МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА АЭРОКОСМИЧЕСКИХ КОМПЬЮТЕРНЫХ И ПРОГРАММНЫХ СИСТЕМ

ОЦЕНКА ДОКЛАДА:

РУКОВОДИТЕЛЬ:

Кандидат техн. наук, доцент		Л.Н.Бариков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №14

по курсу: ОСНОВЫ ПРОГРАММИРОВАНИЯ

на тему: ЛИНЕЙНЫЕ СПИСКИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4143		Е.Д.Тегай
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Цель работы

Изучение способов создания и принципов использования односвязных линейных списков; изучение стандартных средств языка C/C++ для работы с динамической памятью; совершенствование навыков модульного программирования на языке C/C++ при решении задач обработки линейных списков; изучение способов разработки многофайловых проектов.

Задание на программирование

Используя технологию процедурного программирования, разработать программу обработки односвязных линейных списков с числом элементов в списке не менее десяти в соответствии с индивидуальным заданием.

Индивидуальное задание

В непустом списке найти максимальную сумму значений двух любых его элементов. Заменить значение первого элемента списка найденной суммой.

Порядок выполнения работы

1. Получить у преподавателя индивидуальное задание.

2. Провести структуризацию задачи. С этой целью выделить подзадачи, которые будут реализованы в виде отдельных функций. При этом запрещается совмещать в одной функции решение нескольких подзадач.

3. Составить описание процесса решения задачи.

4. Построить схему алгоритма решения задачи (функции main()) с использованием функций создания, просмотра, обработки списка, удаления списка из динамической памяти.

5. Обосновать перечень и типы параметров всех функций.

6. Построить схему алгоритма функции обработки списка.

7. Создать многофайловый проект на языке C/C++.

8. Проверить и продемонстрировать преподавателю работу программы на полном наборе тестов. Обеспечить одновременный показ на экране исходного и результирующего списков.

9. Выходные данные должны выводиться на экран с пояснениями. Операторы вывода результатов работы должны находиться либо в функции

main(), либо в специальной функции вывода (например, преобразованного списка), вызов которой осуществляется из функции main().

10. Оформить отчет о работе в составе: постановка задачи, описание процесса решения, схемы алгоритмов функции main() и функции обработки списка, текст всех модулей проекта, контрольные примеры (скриншоты).

11. Текст программы в отчете не должен представлять из себя скриншот.

12. Скриншоты тестов должны легко читаться. Все их неинформативные части должны быть удалены.

Описание процесса решения

В разрабатываемый проект будут входить три файла:

– заголовочный файл ModSp1.h, содержащий определения новых типов и объявления функций;

– исходный файл ModSp1.cpp, содержащий реализацию набора функций для обработки списка;

– основной файл spis1.cpp с программой (функция main()).

При решении задачи, а значит в тексте проекта, кроме функции main(), необходимо реализовать следующие функции:

– функцию инициализации списка;

– функцию добавления нового элемента в конец списка;

– функцию обработки списка в соответствии с заданием;

– функцию просмотра содержимого списка;

– функцию удаления списка из динамической памяти.

Конкретный тип значений информационной части элементов списка в задании не указан, поэтому для получения легко модернизируемой программы с целью изменения типа обрабатываемых данных определяем новый тип с использованием typedef: тип значений информационной части элементов списка (telem).

Определяем новый тип – структуру list, которая будет описывать элементы списка. У нее два поля: информационное поле типа telem и поле адреса (указатель на следующий элемент списка (list*)).

Эти определения новых типов помещаем в заголовочный файл ModSp1.h. Сюда же будут помещены объявления функций, необходимых для решения задачи.

Функцию инициализации списка называем init_spis(list**,list**). Тип возвращаемого функцией значения – void. В задании не сказано, как должен формироваться список: подключением новых элементов к голове или к хвосту списка. Выбираем способ формирования с подключением новых элементов к концу списка. Поэтому функция инициализации будет иметь два параметра типа «указатель на указатель на тип list». Первый хранит адрес адреса головного элемента списка, а второй – адрес адреса концевого элемента. Под инициализацией списка будем понимать присваивание адресам головного и концевого элементов списка значения NULL (т. е. список пуст).

Функцию добавления нового элемента в конец списка называем add_ spis(telem,list**,list**). Тип возвращаемого функцией значения – void. У нее три параметра: значение информационной части нового элемента списка типа telem и два параметра типа «указатель на указатель на тип list» (адреса головного и концевого элементов списка будут меняться при подключении нового элемента и передаются через указатели).

