**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**КАФЕДРА САПР**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Программирование»**

**Тема: «Обработка текстов. Реализация длинной арифметики на C++»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3352 |  | Рябов В. А. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Ягодка Т.Д. | | |
| Группа 3352 | | |
| Тема работы: Использование мультисписков для работы с данными | | |
| Исходные данные:  Дано N строк с данными – дата, тип самолета, пункт назначения, номер рейса | | |
| |  | | --- | | Содержание пояснительной записки:  “Исходная формулировка”, “Математическая постановка”, “Контрольные примеры”, “Особенности реализации на компьютере”, “Раздел описания классов/структур” “Формат хранения данных”, “Организация диалога с пользователем”, “Средства обеспечения ввода/вывода”, “Функции”, “Порядок использования пользовательских файлов”, “Алгоритм работы программы”, “Текст программы”, “Результат работы программы”. | | | | |
| Предполагаемый объем пояснительной записки:  Не менее 30 страниц. | | |
| Дата выдачи задания: 20.02.2024 | | |
| Дата сдачи реферата: .05.2024 | | |
| Дата защиты реферата: .05.2024 | | |
| Студент |  | Ягодка Т.Д. |
| Преподаватель |  | Калмычков В.А. |

**Аннотация**

В данной работе была создана программа, которая способна эффективно хранить “данные службы аэропорта”, а также выполнять действия, запрошенные пользователем по поиску информации, такие как поиск по пункту назначения, по датам, по типу самолета. Методами создания программы стали изученные в ходе семестра возможности языка программирования C++, а именно классы, структуры и списки.

**Summary**

In this work, a program was created that is capable of efficiently storing "airport service data" and performing actions requested by the user to search for information, such as searching by destination, dates, and aircraft type. The methods used to create the program were the features of the C++ programming language studied during the semester, specifically classes, structures, and lists.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Исходная формулировка | 5 |
| 2. | Математическая постановка | 5 |
| 2.1. | Описание списка | 6 |
| 3. | Контрольный пример | 8 |
| 4. | Особенности реализации на компьютере | 9 |
| 5. | Раздел описания классов/структур | 9 |
| 6. | Формат хранения данных | 11 |
| 7. | Организация диалога с пользователем | 12 |
| 8. | Средства обеспечения ввода/вывода | 13 |
| 9. | Функции | 13 |
| 10. | Порядок использования пользовательских файлов | 14 |
| 11. | Алгоритм работы программы | 15 |
| 12. | Текст программы | 20 |
| 13. | Результат работы программы | 31 |
|  | Заключение | 31 |

ВВЕДЕНИЕ

Целью данной курсовой работы является написание программы, способной эффективно хранить данные табличного формата, а также производить поиск и вывод данных по запросу пользователя.

Основной задачей работы является создание программы, хранящей данные о рейсах и отвечающей требованиям, описанным выше.

Метод решения: Создание мультисписков, которые позволят через один из параметров о рейсе находить остальные, относящиеся к нему. Так как эффективный формат хранения подразумевает собой единоразовое сохранение данных, все списки сортируются и не принимают повторяющиеся данные. Для передачи информации между классами используются указатели, не нарушающие требований к реализации программы.

Исходная формулировка.

7.25. Служба аэропорта имеет сведения, размещенные в эффективном формате хранения, о номере и дате рейса и пункте назначения, типе самолета.

Найти:

- номера рейсов и типы самолетов, вылетающих в конкретный пункт назначения (в конкретную дату, в пределах месяца, квартала), отсортировать по датам вылета

- пункты назначения и номера рейсов для конкретного типа самолета, сортировка по наименованию пункта назначения.

Математическая постановка.

Даны 4 параметра о рейсе – дата, тип самолета, пункт назначения, номер рейса. Данные подаются во входной файл в таком-же порядке без пробелов с разделителем “|”.

Задача - сохранить данные в эффективном формате хранения, обеспечить возможность поиска данным по дате (конкретной дате, кварталу, месяцу), по пункту назначения, по типу самолета. Учесть, что данные должны выводиться в определенном порядке – сортировка по датам, сортировка по наименованию пункта назначения. Создать меню для взаимодействия пользователя с программой, вывести текст из меню в результирующий файл.

Для выполнения требований сделаны следующие действия:

1. Для эффективного формата хранения каждый созданный список будем использовать в качестве “картотеки”. Это значит, что элементы в каждом списке будут отсортированы по возрастанию, а любые повторы информации исключены.

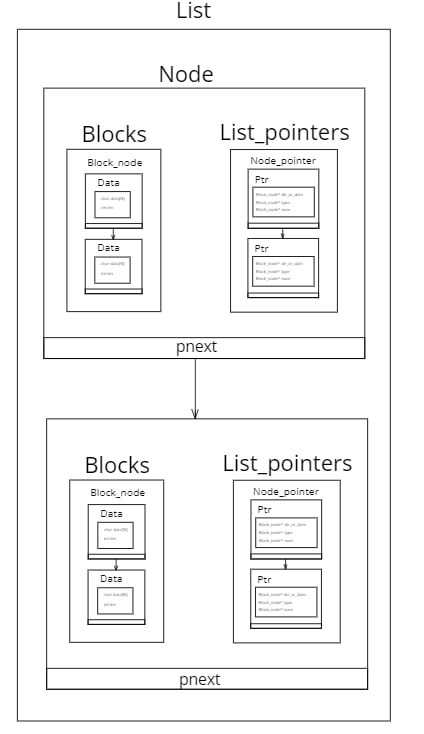
Это будет реализовано с помощью алгоритма для чтения, где каждый новый полученный элемент будет сравниваться с элементами, уже добавленными в новый список, и будет либо добавляться в самый конец (если это самый первый элемент или если по алфавиту он больше всех), добавляться внутрь списка (если по алфавиту он стоит раньше какого-либо добавленного в список элемента) или удаляться из памяти (если будет находиться такой-же элемент)

Сравнения производятся посимвольно до первого отличия (где один из символов будет стоять раньше по алфавиту), для дат сравнения проводится иначе – текстовая информация переводится в числовую и сравнивается.

1. Для вывода данных в сортированном порядке поиск ведется именно через тот список, по которому должна сортироваться вся информация. Например, если нужно найти информацию для конкретного пункта назначения, а данные должны быть отсортированы по датам, то поиск ведется через список дат, где при нахождении нужного города будет выводиться остальная информация о рейсе.
2. Для взаимодействия пользователя и программы написана функция menu(), в которой прописаны варианты запросов. Вся информация, выведенная на экран и вписанная человеком дублируется в результирующий файл.

