Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

Высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики механики

Выполнил:

студент ИИТММ гр. 381706-2

Антипин А.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Список на указателях

Отчет по лабораторной работе

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc532933808)

[Постановка целей и задач 4](#_Toc532933809)

[Руководство пользователя 5](#_Toc532933810)

[Руководство программиста 6](#_Toc532933811)

[Описание структуры программы 6](#_Toc532933812)

[Описание структур данных 7](#_Toc532933813)

[Описание алгоритмов 7](#_Toc532933814)

[Эксперименты 10](#_Toc532933815)

[Результаты работы программы 10](#_Toc532933816)

[Время выполнения 11](#_Toc532933817)

[Заключение 13](#_Toc532933818)

[Литература 14](#_Toc532933819)

[Приложение 15](#_Toc532933820)

# Введение

Стек и очередь являются очень удобными и незаменимыми структурами хранения данных в некоторых задачах, но они имеют один существенный недостаток – ограниченность выделяемой памяти под элементы этих структур. А что если заранее неизвестно сколько элементов будет входить в структуру? Тогда целесообразно создать новую структуру хранения данных типа «Список» или List на английском. Список – это структура хранения данных, в которую можно добавлять и из которой можно удалять элементы сколько угодно раз. Единственное ограничение – это объем физической памяти на компьютере.

Разберем устройство этой структуры данных. Список состоит из элементов списка, которые состоят из самого значения, которое надо хранить, и указателя на следующий элемент списка. По указателю на следующий элемент можно перейти на него и выполнить необходимые действия. Тем самым происходит работа со списком. Единственным недостатком такого способа работы является то, что постоянно приходится использовать циклы, что в случае большого списка замедляет время работы программы.

# Постановка целей и задач

Целью лабораторной работы является создание структуры хранения типа «Список» и методов работы с ним, таких как:

* Добавление элементов в начало, конец и произвольное место списка;
* Удаление элементов из начала и произвольного места списка;
* Получение доступа к элементу списка.

Для реализации алгоритмов будет использоваться 2 шаблонных класса:

* TDatLink;
* List.

Для проверки правильности работы этих классов будут написаны тесты с использованием фреймворка Google Test, а также тестовый образец программы, которая будет использует класс список.

