МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**КУРСОВАЯ РАБОТА**

по дисциплине:”Основы алгоритмизации и программирования”

на тему: ***”Алгоритмизация с использованием динамических структур «Списки и очереди». ”***

Выполнила**:** ст. гр. 10701117 Берестнева О.Я.

Приняла**:** ст**.**преподаватель Борисова И.М.

Минск 2018

Белорусский национальный технический университет

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту (работе)**

**по дисциплине «Основы алгоритмизации и программирования»**

Тема ***”Алгоритмизация с использованием динамических структур «Списки и очереди».”***

**Исполнитель**:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Берестнева О.Я.

(подпись)

**Студентка 1 курса 10701117 группы**

**Руководитель** :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Борисова И.М.

(подпись)

Минск 2018

**Оглавление**

[ВВЕДЕНИЕ 5](#_Toc525590242)

[1 Односвязный список 6](#_Toc525590243)

[1.1 Краткая теоретическая часть 6](#_Toc525590244)

[1.2 Постановка задачи 7](#_Toc525590245)

[1.3 Блок-схемы работы алгоритмов 7](#_Toc525590246)

[1.4 Описание программы и ее решение 8](#_Toc525590247)

[2 Очередь 11](#_Toc525590248)

[2.1 Краткая теоретическая часть 11](#_Toc525590249)

[2.2 Постановка задачи 12](#_Toc525590250)

[2.3 Описание программы и ее решение 13](#_Toc525590251)

[2.3 Блок-схемы работы алгоритмов 14](#_Toc525590252)

[3 Тестирование и руководство пользователя 15](#_Toc525590253)

[ЗАКЛЮЧЕНИЕ 20](#_Toc525590254)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 21](#_Toc525590255)

[ПРИЛОЖЕНИЕ А Программная реализация односвязанного списка 22](#_Toc525590256)

[ПРИЛОЖЕНИЕ Б Программаная реализаци очереди 27](#_Toc525590257)

# ВВЕДЕНИЕ

Программирование – процесс и искусство создания компьютерных программ с помощью языков программирования. Программирование сочетает в себе элементы искусства, науки, математики и инженерии.

В узком смысле слова, программирование рассматривается как кодирование – реализация одного или нескольких взаимосвязанных алгоритмов на некотором языке программирования. В более широком смысле, программирование – процесс создания программ, то есть разработка программного обеспечения.

Довольно часто в процессе работы с различными данными невозможно определить, сколько памяти потребуется для их хранения, поэтому память лучше всего распределять во время выполнения программы отдельными блоками. Блоки связываются друг с другом с помощью указателей. Такой способ организации данных называется динамической структурой данных. Это структуры данных, память под которые выделяется и освобождается по мере необходимости.

Динамическая структура данных характеризуется тем что она не имеет имени, ей выделяется память в процессе выполнения программы, количество элементов структуры может не фиксироваться, размерность структуры может меняться в процессе выполнения программы, в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

Достоинствами представления данных является то, что размер структуры ограничивается только доступным объемом машинной памяти; при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей, а также большая гибкость структуры.

Таким образом целью курсовой работы является закрепление и углубление знаний, полученных при изучении курса «Основы алгоритмизации и программирования» посредством разработки программного обеспечения для компьютера. Разработка программы осуществляется при помощи языка программирования C++ в среде разработки программного обеспечения Microsoft Visual 2017.

# 1 Односвязный список

## 1.1 Краткая теоретическая часть

Линейный список — это динамическая структура данных, каждый элемент которой посредством указателя связывается со следующим элементом. Из определения следует, что каждый элемент списка содержит поле данных (оно может иметь сложную структуру) и поле ссылки на следующий элемент. Поле ссылки последнего элемента должно содержать пустой указатель (NULL). Так как ссылка всего одна (только на следующий элемент), то такой список является односвязным.

Когда говорят о линейном списке, то, как правило, подразумевают именно односвязный список.

Каждый узел однонаправленного (односвязного) линейного списка (ОЛС) содержит одно поле указателя на следующий узел (рисунок 1.1). Поле указателя последнего узла содержит нулевое значение (указывает на NULL).



Рисунок 1.1 ­ Графическое представление линейного списка

struct имя\_типа { информационное поле; адресное поле; };

где информационное поле – это поле любого, ранее объявленного или стандартного, типа;

адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.

Основные действия, производимые над элементами ОЛС: инициализация списка, добавление узла в список, удаление узла из списка, удаление корня списка, вывод элементов списка, взаимообмен двух узлов списка.

Функция добавления узла в список принимает два аргумента: указатель на узел, после которого происходит добавление и данные для добавляемого узла.

В качестве аргументов функции удаления элемента ОЛС передаются указатель на удаляемый узел, а также указатель на корень списка. Функция возвращает указатель на узел, следующий за удаляемым.

В качестве аргумента в функцию вывода элементов передается указатель на корень списка.Функция осуществляет последовательный обход всех узлов с выводом их значений.

## 1.2 Постановка задачи

Составить программу, которая содержит динамическую информацию о наличии автобусов в автобусном парке.

Сведения о каждом автобусе включают:

* + номер автобуса;
  + фамилию и инициалы водителя;
  + номер маршрута;
  + признак того, где находится автобус — на маршруте или в парке.

Программа должна обеспечивать:

* начальное формирование данных обо всех автобусах в виде списка;
* при выезде каждого автобуса из парка вводится номер автобуса, и программа устанавливает значение признака «автобус на маршруте»;
* при въезде каждого автобуса в парк вводится номер автобуса, и программа ус­танавливает значение признака «автобус в парке»;
* по запросу выдаются сведения об автобусах, находящихся в парке, или об авто­бусах, находящихся на маршруте.

## 1.3 Блок-схемы работы алгоритмов

Для изменения статуса месторасположения текущего автобуса была разработана следующая блок-схема, представленная на рисунке 1.2



Рисунок 1.2 – Изменения статуса местонахождения автобуса

Данная блок-схема использует поиск введенного значения или номера автобуса, и изменяет статус на маршруте или в парке.

