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 массивах

Отчет по лабораторной работе

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc536630406)

[Постановка целей и задач 4](#_Toc536630407)

[Руководство пользователя 5](#_Toc536630408)

[Руководство программиста 6](#_Toc536630409)

[Описание структуры программы 6](#_Toc536630410)

[Описание структур данных 7](#_Toc536630411)

[Описание алгоритмов 8](#_Toc536630412)

[Заключение 9](#_Toc536630413)

[Литература 10](#_Toc536630414)

[Приложение 11](#_Toc536630415)

# Введение

Самым главным преимуществом списка является то, что ему не надо знать заранее сколько элементов он будет хранить. Элементы можно добавлять пока не кончится физическая память на компьютере. А что если заранее известно максимальное количество элементов, которое может храниться, такие структуры данных, как стек и очередь являются неудобными для работы с этими данными? Тогда можно попробовать создать новую структуру данных «Список на массивах», в котором известно максимальное количество элементов в списке, и присутствуют разные способы доступа к данным.

Разберем устройство этой структуры данных. Список на массивах состоит из двух массивов, один из которых – это массив элементов, которые необходимо хранить, а второй – это целочисленный массив из индексов элементов первого массива. Данный массив позволяет упорядочивать элементы первого, даже если они в разном порядке. По индексу следующего элемента можно положить или забрать элемент. Тем самым происходит работа со списком.

# Постановка целей и задач

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

* Добавление элементов в начало и конец списка;
* Удаление элементов из начала конца списка с их возвратом;

Для реализации алгоритмов будет использоваться шаблонный класс:

* ArrList.

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

# Руководство пользователя

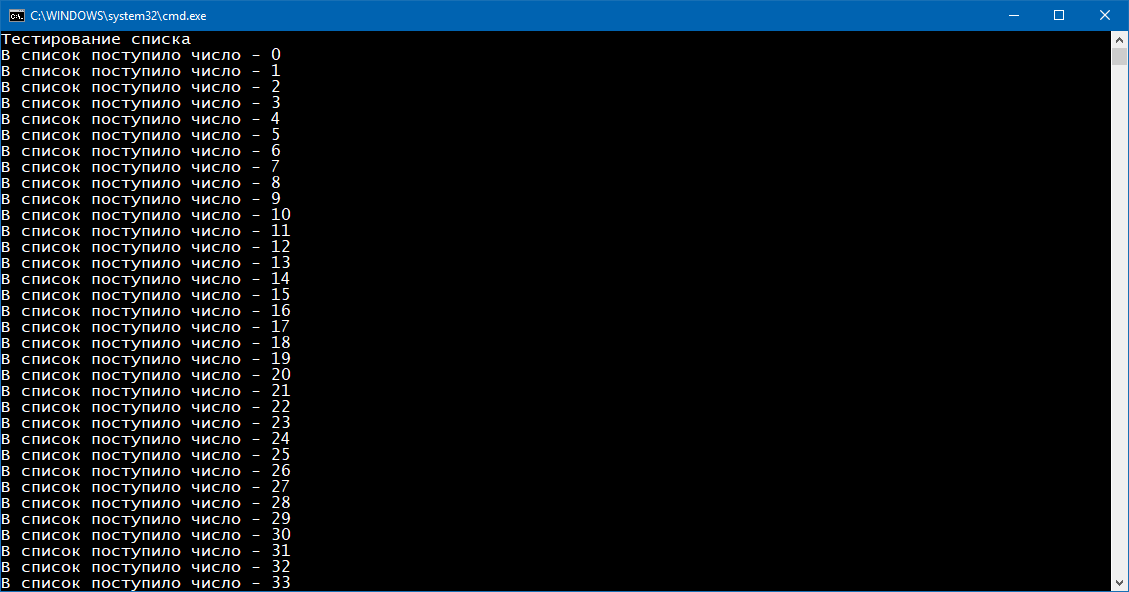
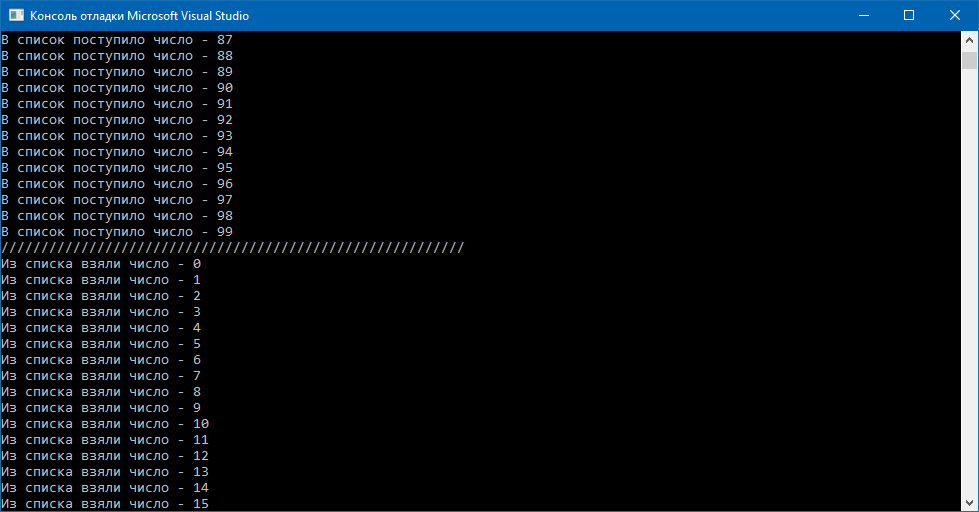
После запуска программы пользователя встречает консольное окно (рис. 1):

