Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

Высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики механики

Стек на списке

Отчет по лабораторной работе

Выполнил:

студент ИИТММ гр. 381706-2

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

Проверил:

ассистент каф. МОСТ, ИИТММ

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

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc534290272)

[Постановка целей и задач 4](#_Toc534290273)

[Руководство пользователя 5](#_Toc534290274)

[Руководство программиста 6](#_Toc534290275)

[Описание структуры программы 6](#_Toc534290276)

[Описание структур данных 7](#_Toc534290277)

[Описание алгоритмов 8](#_Toc534290278)

[Заключение 9](#_Toc534290279)

[Литература 10](#_Toc534290280)

[Приложение 11](#_Toc534290281)

# Введение

Как уже было сказано раньше одной из самых распространенных структур данных является стек. Он очень удобен при решении некоторых задач на графы и других математических задач. Самой главной проблемой стека является ограниченность выделяемой памяти, а в некоторых задачах требуется принцип доступа к элементам в формате стека, но при этом заранее не известно сколько будет элементов в нем.

Основываться стек на списках будет на элементе списка «TDatLink», который был создан для класса List, но реализовать доступ к данным в формате стека, т.е. элемент вошедший последним должен выйти раньше, чем элемент поведший раньше раньше.

# Постановка целей и задач

Основной целью лабораторной работы является создание структуры данных «Стек на списках» и реализация таких методов, как доступ к элементам стека:

* Положить элемент в стек;
* Вернуть элемент с удалением;

Для реализации алгоритмов будет использоваться шаблонный класс tStackList, который использует класс tDatLink.

Для проверки правильности работы этого класса будут написаны тесты с использованием фреймворка Google Test, а также тестовый образец программы, которая использует данный класс.

# Руководство пользователя

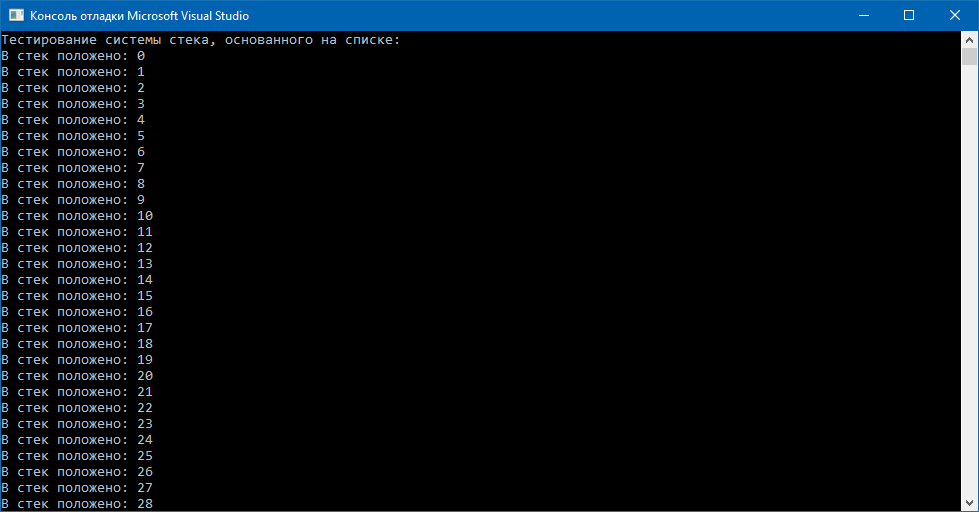
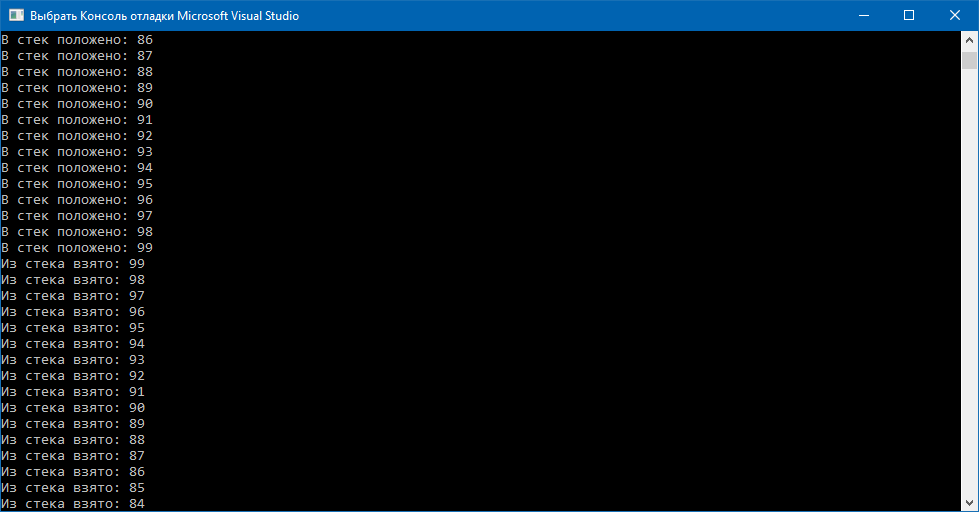
После запуска программы пользователя встречает консольное окно (рис. 1):

