Laboratorium 9

Wygenerowano przez Doxygen 1.8.6

So, 30 maj 2015 11:01:28

Spis treści

1 Indeks hierarchiczny

1.1 Hierarchia klas

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

2 Indeks klas

2.1 Lista klas

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

3 Indeks plików

3.1 Lista plików

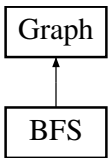
Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

4 Dokumentacja klas

4.1 Dokumentacja klasy BFS

`#include <bfs.h>`

Diagram dziedziczenia dla BFS



Metody publiczne

- BFS ()

  Reprezentuje kolejnke elementów po wizycie.
- virtual ~BFS ()
- void bfs (int fromNumber, int toNumber)
- void findShortestPath (int fromNumber, int toNumber)
- void clean ()

  Zwalniam zasoby i przygotowuje do nowego rozruchu.

Atrybuty publiczne

- int pi [MAX]
- LinkedList< int > q

  rodzic nadego elementu

4.1.1 Opis szczegółowy

Definicja w linii 18 pliku bfs.h.

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 BFS::BFS ( ) [inline]

Definicja w linii 24 pliku bfs.h.

```
00025     {
00026                 clean();
00027     }
```

4.1.2.2 virtual BFS::~BFS ( ) [inline], [virtual]

Definicja w linii 28 pliku bfs.h.

```
00028 {         }
```

4.1.3 Dokumentacja funkcji składowych

4.1.3.1 void BFS::bfs (int *fromNumber,* int *toNumber*) [inline]

Definicja w linii 30 pliku bfs.h.

```
00031     {
00032         colour[fromNumber] = 1;
00033         pi[fromNumber]     = 1; // czyli brak rodzica
00034         q.push_back( fromNumber );
00035         while ( q.size() ) {
00036                 int u = q[0];
00037                 q.pop_front();
00038                 for( int v = 0; v < biggestValue; v++ ) {
00039                         //cout<<"(test) colour:"<<colour[v]<<"\t"<<M[u][v]<<endl;
00040                         if( colour[v] == 0 && M[u][v] ) {
00041                                 colour[v] = 1;
00042                                 pi[v] = u;          // zapisuje gdzie znajduje sie rodzic danej liczby
00043                                 if(toNumber == v) return;
00044                                 q.push_back( v );
00045                         }
00046                 }
00047                 colour[u] = 2;
00048         }
00049     }
```

4.1.3.2 void BFS::clean ( ) [inline],[virtual]

Implementuje Graph.

Definicja w linii 68 pliku bfs.h.

Odwołuje się do MAX.

```
00069     {
00070         num = 0;
00071         biggestValue = 0;
00072         for (int i = 0; i < MAX; i++)
00073         {
00074                 for (int j = 0; j < MAX; j++)         M[i][j] = 0;
00075
00076                 colour[i] = 0;     // ustawia numery jako biale
00077                 pi[i] = 1;    // ustawia rodzicow na 1
00078
00079         }
00080     }
```

4.1.3.3 void BFS::findShortestPath (int *fromNumber,* int *toNumber*) [inline],[virtual]

najkrótszą drogę i wypusuje ją na ekran

Parametry

fromNumber	star poszukiwan
endNumber	Koniec poszukiwan

Implementuje Graph.

Definicja w linii 51 pliku bfs.h.

Odwołuje się do LinkedList< ContentType >::print() i LinkedList< ContentType >::push_front().

```
00052     {
00053         LinkedList<int> listaDoZmianyKolejnosci;
00054         int tmp=toNumber;
00055         bfs(fromNumber, toNumber);
00056         listaDoZmianyKolejnosci.push_front(toNumber);
00057         while(1)
00058         {
00059                 listaDoZmianyKolejnosci.push_front(pi[tmp]);
00060                 tmp = pi[tmp];
00061                 if(tmp == fromNumber) break;
00062         }
00063         listaDoZmianyKolejnosci.print();
00064     }
```

4.1.4 Dokumentacja atrybutów składowych

4.1.4.1 int BFS::pi[MAX]

Definicja w linii 21 pliku bfs.h.

4.1.4.2 LinkedList<int> BFS::q

Definicja w linii 22 pliku bfs.h.

Dokumentacja dla tej klasy została wygenerowana z pliku:

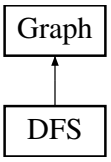
- bfs.h

4.2 Dokumentacja klasy DFS

Klasa przedstawia przeszukiwanie grafu za pomocą DFS Umożliwia wyszukanie najkrótszej trasy do numeru.

`#include <dfs.h>`

Diagram dziedziczenia dla DFS



Metody publiczne

- DFS ()

  wszystkich sciezek
- ~DFS ()
- void findPath (int u, int p, int searchingValue)

  Znajduje wszystkie mozliwe sciezki do liczby.
- void findShortestPath (int fromNumber, int toNumber)
- void clean ()

  Zwalniam zasoby i przygotowuje do nowego rozruchu.

Atrybuty publiczne

- LinkedList< int > currentPath
- LinkedList< int > allPaths [MAX]

  znaleziona sciezka
- int allPathsAmount

  wszystkich ścieżek do liczby

4.2.1 Opis szczegółowy

Definicja w linii 22 pliku dfs.h.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 DFS::DFS ( ) [inline]

Definicja w linii 29 pliku dfs.h.

```
00030   {
00031                   clean();
00032   }
```

4.2.2.2 DFS::~DFS ( ) [inline]

Definicja w linii 33 pliku dfs.h.

```
00033 {        }
```

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 void DFS::clean ( ) [inline], [virtual]

Implementuje Graph.

Definicja w linii 96 pliku dfs.h.

Odwołuje się do MAX.

```
00096                       {
00097                   //n =0;
00098                   biggestValue = 0;
00099                   for (int i = 0; i < MAX; i++)
00100                           for (int j = 0; j < MAX; j++)
00101                                   M[i][j] = 0;
00102                   for (int i = 0; i < MAX; i++) {
00103                           colour[i] = 0;
00104                   }
00105                   num = 0;
00106                   allPathsAmount = 0;
00107           }
```

4.2.3.2 void DFS::findPath (int *u*, int *p*, int *searchingValue*) [inline]

Parametry

u	Od tej liczby zaczynam przeszukiwanie
p	Liczba rodzica

Definicja w linii 39 pliku dfs.h.

```
00039                                                   {
00040           colour[u] = 1;
00041           //cout<<u<<" zmieniam flage na szara\n";
00042           currentPath.push_back(u);
00043           if(u == searchingValue)
00044                   {
00045                           // Znalazlem TĄ LICZBE !!!
00046                           // Dodaje teraz te liste do tablicy list wygranych allPaths
00047                           allPaths[allPathsAmount++].
      cloneFrom(currentPath);
00048                          colour[u] = 0;
00049                           return;
00050                   }
00051
00052           num++;
00053           for( int v = 0; v < biggestValue; v++ ) if( M[u][v] && v != p )
00054           {
00055               if( colour[v] == 0 ) {
00056                   //cout<<u<<" -> "<<v<<"\n";
00057                   findPath( v, u , searchingValue);
00058               }
00059               else if( colour[v] == 1 )
00060               {
00061                   cout<<u<<" znalazlem szare wiec sie cofam\n";
00062
00063               }
00064               else
00065               {
00066                   cout<<u<<" znalazlem czarne wiec sie cofam\n";
00067
00068               }
00069               currentPath.pop_back();
00070           }
00071           //zmieniam flage na czarna;
00072           colour[u] = 0;
00073
00074       }
```

4.2.3.3 void DFS::findShortestPath (int *fromNumber,* int *toNumber*) [inline], [virtual]

najkrótszą drogę i wypusuje ją na ekran

Parametry

fromNumber	star poszukiwan
endNumber	Koniec poszukiwan

Implementuje Graph.

Definicja w linii 80 pliku dfs.h.

```
00081 {
00082         findPath( fromNumber, -1 , toNumber);
00083         int indexOfSmallestList = 0;
00084
00085         for(int i=0; i<allPathsAmount; i++)
00086         {
00087                 if( allPaths[indexOfSmallestList].size() >= allPaths[i].size() )
00088                 {
00089                         indexOfSmallestList = i;
00090                 }
00091         }
00092         allPaths[indexOfSmallestList].print();
00093 }
```

4.2.4 Dokumentacja atrybutów składowych

4.2.4.1 LinkedList<int> DFS::allPaths[MAX]

Definicja w linii 26 pliku dfs.h.

4.2.4.2 int DFS::allPathsAmount

Definicja w linii 27 pliku dfs.h.

4.2.4.3 LinkedList<int> DFS::currentPath

Definicja w linii 25 pliku dfs.h.

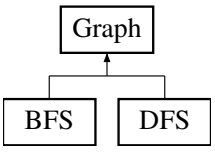
Dokumentacja dla tej klasy została wygenerowana z pliku:

- dfs.h

4.3 Dokumentacja klasy Graph

`#include <graph.h>`

Diagram dziedziczenia dla Graph



Metody publiczne

- virtual void findShortestPath (int fromNumber, int toNumber)=0
- virtual ~Graph ()
- void addEdge (int a, int b)
- void print ()

  Wyświetla graf.
- virtual void clean ()=0

  Zwalniam zasoby i przygotowuje do nowego rozruchu.

Atrybuty publiczne

- bool M [MAX][MAX]
- int colour [MAX]

  macierz numerów/indexów
- int biggestValue

  tablica kolorów
- int num

4.3.1 Opis szczegółowy

Definicja w linii 14 pliku graph.h.

4.3.2 Dokumentacja konstruktora i destruktora

4.3.2.1 virtual Graph::~Graph ( ) [inline], [virtual]

Definicja w linii 28 pliku graph.h.

```
00028 {}
```

4.3.3 Dokumentacja funkcji składowych

4.3.3.1 void Graph::addEdge (int *a*, int *b*) [inline]

połączenia miedzy elementami

Dodaje je obustronne powiązenie między nimi aby łatwiej bylo potem generować dane do testowania

Parametry

a	jedna krawędz
b	druga krawędz

Definicja w linii 36 pliku graph.h.

Odwołania w main().

```
00037         {
00038                 M[a][b] = 1;
00039                 M[b][a] = 1;
00040                 //n++;
00041                 if(a>=biggestValue) biggestValue=a+1;
00042                 if(b>=biggestValue) biggestValue=b+1;
00043         }
```

4.3.3.2 virtual void Graph::clean ( ) [pure virtual]

Implementowany w DFS i BFS.

Odwołania w main().

4.3.3.3 virtual void Graph::findShortestPath (int *fromNumber*, int *toNumber*) [pure virtual]

najkrótszą drogę i wypusuje ją na ekran

Parametry

fromNumber	star poszukiwan
endNumber	Koniec poszukiwan

Implementowany w DFS i BFS.

Odwołania w main().

4.3.3.4 void Graph::print ( ) [inline]

Definicja w linii 48 pliku graph.h.

```
00049           {
00050                   cout<<"\t";
00051                   for(int i=0; i<biggestValue; i++) cout<<i<<"\t";
00052                   cout<<"\n";
00053                   for(int i=0; i<biggestValue; i++)
00054                   {
00055                           cout<<i<<"\t";
00056                           for(int j=0; j<biggestValue; j++)
00057                           {
00058                                   cout<<M[i][j]<<"\t";
00059                           }
00060                           cout<<"\n";
00061                   }
00062           }
```

4.3.4 Dokumentacja atrybutów składowych

4.3.4.1 int Graph::biggestValue

Definicja w linii 19 pliku graph.h.

4.3.4.2 int Graph::colour[MAX]

Definicja w linii 18 pliku graph.h.

4.3.4.3 bool Graph::M[MAX][MAX]

Definicja w linii 17 pliku graph.h.

4.3.4.4 int Graph::num

Definicja w linii 20 pliku graph.h.

Dokumentacja dla tej klasy została wygenerowana z pliku:

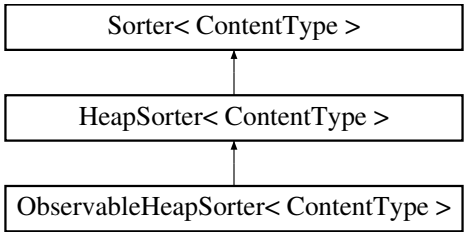
- graph.h

4.4 Dokumentacja szablonu klasy HeapSorter< ContentType >

Klasa sluzaca do obslugi sortowania przez kopcowanie.

`#include <heapsorter.h>`

Diagram dziedziczenia dla HeapSorter< ContentType >



Metody publiczne

- HeapSorter (List< ContentType > &myList)

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

  Sortuje przez kopcowanie.