## 1.4 Описание программы и ее решение

Линейный список – это динамическая структура данных, каждый элемент которой связан с последующим при помощи указателя. От этого определения отталкиваемся при реализации данного задания. В программе создаётся пользовательский тип данных на основе структуры Bus (рисунок 1.3):

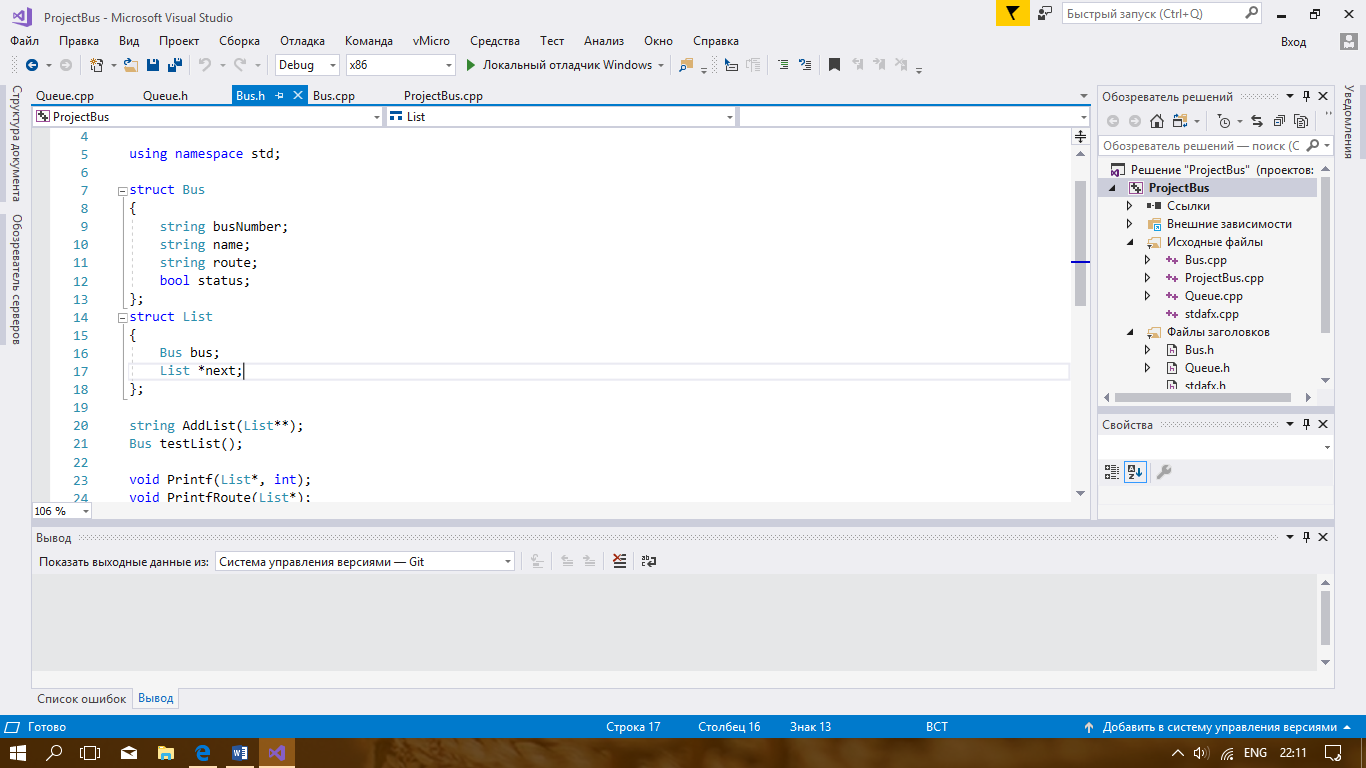


Рисунок 1.3 – Структура проекта

В данной структуре используется несколько полей типа строковой переменной, такие как: номер автобуса, ФИО водителя и маршрут, по которому будет двигаться автобус. Единственны полем, для определения месторасположения автобуса, была использована переменная логического типа, которая определяет, находиться ли данный автобус на маршруте или в парке.

Для описание линейного списка используется структура следующего вида, представленной на рисунке 1.3.

На рисунке 1.3 для хранение информационных полей используется переменная bus, а указателем на следующий элемент списка – переменная next.

Для удобства были разработаны следующие функции:

* AddList – служит для заполнения линейного списка параметрами;
* PrintfRoute – для вывода автобусов, находящиеся на маршруте;
* PrintfPark – для вывода автобусов, находящиеся в парке;
* PrintfList – для вывода всех существующий автобусов
* DeleteBus – для удаление автобуса
* SearchBus – для поиска автобуса
* setRoute – для отправки автобуса на маршрут
* setPark – для возврата автобуса в парк

По заданию нужно реализовать функцию отправки автобуса на маршрут или вернуть автобус в парк, для этого были реализованный два метода, которые изменяют текущее положение. Для этого использует переменная логического типа. Если значение является ложью (false), то автобус прибыл или находится в парке, если истина – то покинул парк или находится на текущем маршруте. На рисунке 1.4 представлен программный код прибытия автобуса в парк.