рис. 2 (вывод работы стека для пользователя, из стека забираются элементы с 99 до 0)

рис. 1 (вывод работы стека для пользователя, в стек кладутся числа от 0 до 99)

В стек кладутся числа от 0 до 99, а затем из стека забираются эти элементы, причем сначала забирается 99, 98, и т.д., а уже потом 0 (рис. 2).

# Руководство программиста

## Описание структуры программы

Объект типа «стек на списке» будет выделять необходимое количество памяти под объекты по мере добавления этих объектов в стек. Таким образом максимальное количество объектов в стеке будет ограничено только физической объемом памяти компьютера

Для реализации алгоритмов будет создан класс tStackList, одним из полей которого будет являться элемент типа TDatLink.

А также проект использующий фреймворк Google Test, для проверки правильности работы класса данного класса, а также тесовый проект, который будет показываться пользователю.

Древо классов

**Класс gtest**

**test\_main.cpp**

**list\_item\_test.cpp**

**Stack\_list\_test.cpp**

**Проект для пользователя**

**main.cpp**

**Stack\_on\_List.h**

**Класс tStackList**

**List\_Item.h**

**Класс TDatLink**

**Класс TDatLink:**

Этот класс был составлен для реализации структуры данных список, но также он удобен для реализации стека на списках, т.к. они оба имеют схожую структуру работы с данными.

**Класс tStackList:**

В этом классе реализованы методы работы со стеком, такие как: положить элемент, забрать элемент, вернуть текущее количество элементов в стеке, проверить является ли стек пустым или полным в данный момент.

**Класс gtest:**

Класс gtest реализует тестирование класса Stack, по средствам фреймворка Google Test. Тесты пишутся для каждого метода классов, каждого ветвления этих методов и для всех возможных исключений этих методов.

**Проект Stack on list:**

В данном проекте реализован примет использования стека на массивах для заполнения элементами и их последующего забора.

## Описание структур данных

Описание структуры данных TDatLink приведено в отчете работы «Список». Разберем только методы, реализованные в классе tStackList.

**Класс tStackList:**

TDatLink<T>\* pFirstItem – указатель на первый элемент в стеке;

int ListLen – текущее количество элементов в стеке;

Exceptions\_from\_stack\_queue\_multystack exception – переменная для вызова исключений;

**Описание методов:**

|  |  |
| --- | --- |
| Метод: | Описание: |
| tStackList<T>::tStackList() | Конструктор по умолчанию для стека на списках. |
| tStackList<T>::~tStackList() | Деструктор для данного класса. |
| int tStackList<T>::GetLen() | Метод который возвращает текущее количество элементов в стеке. |
| bool tStackList<T>::IsEmpty() | Проверка стека на пустоту. |
| void tStackList<T>::Put(T A) | Метод, который позволяет положить элемент в стек |
| T tStackList<T>::Get() | Метод, возвращает элемент из стека с удалением. |

## Описание алгоритмов

**Подробное описание некоторых методов**

Добавление элемента в стек:

* Создание нового элемента типа TDatLink с указателем на NULL;
* Если в стеке до этого не было элементов, то указателю на первый элемент присваивается наш созданный элемент;
* Если в стеке уже лежали элементы, то по указателям доходим до последнего элемента и присваиваем его указателю на следующий элемент созданный элемент в начале работы метода.

Возврат элемента из стека:

* Проверка на пустоту, если стек пустой, то выдается ошибка о пустоте стека;
* Если в стеке один элемент, то возвращается значение, которое лежало в первом TDatLink.
* В противном случае, по циклу доходим до последнего элемента стека и возвращаем его значение, а сам элемент удаляем.

# Заключение

Подводя итоги можно сказать, что стек на списках лишен главного недостатка обычного стека, а именно ограниченность выделяемой памяти. Тем не менее стек на списках медленнее обычного стека, т.к. постоянно приходится использовать циклы.