Функцию обработки списка называем obrabotka(list* beg). В соответствии с условиями задачи адрес головного и концевого элементов при обработке измениться не могут. Тип возвращаемого функцией значения – void. У нее один параметр типа «указатель на тип list». В функцию передается значение адреса головного элемента списка.

Функцию просмотра содержимого списка называем view_ spis(list*). Тип возвращаемого функцией значения – void. У нее один параметр типа «указатель на тип list». В функцию передается значение адреса головного элемента списка. Под просмотром содержимого списка здесь понимается вывод пользователю на экран значений информационных частей всех элементов списка.

Функцию удаления списка из динамической памяти называем udal_spis(list**,list**). Тип возвращаемого функцией значения – void. У нее два параметра типа «указатель на указатель на тип list» (адреса головного и концевого элементов списка будут меняться при удалении списка из динамической памяти), при этом инициализация списка сохраняется.

На этом формирование заголовочного файла ModSp1.h заканчивается.

В исходный файл ModSp1.cpp помещаем реализацию набора функций для работы со списком.

Остановимся подробнее на разработке функции obrabotka() (реализации процесса решения задачи).

Согласно заданию на разработку необходимо в созданном списке заменить первый элемент на максимальную сумму двух любых элементов. Для этого находим значения максимумов.

Блок схема функции показана на рисунке 1.