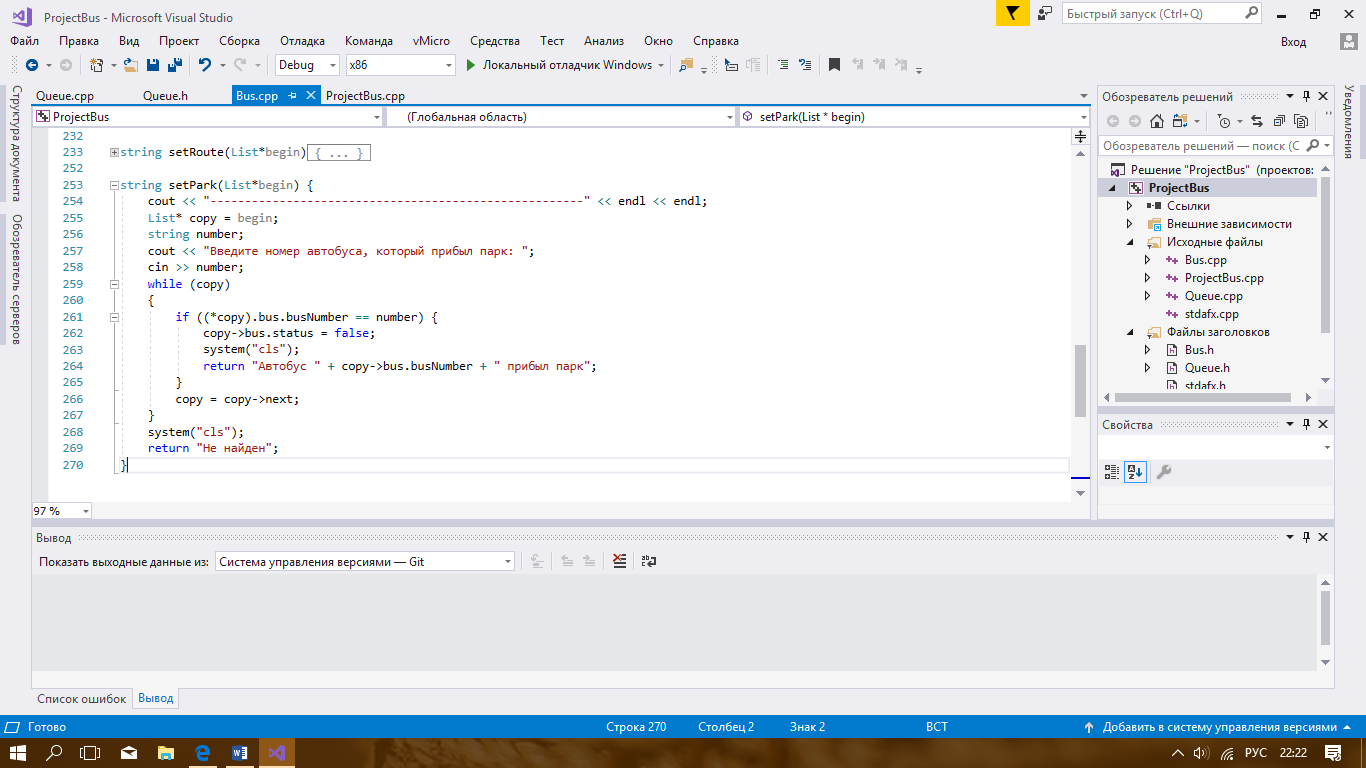


Рисунок 1.4 – Метод прибытия автобуса в парк

Как можно заметить из рисунка 1.4 то, что мы ищем автобус по номеру, если находим такой автобус, то изменяем его текущее положение.

Метод отправки автобуса на маршрут похож на метод прибытия, отличие заключается в том, что при поиске автобуса изменяется логическая переменная на истину.

# 2 Очередь

## 2.1 Краткая теоретическая часть

Очередь – структура данных типа «список», позволяющая добавлять элементы лишь в конец списка, и извлекать их из его начала. Она функционирует по принципу FIFO (First In, First Out — «первым пришёл — первым вышел»), для которого характерно, что все элементы a1, a2, …, an-1, an, добавленные раньше элемента an+1, должны быть удалены прежде, чем будет удален элемент an+1. Также очередь может быть определена как частный случай односвязного списка, который обслуживает элементы в порядке их поступления. Как и в «живой» очереди, здесь первым будет обслужен тот, кто пришел первым (рисунок 2.1).

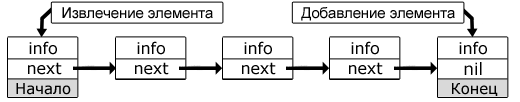


Рисунок 2.1­­­­­­ – Графическое представление очереди

Стандартный набор операций (часто у разных авторов он не идентичен), выполняемых над очередями, совпадает с тем, что используется при обработке стеков:

* добавление элемента;
* удаление элемента;
* чтение первого элемента.

Только, если в отношении стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала. Далее, при реализации интерфейса очереди, список стандартных операций будет расширен.

Выделяют два способа программной реализации очереди. Первый из них основан на базе массива, а второй на базе указателей (связного списка). Первый способ – статический, т. к. очередь представляется в виде простого статического массива, второй – динамический.

## 2.2 Постановка задачи

Вычисления значения многочлена f(x) = a0 + a1 x + a2 x2 + … + an xn  в целочисленной точке х (при этом значения коэффициентов a0, a1, a2, an  вводятся с клавиатуры и динамически размещаются в памяти в форме очереди);

## 2.3 Блок-схемы работы алгоритмов

Для реализации поставленной задачи была создана блок-схема подсчеты суммы ряда (рисунок 2.2).



Рисунок 2.2 – Работа функции подсчета ряда

В данной блок-схеме есть проверка на пустоту очереди. Если очередь не пуста, то идет подсчет суммы ряда.

## 2.4 Описание программы и ее решение

Для реализации поставленной задачи было решено создать две структуры представленной на рисунке 2.3.

