end
00050                           x+=j;
00051                   }
00052 }
```

6.16 main.cpp

```
00001 /*
00002  * main.cpp
00003  *
00004  *  Created on: Mar 6, 2015
00005  *      Author: serek8
00006  */
00008 #include <iostream>
00009 #include <unistd.h>
00010 #include "numbergenerator.h"
00011 #include "observableavltree.h"
00012 #include "mybenchmark.h"
00013 #include "mylist.h"
00014
00015 #define ILE_WYRAZOW_DO_WSTAWIENIA 1000000000    // w razie potrzeby prosze zmniejszyc o jedno zero
00016                                                                                                 // wtedy
       nalezy czekac ok 22sek
00017
00018 int main(int argc, char *argv[])
00019 {
00020
00021         MyList<int> lista;
00022         MyBenchmark benchmark;
00023         benchmark.timerStart();
00024         cout<<"Generuje "<<ILE_WYRAZOW_DO_WSTAWIENIA<<" losowych liczb. Prosze o
       cierpliwosc... \n";
00025         lista = NumberGenerator::generateNumbers<int>(100000, ILE_WYRAZOW_DO_WSTAWIENIA/10);
00026         cout<<"Wygenerowalem losowe liczby w "<<benchmark.timerStop()<<" sekund\n";
00027         AVLTree<int> tree;
00028         int x=0;
00029
00031         for(int j=10; j<ILE_WYRAZOW_DO_WSTAWIENIA; j=j*10)
00032                 {
00033
00034                 benchmark.timerStart();
00035                         for(int i=0; i<j-x; i++)
00036                         {
00037                                 tree.insert(lista[i].content);
00038                         }
00039                         cout<<j<<" "<<benchmark.timerStopAndSaveToFile()<<" ";
00040                         benchmark.tab();
00041                         //szukanie
00042                 benchmark.timerStart();
00043                         for(int i=0; i<j-x; i++)
00044                         {
00045                                 tree.find(lista[i].content);
00046                         }
00047                         cout<<benchmark.timerStopAndSaveToFile()<<endl;
00048                         benchmark.newL();
00049                         //szukanie end
00050                         x+=j;
00051                 }
00052 }
```

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(
      MyList<MyListElementType> left,
      MyList<MyListElementType> right)
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                         }
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 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         }
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
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 }
00021
00022 double MyBenchmark :: timerStopAndSaveToFile()
00023 {
00024         double x= timerStop();
00025                         streamToFile<<x;
00026                         return x;
00027 }
00028 void MyBenchmark :: tab()
00029 {
00030         streamToFile<<" ";
00031
00032 }
00033 void MyBenchmark :: newL()
00034 {
00035         streamToFile<<"\n";
00036
00037 }
```

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         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         void newL();
00052         void tab();
00053
00055         void  timerStart();
00056
00061         double  timerStop();
00062         double  timerStopAndSaveToFile();
00066         virtual ~MyBenchmark() {};
00067         //using DataFrame::operator=;
00068
00069 };
00070
00071
00072 class MyBenchmarkObserver : public MyBenchmark, public
      Observer
00073 {
00074 public:
00075         MyBenchmarkObserver(){};
00076         double getTimerValue() {return this->timerValue;}
00077         void receivedStartUpdate () {
00078                 //std::cout<<"\nWlaczam stoper...";
00079                 timerStart();
00080         }
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
00066         int &size()
00067         {
00068                 return sizeOfList;
00069         }
00074         /*MyListElement<MyListElementType> &pop_back()
00075         {
00076                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new MyListElement<MyListElementType>)); }
00077                 MyListElement<MyListElementType> tmpNumber = *(this -> lastElement);
00078                 MyListElement<MyListElementType> *originMyListElement = this -> lastElement;
00079                 this -> lastElement = this -> lastElement -> previousElement;
00080                 delete originMyListElement;
00081                 isIteratorAfterPop=1;
00082                 return tmpNumber;
00083         }*/
00084         ListElement<MyListElementType> pop_back()
00085         {
00086                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new
      MyListElement<MyListElementType>)); }
00087                 MyListElement<MyListElementType> tmpNumber = *(this ->
      lastElement);
00088                 MyListElement<MyListElementType> *originMyListElement =
      this -> lastElement;
00089                 this -> lastElement = this -> lastElement -> previousElement;
00090                 delete originMyListElement;