# Руководство пользователя

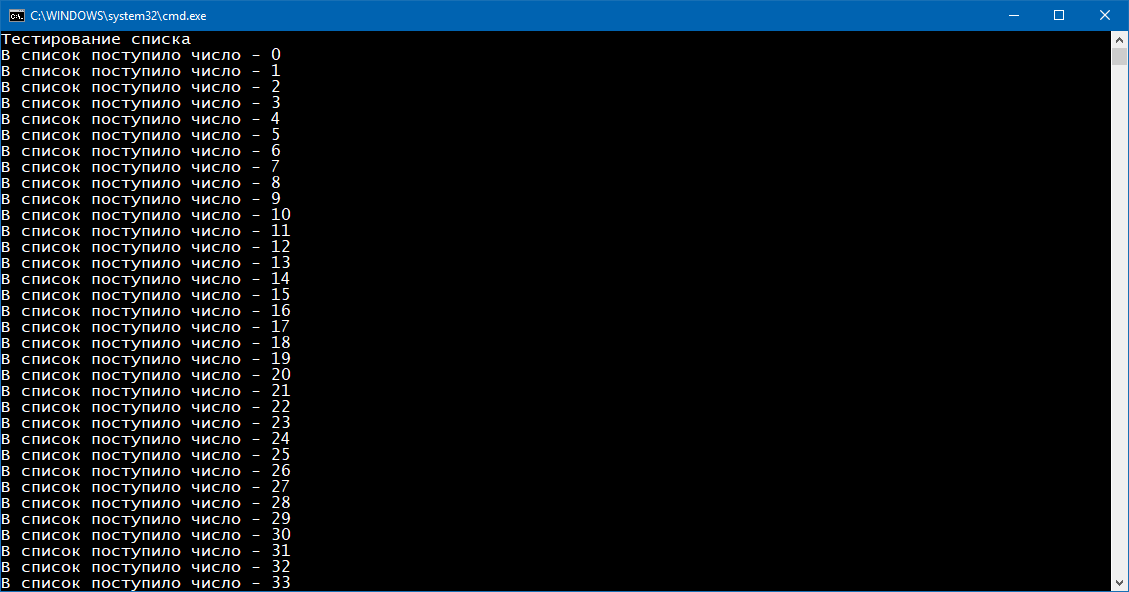
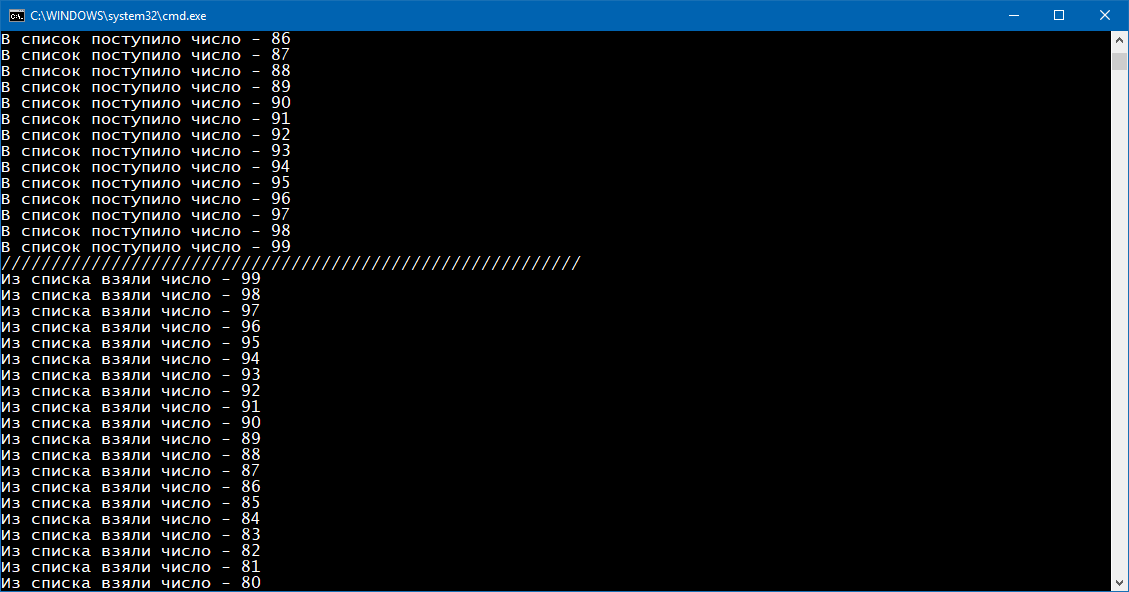
После запуска программы пользователя встречает консольное окно (рис. 1):

рис. 2 (возвращение чисел от 99 до 0)

рис. 1 (вывод программы тестирования списка для пользователя)

в которой сначала в список кладется числа от 0 до 99, а затем значения этих чисел снова выводятся на экран в обратном порядке без удаления (рис. 2).

# Руководство программиста

## Описание структуры программы

Список будет реализован с помощью набора элементов списка:

То есть для реализации алгоритмов будет использовано 2 класса:

* Класс «Звено списка» (TDatLink).
* Класс «Список» (List), который будет использовать класс TDatLink.

А также проект использующий фреймворк Google Test, для проверки правильности работы этих классов и тесовый проект, который будет показываться пользователю.

Древо классов

**Класс gtest**

**test\_main.cpp**

**list\_item\_test.cpp**

**list\_test.cpp**

**Проект для пользователя**

**main.cpp**

**List.h**

**Класс List**

**List\_Item.h**

**Класс TDatLink**

**Класс TDatLink:**

Класс звено списка задает работу с классом List посредством возможности создания элемента этого списка и работы с ним. Также можно выполнять операции, связанные с элементом списка, такие как получение следующего звена, задать указатель на следующий элемент др.