Atrybuty publiczne

- List< ContentType > & list

  Skopiowana lista do przeprowadzania sortowania.

4.4.1 Opis szczegółowy

template<class ContentType>class HeapSorter< ContentType >

Definicja w linii 17 pliku heapsorter.h.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 template<class ContentType > HeapSorter< ContentType >::HeapSorter (List< ContentType > & *myList*) [inline]

Parametry

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

Definicja w linii 26 pliku heapsorter.h.

Odwołuje się do HeapSorter< ContentType >::list.

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

4.4.2.2 template<class ContentType > virtual HeapSorter< ContentType >::~HeapSorter ( ) [inline], [virtual]

Definicja w linii 38 pliku heapsorter.h.

```
00038 {};
```

4.4.3 Dokumentacja funkcji składowych

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

Implementuje Sorter< ContentType >.

Reimplementowana w ObservableHeapSorter< ContentType >.

Definicja w linii 42 pliku heapsorter.h.

Odwołuje się do HeapSorter< ContentType >::list.

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

```
00043         {
00044                 int n = this->list.size();
00045             int parent = n/2, index, child, tmp; /* heap indexes */
00046             /* czekam az sie posortuje */
00047             while (1) {
00048                 if (parent > 0)
00049                 {
00050                     tmp = (this->list)[--parent];  /* kobie kopie do tmp */
00051                 }
00052                 else {
00053                     n--;
00054                     if (n == 0)
00055                     {
00056                         return this->list; /* Zwraca posortowane */
00057                     }
00058                     tmp = this->list[n];
00059                     //int tmp = this->list[0];
00060                     this->list[n] = this->list[0];
00061                 }
00062                 index = parent;
00063                 child = index * 2 + 1;
00064                 while (child < n) {
00065                     if (child + 1 < n  &&  this->list[child + 1] > this->
      list[child]) {
00066                         child++;
00067                     }
00068                     if (this->list[child] > tmp) {
00069                         this->list[index] = this->list[child];
00070                         index = child;
00071                         child = index * 2 + 1;
00072                     } else {
00073                         break;
00074                     }
00075                 }
00076                 this->list[index] = tmp;
00077             }
00078             return this->list;
00079         }
```

4.4.4 Dokumentacja atrybutów składowych

4.4.4.1 template<class ContentType > List<ContentType>& HeapSorter< ContentType >::list

Definicja w linii 21 pliku heapsorter.h.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

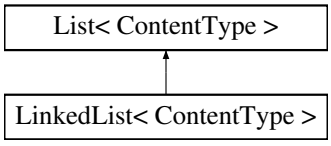
- heapsorter.h

4.5 Dokumentacja szablonu klasy LinkedList< ContentType >

Lista dwukierunkowa.

`#include <linkedlist.h>`

Diagram dziedziczenia dla LinkedList< ContentType >



Metody publiczne

- LinkedList ()

  Konstruktor listy.
- LinkedList (List< ContentType > &list)
- virtual ~LinkedList ()
- int & size ()

  Zwraca ilosc elementow listy.
- ContentType pop_back ()

  Zwraca element ostatni w liscie.
- ContentType pop_front ()

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

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

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

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

  Pokazuje element po koncu listy.
- void print ()

  Wyswietla elementy listy.
- ContentType & operator[] (int numberOfElement)

  Pobiera element z listy.
- LinkedListElement< ContentType > & getLinkedListElementById (int numberOfElement)
- void insertAfter (ContentType &arg, int iteratorID)

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

  Wzorzec projektowy - fabryki abstrakcyjnej.

Atrybuty publiczne

- int sizeOfList

  liczba elementow listy
- LinkedListElement< ContentType > * firstElement

  wskaznik do 'malej struktury' ktora jest pierwsza na liscie
- LinkedListElement< ContentType > * lastElement

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

4.5.1 Opis szczegółowy

template<class ContentType>class LinkedList< ContentType >

Klasa przedstawia liste dwukierunkową dynamiczna

Definicja w linii 22 pliku linkedlist.h.

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 template<class ContentType> LinkedList< ContentType >::LinkedList ( ) [inline]

Definicja w linii 38 pliku linkedlist.h.

```
00039         {
00040                 firstElement = lastElement = new
      LinkedListElement<ContentType>;
00041                 sizeOfList = 0;
00042                 iteratorElementId =0;
00043                 iterator=NULL;
00044                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00045         }
```

4.5.2.2 template<class ContentType> LinkedList< ContentType >::LinkedList (List< ContentType > & *list*) [inline]

Definicja w linii 47 pliku linkedlist.h.

```
00048         {
00049                 firstElement = lastElement = new
      LinkedListElement<ContentType>;
00050                 sizeOfList = 0;
00051                 iteratorElementId =0;
00052                 iterator=NULL;
00053                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00054                 for(int i=0; i<list.size(); i++)
00055                 {
00056                         this->push_back(list[i]);
00057                 }
00058         }
```

4.5.2.3 template<class ContentType> virtual LinkedList< ContentType >::~LinkedList ( ) [inline], [virtual]

Definicja w linii 59 pliku linkedlist.h.

```
00059 {};
```

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 template<class ContentType> List<ContentType>& LinkedList< ContentType >::createObjectFromAbstractReference ( ) [inline], [virtual]

Implementuje List< ContentType >.

Definicja w linii 295 pliku linkedlist.h.

```
00296         {
00297                 return *new LinkedList<ContentType>;
00298         }
```

4.5.3.2 template<class ContentType> LinkedListElement<ContentType>& LinkedList< ContentType >::getLinkedListElementById (int *numberOfElement*) [inline]

Definicja w linii 197 pliku linkedlist.h.

```
00198         {
00199                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00200                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00201                         {
00202                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
      !";
00203                                 return *iterator;
```

```
00204                           }
00205                   if(isIteratorAfterPop)
00206                           {
00207                                   iteratorElementId=0;  // czyli iterator byl zpopowany
00208                                   iterator = firstElement;
00209                                   isIteratorAfterPop=0;
00210                           }
00211                   //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00212                   if((numberOfElement <= iteratorElementId-numberOfElement) &&(
      iteratorElementId-numberOfElement>=0))
00213                   {
00214                           //std::cerr<<"\nJestem w if_1";
00215                           iterator = (this->firstElement);
00216                           iteratorElementId = 0;
00217                           for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00218                                   iterator = (iterator->nextElement);
00219                   }
00220                   else if(numberOfElement > iteratorElementId)
00221                   {
00222                           //std::cerr<<"\nJestem w if_2";
00223                           for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00224                                                   iterator = (iterator->nextElement);
00225                   }
00226                   else if( numberOfElement < iteratorElementId)
00227                   {
00228                           //std::cerr<<"\nJestem w if_3";
00229                           for (; iteratorElementId> numberOfElement ;
      iteratorElementId--)
00230                                                   iterator = (iterator->previousElement);
00231                   }
00232                   return *iterator;
00233           }
```

4.5.3.3 template<class ContentType> void LinkedList< ContentType >::insertAfter (ContentType & *arg,* int *iteratorID*) [inline], [virtual]

Implementuje List< ContentType >.

Definicja w linii 238 pliku linkedlist.h.

```
00239           {
00240                   if(iteratorID==0 && this->sizeOfList==0)  {push_front(arg); return;}
00241                   if(iteratorID==this->sizeOfList-1)  {push_back(arg); return;}
00242                   LinkedListElement<ContentType> *newLinkedListElement = new
      LinkedListElement<ContentType>(arg);
00243                   LinkedListElement<ContentType>
00244                           &tmpThis=(*this).getLinkedListElementById(iteratorID),
00245                           &tmpNext=(*this).getLinkedListElementById(iteratorID+1);
00246                   if(!sizeOfList++) {firstElement =
      lastElement = newLinkedListElement;}
00247                   newLinkedListElement -> nextElement = tmpThis.nextElement;
00248                   newLinkedListElement -> previousElement = &tmpThis;
00249                   tmpThis.nextElement = newLinkedListElement;
00250                   tmpNext.previousElement = newLinkedListElement;
00251                   isIteratorAfterPop=1;
00252           }
```

4.5.3.4 template<class ContentType> LinkedList<ContentType>& LinkedList< ContentType >::operator= (const LinkedList< ContentType > & *pattern*) [inline]

Definicja w linii 262 pliku linkedlist.h.

```
00263           {
00264                   //std::cerr<<" @@@";
00265                   this->sizeOfList = pattern.sizeOfList;
00266                   this->firstElement = pattern.firstElement;
00267                   this->lastElement = pattern.lastElement;
00268                   this->iterator=pattern.iterator;
00269                   this->isIteratorAfterPop = pattern.
      isIteratorAfterPop;
00270                   return *this;
00271           }
```

4.5.3.5 template<class ContentType> ContentType& LinkedList< ContentType >::operator[] (int *numberOfElement*) [inline], [virtual]

Zwraca

Zwraca 0 gdy zapisywanie powiodlo sie

Implementuje List< ContentType >.

Definicja w linii 159 pliku linkedlist.h.

```
00160         {
00161                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00162                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00163                         {
00164                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
      !";
00165                                 return (*iterator).content;
00166                         }
00167                 if(isIteratorAfterPop)
00168                         {
00169                                 iteratorElementId=0;  // czyli iterator byl zpopowany
00170                                 iterator = firstElement;
00171                                 isIteratorAfterPop=0;
00172                         }
00173                 //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00174                 if((numberOfElement <= iteratorElementId-numberOfElement) &&(
     iteratorElementId-numberOfElement>=0))
00175                 {
00176                         //std::cerr<<"\nJestem w if_1";
00177                         iterator = (this->firstElement);
00178                         iteratorElementId = 0;
00179                         for (; iteratorElementId< numberOfElement ;
     iteratorElementId++)
00180                                 iterator = (iterator->nextElement);
00181                 }
00182                 else if(numberOfElement > iteratorElementId)
00183                 {
00184                         //std::cerr<<"\nJestem w if_2";
00185                         for (; iteratorElementId< numberOfElement ;
     iteratorElementId++)
00186                                                 iterator = (iterator->nextElement);
00187                 }
00188                 else if( numberOfElement < iteratorElementId)
00189                 {
00190                         //std::cerr<<"\nJestem w if_3";
00191                         for (; iteratorElementId> numberOfElement ;
     iteratorElementId--)
00192                                                 iterator = (iterator->previousElement);
00193                 }
00194                 return (*iterator).content;
00195         }
```

4.5.3.6 template<class ContentType> ContentType LinkedList< ContentType >::pop_back ( ) [inline], [virtual]

Zwraca

Zwraca element ostatni w liscie

Implementuje List< ContentType >.

Definicja w linii 73 pliku linkedlist.h.

```
00074         {
00075                 if(!(sizeOfList--)) { sizeOfList=0; std::cerr<<"Nie ma takiego elementu
     \n";}
00076                 ContentType tmpNumber = (*(this -> lastElement)).content;
00077                 LinkedListElement<ContentType> *originLinkedListElement =
     this -> lastElement;
00078                 this -> lastElement = this -> lastElement -> previousElement;
00079                 delete originLinkedListElement;
00080                 isIteratorAfterPop=1;
00081                 return tmpNumber;
00082         }
```

4.5.3.7 template<class ContentType> ContentType LinkedList< ContentType >::pop_front ( ) [inline], [virtual]

Zwraca

Zwraca element pierwszy w liscie

Implementuje List< ContentType >.

Definicja w linii 87 pliku linkedlist.h.

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

```
00088         {
00089                 if(!(sizeOfList--)) { sizeOfList=0; std::cerr<<"Nie ma takiego elementu
      \n";}
00090                 ContentType tmpNumber = (*(this -> firstElement)).content;
00091                 LinkedListElement<ContentType> *originLinkedListElement =
      this -> firstElement;
00092                 this -> firstElement = this -> firstElement -> nextElement;
00093                 delete originLinkedListElement;
00094                 isIteratorAfterPop=1;
00095                 return tmpNumber;
00096         }
```

4.5.3.8 template<class ContentType> void LinkedList< ContentType >::print ( ) [inline], [virtual]

Implementuje List< ContentType >.

Definicja w linii 144 pliku linkedlist.h.

Odwołania w BFS::findShortestPath().

```
00145         {
00146                 LinkedListElement<ContentType> *elem = (this->
      firstElement);
00147                 std::cout<<"\nWyswietlam liste (size:"<<this->sizeOfList<<"): ";
00148                 for(int i=0; i< this->sizeOfList; i++)
00149                 {
00150                         std::cout<<" "<<elem->content;
00151                         elem = elem->nextElement;
00152                 }
00153         }
```

4.5.3.9 template<class ContentType> void LinkedList< ContentType >::push_back (ContentType & *arg*) [inline], [virtual]

Implementuje List< ContentType >.