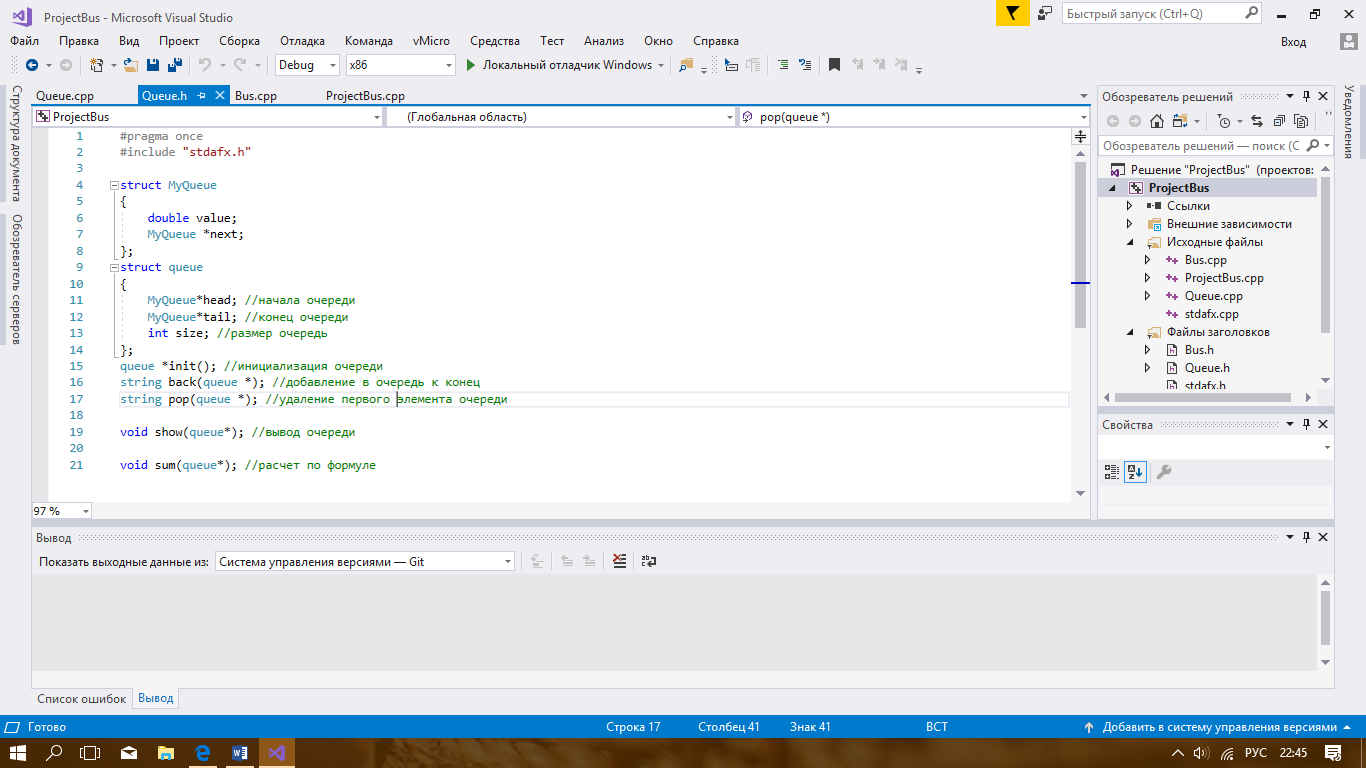


Рисунок 2.3 – Структура очереди

Для облегчения используем числовую переменную value, в которой будет храниться значение переменной очереди, а следующие параметры – для указания на следующий элемент в очереди. Основной принцип работы очереди, что у него есть указатель на начальный элемент и на последний, для этого было создана структура queue, в которой создано указатели на начальный и последний элементы очереди, а также размерность очереди, что является не обязательным параметром в данной структуре и служит для облегчения работы с очередью.

Для облегчения работы с очередью были созданы следующие функции:

* init – инициализация очереди;
* back – метод добавления элемента в конец очереди;
* pop – метод удаления (выхода) первого элемента из очереди;
* show – функция вывода очереди;
* sum – функция расчета по формуле.

При создании очереди первым делом инициализируем переменную используя метод init (рисунок 2.4). Данный метод настаивает указатели для начального и конечного положения элемента очереди, а также размер данной очереди, чтобы не проделывать данные операции самостоятельно. Данный метод возвращает созданную очередь.

## 

Рисунок 2.4 ­– Метод инициализации очереди

В задании просят посчитать сумму ряда. Для этого был создан метод, который рассчитывает данную сумму (рисунок 2.5).

В данном методе есть проверка на пустоту очереди, что облегчает дальнейший расчет. Если очередь не пуста, то начинается подсчёт элементов. Для этого попросят ввести число (целочисленную точку), которая необходима для подсчёта. Потом мы получим получившийся ряд, в которых входят значения очереди и введенного числа в нужной степени.

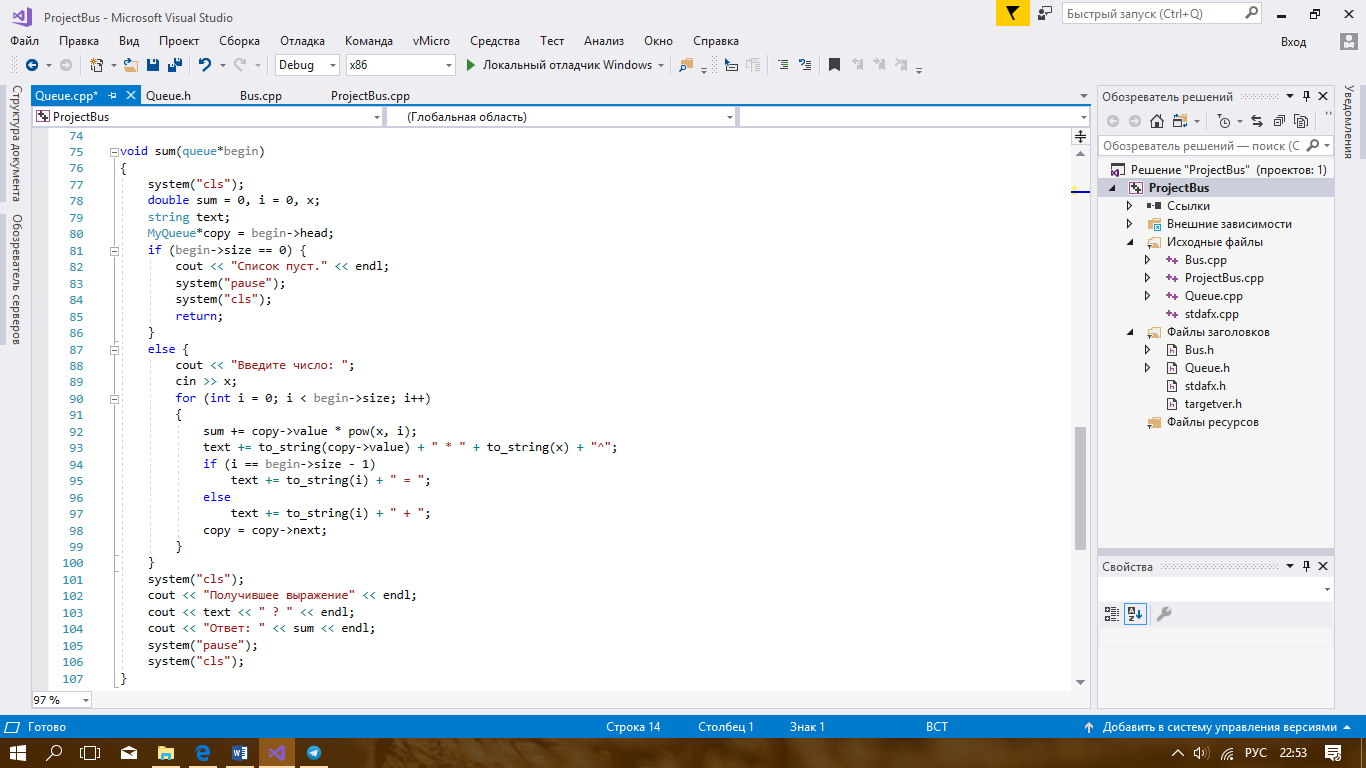


Рисунок 2.5 – Метод нахождение суммы числового ряда

В данной функции присутствуют два условных оператора if, которые служат для определения знака «+» или «=». Если символ очереди находится в конце и больше чисел нету, то выводится завершающий символ «=».