**Класс List:**

Класс List содержит реализацию работы со списками. В нем реализованы такие методы, как положить в список, удалить из списка, посмотреть значение и др.

**Класс gtest:**

Класс gtest реализует тестирование классов TDatLink и List, по средствам фреймворка Google Test. Тесты пишутся для каждого метода классов, каждого ветвления этих методов и для всех возможных исключений этих методов. В файле «list\_item\_test.cpp» реализованы тесты для класса TDatLink, а в файле «list\_test.cpp» соответственно для класса List.

**Проект List:**

В данном проекте реализован примет использования списка, в который поочередно кладутся и забираются элементы.

## Описание структур данных

**Класс TDatLink:**

template <class T> - шаблон класса T

T Mem; - элемент, который нужно сохранить;

TDatLink<T>\* m; - указатель на следующий элемент списка.

**Класс List:**

template< class T > - шаблон класса А

TDatLink<T>\* FirstItem; - указатель на первый элемент списка;

TDatLink<T>\* LastItem; - указатель на текущий последний элемент списка;

int ListLen; - текущее количество элементов в списке.

**Описание методов:**

|  |  |
| --- | --- |
| Метод: | Описание: |
| TDatLink<T>::TDatLink() | Конструктор по умолчанию для класса TDatLink. |
| TDatLink<T>::TDatLink(const T \_Mem, TDatLink<T>\* \_m) | Конструктор с параметром для класса TDatLink, который принимает элемент типа T и указатель на следующий элемент списка. |
| TDatLink<T>::TDatLink(const TDatLink<T>& A) | Конструктор копирования класса TDatLink. |
| void TDatLink<T>::SetMem(T \_Mem) | Метод, позволяющий изменить элемент, хранящийся в звене списка. |
| T TDatLink<T>::GetMem() | Метод, который возвращает элемент, хранящийся в звене списка. |
| void TDatLink<T>::Set\_m(TDatLink<T>\* \_m) | Метод, позволяющий изменить указатель на следующий элемент списка. |
| TDatLink<T>\* TDatLink<T>::GetNextLink() | Метод, который возвращает указатель на следующий элемент списка. |
| List<T>::List() | Конструктор по умолчанию для класса List. |
| List<T>::~List() | Деструктор для класса List. |
| TDatLink<T>\* List<T>::GetLink(T \_Mem, TDatLink<T>\* \_m) | Метод, позволяющий создать звено списка. |
| void List<T>::DelLink(TDatLink<T>\* Link) | Метод, позволяющий удалить звено списка. |
| T List<T>::GetValue(const int Pos) const | Получение значения элемента списка по его номеру. |
| bool List<T>::IsEmpty() const | Проверка списка на пустоту. |
| int List<T>::GetListLen() const | Возвращение текущей длины списка |
| void List<T>::InstFirst(T A) | Устанавливает элемент типа T в начало списка. |
| void List<T>::InstLast(T A) | Устанавливает элемент типа Т в конец списка. |
| void List<T>::InstCustom(const int Pos, T A) | Устанавливает элемент типа Т в произвольное место списка по номеру Pos, Pos не должен выходить за текущие границы списка. |
| void List<T>::DelFirst() | Удаление первого элемента списка. |
| void List<T>::DelCustom(const int Pos) | Удаление произвольного элемента в списке по его позиции в нем. |
| void List<T>::DelList() | Удаление всего списка целиком. |

## Описание алгоритмов

**Подробное описание некоторых методов**

Установка элемента в произвольное место списка:

* Проверка на выход индекса за текущие границы списка;
* Если индекс не равен текущим границам списка, т.е. 0 и последнему, то:
  + Создается указатель на первый элемент списка, указатель на новый элемент списка и звено списка с помещенным в него элемент типа Т (не указатель);
  + В цикле от 0 до необходимой позиции пропускаются элементы списка, а когда доходим до нужной позиции, то меняем указатели в соседних элементах;
  + Увеличение количества элементов в списке на 1;
* Если индекс равен 0, то вызывается метод вставки в начало списка;
* Если индекс равен текущей длине списка, то вызывается метод вставки в конец списка.