Definicja w linii 100 pliku linkedlist.h.

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

```
00101         {
00102                 //std::cerr<<"\n(push_back): arg.content="<<arg.content;
00103                 LinkedListElement<ContentType> *newLinkedListElement = new
      LinkedListElement<ContentType>(arg);
00104                 if(!sizeOfList++) {firstElement =
      lastElement = newLinkedListElement;}
00105                 //newLinkedListElement -> nextElement = 0;
00106                 newLinkedListElement -> previousElement = this -> lastElement;
00107                 this -> lastElement -> nextElement = newLinkedListElement;
00108                 this->lastElement = newLinkedListElement;
00109         }
```

4.5.3.10 template<class ContentType> void LinkedList< ContentType >::push_front (ContentType & *arg*) [inline], [virtual]

Implementuje List< ContentType >.

Definicja w linii 113 pliku linkedlist.h.

Odwołania w BFS::findShortestPath() i LinkedList< Observer * >::insertAfter().

```
00114         {
00115                 LinkedListElement<ContentType> *newLinkedListElement = new
      LinkedListElement<ContentType>(arg);
00116                 if(!sizeOfList++) {firstElement =
      lastElement = newLinkedListElement;}
00117                 //newLinkedListElement -> previousElement =  0;
00118                 newLinkedListElement -> nextElement = this -> firstElement;
00119                 this -> firstElement -> previousElement = newLinkedListElement;
00120                 this->firstElement = newLinkedListElement;
00121                 ++iteratorElementId;
00122         }
```

4.5.3.11 template<class ContentType> ContentType& LinkedList< ContentType >::show_back ( ) [inline], [virtual]

Zwraca

zwraca kopie tego elementu

Implementuje List< ContentType >.

Definicja w linii 135 pliku linkedlist.h.

```
00136         {
00137                 return lastElement->content;
00138         }
```

4.5.3.12 template<class ContentType> ContentType& LinkedList< ContentType >::show_front ( ) [inline], [virtual]

Zwraca

zwraca kopie tego elementu

Implementuje List< ContentType >.

Definicja w linii 127 pliku linkedlist.h.

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

```
00128         {
00129                 return firstElement->content;
00130         }
```

4.5.3.13 template<class ContentType> int& LinkedList< ContentType >::size ( ) [inline], [virtual]

Zwraca

ilosc elementow tablicy

Implementuje List< ContentType >.

Definicja w linii 65 pliku linkedlist.h.

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

```
00066         {
00067                 return sizeOfList;
00068         }
```

4.5.4 Dokumentacja atrybutów składowych

4.5.4.1 template<class ContentType> LinkedListElement<ContentType>* LinkedList< ContentType >::firstElement

Definicja w linii 30 pliku linkedlist.h.

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

4.5.4.2 template<class ContentType> int LinkedList< ContentType >::isIteratorAfterPop

Definicja w linii 35 pliku linkedlist.h.

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

4.5.4.3 template<class ContentType> LinkedListElement<ContentType>* LinkedList< ContentType >::iterator

Definicja w linii 33 pliku linkedlist.h.

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

4.5.4.4 template<class ContentType> int LinkedList< ContentType >::iteratorElementId

Definicja w linii 34 pliku linkedlist.h.

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

4.5.4.5 template<class ContentType> LinkedListElement<ContentType>* LinkedList< ContentType >::lastElement

Definicja w linii 32 pliku linkedlist.h.

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

4.5.4.6 template<class ContentType> int LinkedList< ContentType >::sizeOfList

Definicja w linii 26 pliku linkedlist.h.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

- linkedlist.h

4.6 Dokumentacja szablonu klasy LinkedListElement< ContentType >

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

`#include <linkedlistelement.h>`

Metody publiczne

- LinkedListElement ()

  Konstruktor wewnetrznej klasy 'malych struktur'.
- LinkedListElement (ContentType &arg)

  Konstruktor wewnetrznej klasy 'malych struktur'.

- LinkedListElement (const LinkedListElement &linkedListElement)

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

  Ustawia liczbe oraz klucz slowanika dla elementu.

Atrybuty publiczne

- LinkedListElement * nextElement

  Liczba przechowywana.
- LinkedListElement * previousElement

  wskaznik do poprzedniej 'malej struktury' w liscie
- ContentType content

4.6.1 Opis szczegółowy

template<class ContentType>class LinkedListElement< ContentType >

Definicja w linii 15 pliku linkedlistelement.h.

4.6.2 Dokumentacja konstruktora i destruktora

4.6.2.1 template<class ContentType> LinkedListElement< ContentType >::LinkedListElement ( ) [inline]

Definicja w linii 27 pliku linkedlistelement.h.

```
00028         {
00029                 this -> nextElement =0;
00030                 this -> previousElement =0;
00031         }
```

4.6.2.2 template<class ContentType> LinkedListElement< ContentType >::LinkedListElement (ContentType & *arg*) [inline]

Parametry

arg	liczba do zapisania w kolejnym elemencie listy

Definicja w linii 36 pliku linkedlistelement.h.

```
00037         {
00038                 this -> content = arg;
00039                 this -> nextElement =0;
00040                 this -> previousElement =0;
00041                 //std::cerr<<"\n(konstruktor LinkedListElement): content="<<arg;
00042         }
```

4.6.2.3 template<class ContentType> LinkedListElement< ContentType >::LinkedListElement (const LinkedListElement< ContentType > & *linkedListElement*) [inline]

Parametry

linkedList-Element	Element o przekopiowania

Definicja w linii 47 pliku linkedlistelement.h.

```
00048         {
00049                 this->content = linkedListElement.content;
00050                 this->nextElement = linkedListElement.nextElement;
00051                 this->previousElement = linkedListElement.
      previousElement;
00052                 //std::cerr<<"\n(konstruktor kopiujacy LinkedListElement): content="<<content;
00053         }
```

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 template<class ContentType> void LinkedListElement< ContentType >::set (ContentType *arg*) [inline]

Parametry

arg	Liczba do zapisania

Definicja w linii 57 pliku linkedlistelement.h.

```
00058         {
00059                 this -> content = arg;
00060         }
```

4.6.4 Dokumentacja atrybutów składowych

4.6.4.1 template<class ContentType> ContentType LinkedListElement< ContentType >::content

Definicja w linii 23 pliku linkedlistelement.h.

Odwołania w LinkedListElement< Observer * >::LinkedListElement(), LinkedList< Observer * >::print() i LinkedListElement< Observer * >::set().

4.6.4.2 template<class ContentType> LinkedListElement* LinkedListElement< ContentType >::nextElement

wskaznik do nastepnej 'malej struktury' w liscie

Definicja w linii 20 pliku linkedlistelement.h.

Odwołania w LinkedList< Observer * >::insertAfter(), LinkedListElement< Observer * >::LinkedListElement() i LinkedList< Observer * >::print().

4.6.4.3 template<class ContentType> LinkedListElement* LinkedListElement< ContentType >::previousElement

Definicja w linii 22 pliku linkedlistelement.h.

Odwołania w LinkedList< Observer * >::insertAfter() i LinkedListElement< Observer * >::LinkedListElement().

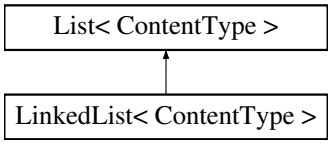
Dokumentacja dla tej klasy została wygenerowana z pliku:

- linkedlistelement.h

4.7 Dokumentacja szablonu klasy List< ContentType >

`#include <list.h>`

Diagram dziedziczenia dla List< ContentType >



Metody publiczne

- virtual int & size ()=0

  Pobiera rozmiar listy.
- virtual ContentType pop_back ()=0

  Zwraca ostatni element z listy.

- virtual ContentType pop_front ()=0

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

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

  Wsadza ContentType do listy na poczatek.
- virtual ContentType & operator[] (int numberOfElement)=0

  Wsadza ContentType do listy na koniec.
- virtual void insertAfter (ContentType &arg, int iteratorID)=0

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

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

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

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

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

  Zwalnia zasoby listy.
- virtual ~List ()

4.7.1 Opis szczegółowy

template<class ContentType>class List< ContentType >

Interface dla klasy przedstawiających listy

Definicja w linii 16 pliku list.h.

4.7.2 Dokumentacja konstruktora i destruktora

4.7.2.1 template<class ContentType> virtual List< ContentType >::~List ( ) [inline], [virtual]

Definicja w linii 83 pliku list.h.

```
00083 {};
```

4.7.3 Dokumentacja funkcji składowych

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

Definicja w linii 68 pliku list.h.

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

```
00069         {
00070                 // release memory from main list
00071                 while(this->size()) pop_back();
00072                 for(int i=0; i<patternList.size(); i++)
00073                         this->push_back(patternList[i]);
00074         }
```

4.7.3.2 template<class ContentType> virtual List<ContentType>& List< ContentType >::createObjectFromAbstractReference ( ) [pure virtual]

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.3 template<class ContentType> virtual void List< ContentType >::free ( ) [inline], [virtual]

Definicja w linii 82 pliku list.h.

```
00082 { while(size()) pop_back(); }
```

4.7.3.4 template<class ContentType> virtual void List< ContentType >::insertAfter (ContentType & *arg*, int *iteratorID*) [pure virtual]

Parametry

arg	Element do wsadzenia
iteratorID	id elementu do wsadzenia

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.5 template<class ContentType> virtual ContentType& List< ContentType >::operator[] (int *numberOfElement*) [pure virtual]

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.6 template<class ContentType> virtual ContentType List< ContentType >::pop_back ( ) [pure virtual]

Zwraca

ostatni element z listy

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

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

4.7.3.7 template<class ContentType> virtual ContentType List< ContentType >::pop_front ( ) [pure virtual]

Zwraca

pierwszy element z listy

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.8 template<class ContentType> virtual void List< ContentType >::print ( ) [pure virtual]

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.9 template<class ContentType> virtual void List< ContentType >::push_back (ContentType & *arg*) [pure virtual]

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

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

4.7.3.10 template<class ContentType> virtual void List< ContentType >::push_front (ContentType & *arg*) [pure virtual]

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.11 template<class ContentType> virtual ContentType& List< ContentType >::show_back ( ) [pure virtual]

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.12 template<class ContentType> virtual ContentType& List< ContentType >::show_front ( ) [pure virtual]

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

4.7.3.13 template<class ContentType> virtual int& List< ContentType >::size ( ) [pure virtual]

Zwraca

Rozmiar listy

Implementowany w LinkedList< ContentType >, LinkedList< int > i LinkedList< Observer * >.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

- list.h

4.8 Dokumentacja szablonu klasy ListSaver< ContentType >

`#include <listsaver.h>`

Metody prywatne

- ListSaver (MyList< ContentType > &listArgument)

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

  Zapisuje liste do pliku.

Atrybuty prywatne

- List< ContentType > & list

  Klasa pozwalająca na zapis Listy do pliku.

4.8.1 Opis szczegółowy

template<class ContentType>class ListSaver< ContentType >

Definicja w linii 15 pliku listsaver.h.

4.8.2 Dokumentacja konstruktora i destruktora

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

Parametry

listArgument	lista do zapisu

Definicja w linii 24 pliku listsaver.h.

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

4.8.3 Dokumentacja funkcji składowych

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

Zwraca

Zwraca 0 gdy zapisywanie powiodlo sie

Definicja w linii 32 pliku listsaver.h.

Odwołuje się do ListSaver< ContentType >::list.

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

4.8.4 Dokumentacja atrybutów składowych

4.8.4.1 template<class ContentType > List<ContentType>& ListSaver< ContentType >::list [private]

Definicja w linii 19 pliku listsaver.h.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

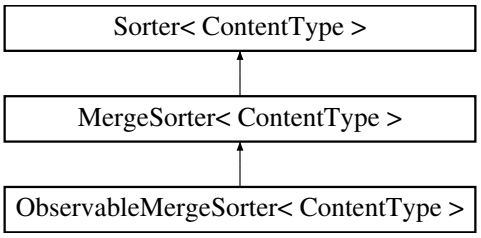
- listsaver.h

4.9 Dokumentacja szablonu klasy MergeSorter< ContentType >

Klasa sluzaca do obslugi sortowania przez Scalanie.

