

SLL

Generated by Doxygen 1.10.0

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Node< T > Class Template Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	5
3.1.2.1 Node()	5
3.2 Person Struct Reference	6
3.3 SinglyLinkedList< T > Class Template Reference	6
3.3.1 Detailed Description	7
3.3.2 Member Function Documentation	7
3.3.2.1 get()	7
3.3.2.2 insertNode()	7
3.3.2.3 ReadFromFile()	7
3.3.2.4 SaveToFile()	8
4 File Documentation	9
4.1 C:/source/repos/3831c825-gr02-repo/Project/SLL_proj/classes.h	9
Index	13

Chapter 1

Class Index

1.1 Class List

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

Node< T >	A class representing node	5
Person	6
SinglyLinkedlist< T >	A class representing a SLL	6

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

C:/source/repos/3831c825-gr02-repo/Project/SLL_proj/[classes.h](#) 9

Chapter 3

Class Documentation

3.1 Node< T > Class Template Reference

A class representing node.

```
#include <classes.h>
```

Public Member Functions

- **Node ()**
A default constructor.
- **Node (T data)**
A constructor.
- **~Node ()**
Destructor.

Public Attributes

- **T data**
Variable which holds data stored in the node.
- **std::shared_ptr< Node< T > > next**
Pointer to another node.

3.1.1 Detailed Description

```
template<typename T>  
class Node< T >
```

A class representing node.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Node()

```
template<typename T >  
Node< T >::Node (  
    T data ) [inline]
```

A constructor.

Parameters

<i>data</i>	information to be stored in the node.
-------------	---------------------------------------

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

- C:/source/repos/3831c825-gr02-repo/Project/SLL_proj/classes.h

3.2 Person Struct Reference

Public Attributes

- std::string **name**
- int **age**

The documentation for this struct was generated from the following file:

- C:/source/repos/3831c825-gr02-repo/Project/SLL_proj/SLL_proj.cpp

3.3 SinglyLinkedList< T > Class Template Reference

A class representing a SLL.

```
#include <classes.h>
```

Public Member Functions

- **SinglyLinkedList** ()
Default constructor.
- **SinglyLinkedList** ([SinglyLinkedList](#) &SLL)
Copy constructor.
- **SinglyLinkedList** ([SinglyLinkedList](#) &&SLL)
Move constructor.
- **~SinglyLinkedList** ()
Destructor.
- void **insertNode** (T data)
A function which adds a new element to the container.
- void **print** ()
A function which prints the whole list.
- void **operator=** (const [SinglyLinkedList](#) &SLL)
An overloaded assignment operator.
- [SinglyLinkedList](#) & **operator=** ([SinglyLinkedList](#) &&SLL)
An overloaded move operator.
- std::shared_ptr< [Node](#)< T > > **get** (T val)
A function searches for a specific element in the container.
- void **sort** ()
A function sorting the list in ascending order, bubble sort algorithm is used.
- bool **SaveToFile** (std::string fname)
A function which saves the current state of the structure to a file.
- bool **ReadFromFile** (std::string fname)
A function which loades state of the structure from a file.

3.3.1 Detailed Description

```
template<typename T>
class SinglyLinkedList< T >
```

A class representing a SLL.

3.3.2 Member Function Documentation

3.3.2.1 get()

```
template<typename T >
std::shared_ptr< Node< T > > SinglyLinkedList< T >::get (
    T val ) [inline]
```

A function searches for a specific element in the container.

Parameters

<i>val</i>	value of the searched element.
------------	--------------------------------

Returns

the searched element, if not presented the head is returned.

3.3.2.2 insertNode()

```
template<typename T >
void SinglyLinkedList< T >::insertNode (
    T data ) [inline]
```

A function which adds a new element to the container.

Parameters

<i>data</i>	information to be stored in the new node.
-------------	---

3.3.2.3 ReadFromFile()

```
template<typename T >
bool SinglyLinkedList< T >::ReadFromFile (
    std::string fname ) [inline]
```

A function which loads state of the structure from a file.

Parameters

<i>fname</i>	name of the file, where data is stored.
--------------	---

Returns

true if succesfuul, false if no.

3.3.2.4 SaveToFile()

```
template<typename T >
bool SinglyLinkedList< T >::SaveToFile (
    std::string fname ) [inline]
```

A function which saves the current state of the structure to a file.

Parameters

<i>fname</i>	name of the file, where data will be stored.
--------------	--

Returns

true if succesfuul, false if no.