# 3 Тестирование и руководство пользователя

Начальное меню экрана представлена на рисунке 3.1. На данном рисунке можно выбрать одно из двух вариантов – это работа со списком или со стеком, третий вариант служит для выхода из консольного приложения. В случаи нажатии на другие цифры, данное окно обновиться и переходов или исключений не сработает.

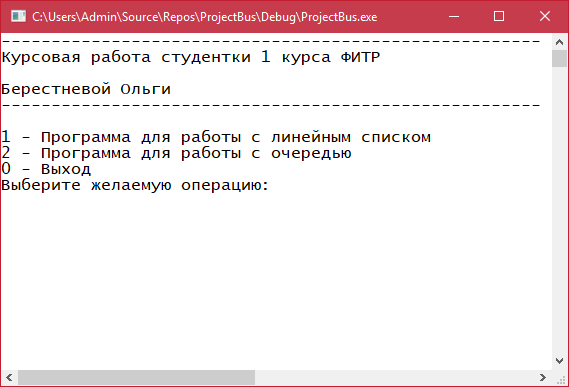


Рисунок 3.1 – Начальное меню приложения

При выборе «работы со списком» будет произведен переход на следующее меню, представленной на рисунке 3.2. В данном окне следует выбрать один из предложенных вариантов. Что и предыдущей окне, при нажатии на цифры, не входящие в данный список, окно будет обновлено.

При выборе опции «Назад» будет совершено перемещение на начальное окно, которое представлено на рисунке 3.1. В случаи выбора опции «Выход» будет совершен выход из самого консольного приложения.

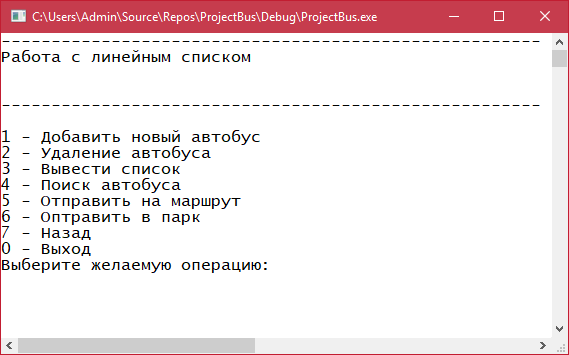


Рисунок 3.2 – Меню работы со списком

Добавление элементов в список происходит при вводе цифры «1», тогда попросят ввести необходимую информацию, представленную на рисунке 3.3. После всех введенных полей будет сделан возврат на начальное окно (рисунок 3.2).

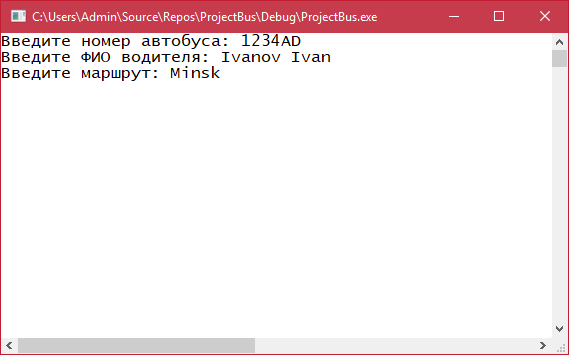


Рисунок 3.3 – Добавление новой записи

Чтобы удалить следует ввести значение «2», дальше выбрать по какому принципу будет происходить удаление (рисунок 3.4). Если передумали удалять, то можно вернуться назад. Если ошибочно было введено не та цифра, то никаких действий не произойдёт.

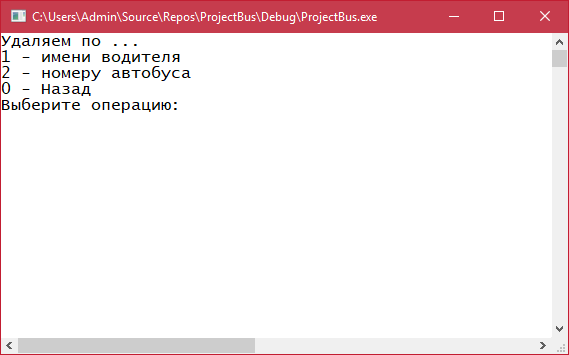


Рисунок 3.4 – Выбор критерия удаления

Если было выбрано удаление по «имени водителя», то попросят ввести ФИО водителя (рисунок 3.5), если выбрать другое – то номер автобуса (рисунок 3.6).

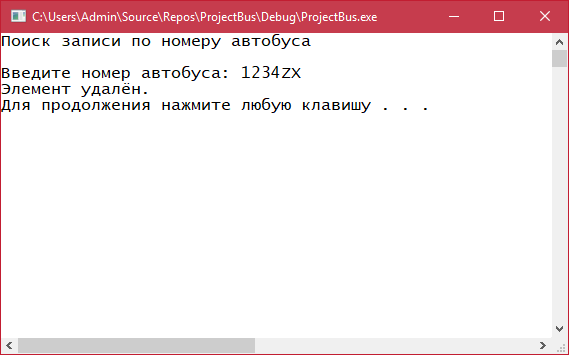


Рисунок 3.5 – Удаление по номеру автобуса

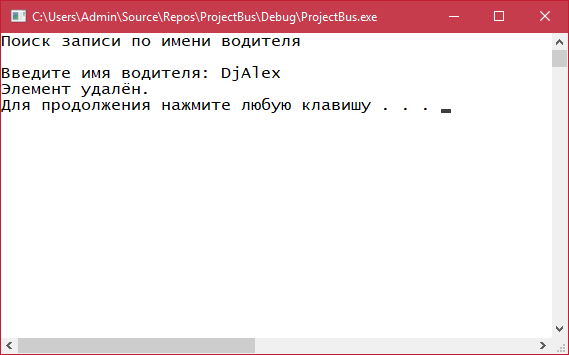


Рисунок 3.6 – Удаление по имени водителя

Для вывода вводится цифра «3» и, как показано на рисунке 3.7, следует по какому принципу выводить наш список.