`#include <mergesorter.h>`

Diagram dziedziczenia dla MergeSorter< ContentType >



Metody publiczne

- MergeSorter (LinkedList< ContentType > &listArg)
  Konstruktor.
- virtual ~MergeSorter ()
- LinkedList< ContentType > merge (LinkedList< ContentType > left, LinkedList< ContentType > right)
  Scalanie list.
- LinkedList< ContentType > mergeSort (LinkedList< ContentType > m)
  Sortuje liste przez scalanie.
- List< ContentType > & sort ()
  Sortuje przez scalanie.

Atrybuty publiczne

- LinkedList< ContentType > & list

  Skopiowana lista do przeprowadzania sortowania.

4.9.1 Opis szczegółowy

template<class ContentType>class MergeSorter< ContentType >

Definicja w linii 17 pliku mergesorter.h.

4.9.2 Dokumentacja konstruktora i destruktora

4.9.2.1 template<class ContentType > MergeSorter< ContentType >::MergeSorter (LinkedList< ContentType > & *listArg*) [inline]

Parametry

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

Definicja w linii 26 pliku mergesorter.h.

```
00027         :list(listArg)      {}
```

4.9.2.2 template<class ContentType > virtual MergeSorter< ContentType >::~MergeSorter ( ) [inline], [virtual]

Definicja w linii 29 pliku mergesorter.h.

```
00029 {}
```

4.9.3 Dokumentacja funkcji składowych

4.9.3.1 template<class ContentType > LinkedList<ContentType> MergeSorter< ContentType >::merge (LinkedList< ContentType > *left,* LinkedList< ContentType > *right*) [inline]

Parametry

left	lewa lista do scalania
right	prawa lista do scalania

Zwraca

zwraca posotrowana liste

Definicja w linii 37 pliku mergesorter.h.

Odwołuje się do LinkedList< ContentType >::pop_front(), LinkedList< ContentType >::push_back(), LinkedList< ContentType >::show_front() i LinkedList< ContentType >::size().

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

```
00038         {
00039                 LinkedList<ContentType> result;
00040                 //Gdy jest jeszcze cos do sortowania
00041                 while (left.size() > 0 || right.size() > 0)
00042                 {
00043                         // Jak oba to zamieniamy
00044                         if (left.size() > 0 && right.size() > 0)
00045                         {
```

```
00046                                   // Sprawdzam czy zamieniac
00047                                   if (left.show_front() <= right.
      show_front())
00048                                           {
00049                                                   result.push_back(left.
      show_front()); left.pop_front();
00050                                           }
00051                                   else
00052                                   {
00053                                           result.push_back(right.
      show_front()); right.pop_front();
00054                                   }
00055                           }
00056                           // pojedyncze listy (nieparzyse)
00057                           else if (left.size() > 0)
00058                           {
00059                                   for (int i = 0; i < left.size(); i++) result.
      push_back(left[i]); break;
00060                           }
00061                           // pojedyncze listy (nieparzyse- taka sama sytuacja jak wyzej)
00062                           else if ((int)right.size() > 0)
00063                           {
00064                                   for (int i = 0; i < (int)right.size(); i++) result.
      push_back(right[i]); break;
00065                           }
00066                   }
00067                   return result;
00068           }
```

4.9.3.2 template<class ContentType > LinkedList<ContentType> MergeSorter< ContentType >::mergeSort (LinkedList< ContentType > *m*) [inline]

Parametry

m	Lista do posotrowania

Zwraca

zwraca posotrowana liste

Definicja w linii 74 pliku mergesorter.h.

Odwołuje się do MergeSorter< ContentType >::merge(), LinkedList< ContentType >::push_back() i LinkedList< ContentType >::size().

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

```
00075           {
00076                   if (m.size() <= 1) return m; // gdy juz nic nie ma do sotrowania
00077                   LinkedList<ContentType> left, right, result;
00078                   int middle = (m.size()+ 1) / 2; // anty-nieparzyscie
00079                   for (int i = 0; i < middle; i++)
00080                           {
00081                                   left.push_back(m[i]);
00082                           }
00083                   for (int i = middle; i < m.size(); i++)
00084                           {
00085                                   right.push_back(m[i]);
00086                           }
00087                   left = mergeSort(left);
00088                   right = mergeSort(right);
00089                   result = merge(left, right);
00090                   return result;
00091           }
```

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

Implementuje Sorter< ContentType >.

Reimplementowana w ObservableMergeSorter< ContentType >.

Definicja w linii 96 pliku mergesorter.h.

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

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

```
00097         {
00098                 this->list=mergeSort(this->list);
00099                 return this->list;
00100         }
```

4.9.4 Dokumentacja atrybutów składowych

4.9.4.1 template<class ContentType > LinkedList<ContentType>& MergeSorter< ContentType >::list

Definicja w linii 21 pliku mergesorter.h.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

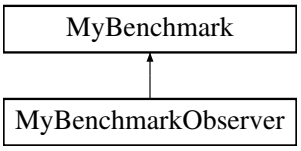
- mergesorter.h

4.10 Dokumentacja klasy MyBenchmark

Klasa bazowa/interface do testowania algorytmu.

`#include <mybenchmark.h>`

Diagram dziedziczenia dla MyBenchmark



Metody publiczne

- MyBenchmark ()
- void timerStart ()

  włączam stoper
- double timerStop ()

  wyłączam stoper
- virtual ~MyBenchmark ()

  Usuwam obiekt test biorąc pod uwage jego prawdziwy typ.

Atrybuty publiczne

- double timerValue

  Czas stopera.

4.10.1 Opis szczegółowy

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

Definicja w linii 20 pliku mybenchmark.h.

4.10.2 Dokumentacja konstruktora i destruktora

4.10.2.1 MyBenchmark::MyBenchmark ( ) [inline]

Definicja w linii 27 pliku mybenchmark.h.

Odwołuje się do timerValue.

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

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

Definicja w linii 44 pliku mybenchmark.h.

```
00044 {};
```

4.10.3 Dokumentacja funkcji składowych

4.10.3.1 void MyBenchmark::timerStart ( )

Definicja w linii 12 pliku mybenchmark.cpp.

Odwołuje się do timerValue.

Odwołania w MyBenchmarkObserver::receivedStartUpdate().

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

4.10.3.2 double MyBenchmark::timerStop ( )

Zwraca

Dlugosc dzialania stopera

Definicja w linii 17 pliku mybenchmark.cpp.

Odwołuje się do timerValue.

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

4.10.4 Dokumentacja atrybutów składowych

4.10.4.1 double MyBenchmark::timerValue

Definicja w linii 25 pliku mybenchmark.h.

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

Dokumentacja dla tej klasy została wygenerowana z plików:

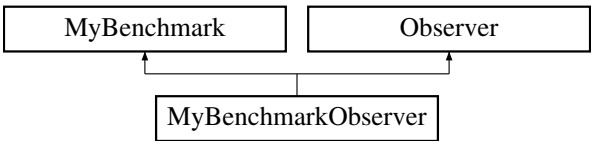
- mybenchmark.h
- mybenchmark.cpp

4.11 Dokumentacja klasy MyBenchmarkObserver

Mybenchmark obserwator Używana jako obserwator klasa sprawdzajaca odpowiednie objekty.

`#include <mybenchmark.h>`

Diagram dziedziczenia dla MyBenchmarkObserver



Metody publiczne

- MyBenchmarkObserver ()
- double getTimerValue ()

  pobiera czas trwania algorytmu
- void receivedStartUpdate ()

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

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

Dodatkowe Dziedziczone Składowe

4.11.1 Opis szczegółowy

Definicja w linii 52 pliku mybenchmark.h.

4.11.2 Dokumentacja konstruktora i destruktora

4.11.2.1 MyBenchmarkObserver::MyBenchmarkObserver ( ) [inline]

Definicja w linii 55 pliku mybenchmark.h.

```
00055 {};
```

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

Definicja w linii 73 pliku mybenchmark.h.

```
00073 {};
```

4.11.3 Dokumentacja funkcji składowych

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

Zwraca

czas trwania algorytmu

Implementuje Observer.

Definicja w linii 60 pliku mybenchmark.h.

Odwołuje się do MyBenchmark::timerValue.

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

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

Implementuje Observer.

Definicja w linii 64 pliku mybenchmark.h.

Odwołuje się do MyBenchmark::timerStart().

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

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

Implementuje Observer.

Definicja w linii 70 pliku mybenchmark.h.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

- mybenchmark.h

4.12 Dokumentacja klasy NumberGenerator

Klasa generujaca losowe liczby.

`#include <numbergenerator.h>`

Statyczne metody publiczne

- template<typename ContentType >
  static LinkedList< ContentType > & generateNumbers (int range, int quantity)

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

  Generuje losowe stringi.

4.12.1 Opis szczegółowy

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

Definicja w linii 27 pliku numbergenerator.h.

4.12.2 Dokumentacja funkcji składowych

4.12.2.1 template<typename ContentType > static LinkedList<ContentType>& NumberGenerator::generateNumbers (int *range,* int *quantity*) [inline], [static]

Definicja w linii 33 pliku numbergenerator.h.

Odwołuje się do LinkedList< ContentType >::push_back().

```
00034 {
00035         LinkedList<ContentType> &myList = *new
      LinkedList<ContentType>();
00036         time_t randomTime = clock();
00037         int randomNumber;
00038         for(int i=0; i<quantity ; i++)
00039         {
00040                 srand (randomTime = clock());
00041                 randomNumber = rand()%range;
00042                 myList.push_back(randomNumber);
00043                 randomTime = clock();
00044         }
00045         return myList;
00046 }
```

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

Parametry

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

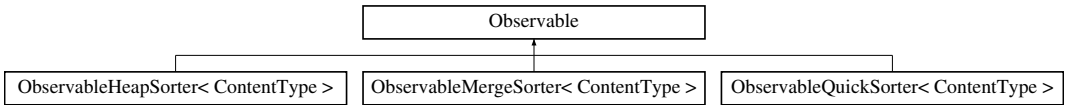
- numbergenerator.h

4.13 Dokumentacja klasy Observable

Klasa abstrakcyjna- bazowa dla objektow do obserowania.

`#include <observable.h>`

Diagram dziedziczenia dla Observable



Metody publiczne

- void add (Observer *obserwator)

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

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

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

Atrybuty publiczne

- LinkedList< Observer * > observaters

  Lista obserwatorow.

4.13.1 Opis szczegółowy

Definicja w linii 16 pliku observable.h.

4.13.2 Dokumentacja konstruktora i destruktora

4.13.2.1 virtual Observable::~Observable ( ) [inline], [virtual]

Definicja w linii 44 pliku observable.h.

```
00044 {}
```

4.13.3 Dokumentacja funkcji składowych

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

Definicja w linii 23 pliku observable.h.

Odwołuje się do observaters i LinkedList< ContentType >::push_back().

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

4.13.3.2 void Observable::sendStartUpdateToObservers ( ) [inline]

Definicja w linii 29 pliku observable.h.

Odwołuje się do observaters i LinkedList< ContentType >::size().

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

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

4.13.3.3 void Observable::sendStopUpdateToObservers ( ) [inline]

Definicja w linii 39 pliku observable.h.

Odwołuje się do observaters i LinkedList< ContentType >::size().

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

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

4.13.4 Dokumentacja atrybutów składowych

4.13.4.1 LinkedList<Observer*> Observable::observaters

Definicja w linii 19 pliku observable.h.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

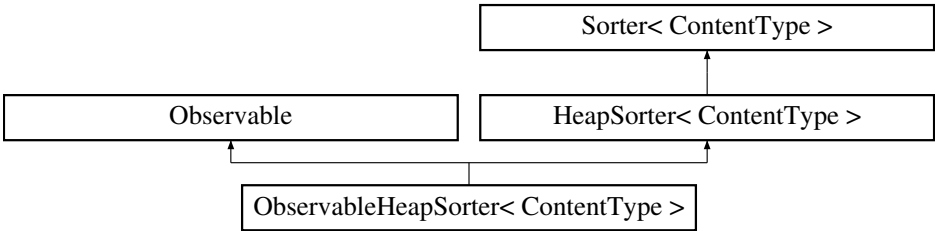
- observable.h

4.14 Dokumentacja szablonu klasy ObservableHeapSorter< ContentType >

Klasa sluzaca do obslugi sortowania przez kopcowanie z dodaniem obserwatora.

`#include <observableheapsorter.h>`

Diagram dziedziczenia dla ObservableHeapSorter< ContentType >



Metody publiczne

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

  sortuje przez kopcowanie
- virtual ~ObservableHeapSorter ()

Dodatkowe Dziedziczone Składowe

4.14.1 Opis szczegółowy

template<class ContentType>class ObservableHeapSorter< ContentType >

Definicja w linii 18 pliku observableheapsorter.h.