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

- C:/source/repos/3831c825-gr02-repo/Project/SLL_proj/classes.h

Chapter 4

File Documentation

4.1 C:/source/repos/3831c825-gr02-repo/Project/SLL_proj/classes.h

```
00001 #pragma once
00002 #ifndef CLASSES_H
00003 #define CLASSES_H
00004 #include <iostream>
00005 #include <fstream>
00006 #include <string>
00007
00008     template <typename T>
00009     class Node
00010     {
00011     public:
00012
00013         T data;
00014         std::shared_ptr< Node<T> > next;
00015
00016         Node()
00017         {
00018             data = 0;
00019             next = nullptr;
00020         }
00021
00022         Node(T data)
00023         {
00024             this->data = data;
00025             this->next = nullptr;
00026         }
00027
00028         ~Node() //destructor
00029         {
00030         };
00031     };
00032
00033     template <typename T>
00034     class SinglyLinkedList
00035     {
00036     public:
00037         std::shared_ptr<Node<T> > head;
00038
00039         SinglyLinkedList() { head = nullptr; };
00040
00041         SinglyLinkedList(SinglyLinkedList& SLL)
00042         {
00043             if (SLL.head != nullptr)
00044             {
00045                 this->head = std::shared_ptr<Node<T> >(new Node<T>);
00046                 this->head->data = SLL.head->data;
00047                 std::shared_ptr<Node<T> > t(SLL.head->next);
00048                 std::shared_ptr<Node<T> > t1(this->head);
00049                 while (t != nullptr)
00050                 {
00051                     std::shared_ptr<Node<T> > NewNode(new Node<T>);
00052                     NewNode->data = t->data;
00053                     t1->next = NewNode;
00054                 }
00055             }
00056         }
00057     };
00058 }
```

```

00071         t1 = t1->next;
00072         t = t->next;
00073     }
00074 }
00075 else
00076 {
00077     head = nullptr;
00078 }
00079 };
00080
00081 SinglyLinkedList(SinglyLinkedList&& SLL) //move constructor
00082 {
00083     std::shared_ptr<Node<T>> t = SLL.head;
00084
00085     if (SLL.head != nullptr)
00086     {
00087         this->head = std::shared_ptr<Node<T>>(new Node<T>);
00088         this->head->data = SLL.head->data;
00089         std::shared_ptr<Node<T>> t(SLL.head->next);
00090         std::shared_ptr<Node<T>> t1(this->head);
00091         while (t != nullptr)
00092         {
00093             std::shared_ptr<Node<T>> NewNode(new Node<T>);
00094             NewNode->data = t->data;
00095             t1->next = NewNode;
00096             t1 = t1->next;
00097             t = t->next;
00098         }
00099         SLL.head = nullptr;
00100     }
00101     else
00102     {
00103         head = nullptr;
00104     }
00105 }
00106 };
00107
00108 ~SinglyLinkedList() //destructor
00109 {
00110 }
00111 };
00112
00113 void insertNode(T data)
00114 {
00115     std::shared_ptr<Node<T>> NewNode(new Node<T>(data));
00116
00117     if (head == nullptr)
00118     {
00119         head = NewNode;
00120         return;
00121     }
00122
00123     std::shared_ptr<Node<T>> t(head);
00124     while (t->next != nullptr)
00125     {
00126         t = t->next;
00127     }
00128     t->next = NewNode;
00129 };
00130
00131 void print()
00132 {
00133     std::shared_ptr<Node<T>> t(head);
00134
00135     if (head == nullptr)
00136     {
00137         std::cout << "List is empty" << std::endl;
00138         return;
00139     }
00140
00141     while (t != nullptr)
00142     {
00143         std::cout << t->data << " ";
00144         t = t->next;
00145     }
00146     std::cout << std::endl;
00147 };
00148
00149 void operator=(const SinglyLinkedList& SLL)
00150 {
00151     if (SLL.head != nullptr) {
00152         this->head = std::shared_ptr<Node<T>>(new Node<T>);
00153         this->head->data = SLL.head->data;
00154         std::shared_ptr<Node<T>> t(SLL.head->next);
00155         std::shared_ptr<Node<T>> t1(this->head);
00156         while (t != nullptr) {