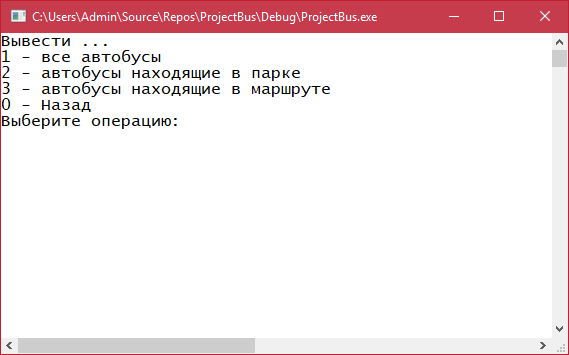


Рисунок 3.7 – Меню вывода

После выбора будет выведен список. На рисунке 3.8 представлен вывод всего списка.

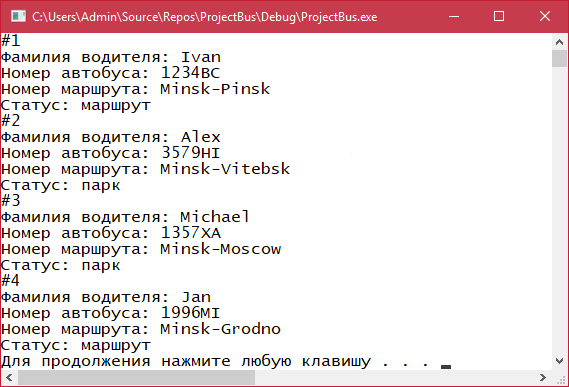


Рисунок 3.8 – Вывод всех автобусов

Чтобы автобус покидает парк, то нужно выбрать «5» и ввести номер автобуса, который покидает парк (рисунок 3.9)

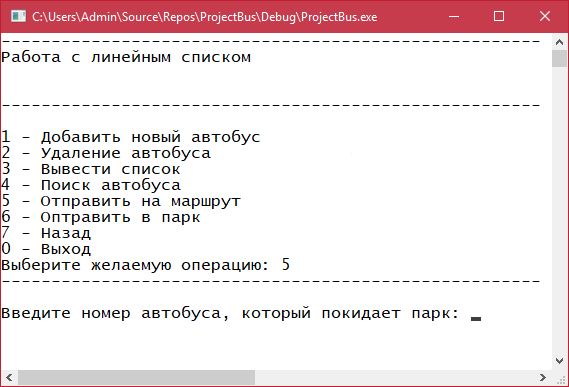


Рисунок 3.9 – Метод отправки автобуса на маршрут

Тоже самое, если автобус прибывает в парк, попросят ввести номер автобуса.

Если нужно выйти из приложения выбираем «0», в дальнейшем при выборе данной цифры, приложение будет выходить. Еще одной стандартной операции является возврат, для этого используется цифра «7».

Для работы с очередью (рисунок 3.1) мы должны вести цифру «2» и перейдем в меню для работы с очередью (рисунок 3.10)

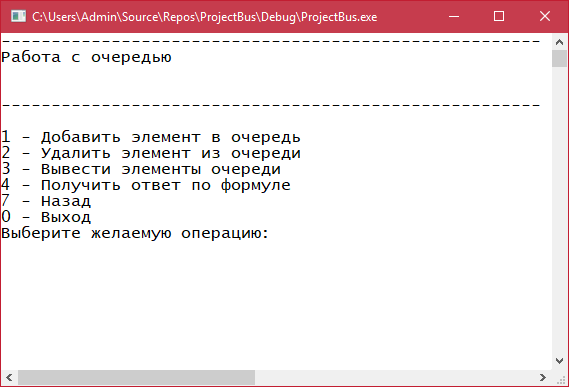


Рисунок 3.10 – Меню очереди

Чтобы добавить элемент нужно ввести цифру «1» и попросят ввести любое число (рисунок 3.11), которое будет использоваться в числовом ряду

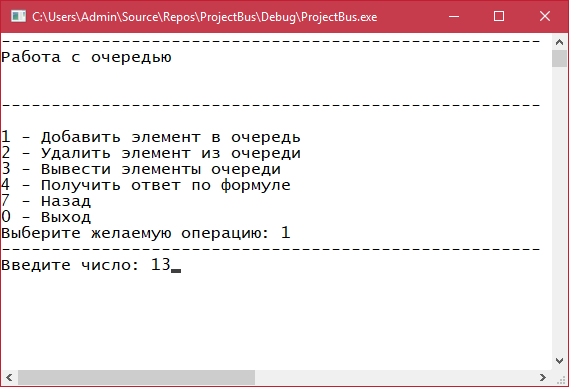


Рисунок 3.11 – Добавление числа в очередь

Для вывода числового ряда вводим число «3» и выводится вся очередь (рисунок 3.12).

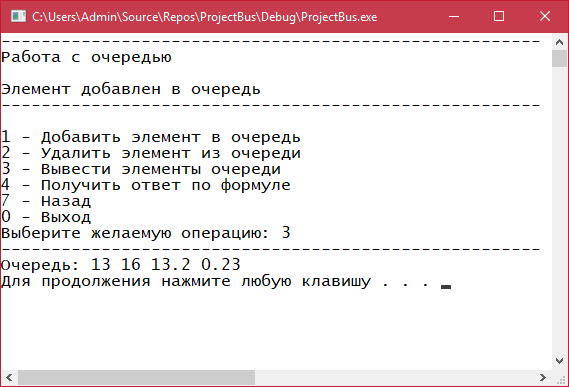


Рисунок 3.12 – Вывод очереди

Чтобы подсчитали значение числового ряда в определенной точке нужно ввести «4», после ввода попросят вести точку. После ввода точки на экран будет выведено полученное выражение и ответ (рисунок 3.13).

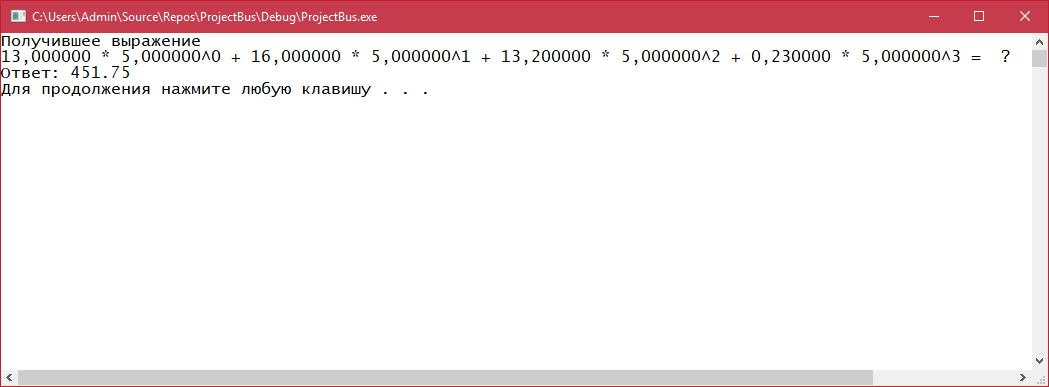


Рисунок 3.13 – Вывод числового ряда

В случаи с удаление элемента из очереди, то будет удален первый элемент и в верхней участке окна появиться сообщение, что элемент вышел из очереди.