Удаление произвольного элемента:

* Если индекс удаляемого элемента выходит за границу текущей длины списка, то вызывается исключение;
* Если список пуст, то также вызывается исключение;
* Если индекс удаляемого элемента равен 0, то вызывается метод удаления начального элемента списка;
* Если индекс удаляемого элемента равен текущей длине списка, то:
  + Создается указатель на первый элемент списка и на пустой элемент списка;
  + В цикле по указателям доходим до предпоследнего элемента;
  + Присваиваем пустому указателю на элемент списка последний элемент списка и удаляем его;
  + Предпоследнему элементу списка ставим указатель на следующий элемент равный NULL и уменьшаем количество элементов на 1;
* Если индекс находится в диапазоне текущей длины списка, то делаем те же действия, что и в предыдущем пункте, только в пустой указатель записываем не последний элемент, а элемент под нужным индексом и предыдущему элементу присваиваем указатель на элемент больший текущего.

# Эксперименты

## Время выполнения

Рассмотрим время выполнения некоторых методов теоретической сложностью O(n) (время приведено в наносекундах): вставка в список в произвольную позицию (рис. 2) и удаление произвольного элемента списка (для максимально наглядного представления вставка и удаление будут производиться в конце списка)

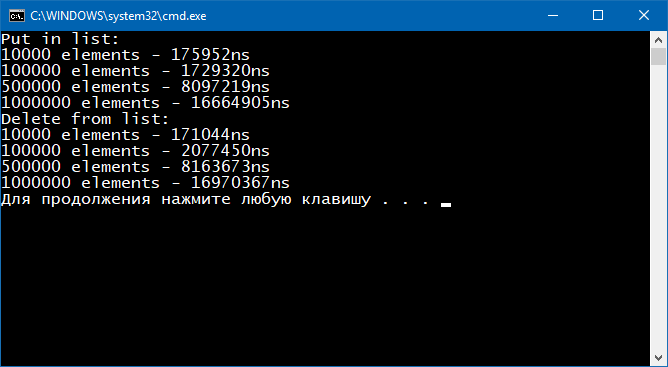
Составим диаграмму с временем работы:

рис. 2 (время работы вставки и удаления элемента из списка)

С учетом погрешности измерения, можно сказать, что теоретическая сложность алгоритмов совпадает с практическими измерениями.

Тесты проводились на системе:

Процессор Intel Core i5 7200U;

Оперативная память 12 GB.

# Заключение

В заключении можно сказать, что все поставленные цели и задачи были выполнены, а именно: созданы классы «TDatLink» и «List» с реализованными методами добавления, удаления и доступа к элементам списка, а также написаны к ним тесты, и они успешно пройдены.

# Литература

* Учебные материалы к учебному курсу «Методы программирования» - Гергель В.П.

# Приложение

**List\_Item.h:**

#pragma once

#include <iostream>

template <class T>

class TDatLink

{

protected:

T Mem;

TDatLink<T>\* m;

public:

TDatLink();

TDatLink(const T \_Mem, TDatLink<T>\* \_m);

TDatLink(const TDatLink<T>& A);

void SetMem(T \_Mem);

T GetMem();

void Set\_m(TDatLink<T>\* \_m);

TDatLink<T>\* GetNextLink();

};

template <class T>

TDatLink<T>::TDatLink()

{

}

template <class T>

TDatLink<T>::TDatLink(const T \_Mem, TDatLink<T>\* \_m)

{

Mem = \_Mem;

m = \_m;

}

template <class T>

TDatLink<T>::TDatLink(const TDatLink<T>& A)

{

Mem = A.Mem;

m = A.m;

}

template <class T>

void TDatLink<T>::SetMem(T \_Mem)