4.14.2 Dokumentacja konstruktora i destruktora

4.14.2.1 template<class ContentType > ObservableHeapSorter< ContentType >::ObservableHeapSorter (List< ContentType > & *myList*) [inline]

Definicja w linii 21 pliku observableheapsorter.h.

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

4.14.2.2 template<class ContentType > virtual ObservableHeapSorter< ContentType >::~ObservableHeapSorter ( ) [inline], [virtual]

Definicja w linii 33 pliku observableheapsorter.h.

```
00033 {};
```

4.14.3 Dokumentacja funkcji składowych

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

Reimplementowana z HeapSorter< ContentType >.

Definicja w linii 26 pliku observableheapsorter.h.

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

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

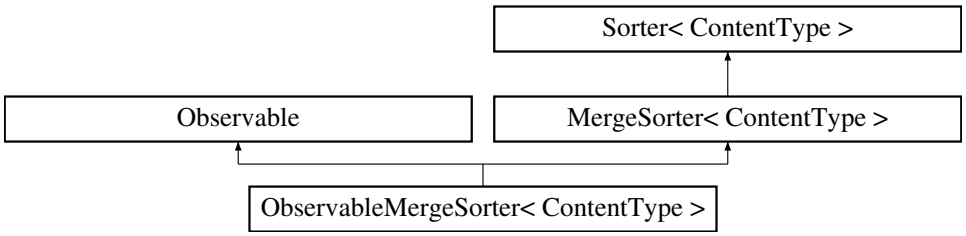
- observableheapsorter.h

4.15 Dokumentacja szablonu klasy ObservableMergeSorter< ContentType >

Klasa sluzaca do obslugi sortowania przez Scalanie z dodaniem obserwatora.

`#include <observablemergesorter.h>`

Diagram dziedziczenia dla ObservableMergeSorter< ContentType >



Metody publiczne

- ObservableMergeSorter (LinkedList< ContentType > &myList)
- List< ContentType > & sort ()

  sortuje przez scalanie
- virtual ~ObservableMergeSorter ()

Dodatkowe Dziedziczone Składowe

4.15.1 Opis szczegółowy

template<class ContentType>class ObservableMergeSorter< ContentType >

Definicja w linii 18 pliku observablemergesorter.h.

4.15.2 Dokumentacja konstruktora i destruktora

4.15.2.1 template<class ContentType > ObservableMergeSorter< ContentType >::ObservableMergeSorter (LinkedList< ContentType > & *myList*) [inline]

Definicja w linii 21 pliku observablemergesorter.h.

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

4.15.2.2 template<class ContentType > virtual ObservableMergeSorter< ContentType >::~ObservableMergeSorter ( ) [inline], [virtual]

Definicja w linii 33 pliku observablemergesorter.h.

```
00033 {};
```

4.15.3 Dokumentacja funkcji składowych

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

Reimplementowana z MergeSorter< ContentType >.

Definicja w linii 26 pliku observablemergesorter.h.

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

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

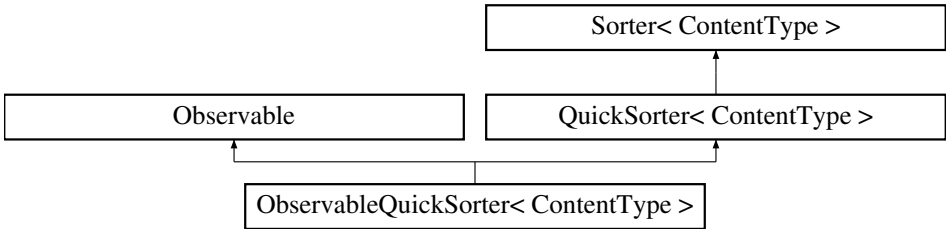
- observablemergesorter.h

4.16 Dokumentacja szablonu klasy ObservableQuickSorter< ContentType >

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

`#include <observablequicksorter.h>`

Diagram dziedziczenia dla ObservableQuickSorter< ContentType >



Metody publiczne

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

  sortuje przez scalanie
- virtual ~ObservableQuickSorter ()

Dodatkowe Dziedziczone Składowe

4.16.1 Opis szczegółowy

template<class ContentType>class ObservableQuickSorter< ContentType >

Definicja w linii 18 pliku observablequicksorter.h.

4.16.2 Dokumentacja konstruktora i destruktora

4.16.2.1 template<class ContentType > ObservableQuickSorter< ContentType >::ObservableQuickSorter (List< ContentType > & *list*) [inline]

Definicja w linii 21 pliku observablequicksorter.h.

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

4.16.2.2 template<class ContentType > virtual ObservableQuickSorter< ContentType >::~ObservableQuickSorter ( ) [inline], [virtual]

Definicja w linii 33 pliku observablequicksorter.h.

```
00033 {};
```

4.16.3 Dokumentacja funkcji składowych

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

Implementuje Sorter< ContentType >.

Definicja w linii 26 pliku observablequicksorter.h.

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

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

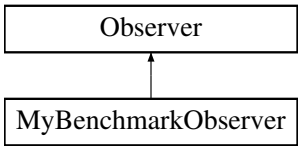
- observablequicksorter.h

4.17 Dokumentacja klasy Observer

obserwator

`#include <observer.h>`

Diagram dziedziczenia dla Observer



Metody publiczne

- virtual double getTimerValue ()=0

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

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

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

4.17.1 Opis szczegółowy

Interfejs obserwatora

Definicja w linii 19 pliku observer.h.

4.17.2 Dokumentacja konstruktora i destruktora

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

Definicja w linii 33 pliku observer.h.

```
00033 {};
```

4.17.3 Dokumentacja funkcji składowych

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

Zwraca

czas trwania algorytmu

Implementowany w MyBenchmarkObserver.

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

Implementowany w MyBenchmarkObserver.

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

Implementowany w MyBenchmarkObserver.

Dokumentacja dla tej klasy została wygenerowana z pliku:

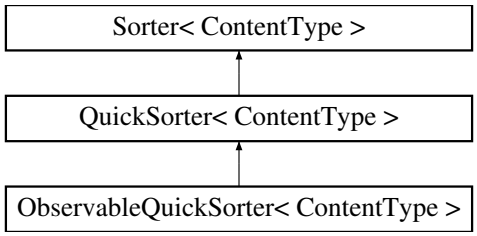
- observer.h

4.18 Dokumentacja szablonu klasy QuickSorter< ContentType >

Klasa sluzaca do obslugi sortowania przez Scalanie.

`#include <quicksorter.h>`

Diagram dziedziczenia dla QuickSorter< ContentType >



Metody publiczne

- QuickSorter (List< ContentType > &list)

  Konstruktor.
- virtual ~QuickSorter ()
- void quicksort (int lewy, int prawy)

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

  Sortuje przez Sortowanie szybkie.

Atrybuty publiczne

- int enablePivot
- List< ContentType > & list

  Skopiowana lista do przeprowadzania sortowania.

4.18.1 Opis szczegółowy

template<class ContentType>class QuickSorter< ContentType >

Definicja w linii 18 pliku quicksorter.h.

4.18.2 Dokumentacja konstruktora i destruktora

4.18.2.1 template<class ContentType > QuickSorter< ContentType >::QuickSorter (List< ContentType > & *list*) [inline]

Parametry

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

Definicja w linii 28 pliku quicksorter.h.

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

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

4.18.2.2 template<class ContentType > virtual QuickSorter< ContentType >::~QuickSorter ( ) [inline], [virtual]

Definicja w linii 35 pliku quicksorter.h.

```
00035 {};
```

4.18.3 Dokumentacja funkcji składowych

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

Parametry

lewy	brzeg poszukiwan
prawy	brzeg poszukiwan

Definicja w linii 42 pliku quicksorter.h.

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

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

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

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

Implementuje Sorter< ContentType >.

Definicja w linii 67 pliku quicksorter.h.

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

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

```
00068         {
00069                 //std::cout<<"(QuickSort)";
00070                 quicksort(0, list.size()-1);
00071                 return list;
00072         }
```

4.18.4 Dokumentacja atrybutów składowych

4.18.4.1 template<class ContentType > int QuickSorter< ContentType >::enablePivot

Definicja w linii 21 pliku quicksorter.h.

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

4.18.4.2 template<class ContentType > List<ContentType>& QuickSorter< ContentType >::list

Definicja w linii 23 pliku quicksorter.h.

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

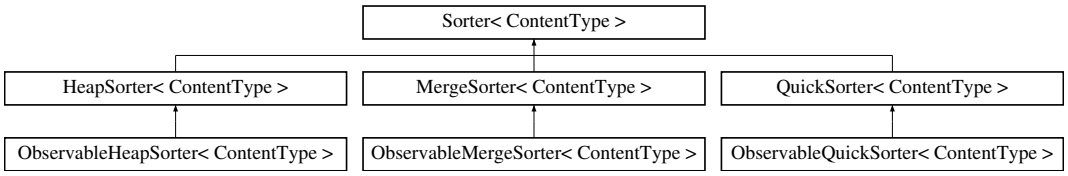
- quicksorter.h

4.19 Dokumentacja szablonu klasy Sorter< ContentType >

interfejs kazdego sortowania

`#include <sorter.h>`

Diagram dziedziczenia dla Sorter< ContentType >



Metody publiczne

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

  Sortuje przez scalanie.

4.19.1 Opis szczegółowy

template<class ContentType>class Sorter< ContentType >

Definicja w linii 15 pliku sorter.h.

4.19.2 Dokumentacja konstruktora i destruktora

4.19.2.1 template<class ContentType > virtual Sorter< ContentType >::~Sorter ( ) [inline], [virtual]

Definicja w linii 23 pliku sorter.h.

```
00023 {};
```

4.19.3 Dokumentacja funkcji składowych

4.19.3.1 template<class ContentType > virtual List<ContentType>& Sorter< ContentType >::sort ( ) [pure virtual]

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

Dokumentacja dla tej klasy została wygenerowana z pliku:

- sorter.h

5 Dokumentacja plików

5.1 Dokumentacja pliku bfs.h

```
#include <iostream>
#include "linkedlist.h"
#include "graph.h"
```

Komponenty

- class BFS

Definicje

- #define MAX 128

5.1.1 Dokumentacja definicji

5.1.1.1 #define MAX 128

Definicja w linii 14 pliku bfs.h.

Odwołania w BFS::clean().

5.2 bfs.h

```
00001 /*
00002  * bsf.h
00003  *
00004  *  Created on: May 28, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef BSF_H_
00009 #define BSF_H_
00010
00011 #include <iostream>
00012 #include "linkedlist.h"
00013 #include "graph.h"
00014 #define MAX 128
00015
00016 using namespace std;
00017
00018 class BFS : public Graph
00019 {
00020 public:
00021         int pi[MAX];
00022     LinkedList<int> q;
00023
00024     BFS()
00025     {
00026                 clean();
00027     }
00028     virtual ~BFS()   {        }
00029
00030     void bfs(int fromNumber, int toNumber)
00031     {
00032         colour[fromNumber] = 1;
00033         pi[fromNumber]     = 1; // czyli brak rodzica
00034         q.push_back( fromNumber );
00035         while ( q.size() ) {
00036                 int u = q[0];
00037                 q.pop_front();
00038                 for( int v = 0; v < biggestValue; v++ ) {
00039                         //cout<<"(test) colour:"<<colour[v]<<"\t"<<M[u][v]<<endl;
00040                         if( colour[v] == 0 && M[u][v] ) {
00041                                 colour[v] = 1;
00042                                 pi[v] = u;          // zapisuje gdzie znajduje sie rodzic danej liczby
00043                                 if(toNumber == v) return;
00044                                 q.push_back( v );
00045                         }
00046                 }
```

```
00047                 colour[u] = 2;
00048         }
00049     }
00050
00051     void findShortestPath(int fromNumber, int toNumber)
00052     {
00053         LinkedList<int> listaDoZmianyKolejnosci;
00054         int tmp=toNumber;
00055         bfs(fromNumber, toNumber);
00056         listaDoZmianyKolejnosci.push_front(toNumber);
00057         while(1)
00058         {
00059                 listaDoZmianyKolejnosci.push_front(pi[tmp]);
00060                 tmp = pi[tmp];
00061                 if(tmp == fromNumber) break;
00062         }
00063         listaDoZmianyKolejnosci.print();
00064     }
00065
00066
00067
00068     void clean()
00069     {
00070         num = 0;
00071         biggestValue = 0;
00072         for (int i = 0; i < MAX; i++)
00073         {
00074                 for (int j = 0; j < MAX; j++)        M[i][j] = 0;
00075
00076                 colour[i] = 0;    // ustawia numery jako biale
00077                 pi[i] = 1;   // ustawia rodzicow na 1
00078
00079         }
00080     }
00081 };
00082
00083 #endif /* BSF_H_ */
```

5.3 Dokumentacja pliku dfs.h

```
#include <iostream>
#include "linkedlist.h"
#include "graph.h"
```

Komponenty

- class DFS

  Klasa przedstawia przeszukiwanie grafu za pomocą DFS Umożliwia wyszukanie najkrótszej trasy do numeru.