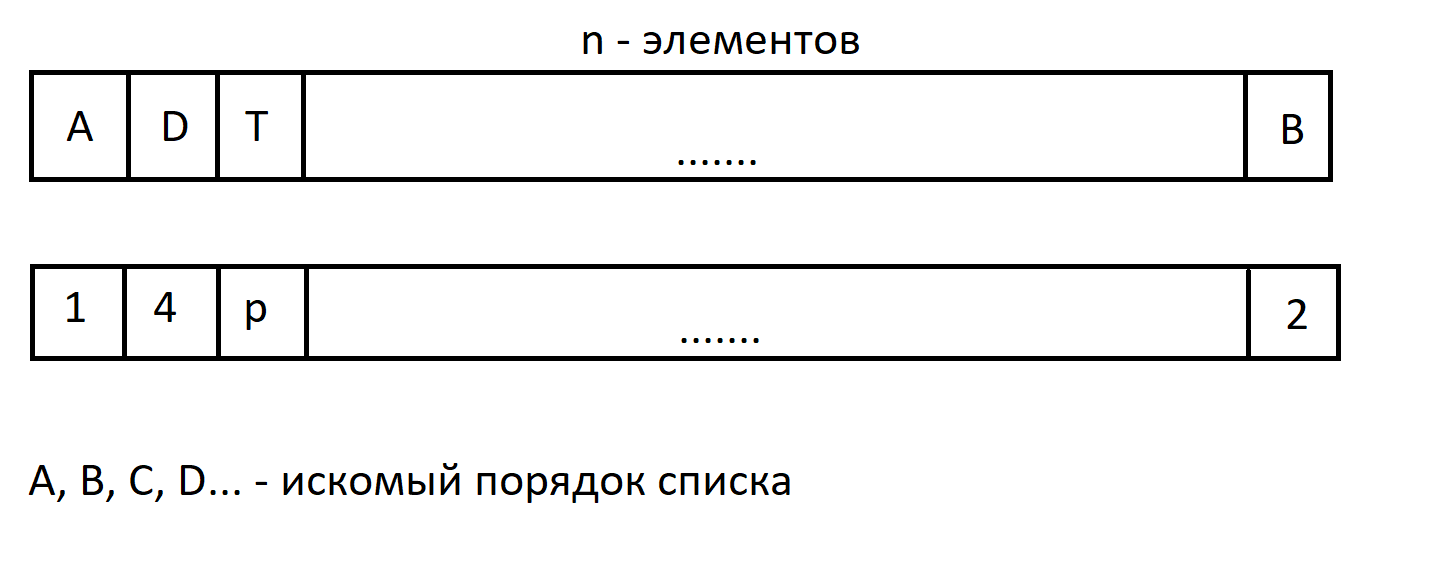
рис. 2 (возвращение чисел от 0 до 99)

рис. 1 (вывод программы тестирования списка для пользователя)

в которой сначала в список кладется числа от 0 до 99, а затем эти числа забираются из списка с удалением (рис. 2).