{

Mem = \_Mem;

}

template <class T>

T TDatLink<T>::GetMem()

{

return Mem;

}

template <class T>

void TDatLink<T>::Set\_m(TDatLink<T>\* \_m)

{

m = \_m;

}

template<class T>

TDatLink<T>\* TDatLink<T>::GetNextLink()

{

return m;

}

**List.h:**

#pragma once

#include "List\_Item.h"

#include "Exception\_class.h"

#include <iostream>

Exceptions\_from\_list\_arrlist exception;

template <class T>

class List

{

protected:

TDatLink<T>\* FirstItem;

TDatLink<T>\* LastItem;

int ListLen;

TDatLink<T>\* GetLink(T \_Mem, TDatLink<T>\* \_m);

void DelLink(TDatLink<T>\* Link);

public:

List();

~List();

T GetValue(const int Pos) const;

bool IsEmpty() const;

int GetListLen() const;

void InstFirst(T A);

void InstCustom(const int Pos, T A);

void InstLast(T A);

void DelFirst();

void DelCustom(const int Pos);

void DelList();

friend TDatLink<T>;

};

template <class T>

List<T>::List()

{

FirstItem = NULL;

LastItem = NULL;

ListLen = 0;

}

template <class T>

List<T>::~List()

{

DelList();

}

template <class T>

TDatLink<T>\* List<T>::GetLink(T \_Mem, TDatLink<T>\* \_m)

{

TDatLink<T>\* tmp = new TDatLink<T>(\_Mem, \_m);

if (tmp->GetMem() == NULL && tmp->GetNextLink() == NULL)

exception.except\_throw(101);

else

return tmp;

}

template <class T>

void List<T>::DelLink(TDatLink<T>\* Link)

{

if (Link->GetMem() != NULL && Link->GetNextLink() != NULL)

{

delete Link->Mem;

delete Link->m;

}

else

exception.except\_throw(102);

}

template <class T>

T List<T>::GetValue(const int Pos) const

{

if (Pos >= ListLen || Pos < 0)

exception.except\_throw(103);

TDatLink<T>\* tmp = FirstItem;

for (int i = 0; i < Pos; i++)

tmp = tmp->GetNextLink();

return tmp->GetMem();

}

template <class T>

bool List<T>::IsEmpty() const

{

if (ListLen == 0)

return true;

else

return false;

}

template <class T>

int List<T>::GetListLen() const

{

return ListLen;

}

template <class T>

void List<T>::InstFirst(T A)

{

TDatLink<T>\* tmp = new TDatLink<T>(A, FirstItem);

FirstItem = tmp;

ListLen++;

if (ListLen == 1)

LastItem = tmp;

}

template <class T>

void List<T>::InstLast(T A)

{

TDatLink<T>\* TMP = new TDatLink<T>(A, NULL);

ListLen++;

if (ListLen == 1)

{

FirstItem = TMP;

LastItem = TMP;

}

else

{

LastItem->Set\_m(TMP);

LastItem = TMP;

}

}

template <class T>

void List<T>::InstCustom(const int Pos, T A)

{

if (Pos > ListLen || Pos < 0)

exception.except\_throw(103);

if (Pos != 0 && Pos != ListLen)

{

TDatLink<T>\* tmp\_1 = FirstItem;

TDatLink<T>\* tmp\_2 = new TDatLink<T>(A, NULL);

TDatLink<T> tmp\_3(A, NULL);

for (int i = 0; i < Pos; i++)

if (i == Pos - 1)

{

tmp\_3 = \*tmp\_1;

tmp\_1->Set\_m(tmp\_2);

tmp\_2->Set\_m(tmp\_3.GetNextLink());

}

else

tmp\_1 = tmp\_1->GetNextLink();

ListLen++;

}

else if (Pos == 0)

InstFirst(A);

else if (Pos == ListLen)

InstLast(A);

}

template <class T>

void List<T>::DelFirst()