Definicje

- #define MAX 128

5.3.1 Dokumentacja definicji

5.3.1.1 #define MAX 128

Definicja w linii 14 pliku dfs.h.

Odwołania w DFS::clean().

5.4 dfs.h

```
00001 /*
00002  * dfs.h
00003  *
00004  *  Created on: May 28, 2015
```

```
00005  *      Author: serek8
00006  */
00007
00008 #ifndef DFS_H_
00009 #define DFS_H_
00010
00011 #include <iostream>
00012 #include "linkedlist.h"
00013 #include "graph.h"
00014 #define MAX 128
00015
00016 using namespace std;
00017
00022 class DFS:public Graph
00023 {
00024 public:
00025   LinkedList<int> currentPath;
00026   LinkedList<int> allPaths[MAX];
00027   int allPathsAmount;
00028
00029   DFS()
00030   {
00031                 clean();
00032   }
00033   ~DFS()   {        }
00034
00039   void findPath( int u, int p , int searchingValue) {
00040         colour[u] = 1;
00041         //cout<<u<<" zmieniam flage na szara\n";
00042         currentPath.push_back(u);
00043         if(u == searchingValue)
00044                 {
00045                         // Znalazlem TĄ LICZBE !!!
00046                         // Dodaje teraz te liste do tablicy list wygranych allPaths
00047                         allPaths[allPathsAmount++].cloneFrom(currentPath);
00048                        colour[u] = 0;
00049                         return;
00050                 }
00051
00052         num++;
00053         for( int v = 0; v < biggestValue; v++ ) if( M[u][v] && v != p )
00054         {
00055             if( colour[v] == 0 ) {
00056                 //cout<<u<<" -> "<<v<<"\n";
00057                 findPath( v, u , searchingValue);
00058             }
00059             else if( colour[v] == 1 )
00060             {
00061                 cout<<u<<" znalazlem szare wiec sie cofam\n";
00062
00063             }
00064             else
00065             {
00066                 cout<<u<<" znalazlem czarne wiec sie cofam\n";
00067
00068             }
00069             currentPath.pop_back();
00070         }
00071         //zmieniam flage na czarna;
00072         colour[u] = 0;
00073
00074     }
00075
00080 void findShortestPath(int fromNumber, int toNumber)
00081 {
00082         findPath( fromNumber, -1 , toNumber);
00083         int indexOfSmallestList = 0;
00084
00085         for(int i=0; i<allPathsAmount; i++)
00086         {
00087                 if( allPaths[indexOfSmallestList].size() >= allPaths[i].size() )
00088                 {
00089                         indexOfSmallestList = i;
00090                 }
00091         }
00092         allPaths[indexOfSmallestList].print();
00093 }
00094
00095
00096         void clean() {
00097                 //n =0;
00098                 biggestValue = 0;
00099                 for (int i = 0; i < MAX; i++)
00100                         for (int j = 0; j < MAX; j++)
00101                                 M[i][j] = 0;
00102                 for (int i = 0; i < MAX; i++) {
00103                         colour[i] = 0;
```

```
00104                 }
00105                 num = 0;
00106                 allPathsAmount = 0;
00107         }
00108 };
00109
00110 #endif /* DFS_H_ */
```

5.5 Dokumentacja pliku filestreamer.h

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

Funkcje

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

5.5.1 Dokumentacja funkcji

5.5.1.1 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 }
```

5.5.1.2 void writeStringToFile (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 }
```

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

Definicja w linii 23 pliku filestreamer.h.

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

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

Definicja w linii 32 pliku filestreamer.h.

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

5.6 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 writeStringToFile(std::string fileName, std::string textToSave)
00016 {
00017         std::ofstream streamToFile;
00018         streamToFile.open (fileName.c_str(), std::ofstream::app);
00019         streamToFile << std::fixed;
00020         streamToFile << std::setprecision(5) <<textToSave;
00021         streamToFile.close();
00022 }
00023 void writeStringToFile(std::string fileName, double textToSave)
00024 {
00025         std::ofstream streamToFile;
00026         streamToFile.open (fileName.c_str(), std::ofstream::app);
00027         streamToFile << std::fixed;
00028         streamToFile<<std::setprecision(5)  << textToSave;
00029         streamToFile.close();
00030 }
00031
00032 void writeStringToFile(std::string fileName, int textToSave)
00033 {
00034         std::ofstream streamToFile;
00035         streamToFile.open (fileName.c_str(), std::ofstream::app);
00036         streamToFile << std::fixed;
00037         streamToFile <<std::setprecision(5) << textToSave;
00038         streamToFile.close();
00039 }
00040
00041 void clearFile(std::string fileName)
00042 {
00043         std::ofstream streamToFile;
00044         streamToFile.open (fileName.c_str(), std::ofstream::out  | std::ofstream::trunc);
00045         streamToFile.close();
00046 }
00047
00048 #endif /* FILESTREAMER_H_ */
```

5.7 Dokumentacja pliku graph.h

```
#include <iostream>
```

Komponenty

- class Graph

Definicje

- #define MAX 128

5.7.1 Dokumentacja definicji

5.7.1.1 #define MAX 128

Definicja w linii 11 pliku graph.h.

5.8 graph.h

```
00001 /*
00002  * graph.h
00003  *
00004  *  Created on: May 30, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef GRAPH_H_
00009 #define GRAPH_H_
00010 #include <iostream>
00011 #define MAX 128
00012 using namespace std;
00013
00014 class Graph
00015 {
00016 public:
00017          bool M[MAX][MAX];
00018          int colour[MAX];
00019          int biggestValue;    //  @najwieksza liczba jaka zostala uzyta - optymalizacja
00020          int num; // po tylku krokach znajdujemy wszystkie sciezki
00021
00026          void virtual findShortestPath(int fromNumber, int toNumber) = 0;
00027
00028          virtual ~Graph() {}
00029
00036          void addEdge(int a, int b)
00037          {
00038                  M[a][b] = 1;
00039                  M[b][a] = 1;
00040                  //n++;
00041                  if(a>=biggestValue) biggestValue=a+1;
00042                  if(b>=biggestValue) biggestValue=b+1;
00043          }
00044
00048           void print()
00049           {
00050                   cout<<"\t";
00051                   for(int i=0; i<biggestValue; i++)     cout<<i<<"\t";
00052                   cout<<"\n";
00053                   for(int i=0; i<biggestValue; i++)
00054                   {
00055                           cout<<i<<"\t";
00056                           for(int j=0; j<biggestValue; j++)
00057                           {
00058                                   cout<<M[i][j]<<"\t";
00059                           }
00060                           cout<<"\n";
00061                   }
00062           }
00066           void virtual clean() = 0 ;
00067 };
00068
00069
00070 #endif /* GRAPH_H_ */
```

5.9 Dokumentacja pliku heapsorter.h

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

Komponenty

- class HeapSorter< ContentType >

  Klasa sluzaca do obslugi sortowania przez kopcowanie.

5.10 heapsorter.h

```
00001 /*
00002  * heapsorter.h
00003  *
00004  *  Created on: May 12, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef HEAPSORTER_H_
00009 #define HEAPSORTER_H_
00010
00011
00012 #include "sorter.h"
00013 #include "list.h"
00014
00016 template <class ContentType>
00017 class HeapSorter: public Sorter<ContentType>
00018 {
00019 public:
00021         List<ContentType> &list;
00022
00026         HeapSorter(List<ContentType> &myList)
00027         :list(myList.createObjectFromAbstractReference())
00028
00029         {
00030                 this->list.cloneFrom(myList);
00031                 /*this->sizeOfList = myList.sizeOfList;
00032                 this->firstElement = myList.firstElement;
00033                 this->lastElement = myList.lastElement;
00034                 this->iterator=myList.iterator;
00035                 this->isIteratorAfterPop = myList.isIteratorAfterPop;*/
00036         }
00037
00038         virtual ~HeapSorter(){};
00039
00042         List<ContentType> &sort()
00043         {
00044                 int n = this->list.size();
00045             int parent = n/2, index, child, tmp; /* heap indexes */
00046             /* czekam az sie posortuje */
00047             while (1) {
00048                 if (parent > 0)
00049                 {
00050                     tmp = (this->list)[--parent];  /* kobie kopie do tmp */
00051                 }
00052                 else {
00053                     n--;
00054                     if (n == 0)
00055                     {
00056                         return this->list; /* Zwraca posortowane */
00057                     }
00058                     tmp = this->list[n];
00059                     //int tmp = this->list[0];
00060                     this->list[n] = this->list[0];
00061                 }
00062                 index = parent;
00063                 child = index * 2 + 1;
00064                 while (child < n) {
00065                     if (child + 1 < n  &&  this->list[child + 1] > this->
      list[child]) {
00066                         child++;
00067                     }
00068                     if (this->list[child] > tmp) {
00069                         this->list[index] = this->list[child];
00070                         index = child;
00071                         child = index * 2 + 1;
00072                     } else {
00073                         break;
00074                     }
00075                 }
00076                 this->list[index] = tmp;
00077             }
00078             return this->list;
00079         }
00080
00081
00082
```

```
00083 };
00084
00085
00086 #endif /* HEAPSORTER_H_ */
```

5.11 Dokumentacja pliku linkedlist.h

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

Komponenty

- class LinkedList< ContentType >

  Lista dwukierunkowa.

5.12 linkedlist.h

```
00001 /*
00002  * mylist.h
00003  *
00004  *  Created on: Mar 12, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef LINKEDLIST_H_
00009 #define LINKEDLIST_H_
00010
00011 #include <iostream>
00012 #include <string>
00013 #include "linkedlistelement.h"
00014 #include "observer.h"
00015 #include "list.h"
00021 template <class ContentType>
00022 class LinkedList : public List<ContentType>{
00023
00024 public:
00026         int sizeOfList;
00027
00028
00030         LinkedListElement<ContentType> *
      firstElement;
00032         LinkedListElement<ContentType> *
      lastElement;
00033         LinkedListElement<ContentType> *iterator;
00034         int iteratorElementId; // nie ruszac !
00035         int isIteratorAfterPop;
00037
00038         LinkedList()
00039         {
00040                 firstElement = lastElement = new
      LinkedListElement<ContentType>;
00041                 sizeOfList = 0;
00042                 iteratorElementId =0;
00043                 iterator=NULL;
00044                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00045         }
00046
00047         LinkedList(List<ContentType> &list)
00048         {
00049                 firstElement = lastElement = new
      LinkedListElement<ContentType>;
00050                 sizeOfList = 0;
00051                 iteratorElementId =0;
00052                 iterator=NULL;
00053                 isIteratorAfterPop = 1; //to znaczy ze jeszcze raz trzeba bedzie
       sprawdzic pozycje iteratora 1- znaczy ze trzeba sprawdzic
00054                 for(int i=0; i<list.size(); i++)
00055                 {
00056                         this->push_back(list[i]);
```