# ЗАКЛЮЧЕНИЕ

Подводя итоги курсовой работы по дисциплине «Основы алгоритмизации и программирования», можно сделать выводы о том, что поставленные цели и задачи были успешно выполнены, так же были повторены основы языка программирования C++. На более глубоком уровне изучена работа со списками со стеками. Изучены сложные и нестандартные алгоритмы. Отработаны практические навыки и умения, полученные в процессе изучения базовой части дисциплины «Основы алгоритмизации и программирования». Усовершенствовали теоретические знания.

Во время выполнения курсовой работы использовались полученные знания во время изучения дисциплины «Основы алгоритмизации и программирования». В программе продемонстрированы такие навыки работы, как работа с линейным списком и работа со стеками;

В ходе работы с линейным списком была написана программа, которая обеспечивает хранение терминов и их определений, добавление терминов в список, удаление терминов, вывод определений по заданному термину и последующие удаление, вывод всех терминов.

В ходе работы со стеками, была написана программа, которая производит формирование стека и вывод его элементов в порядке возрастания.

Научилась самостоятельно решать поставленные задачи. Познакомилась со списками и стеками, более углубленно изучила и освоила язык программирования С++.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Стефан Р. Дэвис C++ для «чайников» 4-е издание 2003 с.166
2. Стек [электронный ресурс]. Ссылка на интернет ресурс - http://prog-cpp.ru/data-stack/
3. Линейный список [электронный ресурс]. Ссылка на интернет ресурс -<http://victor192007.narod.ru/files/cpp_d1.html>
4. Программирование на языках высокого уровня Т.А. Павловская издания “Питер” 2003
5. Стек [электронный ресурс]. Ссылка на интернет ресурс - <http://kvodo.ru/stack-data-structure.html>
6. Х. Дейтел, П. Дейтел «Как программировать на С++» 1996, 1037с.

# ПРИЛОЖЕНИЕ А

Заголовочный файл Bus.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct Bus

{

string busNumber;

string name;

string route;

bool status;

};

struct List

{

Bus bus;

List \*next;

};

string AddList(List\*\*);

Bus testList();

void Printf(List\*, int);

void PrintfRoute(List\*);

void PrintfPark(List\*);

void PrintfList(List\*);

void DeleteBus(List\*, int);

void SearchBus(List\*, int);

string setRoute(List\*);

string setPark(List\*);

Файл С++ Bus.cpp

#include "stdafx.h"

string AddList(List\*\*begin) {

cout << "------------------------------------------------------" << endl << endl;

Bus bus = testList();

if (\*begin == nullptr) {

\*begin = new List();

(\*begin)->bus = bus;

(\*begin)->next = nullptr;

}

else //!

{

List \*copy = \*begin;

while (copy->next) copy = copy->next;

copy->next = new List;

copy = copy->next;

copy->bus = bus;

copy->next = nullptr;

}

system("cls");

return "Добавлена запись";

}

Bus testList() {

Bus test;

do {

cout << "Введите номер автобуса: ";

cin >> test.busNumber;

} while (test.busNumber == "");

do {

cout << "Введите ФИО водителя: ";

cin.get();

getline(cin, test.name);

} while (test.busNumber == "");

do {

cout << "Введите маршрут: ";

cin >> test.route;

} while (test.busNumber == "");

test.status = false;

return test;

}

void Printf(List\*begin, int marker) {

switch (marker)

{

case 1: PrintfList(begin); break;

case 2: PrintfPark(begin); break;

case 3: PrintfRoute(begin); break;

case 0: break;

default:

cout << "Введено не правильное значение" << endl;

}

}

void PrintfRoute(List\*begin) {

system("cls");

List\* copy = begin;

int counter = 0;

if (copy == nullptr)

{

cout << "Список пуст." << endl;

system("pause");

system("cls");

return;

}

while (copy != nullptr) {

++counter;

if (copy->bus.status)

{

cout << "#" << counter << endl;

cout << "Фамилия водителя: " << copy->bus.name << endl;

cout << "Номер автобуса: " << copy->bus.busNumber << endl;

cout << "Номер маршрута: " << copy->bus.route << endl;

cout << "Статус: " << (copy->bus.status ? "маршрут" : "парк") << endl;

}

copy = copy->next;

}

system("pause");

system("cls");

}

void PrintfPark(List\*begin) {

system("cls");

List\* copy = begin;

int counter = 0;

if (copy == nullptr)

{

cout << "Список пуст." << endl;

system("pause");

system("cls");

return;

}

while (copy != nullptr) {

++counter;

if (!copy->bus.status)

{

cout << "#" << counter << endl;

cout << "Фамилия водителя: " << copy->bus.name << endl;

cout << "Номер автобуса: " << copy->bus.busNumber << endl;

cout << "Номер маршрута: " << copy->bus.route << endl;

cout << "Статус: " << (copy->bus.status ? "маршрут" : "парк") << endl;

}

copy = copy->next;

}

system("pause");

system("cls");

}

void PrintfList(List\*begin) {

system("cls");

List\* copy = begin;

int counter = 0;

if (copy == nullptr)

{

cout << "Список пуст." << endl;

system("pause");

system("cls");

return;

}

while (copy != nullptr) {

++counter;

cout << "#" << counter << endl;

cout << "Фамилия водителя: " << copy->bus.name << endl;

cout << "Номер автобуса: " << copy->bus.busNumber << endl;

cout << "Номер маршрута: " << copy->bus.route << endl;

cout << "Статус: " << (copy->bus.status ? "маршрут" : "парк") << endl;

copy = copy->next;

}

system("pause");

system("cls");

}

void SearchBus(List\*begin, int marker) {

system("cls");