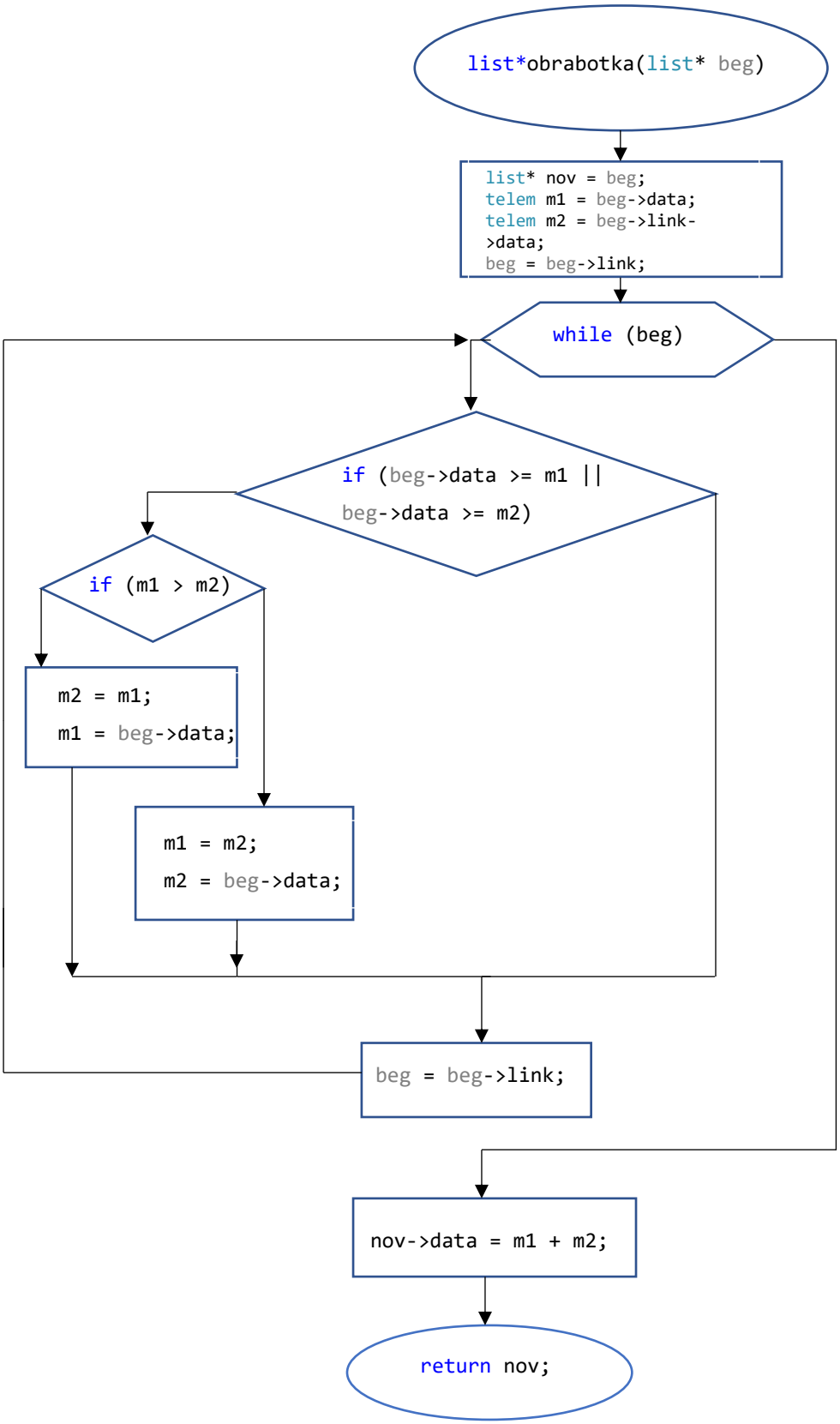

Рисунок 1 – Блок схема функции obrabotka

В основной файл spis1.cpp помещаем функцию main(). Исполнение программы (функции main()) начинается с объявления переменных. Это переменная ch типа telem и два указателя на первый и последний элемент списка.

Инициализируем исходный и результирующий списки. Для этого в функцию main() помещаем три вызова функции init_spis(), параметрами которой являются адреса указателей на первый и последний элементы инициализируемого списка (&l_beg, &l_end).

Приступаем к созданию исходного линейного списка. Поскольку линейный список – это динамическая структура, при создании пользователю выводится подсказка с указанием признака окончания ввода. У элементов списка целочисленная информационная часть, поэтому для примера оговариваем, что признаком окончания ввода значений является значение 0.

Пользователь вводит значения строкой через пробел и нажимает Enter. Считываем первое введенное значение и, если оно отлично от 0, организуем цикл while(ch). В тело этого цикла помещаем вызов функции добавления нового элемента в список add_spis(), передавая ей в качестве параметров значение очередного считанного значения ch и адреса указателей на первый и последний элементы списка (&l_beg, &l_end).

После выхода из цикла проверяем факт создания исходного списка. Если список не создан, выполнение программы заканчивается (return 0).

Если список создан, выводим значения его элементов, вызывая функцию view_spis(), параметром которой является значение указателя на первый элемент созданного исходного списка l_beg.

Проверка создания списка выполнена. Приступаем к обработке созданного списка. Вызываем функцию obrabotka(), передавая ей в качестве параметров значение указателя на первый элемент исходного списка l_beg и указатели на адреса головного и концевого элементов результирующего списка.

После обработки списка выводим значения элементов этого списка, вызывая функцию view_ spis(), параметром которой является значение указателя на первый элемент сформированного списка l_beg.

Программа завершается. Удаляем все списки из динамической памяти, трижды вызывая функцию udal_spis() и передавая ей в качестве параметров

значения адресов указателей на первый и последний элементы списков (&l_beg, &l_end). Анализируем результаты работы проекта и делаем выводы. Блок схема функции main() показана на рисунке 2.