```
00057                 }
00058         }
00059         virtual ~LinkedList(){};
00060
00065         int &size()
00066         {
00067                 return sizeOfList;
00068         }
00073         ContentType pop_back()
00074         {
00075                 if(!(sizeOfList--)) { sizeOfList=0; std::cerr<<"Nie ma takiego elementu
      \n";}
00076                 ContentType tmpNumber = (*(this -> lastElement)).content;
00077                 LinkedListElement<ContentType> *originLinkedListElement =
      this -> lastElement;
00078                 this -> lastElement = this -> lastElement -> previousElement;
00079                 delete originLinkedListElement;
00080                 isIteratorAfterPop=1;
00081                 return tmpNumber;
00082         }
00087         ContentType pop_front()
00088         {
00089                 if(!(sizeOfList--)) { sizeOfList=0; std::cerr<<"Nie ma takiego elementu
      \n";}
00090                 ContentType tmpNumber = (*(this -> firstElement)).content;
00091                 LinkedListElement<ContentType> *originLinkedListElement =
      this -> firstElement;
00092                 this -> firstElement = this -> firstElement -> nextElement;
00093                 delete originLinkedListElement;
00094                 isIteratorAfterPop=1;
00095                 return tmpNumber;
00096         }
00100         void push_back(ContentType &arg)
00101         {
00102                 //std::cerr<<"\n(push_back): arg.content="<<arg.content;
00103                 LinkedListElement<ContentType> *newLinkedListElement = new
      LinkedListElement<ContentType>(arg);
00104                 if(!sizeOfList++) {firstElement =
      lastElement = newLinkedListElement;}
00105                 //newLinkedListElement -> nextElement = 0;
00106                 newLinkedListElement -> previousElement = this -> lastElement;
00107                 this -> lastElement -> nextElement = newLinkedListElement;
00108                 this->lastElement = newLinkedListElement;
00109         }
00113         void push_front(ContentType &arg)
00114         {
00115                 LinkedListElement<ContentType> *newLinkedListElement = new
      LinkedListElement<ContentType>(arg);
00116                 if(!sizeOfList++) {firstElement =
      lastElement = newLinkedListElement;}
00117                 //newLinkedListElement -> previousElement =  0;
00118                 newLinkedListElement -> nextElement = this -> firstElement;
00119                 this -> firstElement -> previousElement = newLinkedListElement;
00120                 this->firstElement = newLinkedListElement;
00121                 ++iteratorElementId;
00122         }
00127         ContentType &show_front()
00128         {
00129                 return firstElement->content;
00130         }
00135         ContentType &show_back()
00136         {
00137                 return lastElement->content;
00138         }
00139
00140
00144         void  print()
00145         {
00146                 LinkedListElement<ContentType> *elem = (this->
      firstElement);
00147                 std::cout<<"\nWyswietlam liste (size:"<<this->sizeOfList<<"): ";
00148                 for(int i=0; i< this->sizeOfList; i++)
00149                 {
00150                         std::cout<<" "<<elem->content;
00151                         elem = elem->nextElement;
00152                 }
00153         }
00154
00159         ContentType &operator[](int numberOfElement)
00160         {
00161                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00162                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00163                         {
00164                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
      !";
00165                                 return (*iterator).content;
00166                         }
```

```
00167                 if(isIteratorAfterPop)
00168                         {
00169                                 iteratorElementId=0;  // czyli iterator byl zpopowany
00170                                 iterator = firstElement;
00171                                 isIteratorAfterPop=0;
00172                         }
00173                 //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00174                 if((numberOfElement <= iteratorElementId-numberOfElement) &&(
      iteratorElementId-numberOfElement>=0))
00175                 {
00176                         //std::cerr<<"\nJestem w if_1";
00177                         iterator = (this->firstElement);
00178                         iteratorElementId = 0;
00179                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00180                                 iterator = (iterator->nextElement);
00181                 }
00182                 else if(numberOfElement > iteratorElementId)
00183                 {
00184                         //std::cerr<<"\nJestem w if_2";
00185                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00186                                                 iterator = (iterator->nextElement);
00187                 }
00188                 else if( numberOfElement < iteratorElementId)
00189                 {
00190                         //std::cerr<<"\nJestem w if_3";
00191                         for (; iteratorElementId> numberOfElement ;
      iteratorElementId--)
00192                                                 iterator = (iterator->previousElement);
00193                 }
00194                 return (*iterator).content;
00195         }
00196
00197         LinkedListElement<ContentType> &
      getLinkedListElementById(int numberOfElement)
00198         {
00199                 //std::cerr<<"\nJestem w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00200                 if(numberOfElement > (sizeOfList-1)) // jezeli wyszedlem poza liste
00201                         {
00202                                 std::cerr<<"\n! Error indeks o numerze: "<<numberOfElement<<" nie istnieje
       !";
00203                                 return *iterator;
00204                         }
00205                 if(isIteratorAfterPop)
00206                         {
00207                                 iteratorElementId=0;  // czyli iterator byl zpopowany
00208                                 iterator = firstElement;
00209                                 isIteratorAfterPop=0;
00210                         }
00211                 //std::cerr<<"\nsprawdzam w ["<<numberOfElement<<"] iterator="<<iteratorElementId;
00212                 if((numberOfElement <= iteratorElementId-numberOfElement) &&(
      iteratorElementId-numberOfElement>=0))
00213                 {
00214                         //std::cerr<<"\nJestem w if_1";
00215                         iterator = (this->firstElement);
00216                         iteratorElementId = 0;
00217                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00218                                 iterator = (iterator->nextElement);
00219                 }
00220                 else if(numberOfElement > iteratorElementId)
00221                 {
00222                         //std::cerr<<"\nJestem w if_2";
00223                         for (; iteratorElementId< numberOfElement ;
      iteratorElementId++)
00224                                                 iterator = (iterator->nextElement);
00225                 }
00226                 else if( numberOfElement < iteratorElementId)
00227                 {
00228                         //std::cerr<<"\nJestem w if_3";
00229                         for (; iteratorElementId> numberOfElement ;
      iteratorElementId--)
00230                                                 iterator = (iterator->previousElement);
00231                 }
00232                 return *iterator;
00233         }
00234
00238         void insertAfter(ContentType &arg, int iteratorID)
00239         {
00240                 if(iteratorID==0 && this->sizeOfList==0)  {push_front(arg); return;}
00241                 if(iteratorID==this->sizeOfList-1)  {push_back(arg); return;}
00242                 LinkedListElement<ContentType> *newLinkedListElement = new
      LinkedListElement<ContentType>(arg);
00243                 LinkedListElement<ContentType>
00244                         &tmpThis=(*this).getLinkedListElementById(iteratorID),
00245                         &tmpNext=(*this).getLinkedListElementById(iteratorID+1);
```

```
00246                 if(!sizeOfList++) {firstElement =
      lastElement = newLinkedListElement;}
00247                 newLinkedListElement -> nextElement = tmpThis.nextElement;
00248                 newLinkedListElement -> previousElement = &tmpThis;
00249                 tmpThis.nextElement = newLinkedListElement;
00250                 tmpNext.previousElement = newLinkedListElement;
00251                 isIteratorAfterPop=1;
00252         }
00253
00254
00255         //LinkedListElement operator[](int numberOfElement);
00256         //virtual LinkedList<ContentType> sort()
00257         //{
00258         //      std::cerr<<"\nError: Sortowanie z klasy LinkedList !!!";
00259         //      //return m;
00260         //}
00261
00262         LinkedList<ContentType> &operator=(const
      LinkedList<ContentType> &pattern)
00263         {
00264                 //std::cerr<<" @@@";
00265                 this->sizeOfList = pattern.sizeOfList;
00266                 this->firstElement = pattern.firstElement;
00267                 this->lastElement = pattern.lastElement;
00268                 this->iterator=pattern.iterator;
00269                 this->isIteratorAfterPop = pattern.
      isIteratorAfterPop;
00270                 return *this;
00271         }
00272 //      List<ContentType> &operator=(const List<ContentType> &pattern)
00273 //      {
00274 //              std::cerr<<" ###";
00280 //              //this->cloneFrom(pattern);
00281 //              return *this;
00282 //      }
00283
00284 /*      void cloneFrom(LinkedList<ContentType> patternList)
00285         {
00286                 LinkedList<ContentType> &clonedList = *new LinkedList<ContentType>;
00287                 // release memory from main list
00288                 while(this->size()) pop_back();
00289                 for(int i=0; i<patternList.size(); i++)
00290                         clonedList.push_back(patternList[i]);
00291                 *this = clonedList;
00292         }
00293 */
00294
00295         List<ContentType> &createObjectFromAbstractReference
      (/*LinkedList<ContentType> abstractPattern*/)
00296         {
00297                 return *new LinkedList<ContentType>;
00298         }
00299
00300
00301
00302 };
00304
00306
00307
00308
00309 /*class LinkedListObserved : public LinkedList, public Observed
00310 {
00311 public:
00312         void mergeSort(LinkedList m)
00313         {
00314         LinkedList::mergeSort(m);
00315         powiadom();
00316
00317         }
00318         LinkedListObserved(){};
00319         ~LinkedListObserved(){};
00320
00321
00322 };*/
00323
00324 #endif /* MYLIST_H_ */
```

5.13 Dokumentacja pliku linkedlistelement.h

```
#include "linkedlist.h"
```

Komponenty

- class LinkedListElement< ContentType >

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

5.14 linkedlistelement.h

```
00001 /*
00002  * mylistelement.h
00003  *
00004  *  Created on: May 11, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef LINKEDLISTELEMENT_H_
00009 #define LINKEDLISTELEMENT_H_
00010
00011 #include "linkedlist.h"
00012
00014 template <class ContentType>
00015 class LinkedListElement{
00017 public:
00018         //ContentType content;
00020         LinkedListElement *nextElement;
00022         LinkedListElement *previousElement;
00023         ContentType content;
00027         LinkedListElement()
00028         {
00029                 this -> nextElement =0;
00030                 this -> previousElement =0;
00031         }
00036         LinkedListElement(ContentType &arg)
00037         {
00038                 this -> content = arg;
00039                 this -> nextElement =0;
00040                 this -> previousElement =0;
00041                 //std::cerr<<"\n(konstruktor LinkedListElement): content="<<arg;
00042         }
00047         LinkedListElement(const LinkedListElement &linkedListElement)
00048         {
00049                 this->content = linkedListElement.content;
00050                 this->nextElement = linkedListElement.nextElement;
00051                 this->previousElement = linkedListElement.
      previousElement;
00052                 //std::cerr<<"\n(konstruktor kopiujacy LinkedListElement): content="<<content;
00053         }
00057         void set(ContentType arg)
00058         {
00059                 this -> content = arg;
00060         }
00061         //friend class LinkedList;
00062 };
00063 #endif /* LINKEDLISTELEMENT_H_ */
```

5.15 Dokumentacja pliku list.h

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

Komponenty

- class List< ContentType >

5.16 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 "list.h"
00012
00015 template <class ContentType>
00016 class List
00017 {
00018 public:
00022         int virtual &size() = 0;
00023
00024         // Zaczerpnite z wzorca projektowego Budowniczy
00025
00029         ContentType virtual pop_back()  =       0;
00030
00034         ContentType virtual pop_front()                 =       0;
00035
00038         void virtual print()       =       0;
00039         void virtual push_back(ContentType &arg)        =       0;
00040
00043         void virtual push_front(ContentType &arg)     =       0;
00044
00047         ContentType virtual &operator[](int numberOfElement)          =       0;
00048
00053         void virtual insertAfter(ContentType &arg, int iteratorID)   =       0;
00054
00057         ContentType virtual &show_front()     =       0;
00058
00061         ContentType virtual &show_back()       =       0;
00062
00063
00064         //List<ContentType> virtual &operator=(const List<ContentType> &pattern)        =       0;
00065
00068         void virtual cloneFrom(List<ContentType> &patternList)
00069         {
00070                 // release memory from main list
00071                 while(this->size()) pop_back();
00072                 for(int i=0; i<patternList.size(); i++)
00073                         this->push_back(patternList[i]);
00074         }
00075
00078         List<ContentType> virtual &
      createObjectFromAbstractReference() = 0;
00079
00082         void virtual free(){ while(size()) pop_back(); }
00083         virtual ~List(){};
00084 };
00085
00086
00087
00088 #endif /* LIST_H_ */
```

5.17 Dokumentacja pliku listsaver.h

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

Komponenty

- class ListSaver< ContentType >

5.18 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 ContentType>
```

```
00015 class ListSaver
00016 {
00019         List<ContentType> &list;
00020
00024         ListSaver(MyList<ContentType> &listArgument):
00025                 list(listArgument)
00026         {}
00027
00032         void saveToFile(std::string nazwaPliku)
00033         {
00034                 std::ofstream streamToFile;
00035                 streamToFile.open (nazwaPliku.c_str(), std::ofstream::out);
00036                 for(int i=0; i<list.size() ; i++)
00037                         streamToFile << '{'<<list[i].content<<"} ";
00038                 streamToFile.close();
00039         }
00040
00041 };
00042
00043
00044
00045
00046
00047 #endif /* LISTSAVER_H_ */
```

5.19 Dokumentacja pliku main.cpp

```
#include <iostream>
#include <unistd.h>
#include "dfs.h"
#include "bfs.h"
#include "graph.h"
```

Funkcje

- int main (int argc, char ∗argv[])

5.19.1 Dokumentacja funkcji

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

Definicja w linii 14 pliku main.cpp.

Odwołuje się do Graph::addEdge(), Graph::clean() i Graph::findShortestPath().