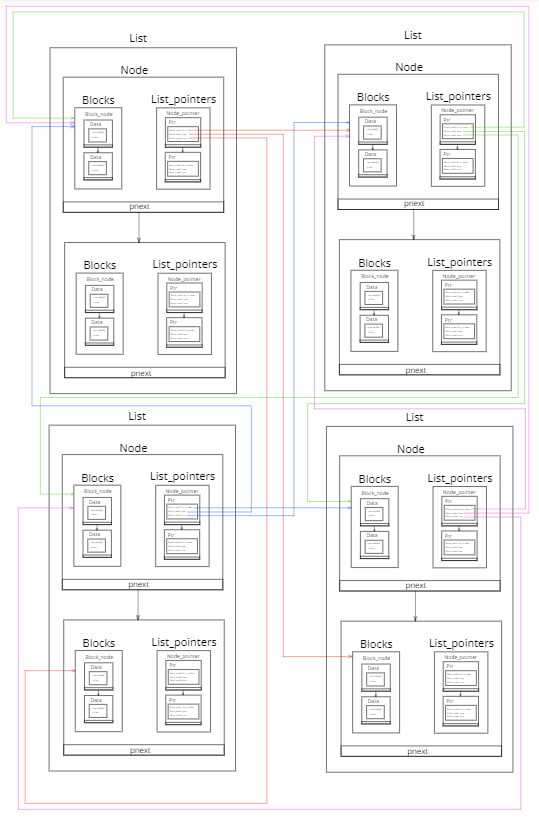
Описание списка.

Для хранения данных используются мультисписки. В каждом элементе списка содержатся свои данные, например для даты – список с датой и т.д., а также содержится список из указателей на элементы других списков.



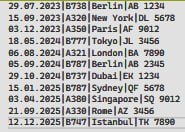
List – основной список, который хранит элементы типа Node. В каждом элементе лежит список Blocks – список для хранения текстовой информации, хранит элементы типа Block\_Node. Внутри каждого элемента Block\_Node лежит char’овсий массив, а также число – количество добавленных символов. Также в элементе Node лежит список List\_pointers – это список из указателей, состоящий из элементов типа Node\_pointer, которые содержат 3 указателя на первый элемент Block\_Node списков Blocks из остальных списков типа List.

Связь между списками изображена на данном рисунке:



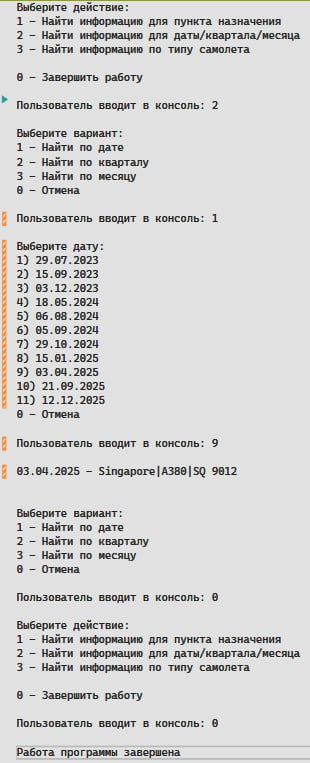
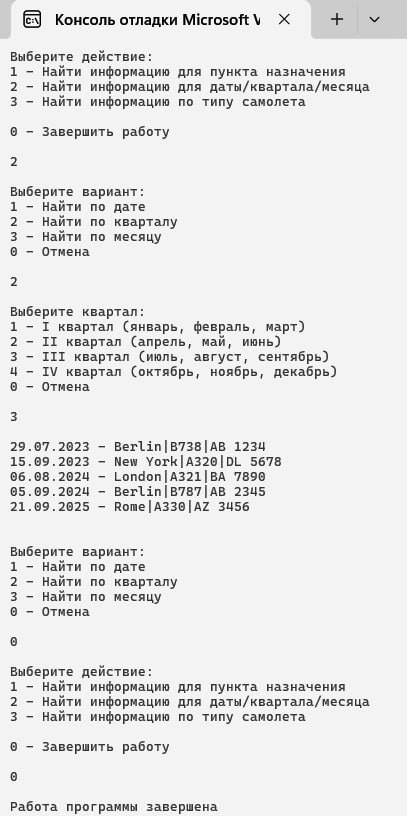
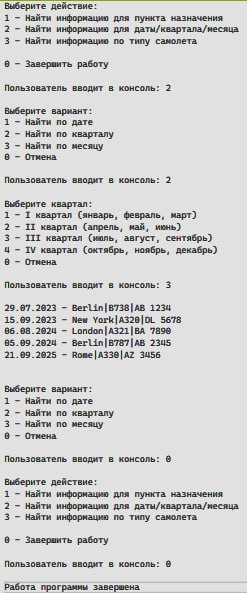
Контрольный пример.

Ввод:

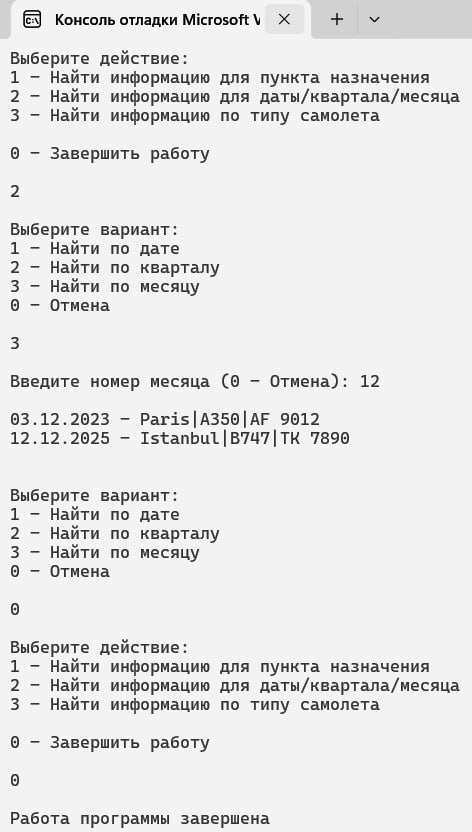
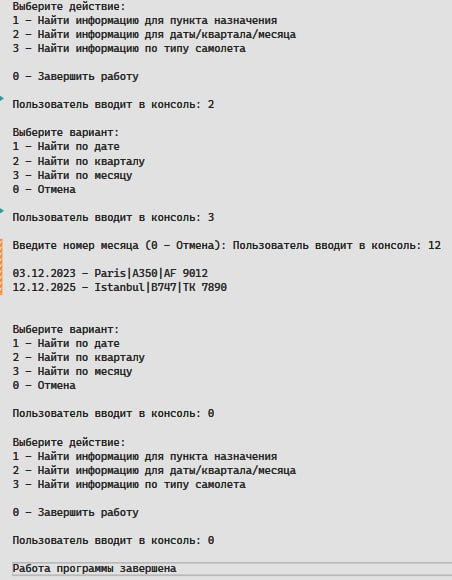
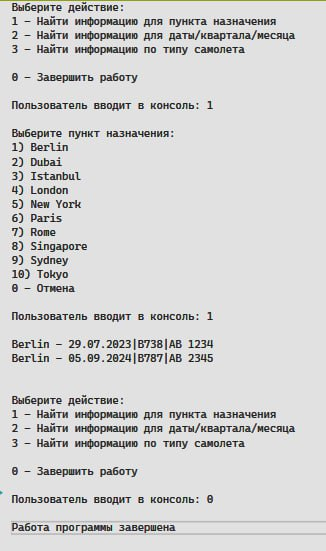
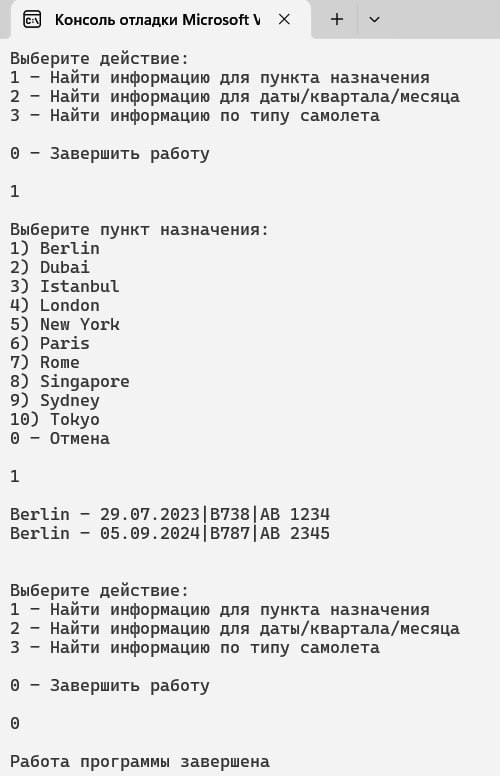