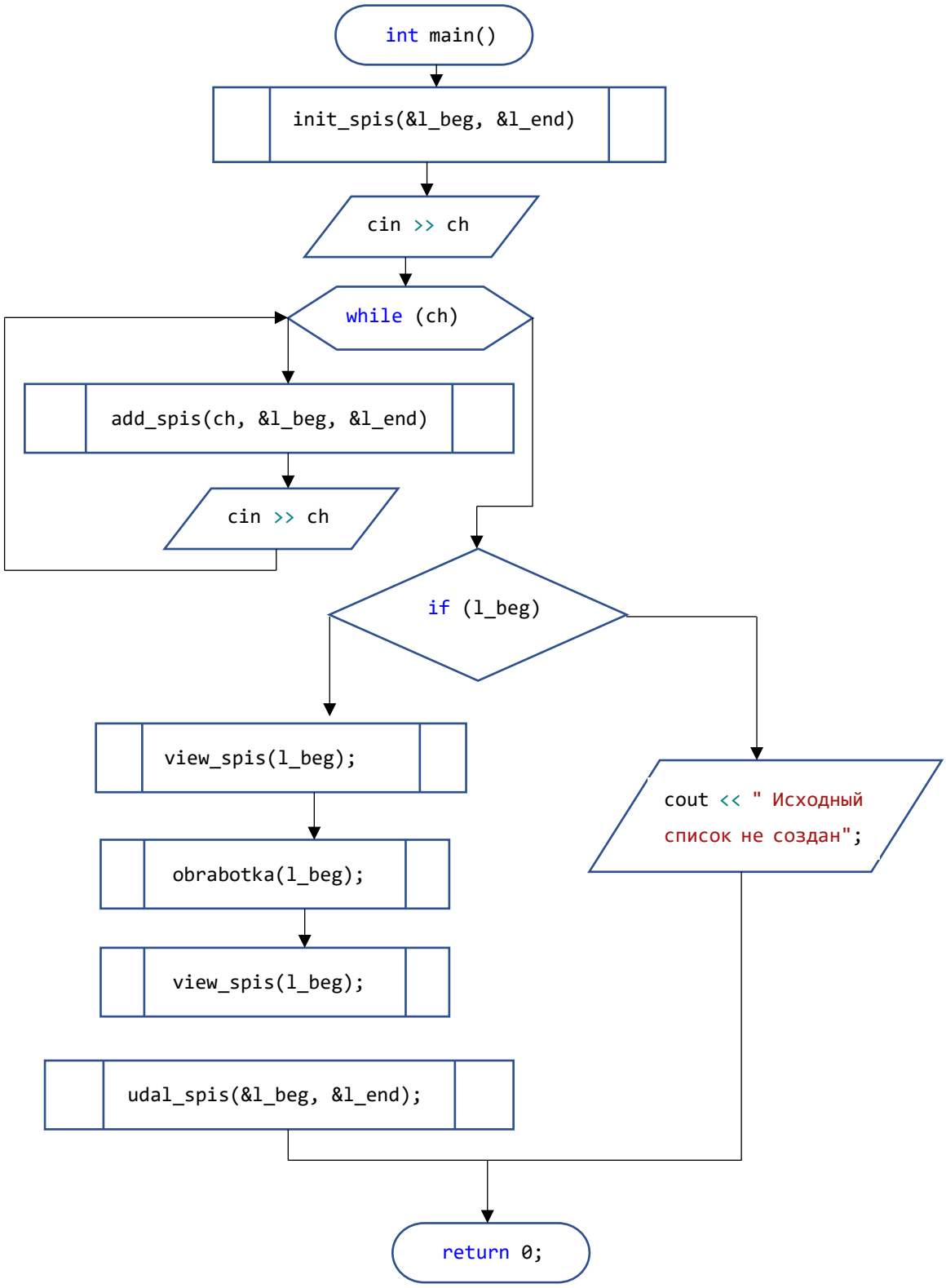

Рисунок 2 – Блок схема функции main

Текст программ проекта

Заголовочный файл ModSp1.h

```
#pragma once
/* Заголовочный файл ModSp1.h
* Содержит определения новых типов и объявления
функций для работы со списком */
// Определение типов
typedef int telem; // определение типа информационного поля
struct list {    // определение типа элемента списка
	telem data;      // информаицонное поле
	list* link;      // поле адреса
};
// Объявления (прототипы) функций
void init_spis(list**, list**);
void add_spis(telem, list**, list**);
list* obrabotka(list*);
void view_spis(list*);
void udal_spis(list**, list**);
```

Основной файл spis1.cpp

```
/* Многофайловый проект. Основной файл spis1.cpp*/
#include <iostream>
#include "ModSp1.h"
using namespace std;

// основная программа
int main() {
	telem ch;
	list * l_beg, * l_end;
	setlocale(LC_ALL, "Rus");
	// инициализация списков
	init_spis(&l_beg, &l_end);
	cout << "\n Задание: " << endl << "Создаётся непустой линейный список чисел." <<
		endl << " После этого находится максимальная сумма двух любых значений" <<
		endl << " и ставится на место первого элемента списка";
	// создание исходного списка
	cout << "\n Вводите элементы исходного списка" <<
		endl << " через пробел одной строкой, а в конце поставьте ноль: " << endl;
	cin >> ch;
	while (ch) {
		add_spis(ch, &l_beg, &l_end);
		cin >> ch;
	}
	if (l_beg) {
		// выводим исходный список на экран
		cout << endl << " Исходный список: " << endl;
		view_spis(l_beg);
		// обрабатываем созданный список и получаем два результирующие
		obrabotka(l_beg);
		// изменённый исходный список
		cout << endl << "Ваш список после обработки: " << endl;
		view_spis(l_beg);
		// удаляем списки из динамической памяти
		udal_spis(&l_beg, &l_end);
	}
	else {
		cout << " Исходный список не создан";
	}
	return 0;
}
```