# Руководство программиста

## Описание структуры программы

Список будет реализован с помощью двух массивов:

Для реализации алгоритмов будет использовано класс ArrList. 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**

**Arr\_list\_test.cpp**

**Проект для пользователя**

**main.cpp**

**Arr\_List.h**

**Класс ArrList**

**Класс List:**

Класс ArrList содержит реализацию работы со списком на массивах. В нем реализованы такие методы, как положить в список, вернуть из списка элемент и др.

**Класс gtest:**

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

**Проект Array List:**

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

## Описание структур данных

**Класс ArrList:**

template< class T > - шаблон класса ArrList

T\* Mas; - указатель для выделения памяти под массив элементов типа T;

int\* Index – указатель для выделения памяти под массив индексов;

int Mem, count, c, size – целочисленные значения: индекс текущего элемента, максимальное количество элементов, индекс текущего свободного слота, количество занятых мест в списке;

**Описание методов:**

|  |  |
| --- | --- |
| Метод: | Описание: |
| ArrList<T>::ArrList(const int n) | Конструктор с параметром для класса ArrList. n – максимальное количество элементов в списке. |
| ArrList<T>::ArrList(const ArrList<T>& A) | Конструктор копирования класса ArrList. |
| ArrList<T>::~ArrList() | Деструктор этого класса. |
| bool ArrList<T>::IsFull() | Метод проверки списка на полноту. |
| bool ArrList<T>::IsEmpty() | Метод проверки списка на пустоту. |
| void ArrList<T>::PutStart(const T& A) | Метод, позволяющий положить элемент в начало списка. |
| void ArrList<T>::PutEnd(const T& A) | Метод, который позволяет положить элемент в конец списка. |
| T ArrList<T>::GetStart() | Метод, который позволяет забрать элемент с удалением из начала списка. |
| T ArrList<T>::GetEnd() | Метод, который позволяет забрать элемент с удалением из конца списка. |

## Описание алгоритмов

**Подробное описание некоторых методов**

Установка элемента в начало списка:

* Проверка списка на полноту;
* Если список не полон, то:
  + В массив элементов записывается элемент в текущую свободную позицию;
  + В массив указателей записывается в текущую свободную позицию индекс текущего элемента (т.е. уже предыдущего);
  + Текущему элементу присваивается значение текущего свободного элемента;
  + Количество занятых элементов увеличивается на единицу;
  + Происходит поиск следующего первого свободного места.

Забрать элемент с удалением из конца списка:

* Происходит проверка списка на пустоту и в случае чего выбрасывается исключение;
* Если в списке всего один элемент, то вызывается метод забора элемента из начала списка;
* Если в списке больше чем один элемент, то:
  + Создаются два целочисленных элемента, один – текущий элемент, второй – следующий по списку за ним элемент;
  + По этим двум элемента находится последний элемент в списке;
  + Индекс последнего элемента освобождается, а сам элемент возвращается пользователю;

# Заключение

В заключении можно сказать, что все поставленные цели и задачи были выполнены, а именно: создан класс «ArrList» с реализованными методами добавления 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

**Arr\_List.h:**

#pragma once

#include <iostream>

#include "Exception\_class.h"

template <class T>

class ArrList

{

protected:

T\* Mas;

int\* Index;

int Mem, count, c, size;//текущий элемент; количество элементов; текущий свободный; размер

Exceptions\_from\_list\_arrlist exception;

public:

ArrList(const int n = 0);

ArrList(const ArrList<T>& A);

~ArrList();

bool IsFull();

bool IsEmpty();

void PutStart(const T& A);

void PutEnd(const T& A);

T GetStart();

T GetEnd();

};

template <class T>

ArrList<T>::ArrList(const int n)

{

if (n < 0)

exception.except\_throw(105);

else if (n == 0)

{

Mas = NULL;

Index = NULL;

Mem = count = c = size = 0;

}

else

{

size = n;

count = c = 0;

Mas = new T[size];

Index = new int[size];

Mem = -1;

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

Index[i] = -2;

}

}

template <class T>

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

{

if (A.size == 0)

{

size = c = Mem = count = 0;

Mas = NULL;

Index = NULL;

}

else

{

size = A.size;

c = A.c;

Mem = A.Mem;

count = A.count;

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

{

Mas[i] = A.Mas[i];

Index[i] = A.Index[i];

}

}

}

template <class T>

ArrList<T>::~ArrList()

{

delete[] Mas;

delete[] Index;

}

template <class T>

bool ArrList<T>::IsFull()

{

if (count == size)

return true;

else

return false;

}

template <class T>

bool ArrList<T>::IsEmpty()