Вывод:

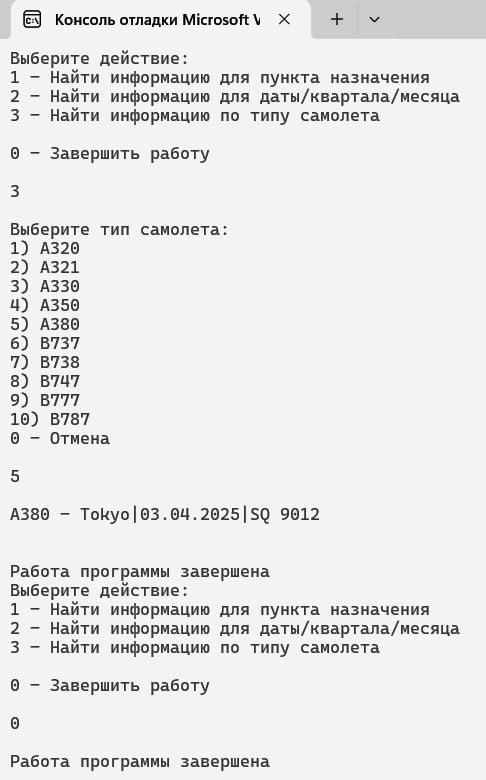
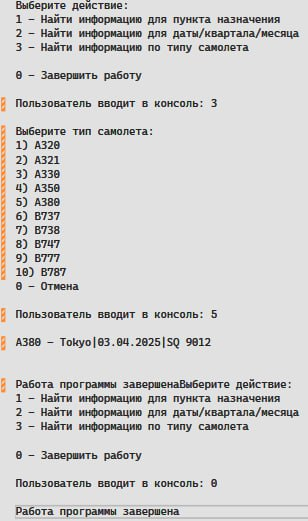
1. Поиск по дате 2) Поиск по кварталу

3) Поиск по месяцу 4) Поиск по городу

5) Поиск по типу самолета



Особенности реализации на компьютере.

В файле Flights\_Data.txt дается табличка из данных. Программа создает 4 списка типа List и с помощью цикла считывает построчно весь файл до самого конца. Каждый список получает только один параметр, а после окончания строки получает указатели на элементы других список, лежащих в той же строке.

После считывания данных программа имеет 4 картотеки. Далее запускается функция меню, где с помощью команд пользователя из консоли программа выполняет определенные действия.

Для поиска по пункту назначения программа идет по списку дат и выводит все данные при совпадении города с информацией в списке дат. Для типов самолета аналогично, но поиск идет по списку городов. Если поиск идет по датам, то программа двигается по списку дат и ищет совпадения.

Раздел описания классов / структур.

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| struct Data | char data | Массив символов |
| int len | Количество символов в массиве |

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| struct Block\_node | Block\_node\* pnext | Указатель на следующий элемент |
| Data material | Данные |

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| class Blocks | Block\_node\* head | Указатель на первый элемент |
| int size | Количество элементов в списке |
| Block\_node\* get\_ptr() | Функция, возвращающая указатель на первый элемент |
| void push\_back() | Добавление в конец списка |
| void get\_str() | Добавление информации из файла |
| int compare() | Сравнение элементов |
| int is\_date() | Сравнение дат |
| void print\_data() | Вывод информации |
| Block\_node\* getList() | Функция для перемещения по элементам списка |
| void del() | Удаление данных из памяти |

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| struct Ptr | Block\_node\* dir\_or\_date | Указатель на первый элемент списка, хранящего дату или город |
| Block\_node\* type | Указатель на первый элемент списка, хранящего тип самолета |
| Block\_node\* num | Указатель на первый элемент списка, хранящего номер рейса |

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| struct Node\_pointer | Node\_poiner\* pnext | Указатель на следующий элемент |
| Prt inf | Указатели |

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| class List\_pointer | Node\_pointer\* head | Указатель на первый элемент |
| int size | Количество элементов в списке |
| void push\_back() | Добавление элемента в конец списка |
| void add() | Функция для добавления элемента |
| bool search() | Поиск совпадающих элементов |
| void out() | Функция вывода |
| void print() | Функция производящая вывод |
| void print\_help() | Вспомогательная функция вывода |
| Node\_pointer\* getList() | Функция для перемещения по списку |
| void del() | Удаление данных из памяти |

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| struct Node | Node\* pnext | Указатель на следующий элемент |
| Blocks info | Список с текстом |
| List\_pointers connected\_info | Список с указателями |

|  |  |  |
| --- | --- | --- |
| Название | Входящие переменные | Назначение входящих переменных |
| class List | Node\* head | Указатель на первый элемент |
| int size | Количество элементов в списке |
| int type | Так как каждый список создается из этого класса, для того чтобы для каждого делать свои действия всем добавляется своя цифра |
| Node\* last | Указатель на последний добавленный элемент |
| Block\_node\* last\_added | Указатель на последний добавленный список текста |
| void push\_back() | Добавление в конец списка |
| void push\_in() | Добавление внутрь списка |
| void get\_data() | Добавление данных из файла |
| void connect() | Соединение списков между собой |
| Node\* getList() | Перемещение по элементам списка |
| void output() | Вывод данных |
| void find\_info() | Поиск по городу или типу |
| void find\_for\_date() | Поиск по конкретной дате |
| void find\_for\_m\_q() | Поиск по месяцу или кварталу |
| void del() | Удаление данных из памяти |

Формат хранения данных.