```

```

00165         std::shared_ptr< Node<T> > NewNode(new Node<T>);
00166         NewNode->data = t->data;
00167         t1->next = NewNode;
00168         t1 = t1->next;
00169         t = t->next;
00170     }
00171 }
00172 else
00173 {
00174     head = nullptr;
00175 }
00176 };
00177
00178 SinglyLinkedList& operator=(SinglyLinkedList&& SLL)
00179 {
00180     std::shared_ptr<Node<T> > t(SLL.head);
00181     if (SLL.head != nullptr)
00182     {
00183         this->head = std::shared_ptr<Node<T> >(new Node<T>);
00184         this->head->data = SLL.head->data;
00185         std::shared_ptr<Node<T> > t(SLL.head->next);
00186         std::shared_ptr<Node<T> > t1(this->head);
00187         while (t != nullptr)
00188         {
00189             std::shared_ptr<Node<T> > NewNode(new Node<T>);
00190             NewNode->data = t->data;
00191             t1->next = NewNode;
00192             t1 = t1->next;
00193             t = t->next;
00194         }
00195         SLL.head = nullptr;
00196     }
00197     else
00198     {
00199         head = nullptr;
00200     }
00201     return *this;
00202 };
00203
00204 std::shared_ptr<Node<T> > get(T val)
00205 {
00206     std::shared_ptr<Node<T> > t(head);
00207     if (head == nullptr)
00208     {
00209         std::cout << "List is empty" << std::endl;
00210         return nullptr;
00211     }
00212     while (t != nullptr)
00213     {
00214         if (t->data == val)
00215         {
00216             return t;
00217         }
00218         t = t->next;
00219     }
00220     std::cout << "Element is not presented" << std::endl;
00221     return head;
00222 };
00223
00224 void sort()
00225 {
00226     std::shared_ptr<Node<T> > t(head);
00227     std::shared_ptr<Node<T> > c1(head);
00228     std::shared_ptr<Node<T> > c2(head);
00229     int l = 0;
00230     while (t != nullptr)
00231     {
00232         t = t->next;
00233         l++;
00234     }
00235     for (int i = 0; i < l; i++)
00236     {
00237         for (int j = 0; j < l - 1; j++)
00238         {
00239             if (c1->data < c2->data)
00240             {
00241                 std::swap(c1->data, c2->data);
00242             }
00243         }
00244     }
00245 }

```

```

00258             c2 = c2->next;
00259         }
00260
00261         c2 = head;
00262         c1 = c2->next;
00263         for (int k = 0; k < i; k++)
00264         {
00265             c1 = c1->next;
00266         }
00267     }
00268
00269 };
00270
00271
00272
00273
00274 bool SaveToFile(std::string fname)
00275 {
00276     std::ofstream fout(fname, std::ios::out | std::ios::binary);
00277
00278     std::shared_ptr<Node<T> t(head);
00279     int l = 0;
00280     while (t != nullptr)
00281     {
00282         t = t->next;
00283         l++;
00284     }
00285
00286     t = head;
00287
00288     if (!fout)
00289     {
00290         return false;
00291     }
00292
00293     fout.write((char*)&l, sizeof(l));
00294
00295     while (t != nullptr)
00296     {
00297         fout.write((char*)&t->data, sizeof(t->data));
00298         t = t->next;
00299     }
00300
00301     fout.close();
00302     return true;
00303 };
00304
00305
00306
00307
00308
00309 bool ReadFromFile(std::string fname)
00310 {
00311     std::ifstream fin(fname, std::ios::in | std::ios::binary);
00312     int l;
00313     T buffer;
00314
00315     if (!fin)
00316     {
00317         return false;
00318     }
00319
00320     head = nullptr;
00321
00322     fin.read((char*)&l, sizeof(int));
00323
00324     for (int i = 0; i < l; i++)
00325     {
00326         fin.read((char*)&buffer, sizeof(T));
00327         insertNode(buffer);
00328     }
00329
00330     fin.close();
00331
00332     return true;
00333 };
00334
00335 };
00336
00337
00338
00339
00340 #endif

```


Index

get

 SinglyLinkedList< T >, [7](#)

insertNode

 SinglyLinkedList< T >, [7](#)

Node

 Node< T >, [5](#)

Node< T >, [5](#)

 Node, [5](#)

Person, [6](#)

ReadFromFile

 SinglyLinkedList< T >, [7](#)

SaveToFile

 SinglyLinkedList< T >, [8](#)

SinglyLinkedList< T >, [6](#)

 get, [7](#)

 insertNode, [7](#)

 ReadFromFile, [7](#)

 SaveToFile, [8](#)