Исходный файл ModSp1.cpp

```
/* Исходный файл ModSp1.cpp
Реализация набора функций для работы со списком*/
#include <iostream>
#include "ModSp1.h"
using namespace std;

// Функция, отвечающая за добавление нового элемента в конец списка
void add_spis(int ch, list** beg, list** end) {
	list* nov = new list; // выделяем память под элемент списка
	nov->data = ch; // заполняем информационную часть
	nov->link = 0; // в ссылочную - NULL
	if (*beg) // если список не пуст,
		(*end)->link = nov; // добавляем в конец списка
	else *beg = nov; // если список был пуст
	*end = nov; // новый элемент - последний
	return;
}

/* нахождение максимальной суммы значений двух любых элементов списка */
list* obrabotka(list* beg) {
	list* nov = beg;
	telem m1 = beg->data;
	telem m2 = beg->link->data;
	beg = beg->link;
	// ищем максимумы
	while (beg) {
		if (beg->data >= m1 || beg->data >= m2) {
			if (m1 > m2) {
				m2 = m1; // максимальный элемент
				m1 = beg->data; // значение максимального элемента
			}
			else {
				m1 = m2; // максимальный элемент
				m2 = beg->data; // значение максимального элемента
			}

		}
		beg = beg->link; // след. элемент
	}
	nov->data = m1 + m2;
	return nov;
}

// Функция, отвечающая за просмотр списка
void view_spis(list* beg) {
	while (beg) {
		cout << beg->data << ' ';
		beg = beg->link;
	}
	cout << endl;
	return;
}

// Функция, отвечающая за инициализацию списка
void init_spis(list** beg, list** end) {
	*beg = *end = 0;
	return;
}

// Функция, отвечающая за удаление списка
void udal_spis(list** beg, list** end) {
```

```
        list* tec;
        while (*beg) {
               tec = *beg;
               *beg = (*beg)->link;
               delete tec;
        }
        *end = 0;
        // список пуст
        return;
}
```

Результаты работы программы

Результаты показаны на рисунках 3 - 6.