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Назначение |
| ifstream | departures | Файл с данными |
| ofstream | res, protocol | Файлы для вывода |
| List | type, date, flight, direction | Основные списки |
| int | action, choise, choise\_in | Переменные для работы меню |
| type | Тип списка |
| size, len | Длина списка/массива |
| add, i, j, k, count, blc, last, var | Переменные для циклов |
| month, m1, m2, dt | Переменные для работы с датами |
| Node\* | head, last, current, tmp | Указатели на элемент Node |
| Block\_node\* | last\_added, head, pnext, current, tmp, dir\_or\_date, type, num | Указатели на элемент Block\_node\* |
| Blocks | line, added, info | Список текста |
| List\_pointes | connected\_info | Список указателей |
| char | tmp, tmp1, tmp2 | Переменные для работы с символами |
| data | Массив символов |
| bool | f, f1 | Флаги |
| Node\_pointer\* | head, pnext, tmp, current | Указатели на элемент Node\_pointer |
| Ptr | inf, a | Указатели |

Организация диалога с пользователем.

|  |  |
| --- | --- |
| Варианты ввода | Для файла: данные формата  дд.мм.гггг|тип самолета|город|номер рейса  Для консоли: на ввод принимаются только числовые значения |
| Варианты вывода | Меню принимает разные формы.   1. Основная   Выберите действие:  1 - Найти информацию для пункта назначения  2 - Найти информацию для даты/квартала/месяца  3 - Найти информацию по типу самолета  0 - Завершить работу   1. Для выбора города   Выберите пункт назначения:  1) Berlin  2) Dubai  3) Istanbul  4) London  5) New York  6) Paris  7) Rome  8) Singapore  9) Sydney  10) Tokyo  0 – Отмена   1. Для выбора даты/квартала/месяца   Выберите вариант:  1 - Найти по дате  2 - Найти по кварталу  3 - Найти по месяцу  0 - Отмена  3.1) 3.2) 3.3)  Выберите дату: Выберите квартал: Введите номер месяца (0 - Отмена):  1) 29.07.2023 1 - I квартал (январь, февраль, март)  2) 15.09.2023 2 - II квартал (апрель, май, июнь)  3) 03.12.2023 3 - III квартал (июль, август, сентябрь)  4) 18.05.2024 4 - IV квартал (октябрь, ноябрь, декабрь)  5) 06.08.2024 0 - Отмена  6) 05.09.2024  7) 29.10.2024  8) 15.01.2025  9) 03.04.2025  10) 21.09.2025  11) 12.12.2025  0 - Отмена   1. Для выбора типа   Выберите тип самолета:  1) A320  2) A321  3) A330  4) A350  5) A380  6) B737  7) B738  8) B747  9) B777  10) B787  0 – Отмена   1. При неверно введенном числа программа выводит   ОШИБКА! Некорректный номер   1. При завершении работы программа уведомляет об этом 2. Файл протокола: информирует о процессе заполнения списков |

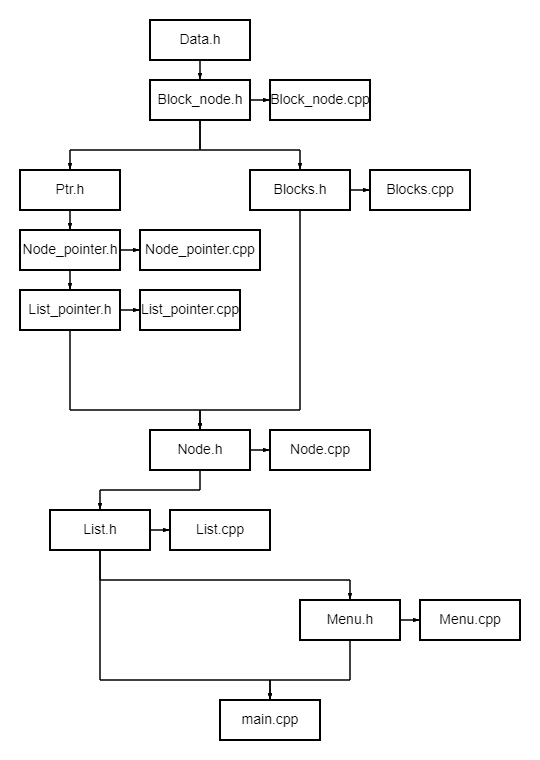
**Средства обеспечения ввода/вывода.**

|  |  |
| --- | --- |
| Библиотека | Команды |
| iostream | cout |
| fstream | ifstream, ofstream, .eof(), .open() |
| locale | setloacle |

**Функции.**

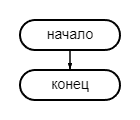
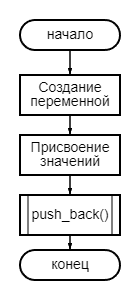
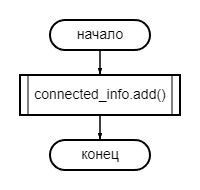
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя функции | Тип | Назначение | Параметры | | |
| Входные | Выходные | Модифицируемые |
| menu | void | Функциональное меню в консоли | List& type, List& date, List& flight, List& direction, ofstream& res |  | res |
| push\_back | void | Добавление элемента в конец списка | info |  |  |
| push\_in | void | Добавление элемента внутрь списка | info, added |  |  |
| connect | void | Соединение списков между собой | Node\* to, Block\_node\* dir\_or\_date, Block\_node\* type, Block\_node\* num |  |  |
| getList | Node\*/  Block\_node\*/  Node\_pointer\* | Перемещение по списку | int k | tmp |  |
| output | void | Вывод данных списка | ofstream& res |  | res, protocol |
| find\_info | void | Поиск по слову | ofstream& res, Block\_node\* to\_find |  | res |
| find\_for\_date | void | Поиск по дате | ofstream& res, int k |  | res, protocol |
| find\_for\_m\_q | void | Поиск по месяцу или кварталу | ofstream& res, int month, int m2=0, int m3=0 |  | res |
| get\_data | void | Добавление данных из текста | ifstream& departures |  | last\_added, protocol |
| get\_str | void | Добавление данных из строки | ifstream& departures |  |  |
| get\_ptr | Block\_node\* | Получение указателя | - | head |  |
| compare | int | Сравнение строк | Blocks l1, Blocks l2, int type | 1,2,0 |  |
| is\_date | int | Проверка на совпадение дат | Blocks l, int m, int m1 = 0, int m2 = 0 | 1,0 |  |
| print\_data | void | Вывод информации | ofstrean& res |  | res, protocol |
| add | void | Добавление указателей | Block\_node\* inf1, Block\_node\* inf2, Block\_node\* inf3 |  |  |
| search | bool | Поиск совпадающих элементов | ofstream& res, Block\_node\* given, Block\_node\* obj | f | res |
| out | void | Вывод информации | ofstream& res, Block\_node\* given |  | res, protocol |
| print | void | Вывод информации | ofstream& res, Block\_node\* given, Node\_pointer\* tmp, int var |  | res, protocol |
| print\_help | void | Вспомогательная программа для вывода | ofstream& res, Block\_node\* data |  | res, protocol |
| del | void | Удаление данных из памяти |  |  | Node\* current, Block\_node\* current, Node\_pointer\* current |