# Литература

* Учебные материалы к учебному курсу «Методы программирования» - Гергель В.П.
* http://www.cyberforum.ru/cpp-beginners/thread587712.html

# Приложение

**Stack\_on\_List.h:**

#pragma once

#include "List\_Item.h"

#include <iostream>

#include "Exception\_class.h"

template <class T>

class tStackList

{

protected:

TDatLink<T>\* pFirstItem;

int ListLen;

Exceptions\_from\_stack\_queue\_multystack exception;

public:

tStackList();

~tStackList();

int GetLen();

bool IsEmpty();

void Put(T A);

T Get();

};

template <class T>

tStackList<T>::tStackList()

{

pFirstItem = NULL;

ListLen = 0;

}

template <class T>

tStackList<T>::~tStackList()

{

while (IsEmpty() == false)

{

TDatLink<T>\* tmp = pFirstItem;

pFirstItem = pFirstItem->GetNextLink();

delete tmp;

}

}

template<class T>

int tStackList<T>::GetLen()

{

return ListLen;

}

template <class T>

bool tStackList<T>::IsEmpty()

{

if (pFirstItem == NULL)

return true;

else

return false;

}

template <class T>

void tStackList<T>::Put(T A)

{

TDatLink<T>\* tmp\_1 = new TDatLink<T>(A, NULL);

if (ListLen == 0)

pFirstItem = tmp\_1;

else

{

TDatLink<T>\* tmp\_2 = pFirstItem;

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

tmp\_2 = tmp\_2->GetNextLink();

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

}

ListLen++;

}

template <class T>

T tStackList<T>::Get()

{

if (ListLen == 0)

exception.except\_throw(103);

T A;

if (ListLen == 1)

{

A = pFirstItem->GetMem();

pFirstItem = NULL;

}

else

{

TDatLink<T>\* tmp\_2 = pFirstItem;

for (int i = 0; i < ListLen - 2; i++)

tmp\_2 = tmp\_2->GetNextLink();

A = tmp\_2->GetNextLink()->GetMem();

delete tmp\_2->GetNextLink();

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

}

ListLen--;

return A;

}

**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;

}

**Stack\_list\_test.cpp:**

#include "Stack\_on\_List.h"

#include <gtest.h>

TEST(StackList, can\_create\_stack\_list)

{

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

}

TEST(StackList, can\_put\_it\_empty\_stack\_list)

{

tStackList<int> A;

ASSERT\_NO\_THROW(A.Put(10));

}

TEST(StackList, can\_put\_stack\_list)

{

tStackList<int> A;

A.Put(20);

A.Put(15);

ASSERT\_NO\_THROW(A.Put(10));

}

TEST(StackList, can\_return\_false\_if\_stack\_is\_not\_empty)

{

tStackList<int> A;

A.Put(20);

A.Put(15);

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

}

TEST(StackList, can\_return\_true\_if\_stack\_is\_empty)

{

tStackList<int> A;

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

}

TEST(StackList, can\_throw\_if\_get\_item\_from\_empty\_stack\_list)

{

tStackList<int> A;

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

}

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

{

tStackList<int> A;

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

}

TEST(StackList, can\_return\_stack\_item\_1)

{

tStackList<int> A;

A.Put(20);

ASSERT\_NO\_THROW(A.Get());

}

TEST(StackList, can\_return\_stack\_item)

{

tStackList<int> A;

A.Put(20);

A.Put(15);

ASSERT\_NO\_THROW(A.Get());

}

TEST(StackList, can\_right\_return\_stack\_item)

{

tStackList<int> A;

A.Put(20);

A.Put(15);

EXPECT\_EQ(A.Get(), 15);

}

TEST(StackList, can\_right\_put\_and\_return\_stack\_item)

{

tStackList<int> A;

A.Put(20);

A.Put(15);

A.Get();

A.Put(25);

EXPECT\_EQ(A.Get(), 25);

EXPECT\_EQ(A.Get(), 20);

}

**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());

}

**main.cpp:**

#include "Stack\_on\_List.h"

#include <iostream>

void main()

{

tStackList<int> sl;

std::cout << "Тестирование системы стека, основанного на списке:" << std::endl;

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

{

sl.Put(i);

std::cout << "В стек положено: " << i << std::endl;

}

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

std::cout << "Из стека взято: " << sl.Get() << std::endl;

return 0;

}