{

if (count == 0)

return true;

else

return false;

}

template <class T>

void ArrList<T>::PutStart(const T& A)

{

if (IsFull() == true)

exception.except\_throw(106);

else

{

Mas[c] = A;

Index[c] = Mem;

Mem = c;

count++;

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

if (Index[i] == -2)

{

c = i;

break;

}

}

}

template <class T>

void ArrList<T>::PutEnd(const T& A)

{

if (IsFull() == true)

exception.except\_throw(106);

else if (IsEmpty() == true)

PutStart(A);

else

{

int i = Mem;

while (Index[i] != -1)

i = Index[i];

Mas[c] = A;

Index[c] = -1;

Index[i] = c;

count++;

for (int k = 0; k < size; k++)

if (Index[k] == -2)

{

c = k;

break;

}

}

}

template <class T>

T ArrList<T>::GetStart()

{

if (IsEmpty() == true)

exception.except\_throw(107);

int i = Mem;

Mem = Index[Mem];

Index[i] = -2;

count--;

return Mas[i];

}

template <class T>

T ArrList<T>::GetEnd()

{

if (IsEmpty() == true)

exception.except\_throw(107);

else if (count == 1)

GetStart();

else

{

int i = Mem;

int j = Index[i];

while (Index[j] != -1)

{

i = j;

j = Index[j];

}

Index[j] = -2;

count--;

c = j;

Index[i] = -1;

return Mas[j];

}

}

**Arr\_list\_test.cpp:**

#include "Arr\_List.h"

#include <gtest.h>

TEST(arr\_list, can\_create\_array\_list)

{

ASSERT\_NO\_THROW(ArrList<int> A(5));

}

TEST(arr\_list, can\_throw\_if\_create\_array\_list\_with\_negative\_size)

{

ASSERT\_ANY\_THROW(ArrList<int> A(-5));

}

TEST(arr\_list, can\_report\_if\_array\_list\_is\_empty)

{

ArrList<int> A(5);

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

}

TEST(arr\_list, can\_report\_if\_array\_list\_is\_not\_empty)

{

ArrList<int> A(5);

A.PutEnd(100);

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

}

TEST(arr\_list, can\_report\_if\_array\_list\_is\_full)

{

ArrList<int> A(5);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

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

}

TEST(arr\_list, can\_report\_if\_array\_list\_is\_not\_full)

{

ArrList<int> A(5);

A.PutEnd(100);

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

}

TEST(arr\_list, can\_put\_element\_to\_start\_of\_array\_list)

{

ArrList<int> A(5);

ASSERT\_NO\_THROW(A.PutStart(100));

}

TEST(arr\_list, can\_throw\_if\_put\_element\_to\_start\_of\_full\_array\_list)

{

ArrList<int> A(5);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

ASSERT\_ANY\_THROW(A.PutStart(100));

}

TEST(arr\_list, can\_return\_element\_from\_start\_of\_array\_list)

{

ArrList<int> A(5);

A.PutEnd(100);

A.PutEnd(100);

A.PutStart(50);

EXPECT\_EQ(A.GetStart(), 50);

}

TEST(arr\_list, can\_throw\_if\_put\_element\_from\_start\_of\_empty\_array\_list)

{

ArrList<int> A(5);

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

}

TEST(arr\_list, can\_put\_element\_to\_the\_end\_of\_array\_list)

{

ArrList<int> A(5);

ASSERT\_NO\_THROW(A.PutEnd(100));

}

TEST(arr\_list, can\_throw\_if\_put\_element\_to\_the\_end\_of\_full\_array\_list)

{

ArrList<int> A(5);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

A.PutEnd(100);

ASSERT\_ANY\_THROW(A.PutEnd(100));

}

TEST(arr\_list, can\_return\_element\_from\_the\_end\_of\_array\_list)

{

ArrList<int> A(5);

A.PutStart(100);

A.PutStart(100);

A.PutEnd(50);

EXPECT\_EQ(A.GetEnd(), 50);

}

TEST(arr\_list, can\_throw\_if\_put\_element\_from\_the\_end\_of\_empty\_array\_list)

{

ArrList<int> A(5);

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

}

**main.cpp:**

#include "List.h"

#include <iostream>

int main()

{

setlocale(LC\_ALL, "RUS");

ArrList<int> al(100);

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

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

{

al.PutStart(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 - " << al.GetEnd() << std::endl;

}

return 0;

}