**Порядок использования пользовательских файлов.**

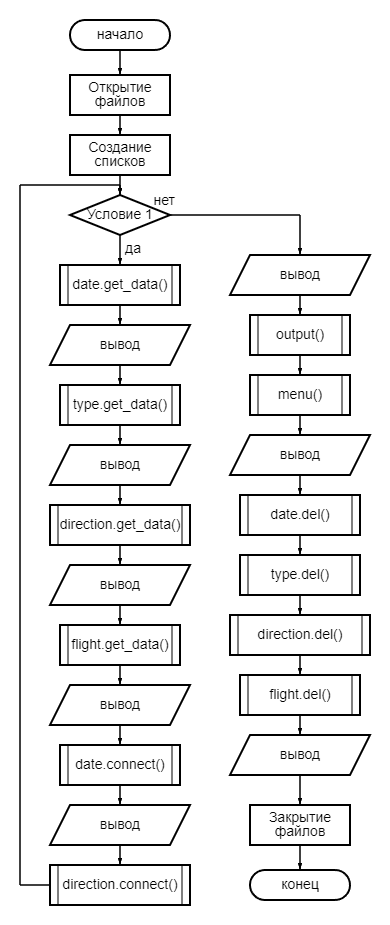
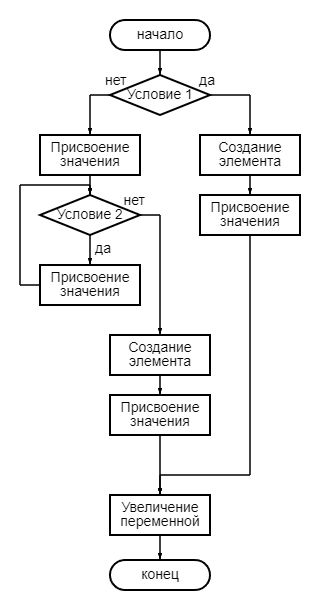
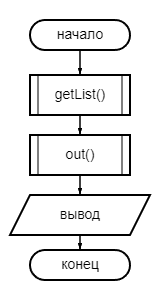


Алгоритм работы программы.

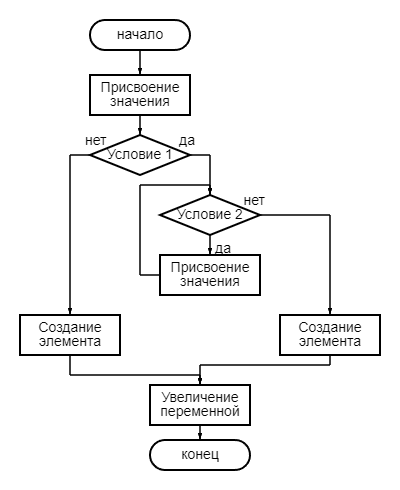
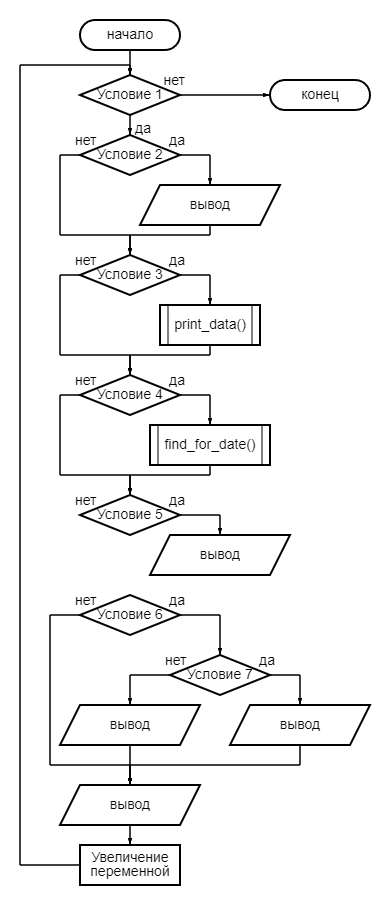
Blocks::get\_ptr() List\_pointers::add() List::connect()

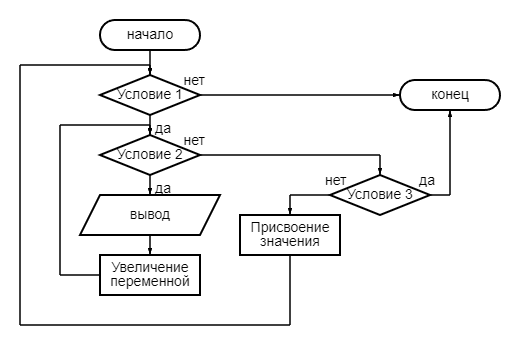
main() push\_back() (одинаковая) List::find\_for\_date()

getList() (одинаковая) List::output()

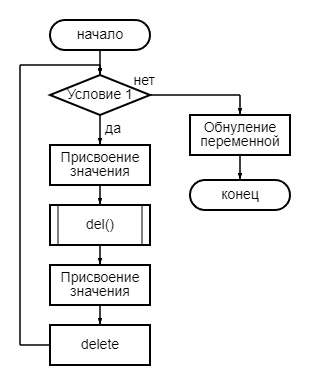
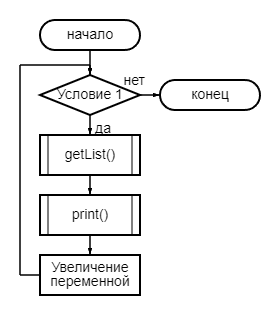
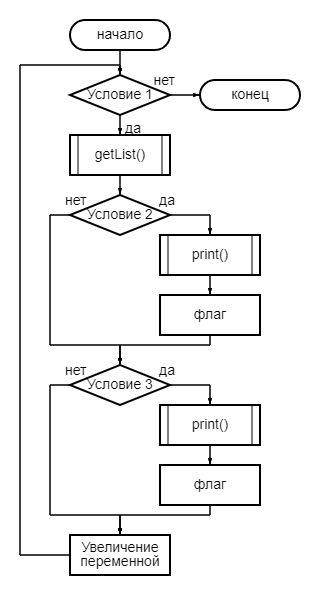
List\_pointers::print\_help()