```
00015 {
00016
00017         Graph &graf = *new DFS();
00018         graf.addEdge(0, 1);
00019         graf.addEdge(1, 2); graf.addEdge(2, 4);
00020         graf.addEdge(1, 3); graf.addEdge(3, 5); graf.addEdge(5, 6);
00021         graf.addEdge(1, 12); graf.addEdge(12, 16); graf.addEdge(16, 6);
00022         graf.addEdge(12, 6);
00023         graf.findShortestPath(1, 6);
00024         graf.clean();
00025
00026
00027
00028         Graph &grafBFS = *new BFS();
00029         grafBFS.addEdge(1, 2); grafBFS.addEdge(2, 4);
00030         grafBFS.addEdge(1, 3); grafBFS.addEdge(3, 5); grafBFS.
      addEdge(5, 6);
00031         grafBFS.addEdge(1, 12); grafBFS.addEdge(12, 16); grafBFS.
      addEdge(16, 6);
00032         grafBFS.addEdge(12, 6);
00033         grafBFS.findShortestPath(1, 6);
00034
00035
00036         std::cout<<std::endl;
00037
00038         return 0;
00039 }
```

5.20 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 "dfs.h"
00011 #include "bfs.h"
00012 #include "graph.h"
00013
00014 int main(int argc, char *argv[])
00015 {
00016
00017         Graph &graf = *new DFS();
00018         graf.addEdge(0, 1);
00019         graf.addEdge(1, 2); graf.addEdge(2, 4);
00020         graf.addEdge(1, 3); graf.addEdge(3, 5); graf.addEdge(5, 6);
00021         graf.addEdge(1, 12); graf.addEdge(12, 16); graf.addEdge(16, 6);
00022         graf.addEdge(12, 6);
00023         graf.findShortestPath(1, 6);
00024         graf.clean();
00025
00026
00027
00028         Graph &grafBFS = *new BFS();
00029         grafBFS.addEdge(1, 2); grafBFS.addEdge(2, 4);
00030         grafBFS.addEdge(1, 3); grafBFS.addEdge(3, 5); grafBFS.
    addEdge(5, 6);
00031         grafBFS.addEdge(1, 12); grafBFS.addEdge(12, 16); grafBFS.
    addEdge(16, 6);
00032         grafBFS.addEdge(12, 6);
00033         grafBFS.findShortestPath(1, 6);
00034
00035
00036         std::cout<<std::endl;
00037
00038         return 0;
00039 }
```

5.21 Dokumentacja pliku mergesorter.h

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

Komponenty

- class MergeSorter< ContentType >

  Klasa sluzaca do obslugi sortowania przez Scalanie.

5.22 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 #include "linkedlist.h"
00014
00016 template <class ContentType>
00017 class MergeSorter: public Sorter<ContentType> {
00018 public:
00019
```

```
00021         LinkedList<ContentType> &list;
00022
00026         MergeSorter(LinkedList<ContentType> &listArg)
00027         :list(listArg)      {}
00028
00029         virtual ~MergeSorter(){}
00030
00037         LinkedList<ContentType> merge(
      LinkedList<ContentType> left, LinkedList<ContentType> right)
00038         {
00039                 LinkedList<ContentType> result;
00040                 //Gdy jest jeszcze cos do sortowania
00041                 while (left.size() > 0 || right.size() > 0)
00042                 {
00043                         // Jak oba to zamieniamy
00044                         if (left.size() > 0 && right.size() > 0)
00045                         {
00046                                 // Sprawdzam czy zamieniac
00047                                 if (left.show_front() <= right.
      show_front())
00048                                         {
00049                                                 result.push_back(left.
      show_front()); left.pop_front();
00050                                         }
00051                                 else
00052                                 {
00053                                         result.push_back(right.
      show_front()); right.pop_front();
00054                                 }
00055                         }
00056                         // pojedyncze listy (nieparzyse)
00057                         else if (left.size() > 0)
00058                         {
00059                                 for (int i = 0; i < left.size(); i++) result.
      push_back(left[i]); break;
00060                         }
00061                         // pojedyncze listy (nieparzyse- taka sama sytuacja jak wyzej)
00062                         else if ((int)right.size() > 0)
00063                         {
00064                                 for (int i = 0; i < (int)right.size(); i++) result.
      push_back(right[i]); break;
00065                         }
00066                 }
00067                 return result;
00068         }
00074         LinkedList<ContentType> mergeSort(
      LinkedList<ContentType> m)
00075         {
00076                 if (m.size() <= 1) return m; // gdy juz nic nie ma do sotrowania
00077                 LinkedList<ContentType> left, right, result;
00078                 int middle = (m.size()+ 1) / 2; // anty-nieparzyscie
00079                 for (int i = 0; i < middle; i++)
00080                         {
00081                                 left.push_back(m[i]);
00082                         }
00083                 for (int i = middle; i < m.size(); i++)
00084                         {
00085                                 right.push_back(m[i]);
00086                         }
00087                 left = mergeSort(left);
00088                 right = mergeSort(right);
00089                 result = merge(left, right);
00090                 return result;
00091         }
00092
00093
00096         List<ContentType> &sort()
00097         {
00098                 this->list=mergeSort(this->list);
00099                 return this->list;
00100         }
00101
00102 };
00103
00104 #endif /* MERGESORT_H_ */
```

5.23 Dokumentacja pliku mybenchmark.cpp

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

5.24 mybenchmark.cpp

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

5.25 Dokumentacja pliku mybenchmark.h

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

Komponenty

- class MyBenchmark

  Klasa bazowa/interface do testowania algorytmu.
- class MyBenchmarkObserver

  Mybenchmark obserwator Używana jako obserwator klasa sprawdzajaca odpowiednie objekty.

5.26 mybenchmark.h

```
00001 /*
00002  * mybenchmark.h
00003  *
00004  *  Created on: Mar 6, 2015
00005  *      Author: serek8
00006  */
00008 #ifndef MYBENCHMARK_H_
00009 #define MYBENCHMARK_H_
00010
00011 #include <ctime>
00012 #include "observer.h"
00013 #include <iostream>
00020 class MyBenchmark
00021 {
00022 public:
00023
00025         double timerValue;
00026
00027         MyBenchmark()
00028         {
00029                 timerValue = 0;
00030         }
00031
00033         void timerStart();
00034
00039         double timerStop();
00040
00044         virtual ~MyBenchmark() {};
00045         //using DataFrame::operator=;
00046 };
00047
00052 class MyBenchmarkObserver : public MyBenchmark, public
      Observer
00053 {
00054 public:
00055         MyBenchmarkObserver(){};
```

```
00056
00060         double getTimerValue() {return this->timerValue;}
00061
00064         void receivedStartUpdate () {
00065                 timerStart();
00066         }
00067
00070         void receivedStopUpdate () {
00071                 // std::cout<<"\nCzas wykonywania operacji: "<<timerStop();
00072         }
00073         virtual ~MyBenchmarkObserver(){};
00074
00075 };
00076
00077
00078
00079 #endif /* MYBENCHMARK_H_ */
```

5.27 Dokumentacja pliku numbergenerator.h

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

Komponenty

- class NumberGenerator

  Klasa generujaca losowe liczby.

Definicje

- #define MAX_HEX_ASCII_KOD 127
- #define ROZMIAR_STRINGU 20

5.27.1 Dokumentacja definicji

5.27.1.1 #define MAX_HEX_ASCII_KOD 127

Definicja w linii 17 pliku numbergenerator.h.

5.27.1.2 #define ROZMIAR_STRINGU 20

Definicja w linii 18 pliku numbergenerator.h.

5.28 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 "linkedlist.h"
00015 #include <string>
00016
00017 #define MAX_HEX_ASCII_KOD 127
```

```
00018 #define ROZMIAR_STRINGU 20
00019
00027 class NumberGenerator
00028 {
00029 public:
00032 template <typename ContentType>
00033 LinkedList<ContentType> static &generateNumbers(int range, int
      quantity)
00034 {
00035         LinkedList<ContentType> &myList = *new
      LinkedList<ContentType>();
00036         time_t randomTime = clock();
00037         int randomNumber;
00038         for(int i=0; i<quantity ; i++)
00039         {
00040                 srand (randomTime = clock());
00041                 randomNumber = rand()%range;
00042                 myList.push_back(randomNumber);
00043                 randomTime = clock();
00044         }
00045         return myList;
00046 }
00047
00054 static std::string *generateStrings(int ileStringow);
00055
00056
00057
00058 //using DataFrame::operator=;
00059
00060 };
00061
00062 #endif /* NUMBERGENERATOR_H_ */
```

5.29 Dokumentacja pliku observable.h

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

Komponenty

- class Observable

  Klasa abstrakcyjna- bazowa dla objektow do obserowania.

5.30 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 "linkedlist.h"
00013
00016 class Observable {
00017 public:
00019         LinkedList<Observer*> observaters;
00020
00023     void add(Observer *obserwator) {
00024         observaters.push_back(obserwator);
00025     }
00026
00029     void sendStartUpdateToObservers () {
00030         for(int i=0; i<observaters.size(); i++)
00031         {
00032                 //std::cout<<"Wysylam start update";
00033                 observaters[i]->receivedStartUpdate();
00034         }
00035     }
00036
00039     void sendStopUpdateToObservers () {
```

```
00040         for(int i=0; i<observaters.size(); i++)
00041                 observaters[i]->receivedStopUpdate();
00042     }
00043
00044         virtual ~Observable(){}
00045
00046
00047
00048 };
00049
00050 #endif /* OBSERVABLE_H_ */
```

5.31 Dokumentacja pliku observableheapsorter.h

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

Komponenty

- class ObservableHeapSorter< ContentType >

  Klasa sluzaca do obslugi sortowania przez kopcowanie z dodaniem obserwatora.

5.32 observableheapsorter.h

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

5.33 Dokumentacja pliku observablemergesorter.h

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

Komponenty

- class ObservableMergeSorter< ContentType >

  Klasa sluzaca do obslugi sortowania przez Scalanie z dodaniem obserwatora.

5.34 observablemergesorter.h

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

5.35 Dokumentacja pliku observablequicksorter.h

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

Komponenty

- class ObservableQuickSorter< ContentType >

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

5.36 observablequicksorter.h

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

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

5.37 Dokumentacja pliku observer.h

Komponenty

- class Observer

  obserwator

5.38 observer.h

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

5.39 Dokumentacja pliku quicksorter.h

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

Komponenty

- class QuickSorter< ContentType >

  Klasa sluzaca do obslugi sortowania przez Scalanie.

5.40 quicksorter.h

```
00001 /*
00002  * quicksort.h
00003  *
00004  *  Created on: May 12, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef QUICKSORT_H_
00009 #define QUICKSORT_H_
00010
00011 #include "sorter.h"
00012 #include "list.h"
00013 #include <iostream>
00014
00015
00017 template <class ContentType>
00018 class QuickSorter : public Sorter<ContentType>
00019 {
00020 public:
00021         int enablePivot;
00023         List<ContentType> &list;
00024
00028         QuickSorter(List<ContentType> &list)
00029         :list(list.createObjectFromAbstractReference())
00030         {
00031                 this->list.cloneFrom(list);
00032                 this->enablePivot=1;
00033         }
00034
00035         virtual ~QuickSorter(){};
00036
00042         void quicksort(int lewy, int prawy)
00043         {
00044             int pivot=list[(int)(lewy+prawy)/2];
00045             int i=lewy,j=prawy, x;
00046             if(enablePivot) pivot=(list[(int)(lewy+prawy)/2] +
      list[lewy] + list[prawy])/3;
00047             do
00048             {
00049                 while(list[i]<pivot) {i++; }
00050                 while(list[j]>pivot) {j--; }
00051                 if(i<=j)
00052                 {
00053                         x =list[i];
00054                     list[i]=list[j];
00055                     list[j]=x;
00056                     i++;
00057                     j--;
00058                 }
00059             }
00060             while(i<=j);
00061             if(j>lewy) quicksort(lewy, j);
00062             if(i<prawy) quicksort(i, prawy);
00063         }
00064
00067         List<ContentType> &sort()
00068         {
00069                 //std::cout<<"(QuickSort)";
00070                 quicksort(0, list.size()-1);
00071                 return list;
00072         }
00073 };
00074
00075
00076
00077 #endif /* QUICKSORT_H_ */
```

5.41 Dokumentacja pliku sorter.h

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

Komponenty

- class Sorter< ContentType >

  interfejs kazdego sortowania

5.42 sorter.h

```
00001 /*
00002  * Sorter.h
00003  *
00004  *  Created on: May 13, 2015
00005  *      Author: serek8
00006  */
00007
00008 #ifndef SORTER_H_
00009 #define SORTER_H_
00010
00011 #include "list.h"
00012
00014 template <class ContentType>
00015 class Sorter
00016 {
00017 public:
00018
00019         virtual List<ContentType> &sort() = 0;
00020
00023         virtual ~Sorter(){};
00024 };
00025
00026
00027 #endif /* SORTER_H_ */
```