{

if (IsEmpty() == true)

exception.except\_throw(104);

TDatLink<T>\* tmp = FirstItem;

FirstItem = tmp->GetNextLink();

delete tmp;

ListLen--;

}

template <class T>

void List<T>::DelCustom(const int Pos)

{

if (Pos > ListLen - 1 || Pos < 0)

exception.except\_throw(103);

if (IsEmpty() == true)

exception.except\_throw(104);

if (Pos == 0)

DelFirst();

else if (Pos == ListLen - 1)

{

TDatLink<T>\* tmp = FirstItem;

TDatLink<T>\* TMP = new TDatLink<T>(NULL, NULL);

for (int i = 0; i < Pos - 1; i++)

tmp = tmp->GetNextLink();

TMP = tmp->GetNextLink();

delete TMP;

tmp->Set\_m(NULL);

ListLen--;

}

else

{

TDatLink<T>\* tmp = FirstItem;

TDatLink<T>\* TMP = new TDatLink<T>(NULL, NULL);

for (int i = 0; i < Pos - 1; i++)

tmp = tmp->GetNextLink();

TMP = tmp->GetNextLink();

tmp->Set\_m(tmp->GetNextLink()->GetNextLink());

delete TMP;

ListLen--;

}

}

template <class T>

void List<T>::DelList()

{

while (IsEmpty() == false)

DelFirst();

FirstItem = LastItem = NULL;

}

**list\_item\_test.cpp:**

#include <gtest.h>

#include "List\_Item.h"

TEST(List\_Item, can\_create\_list\_item)

{

ASSERT\_NO\_THROW(TDatLink<int> li(10, NULL));

}

TEST(List\_Item, can\_link\_2\_list\_items)

{

TDatLink<int> li\_1(10, NULL);

ASSERT\_NO\_THROW(TDatLink<int> li\_2(10, &li\_1));

}

TEST(List\_Item, can\_set\_mem\_without\_any\_throw)

{

TDatLink<int> li(10, NULL);

ASSERT\_NO\_THROW(li.SetMem(15));

}

TEST(List\_Item, can\_get\_mem\_without\_any\_throw)

{

TDatLink<int> li(10, NULL);

ASSERT\_NO\_THROW(li.GetMem());

}

TEST(List\_Item, can\_get\_right\_mem)

{

TDatLink<int> li(10, NULL);

EXPECT\_EQ(10, li.GetMem());

}

TEST(List\_Item, can\_set\_item\_and\_return\_it\_right)

{

TDatLink<int> li(10, NULL);

li.SetMem(15);

EXPECT\_EQ(15, li.GetMem());

}

TEST(List\_Item, can\_right\_set\_TDatLink)

{

TDatLink<int> li\_1(10, NULL);

TDatLink<int> li\_2(5, NULL);

ASSERT\_NO\_THROW(li\_2.Set\_m(&li\_1));

}

TEST(List\_Item, can\_set\_right\_TDatLink)

{

TDatLink<int> li\_1(10, NULL);

TDatLink<int> li\_2(5, NULL);

li\_2.Set\_m(&li\_1);

EXPECT\_EQ(&li\_1, li\_2.GetNextLink());

}

**list\_test.cpp:**

#include <gtest.h>

#include "List.h"

TEST(List, can\_create\_list\_without\_throws)

{

ASSERT\_NO\_THROW(List<int> A);

}

TEST(List, can\_return\_true\_if\_list\_is\_empty)

{

List<int> A;

EXPECT\_EQ(A.IsEmpty(), 1);

}

TEST(List, can\_return\_false\_if\_list\_is\_not\_empty)

{

List<int> A;

A.InstFirst(5);

EXPECT\_EQ(A.IsEmpty(), 0);

}

TEST(List, can\_return\_wright\_value)

{

List<int> A;

A.InstFirst(5);

EXPECT\_EQ(A.GetValue(0), 5);

}

TEST(List, can\_return\_list\_lenght)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(5);

A.InstFirst(5);