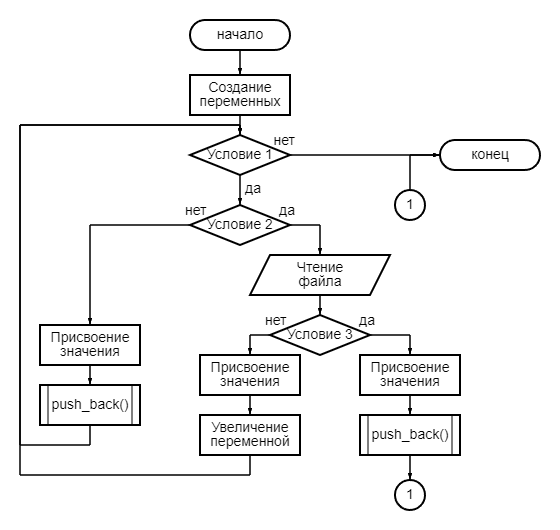
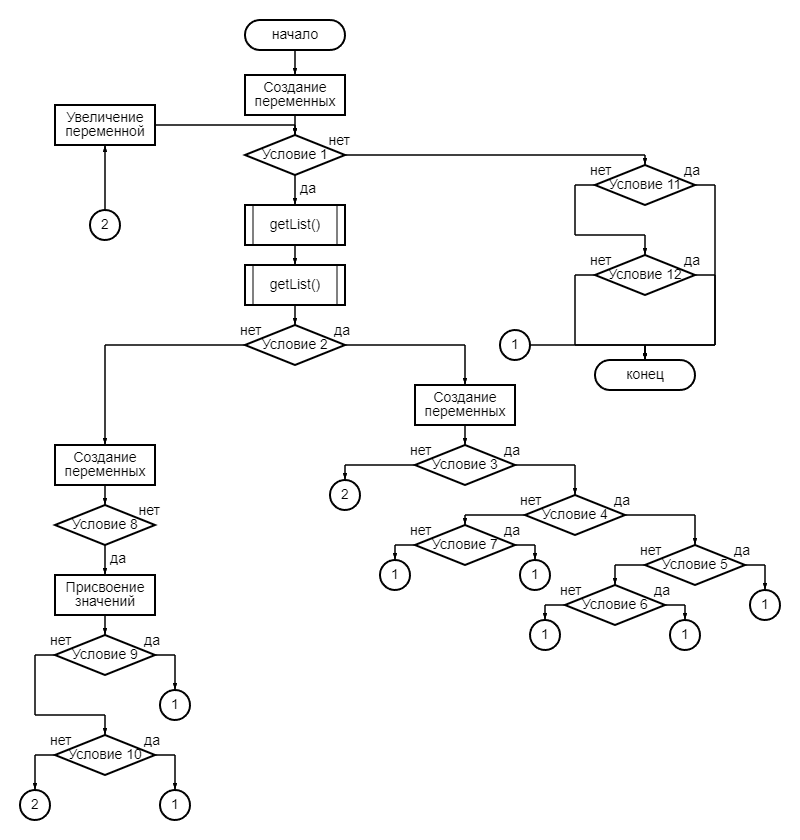
del()

(отличается наличией

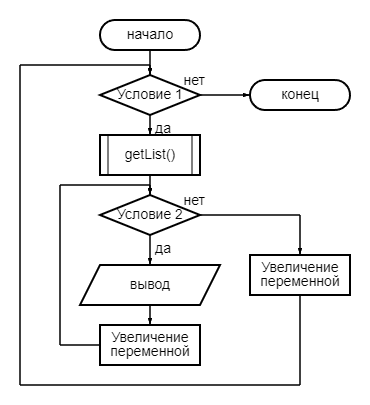
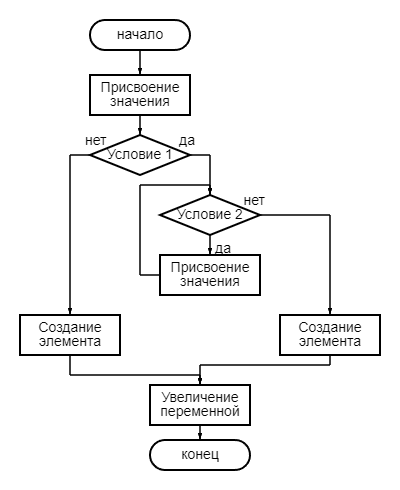
функуии del внутри) List\_pointers::out() List\_pointers::search()

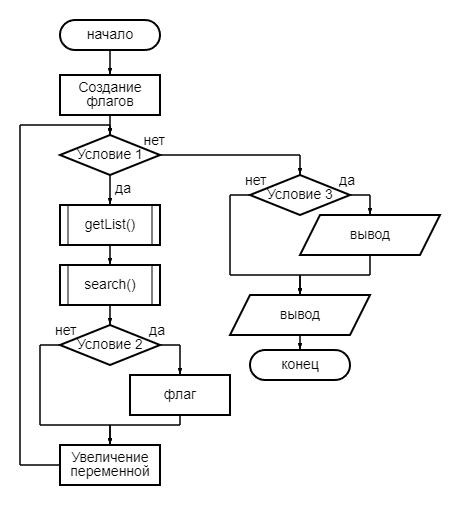
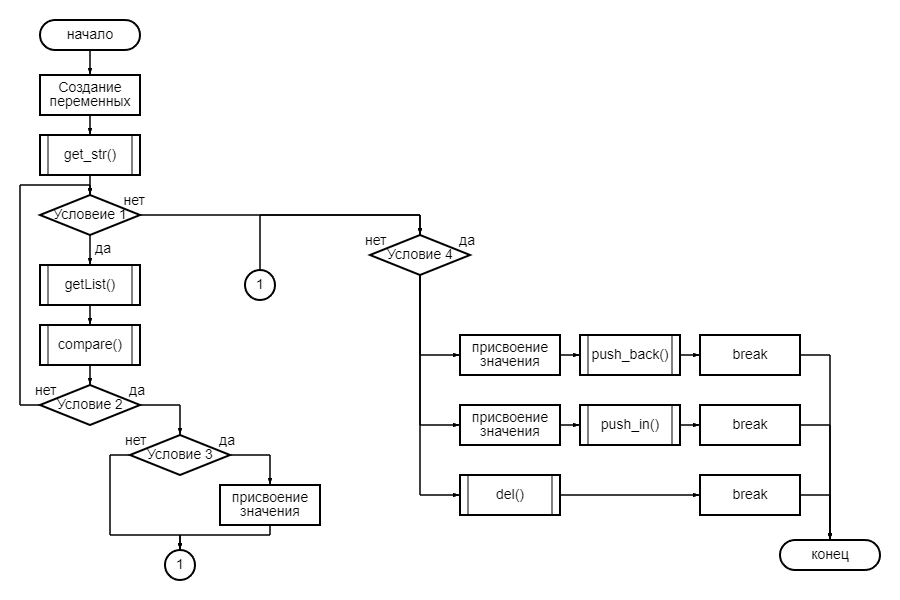
Blocks::get\_str() Blocks::compare()

