МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по учебной практике**

**«Список на массивах»**

**Выполнил:** студент группы 381706-1

Корнев Никита Алексеевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc533799739)

[2. Постановка задачи 4](#_Toc533799740)

[3. Руководство пользователя 5](#_Toc533799741)

[4. Руководство программиста 6](#_Toc533799742)

[4.1. Описание структуры программы 6](#_Toc533799743)

[4.2. Описание структур данных 7](#_Toc533799744)

[4.3 Описание алгоритмов 8](#_Toc533799745)

[5. Заключение 9](#_Toc533799746)

[6. Литература 10](#_Toc533799747)

# Введение

Структура данных (англ. data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс. Одной из важных структур является список.

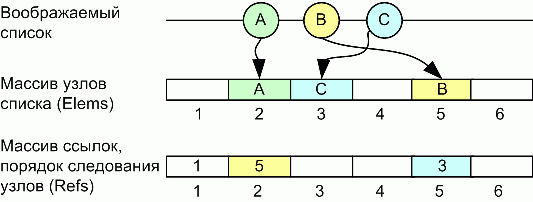
Список — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

Линейный однонаправленный список — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на NULL. Элемент, на который нет указателя, является первым (головным) элементом списка. Здесь ссылка в каждом узле указывает на следующий узел в списке. В односвязном списке можно передвигаться только в сторону конца списка. Узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.

В данной лабораторной работе нам предстоит написать собственную реализацию данной структуры.

# Постановка задачи

1. Написать класс, реализующий список на массивах.
2. Написать тесты на основе Google Tests для проверки работы класса.



# Руководство пользователя

Чтобы использовать данный класс в своем проекте, необходимо подключить библиотеку «ArrayList.h». Библиотека позволяет:

* Создавать объекты типа *список*:

TArrList li1; **конструктор по умолчанию**

TArrList li(10); **конструктор с параметром**

TAList li3 (li22); **конструктор копирования**

* Добавлять элементы в начало, n-ую позицию и конец *списка*:

li.PutStart(T A);

li.PutEnd(T A);

li.Put(n, T A);

* Брать элементы из начала и конца *списка*:

A = li1.GetStart();

B = li1.GetEnd();

* Проверять на *пустоту*:

A = li1.IsEmpty();

* Проверятьна *полноту*:

A = li1.IsFull();

# Руководство программиста

## Описание структуры программы

Проект состоит из следующих модулей:

1. ArrayList

Модуль cодержит файл код «main.cpp», в котором продемонстрирован пример использования структуры.

1. ArrayListLib

Модуль содержит файл заголовок «ArrayList.h», описывающий структуру «список на массивах», а также файл кода «main.cpp». Так как класс шаблонный, main содержит лишь подключение.

1. ArrayListTest

Модуль содержит файлы кода «test\_main.cpp», «test\_ArrayList.cpp». В последнем реализованы тесты для проверки корректности работы методов данного класса.

## Описание структур данных

#### Структура List

Поля со спецификатором доступа «private»:

* ***T\* mas*;** //указатель на область памяти для хранения элементов списка.
* ***int \*nextInd*;** // указатель на область памяти для хранения индексов, указывающих на следующий элемент списка.
* ***int \*predInd***; // указатель на область памяти для хранения индексов, указывающих на предыдущий элемент списка.
* ***int size***; // максимальный размер списка.
* ***int count***; // текущее количество элементов в списке.
* ***int start***; // индекс первого элемента списка.
* ***int end***; // индекс последнего элемента списка.
* ***TQueue <int> freeElem***; // Очередь свободный ячеек в массиве mas.

Поля со спецификатором доступа «public»:

* ***TArrList(int \_size = 1)*** // конструктор по умолчанию.
* ***TArrList(TArrList<T> &A)*** // конструктор копирования.
* ***~ TArrList()*** // деструктор.
* ***void PutStart(T elem)*** // метод, позволяющий добавить элемент в начало списка.
* ***void PutEnd(T elem)*** // метод, позволяющий добавить элемент в конец списка.
* ***void Put(int n, T elem)*** // добавить промежуточный элемент на позицию n в списке
* ***T Get(int n)***// извлечь из списка элемент на позиции n.
* ***T GetStart()*** // метод, позволяющий получить с удалением элемент из начало списка.
* ***T GetEnd()*** // метод, позволяющий получить с удалением элемент из конца списка.
* ***bool IsFull()*** // проверка списка на полноту.
* ***bool IsEmpty()*** // проверка списка на пустоту.

## Описание алгоритмов

Описание некоторых алгоритмов PutStart() и PutEnd():

template <class T>

void TArrList<T>::PutStart(T elem)

{

if (IsFull()) // проверяем список на наличие свободного места

{

MyException ex(4, "Список полон");

throw ex;

}

int ifree = freeElem.Get(); // берем первую свободную ячейку i

mas[ifree] = elem; // в mas записываем значение, которое хотим положить в список

nextInd[ifree] = start; // Определяем, что следующим для этого элемента, будет элемент с текущим индексом start

if (start != -1) // предыдущим для первого элемента списка делаем только что добавленный элемент

predInd[start] = ifree;

else

end = ifree; // определяем, что добавленный элемент является и последним элементом в списке

start = ifree; // индекс start переопределяем на только что добавленный элемент: start = i.

count++; // Увеличиваем количество элементов в списке

}

void TArrList<T>::PutEnd(T elem)

{

if (IsFull()) // проверяем список на наличие свободного места

{

MyException ex(4, "Список полон");

throw ex;

}

int ifree = freeElem.Get(); // берем первую свободную ячейку i

mas[ifree] = elem; // в mas записываем значение, которое хотим положить в список

if (end != -1)

nextInd[end] = ifree; // следующим для последнего элемента списка делаем только что добавленный элемент

else

{

start = ifree; // определяем, что добавленный элемент является первым элементом в списке

predInd[ifree] = -1;

}

predInd[ifree] = end; // определяем, что добавленный элемент является следующим после конца

end = ifree;

count++; // Увеличиваем количество элементов в списке

}

# Заключение

В данном лабораторной работе мне удалось:

* Успешно реализовать класс для списка на массивах
* Продемонстрировать пример использования данного класса
* Написать тесты на основе Google Tests для проверки корректной работы данного класса

# Литература

Интернет-источники:

1. <https://ru.wikipedia.org/wiki/Структура_данных>
2. [https://ru.wikipedia.org/wiki/Список\_(информатика)](https://ru.wikipedia.org/wiki/Очередь_(программирование))
3. <https://habr.com/ru/post/232009/>

Книги:

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.
2. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.