List\* copy = begin;

int counter = 0;

string number;

if (marker == 2) {

cout << "Поиск записи по номеру автобуса" << endl << endl;

cout << "Введите номер автобуса: ";

cin >> number;

while (copy)

{

++counter;

if ((\*copy).bus.busNumber == number) {

cout << "#" << counter << endl;

cout << "Фамилия водителя: " << copy->bus.name << endl;

cout << "Номер автобуса: " << copy->bus.busNumber << endl;

cout << "Номер маршрута: " << copy->bus.route << endl;

cout << "Статус: " << (copy->bus.status ? "маршрут" : "парк") << endl;

}

copy = copy->next;

}

}

if (marker == 1) {

cout << "Поиск записи по имени водителя" << endl << endl;

cout << "Введите имя водителя: ";

cin.get();

getline(cin, number);

while (copy)

{

++counter;

if ((\*copy).bus.name == number) {

cout << "#" << counter << endl;

cout << "Фамилия водителя: " << copy->bus.name << endl;

cout << "Номер автобуса: " << copy->bus.busNumber << endl;

cout << "Номер маршрута: " << copy->bus.route << endl;

cout << "Статус: " << (copy->bus.status ? "маршрут" : "парк") << endl;

}

copy = copy->next;

}

}

if (marker == 0) {

system("cls");

return;

}

system("pause");

system("cls");

}

//не работает

void DeleteBus(List\*begin, int marker) {

system("cls");

List\* temp = begin;

List \*helping = NULL;

string number;

if (marker == 0) {

system("cls");

return;

}

if (marker == 2) {

cout << "Поиск записи по номеру автобуса" << endl << endl;

cout << "Введите номер автобуса: ";

cin >> number;

while (temp && (\*temp).bus.busNumber != number)

{

helping = temp; // предыдущее значение temp

temp = temp->next;

}

}

if (marker == 1) {

cout << "Поиск записи по имени водителя" << endl << endl;

cout << "Введите имя водителя: ";

cin.get();

getline(cin, number);

while (temp && (\*temp).bus.name != number)

{

helping = temp; // предыдущее значение temp

temp = temp->next;

}

}

if (begin == temp) // если элемент который надо удалить первый

{

begin = temp->next;

}

else

{

if (temp)

helping->next = temp->next;

}

delete (temp);

cout << "Элемент удалён." << endl;

system("pause");

system("cls");

}

string setRoute(List\*begin) {

cout << "------------------------------------------------------" << endl << endl;

List\* copy = begin;

string number;

cout << "Введите номер автобуса, который покидает парк: ";

cin >> number;

while (copy)

{

if ((\*copy).bus.busNumber == number) {

copy->bus.status = true;

system("cls");

return "Автобус " + copy->bus.busNumber + " покинул парк";

}

copy = copy->next;

}

system("cls");

return "Не найден";

}

string setPark(List\*begin) {

cout << "------------------------------------------------------" << endl << endl;

List\* copy = begin;

string number;

cout << "Введите номер автобуса, который прибыл парк: ";

cin >> number;

while (copy)

{

if ((\*copy).bus.busNumber == number) {

copy->bus.status = false;

system("cls");

return "Автобус " + copy->bus.busNumber + " прибыл парк";

}

copy = copy->next;

}

system("cls");

return "Не найден";

}

# ПРИЛОЖЕНИЕ Б

Заголовочный файл Queue.h

#pragma once

#include "stdafx.h"

struct MyQueue

{

double value;

MyQueue \*next;

};

struct queue

{

MyQueue\*head; //начала очереди

MyQueue\*tail; //конец очереди

int size; //размер очередь

};

queue \*init(); //инициализация очереди

string back(queue \*); //добавление в очередь к конец

string pop(queue \*); //удаление первого элемента очереди

void show(queue\*); //вывод очереди

void sum(queue\*); //расчет по формуле

Файл C++ Queue.h

#include "stdafx.h"

queue \*init()

{

queue \*lt = new queue;

lt->size = 0; //размер ноль

lt->head = NULL; //начальный пустой

lt->tail = lt->head; //последный указывает на первый

return lt;

}

/\* Добавляем элемент в конец списка \*/

string back(queue \*lt)

{

cout << "------------------------------------------------------" << endl;

double data;

cout << "Введите число: ";

cin >> data;

MyQueue \* node = new MyQueue;

node->value = data;

if (lt->tail != NULL) //если последний не пуст

lt->tail->next = node; //то последним становить добавленный элемент

else { //если последнего нету, а значит очередь пуста

lt->head = node; //то первым добавленный эл-т

}

lt->tail = node; //последним становиться добавленный эл-т

lt->size += 1; //увеличиваем размер

system("cls");

return "Элемент добавлен в очередь";

}

/\* Извлекаем элемент из начала очереди \*/

string pop(queue \*lt)

{

if (lt->size == 0) {

/\* Список пуст \*/

return "Очередь пуста!";

}

MyQueue \*node = lt->head; //получаем первый элемент очереди

double ret\_val = node->value;

lt->size -= 1; //уменьшаем размер очереди

lt->head = node->next; //первым становиться второй элемент очереди

free(node); //удаляем

if (lt->size == 0) {

/\* Это был последний элемент \*/

lt->head = NULL; //делаем указатели пустыми

lt->tail = NULL;

}

return "Элемент " + to\_string(ret\_val) + " вышел";

}

void show(queue \* lt)

{

cout << "------------------------------------------------------" << endl;

MyQueue \*tmp = lt->head;

cout << "Очередь: ";

for(int i = 0; i<lt->size; i++)

{

cout << tmp->value << " ";

tmp = tmp->next;

}

cout << endl;

system("pause");

system("cls");

}

void sum(queue\*begin)

{

system("cls");

double sum = 0, i = 0, x;

string text;

MyQueue\*copy = begin->head;

if (begin->size == 0) {

cout << "Список пуст." << endl;

system("pause");

system("cls");

return;

}

else {

cout << "Введите число: ";

cin >> x;

for (int i = 0; i < begin->size; i++)

{

sum += copy->value \* pow(x, i);

text += to\_string(copy->value) + " \* " + to\_string(x) + "^";

if (i == begin->size - 1)

text += to\_string(i) + " = ";

else

text += to\_string(i) + " + ";

copy = copy->next;

}

}

system("cls");

cout << "Получившее выражение" << endl;

cout << text << " ? " << endl;

cout << "Ответ: " << sum << endl;

system("pause");

system("cls");

}