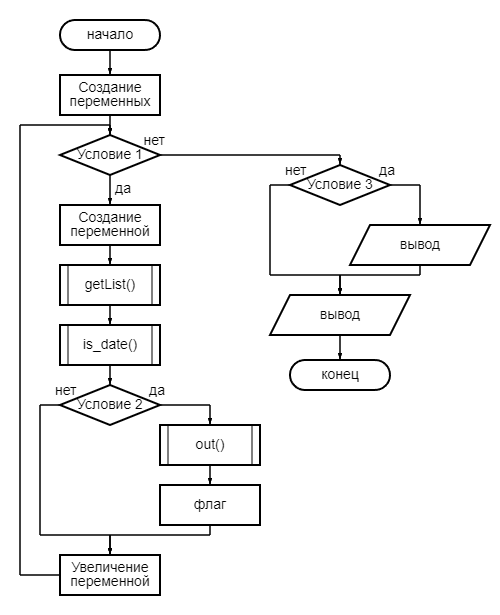
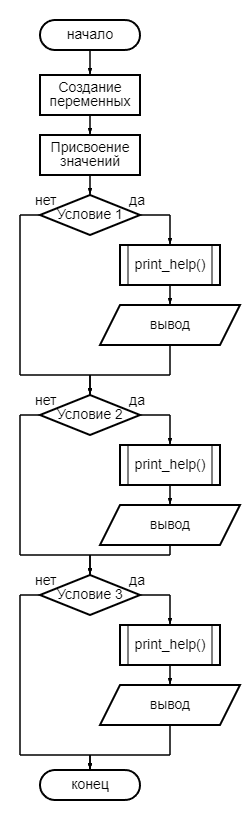
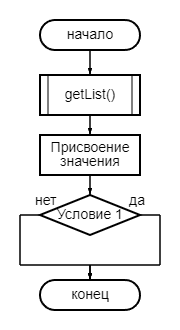
List\_pointers::print\_data() List::push\_in()

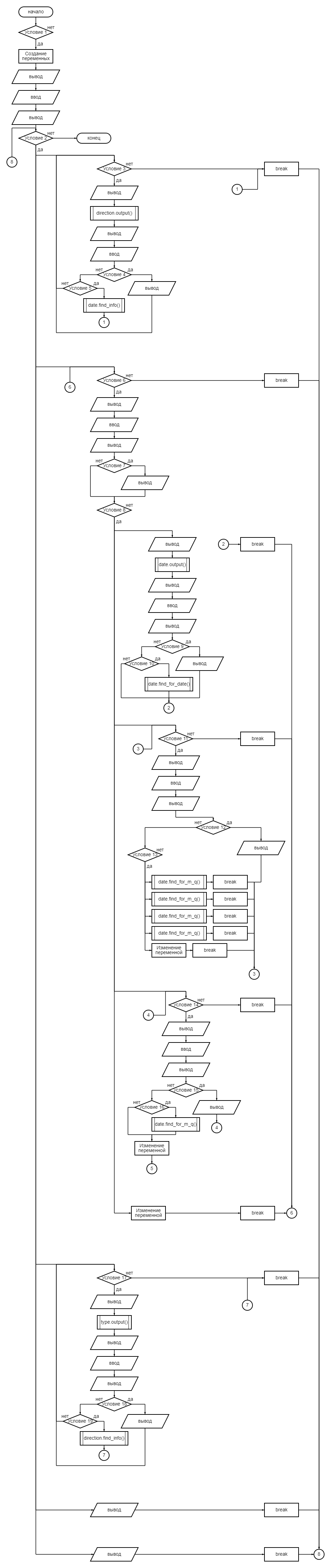
List::find\_info() List::get\_data()

List::find\_for\_m\_q() List\_pointers::print() Blocks::is\_date()

menu()



Текст программы.

main.cpp

#include <iostream>

#include <fstream>

#include <locale>

#include "List.h"

#include "Menu.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

ifstream departures;

ofstream res;

ofstream protocol;

departures.open("Flights\_Data.txt");

res.open("Out.txt");

protocol.open("protocol.txt");

List type(1), date(2), flight(3), direction(4);

protocol << "Созданы списки, начинается чтение файлов\n\n";

while (!departures.eof())

{

protocol << "Чтение дат\n";

date.get\_data(departures, protocol);

protocol << "Чтение типов\n";

type.get\_data(departures, protocol);

protocol << "Чтение городов\n";

direction.get\_data(departures, protocol);

protocol << "Чтение рейсов\n";

flight.get\_data(departures, protocol);

protocol << "\n\nПодключение данных к списку дат\n";

date.connect(date.last, direction.last\_added, type.last\_added, flight.last\_added);

protocol << "\n\nПодключение данных к списку городов\n";

direction.connect(direction.last, date.last\_added, type.last\_added, flight.last\_added);

}

protocol << "Списки заполнены:\n\n";

protocol << "Спискок дат:\n";

date.output(protocol, 2);

protocol << "\n\nСпискок городов:\n";

direction.output(protocol, 2);

protocol << "\n\nСпискок типов:\n";

type.output(protocol, 1);

protocol << "\n\nСпискок рейсов:\n";

flight.output(protocol, 1);

protocol << "\n\nНачало работы меню\n\n";

menu(type, date, flight, direction, res);

protocol << "Работа меню окончена\n\nДанные списков удаляются";

type.del();

date.del();

flight.del();

direction.del();

protocol << "\n\nДанные удалены. Работа программы окончена.";

departures.close();

res.close();

protocol.close();

return 0;

}

Menu.h

#pragma once

#include <iostream>

#include <fstream>

#include <locale>

#include "List.h"

using namespace std;

void menu(List& type, List& date, List& flight, List& direction, ofstream& res);

Menu.cpp

#include "Menu.h"

void menu(List& type, List& date, List& flight, List& direction, ofstream& res)

{

int action = -1;

while (action != 0)

{

int choise\_in = -1;

int choise = -1;

cout << "Выберите действие:\n1 - Найти информацию для пункта назначения\n2 - Найти информацию для даты/квартала/месяца\n3 - Найти информацию по типу самолета\n\n0 - Завершить работу\n\n";

res << "Выберите действие:\n1 - Найти информацию для пункта назначения\n2 - Найти информацию для даты/квартала/месяца\n3 - Найти информацию по типу самолета\n\n0 - Завершить работу\n\n";

cin >> action;

cout << endl;

res << "Пользователь вводит в консоль: " << action << endl << endl;

switch (action)