00091                 isIteratorAfterPop=1;
00092                 return tmpNumber;
00093         }
00098         ListElement<MyListElementType> pop_front()
00099         {
00100                 if(!(sizeOfList--)) { sizeOfList=0; return (*(new
      MyListElement<MyListElementType>())); }
00101                 MyListElement<MyListElementType> tmpNumber = *(this ->
      firstElement);
00102                 MyListElement<MyListElementType> *originMyListElement =
      this -> firstElement;
00103                 this -> firstElement = this -> firstElement -> nextElement;
00104
00105                 delete originMyListElement;
00106                 isIteratorAfterPop=1;
00107                 return tmpNumber;
00108         }
00112         void push_back(MyListElementType arg)
00113         {
00114                 //std::cerr<<"\n(push_back): arg.content="<<arg.content;
00115                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00116                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00117                 //newMyListElement -> nextElement = 0;
00118                 newMyListElement -> previousElement = this -> lastElement;
00119                 this -> lastElement -> nextElement = newMyListElement;
00120                 this->lastElement = newMyListElement;
00121         }
00125         void push_front(MyListElementType arg)
00126         {
00127                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00128                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00129                 //newMyListElement -> previousElement =  0;
00130                 newMyListElement -> nextElement = this -> firstElement;
00131                 this -> firstElement -> previousElement = newMyListElement;
00132                 this->firstElement = newMyListElement;
00133                 ++iteratorElementId;
00134         }
00139         MyListElementType &show_front()
00140         {
00141                 return firstElement->content;
00142         }
00147         MyListElementType &show_back()
00148         {
00149                 return lastElement->content;
00150         }
00151
00152
00156         void  printList()
00157         {
00158                 MyListElement<MyListElementType> *elem = (this->
      firstElement);
00159                 std::cout<<"\nWyswietlam liste (size:"<<this->sizeOfList<<"): ";
00160                 for(int i=0; i< this->sizeOfList; i++)
00161                 {
00162                         std::cout<<" "<<elem->content;
00163                         elem = elem->nextElement;
00164                 }
00165         }
```

```
00166
00171         MyListElement<MyListElementType> &
      operator[](int numberOfElement)
00172         {
00173                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00174                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00175                         {
00176                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
       !";
00177                                 return *iterator;
00178                         }
00179                 if(isIteratorAfterPop)
00180                         {
00181                                 iteratorElementId=0;  // czyli iterator byl zpopowany
00182                                 iterator = firstElement;
00183                                 isIteratorAfterPop=0;
00184                         }
00185                 //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00186                 if((numberOfElement <= iteratorElementId-numberOfElement) &&(
      iteratorElementId-numberOfElement>=0))
00187                 {
00188                         //std::cerr<<"\nJestem w if_1";
00189                         iterator = (this->firstElement);
00190                         iteratorElementId = 0;
00191                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00192                                 iterator = (iterator->nextElement);
00193                 }
00194                 else if(numberOfElement > iteratorElementId)
00195                 {
00196                         //std::cerr<<"\nJestem w if_2";
00197                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00198                                                 iterator = (iterator->nextElement);
00199                 }
00200                 else if( numberOfElement < iteratorElementId)
00201                 {
00202                         //std::cerr<<"\nJestem w if_3";
00203                         for (; iteratorElementId> numberOfElement ;
      iteratorElementId--)
00204                                                 iterator = (iterator->previousElement);
00205                 }
00206                 return *iterator;
00207         }
00208
00212         void insertAfter(MyListElement<MyListElementType> arg,
      int iteratorID)
00213         {
00214                 if(iteratorID==0 && this->sizeOfList==0)  {push_front(arg.
      content); return;}
00215                 if(iteratorID==this->sizeOfList-1)  {push_back(arg.