EXPECT\_EQ(A.GetListLen(), 3);

}

TEST(List, can\_instal\_first\_slot)

{

List<int> A;

ASSERT\_NO\_THROW(A.InstFirst(5));

}

TEST(List, can\_throw\_if\_delete\_first\_item\_in\_empty\_list)

{

List<int> A;

ASSERT\_ANY\_THROW(A.DelFirst());

}

TEST(List, can\_delete\_first\_item)

{

List<int> A;

A.InstFirst(5);

A.DelFirst();

EXPECT\_EQ(1, A.IsEmpty());

}

TEST(List, can\_instal\_last\_slot\_without\_throw)

{

List<int> A;

ASSERT\_NO\_THROW(A.InstLast(5));

}

TEST(List, can\_instal\_last\_slot)

{

List<int> A;

A.InstFirst(10);

ASSERT\_NO\_THROW(A.InstLast(5));

}

TEST(List, can\_throw\_if\_instal\_in\_custom\_slot\_out\_of\_range)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

ASSERT\_ANY\_THROW(A.InstCustom(3, 15));

}

TEST(List, can\_instal\_in\_custom\_slot\_if\_nomber\_is\_0)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

ASSERT\_NO\_THROW(A.InstCustom(0, 15));

}

TEST(List, can\_instal\_in\_custom\_slot)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

ASSERT\_NO\_THROW(A.InstCustom(1, 15));

}

TEST(List, can\_instal\_in\_last\_custom\_slot)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

ASSERT\_NO\_THROW(A.InstCustom(2, 15));

}

TEST(List, can\_return\_a\_custom\_item)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(1, 15);

EXPECT\_EQ(A.GetValue(1), 15);

}

TEST(List, can\_return\_custom\_item\_if\_nomber\_is\_0)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

EXPECT\_EQ(A.GetValue(0), 15);

}

TEST(List, can\_return\_a\_last\_custom\_item)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(2, 15);

EXPECT\_EQ(A.GetValue(2), 15);

}

TEST(List, can\_throw\_if\_delete\_custom\_item\_out\_of\_range)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

ASSERT\_ANY\_THROW(A.DelCustom(3));

}

TEST(List, can\_throw\_if\_delete\_custom\_item\_in\_empty\_list)

{

List<int> A;

ASSERT\_ANY\_THROW(A.DelCustom(3));

}

TEST(List, can\_throw\_if\_delete\_custom\_0)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

ASSERT\_NO\_THROW(A.DelCustom(0));

}

TEST(List, can\_throw\_if\_delete\_custom\_item\_0\_1)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

A.DelCustom(0);

EXPECT\_EQ(10, A.GetValue(0));

}

TEST(List, can\_throw\_if\_delete\_custom)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

ASSERT\_NO\_THROW(A.DelCustom(1));

}

TEST(List, can\_throw\_if\_delete\_custom\_item\_1)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

A.DelCustom(1);

EXPECT\_EQ(5, A.GetValue(1));

}

TEST(List, can\_throw\_if\_delete\_custom\_last)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

ASSERT\_NO\_THROW(A.DelCustom(2));

}

TEST(List, can\_throw\_if\_delete\_custom\_item\_last\_1)

{

List<int> A;

A.InstFirst(5);

A.InstFirst(10);

A.InstCustom(0, 15);

A.DelCustom(2);

ASSERT\_ANY\_THROW(A.GetValue(2));

}

**main.cpp:**

#include "List.h"

#include <iostream>

int main()

{

setlocale(LC\_ALL, "RUS");

List<int> st;

std::cout << "Тестирование списка" << std::endl;

for (int i = 0; i < 100; i++)

{

st.InstLast(i);

std::cout << "В список поступило число - " << i << std::endl;

}

std::cout << "//////////////////////////////////////////////////////////" << std::endl;

for (int i = 99; i >= 0; i--)

{

std::cout << "Из списка взяли число - " << st.GetValue(i) << std::endl;

}

return 0;

}