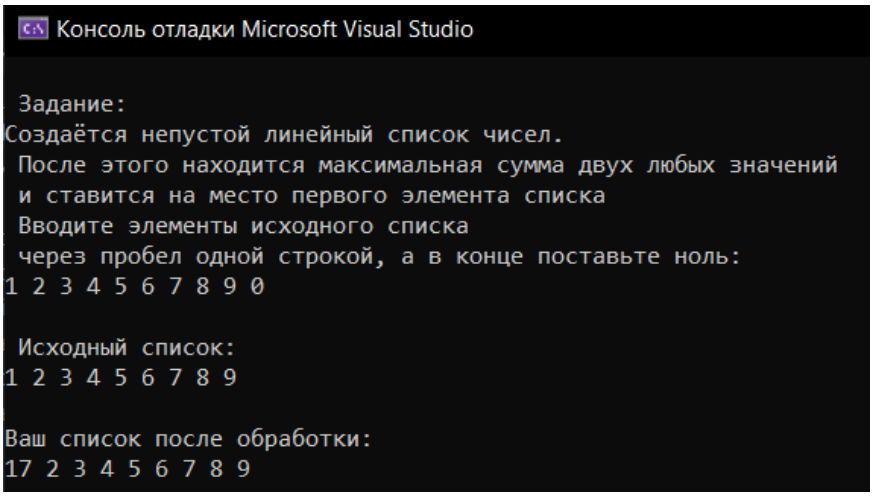

Рисунок 3 – Результаты работы программы

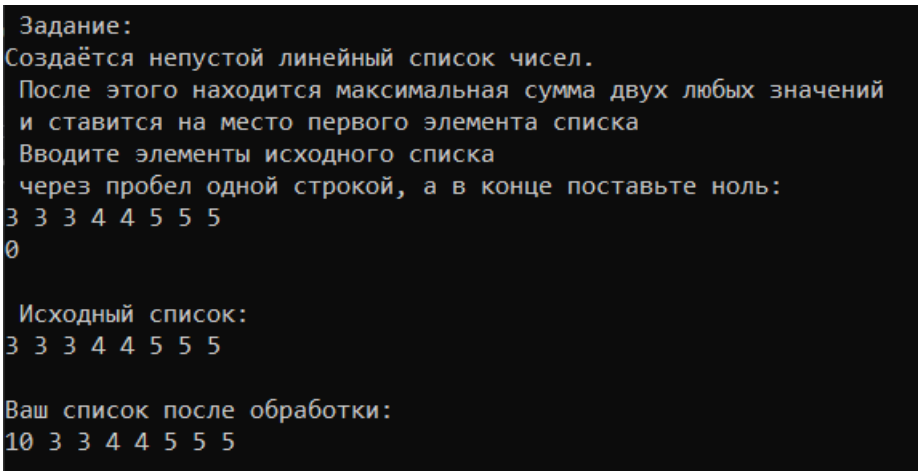

Рисунок 4 – Результаты работы программы

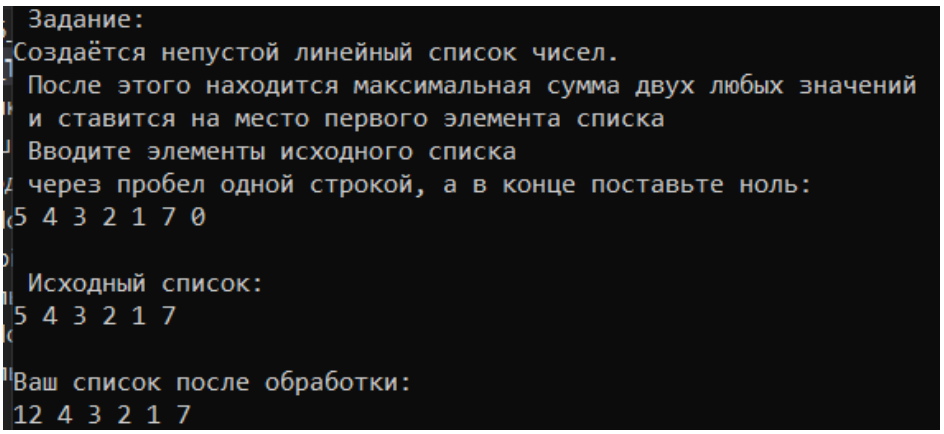

Рисунок 5 - Результаты работы программы

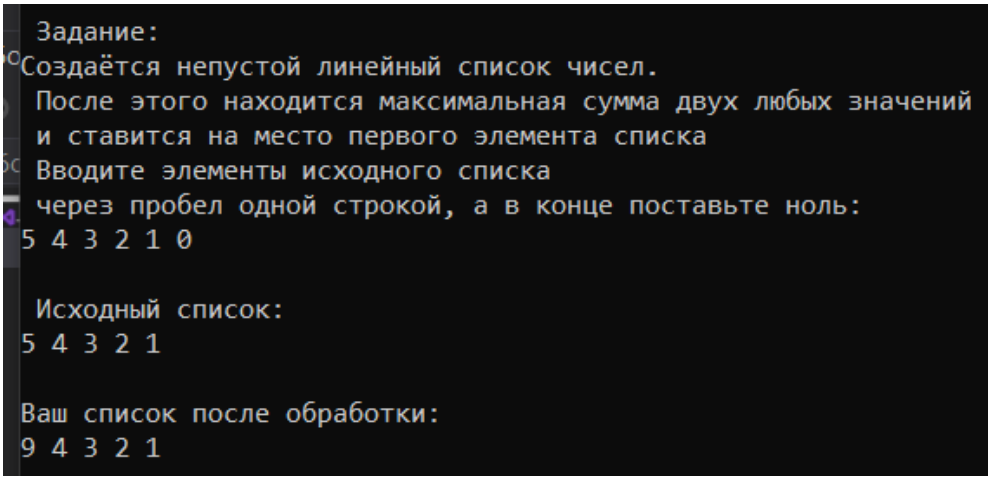

Рисунок 6 - Результаты работы программы

Вывод

В данной лабораторной работе были изучены способы создания и принципы использования односвязных линейных списков; изучены стандартных средств языка C/C++ для работы с динамической памятью; усовершенствованы навыки модульного программирования на языке C/C++ при решении задач обработки линейных списков; изучены способы разработки многофайловых проектов.