      content); return;}
00216                 MyListElement<MyListElementType> *newMyListElement = new
      MyListElement<MyListElementType>(arg);
00217                 MyListElement<MyListElementType> &tmpThis=(*this)[
      iteratorID], &tmpNext=(*this)[iteratorID+1];
00218                 if(!sizeOfList++) {firstElement =
      lastElement = newMyListElement;}
00219                 newMyListElement -> nextElement = tmpThis.nextElement;
00220                 newMyListElement -> previousElement = &tmpThis;
00221                 tmpThis.nextElement = newMyListElement;
00222                 tmpNext.previousElement = newMyListElement;
00223                 isIteratorAfterPop=1;
00224         }
00225
00226
00227         //MyListElement operator[](int numberOfElement);
00228         //virtual MyList<MyListElementType> sort()
00229         //{
00230         //      std::cerr<<"\nError: Sortowanie z klasy MyList !!!";
00231         //      //return m;
00232         //}
00233
00234         MyList<MyListElementType> &operator=(const
      MyList<MyListElementType> &pattern)
00235         {
00236                 //std::cerr<<" @@@";
00237                 this->sizeOfList = pattern.sizeOfList;
00238                 this->firstElement = pattern.firstElement;
00239                 this->lastElement = pattern.lastElement;
00240                 this->iterator=pattern.iterator;
00241                 this->isIteratorAfterPop = pattern.
      isIteratorAfterPop;
00242                 return *this;
00243         }
00244 //      List<MyListElementType> &operator=(const List<MyListElementType> &pattern)
00245 //      {
```

```
00246 //              std::cerr<<" ###";
00252 //              //this->cloneFrom(pattern);
00253 //              return *this;
00254 //      }
00255
00256 /*      void cloneFrom(MyList<MyListElementType> patternList)
00257         {
00258                 MyList<MyListElementType> &clonedList = *new MyList<MyListElementType>;
00259                 // release memory from main list
00260                 while(this->size()) pop_back();
00261                 for(int i=0; i<patternList.size(); i++)
00262                         clonedList.push_back(patternList[i]);
00263                 *this = clonedList;
00264         }
00265 */
00266
00267         List<MyListElementType> &
      createObjectFromAbstractReference(/*MyList<MyListElementType>
       abstractPattern*/)
00268         {
00269                 return *new MyList<MyListElementType>;
00270         }
00271
00272
00273
00274 };
00276
00278
00279
00280
00281 /*class MyListObserved : public MyList, public Observed
00282 {
00283 public:
00284         void mergeSort(MyList m)
00285         {
00286         MyList::mergeSort(m);
00287         powiadom();
00288
00289         }
00290         MyListObserved(){};
00291         ~MyListObserved(){};
00292
00293
00294 };*/
00295
00296 #endif /* MYLIST_H_ */
```

6.25 Dokumentacja pliku mylistelement.h

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

Komponenty

- class MyListElement< MyListElementType >

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

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.
    previousElement;
00056                 //std::cerr<<"\n(konstruktor kopiujacy MyListElement): content="<<content;
00057         }
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 kolejke.

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 generujaca 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
     quantity)
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 }
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> observaterswww;
00017         MyList<Observer*> observaters;
00018
00019     void add(Observer *o) {
00020         observaters.push_back(o);
00021     }
00022
00023     void sendStartUpdateToObservers () {
00024         for(int i=0; i<observaters.size(); i++)
00025         {
00026                 //std::cout<<"Wysylam start update";
00027                 observaters[i].content->receivedStartUpdate();
00028         }
00029     }
00030     void sendStopUpdateToObservers () {
00031         for(int i=0; i<observaters.size(); i++)
00032                 observaters[i].content->receivedStopUpdate();
00033     }
00034     void sendStopUpdateToObserversAndSaveToFile () {
00035         for(int i=0; i<observaters.size(); i++)
00036                 observaters[i].content->receivedStopUpdateAndSaveToFile();
00037     }
00038
00039
00040         virtual ~Observable(){}
00041
00042
00043
00044 };
00045
00046 #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):
00020                 HeapSorter<MyListElementType>::HeapSorter(myList){}
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>
00018 template <class MyListElementType>
00019 class QuickSorter : public Sorter<MyListElementType>
00020 {
00021 public:
00022         int enablePivot;
00023         List<MyListElementType> &list;
00024
00025
00026
00027         QuickSorter(List<MyListElementType> &
      list)
00028         :list(list.createObjectFromAbstractReference())
00029         {
00030                 this->list.cloneFrom(list);
00031                 this->enablePivot=1;
00032         }
00033
00034         virtual ~QuickSorter(){};
00035         //void quicksort(int lewy, int prawy, int enablePivot)
00036         void quicksort(int lewy, int prawy)
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         }
00060
00061         List<MyListElementType> &sort()
00062         {
00063                 //std::cout<<"(QuickSort)";
00064                 quicksort(0, list.size()-1);
00065                 return list;
00066         }
00067 };
00068
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