{

case 1:

while (choise != 0)

{

cout << "Выберите пункт назначения:" << endl;

res << "Выберите пункт назначения:" << endl;

direction.output(res);

cout << "0 - Отмена\n" << endl;

res << "0 - Отмена\n" << endl;

cin >> choise;

cout << endl;

res << "Пользователь вводит в консоль: " << choise << endl << endl;

if (choise > direction.size || choise < 0)

{

cout << "ОШИБКА! Некорректный номер" << endl;

res << "ОШИБКА! Некорректный номер" << endl;

}

else if (choise != 0)

{

date.find\_info(res, direction.getList(choise - 1)->info.get\_prt());

break;

}

}

break;

case 2:

while (choise != 0)

{

cout << "Выберите вариант:\n1 - Найти по дате\n2 - Найти по кварталу\n3 - Найти по месяцу\n0 - Отмена\n\n";

res << "Выберите вариант:\n1 - Найти по дате\n2 - Найти по кварталу\n3 - Найти по месяцу\n0 - Отмена\n\n";

cin >> choise;

res << "Пользователь вводит в консоль: " << choise << endl << endl;

cout << endl;

if (choise > 3 || choise < 0)

{

cout << "ОШИБКА! Некорректный номер" << endl;

res << "ОШИБКА! Некорректный номер" << endl;

}

switch (choise)

{

case 1:

cout << "Выберите дату:" << endl;

res << "Выберите дату:" << endl;

date.output(res);

cout << "0 - Отмена\n" << endl;

res << "0 - Отмена\n" << endl;

cin >> choise\_in;

cout << endl;

res << "Пользователь вводит в консоль: " << choise\_in << endl << endl;

if (choise > date.size || choise < 0)

{

cout << "ОШИБКА! Некорректный номер" << endl;

res << "ОШИБКА! Некорректный номер" << endl;

}

else if (choise != 0)

{

date.find\_for\_date(res, choise\_in - 1);

break;

}

break;

case 2:

while (choise\_in != 0)

{

cout << "Выберите квартал:\n1 - I квартал (январь, февраль, март)\n2 - II квартал (апрель, май, июнь)\n3 - III квартал (июль, август, сентябрь)\n4 - IV квартал (октябрь, ноябрь, декабрь)\n0 - Отмена\n\n";

res << "Выберите квартал:\n1 - I квартал (январь, февраль, март)\n2 - II квартал (апрель, май, июнь)\n3 - III квартал (июль, август, сентябрь)\n4 - IV квартал (октябрь, ноябрь, декабрь)\n0 - Отмена\n\n";

cin >> choise\_in;

cout << endl;

res << "Пользователь вводит в консоль: " << choise\_in << endl << endl;

if (choise > 4 || choise < 0)

{

cout << "ОШИБКА! Некорректный номер" << endl;

res << "ОШИБКА! Некорректный номер" << endl;

}

else

{

switch (choise\_in)

{

case 1:

date.find\_for\_m\_q(res, 1, 2, 3);

break;

case 2:

date.find\_for\_m\_q(res, 4, 5, 6);

break;

case 3:

date.find\_for\_m\_q(res, 7, 8, 9);

break;

case 4:

date.find\_for\_m\_q(res, 10, 11, 12);

break;

default:

choise\_in = -1;

break;

}

break;

}

}

break;

case 3:

while (choise\_in != 0)

{

cout << "Введите номер месяца (0 - Отмена): ";

res << "Введите номер месяца (0 - Отмена): ";

cin >> choise\_in;

cout << endl;

res << "Пользователь вводит в консоль: " << choise\_in << endl << endl;

if (choise\_in < 0 || choise\_in > 12)

{

cout << "ОШИБКА! Некорректный номер" << endl;

res << "ОШИБКА! Некорректный номер" << endl;

}

else

{

if (choise\_in != 0)

date.find\_for\_m\_q(res, choise\_in);

choise\_in = -1;

break;

}

}

break;

case 0:

choise = 0;

break;

default:

break;

}

}

break;

case 3:

while (choise != 0)

{

cout << "Выберите тип самолета:" << endl;

res << "Выберите тип самолета:" << endl;

type.output(res);

cout << "0 - Отмена\n" << endl;

res << "0 - Отмена\n" << endl;

cin >> choise;

cout << endl;

res << "Пользователь вводит в консоль: " << choise << endl << endl;

if (choise > type.size || choise < 0)

{

cout << "ОШИБКА! Некорректный номер" << endl;

res << "ОШИБКА! Некорректный номер" << endl;

}

else if (choise != 0)

{

direction.find\_info(res, type.getList(choise - 1)->info.get\_prt());

break;

}

}

case 0:

cout << "Работа программы завершена" << endl;

res << "Работа программы завершена";

break;

default:

cout << "ОШИБКА! Неизвестная команда" << endl << endl;

res << "ОШИБКА! Неизвестная команда" << endl << endl;

break;

}

}

}

List.h

#pragma once

#include "Node.h"

class List

{

Node\* head;

int type;

public:

ofstream protocol;

int size = 0;

Node\* last;

Block\_node\* last\_added;

List(int type);

void push\_back(Blocks info);

void push\_in(Blocks info, Blocks added);

void get\_data(ifstream& flights, ofstream& protocol);

void connect(Node\* to, Block\_node\* dir\_or\_date, Block\_node\* type, Block\_node\* num);

Node\* getList(int k);

void output(ofstream& res, int t=0);

void find\_info(ofstream& res, Block\_node\* to\_find);

void find\_for\_date(ofstream& res, int k, int t=0);

void find\_for\_m\_q(ofstream& res, int month, int m2=0, int m3=0);

void del();

};

List.cpp

#include "List.h"

#include <iostream>

#include <fstream>

using namespace std;

List::List(int type)

{

size = 0;

head = nullptr;

this->type = type;

}

void List::push\_back(Blocks info)

{

if (head == nullptr)

{

head = new Node(info);

last = head;

}

else

{

Node\* current = head;

while (current->pnext != nullptr)

{

current = current->pnext;

}

current->pnext = new Node(info);

last = current->pnext;

}

size++;

}

void List::push\_in(Blocks info, Blocks added)

{

Node\* current = head;

if (size > 1 && current->info.get\_prt() != added.get\_prt())

{

while (current->pnext->info.get\_prt() != added.get\_prt())

{

current = current->pnext;

}

current->pnext = new Node(info, current->pnext);

last = current->pnext;

}

else

{

head = new Node(info, current);

last = head;

}

size++;

}

void List::get\_data(ifstream& flights, ofstream& protocol)

{

Blocks line;

Blocks added;

int add = 1;

line.get\_str(flights);

protocol << "Получена строка: ";

line.print\_data(protocol, 1);

protocol << endl;

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

{

added = getList(i)->info;

add = added.compare(added, line, type);

if (add == 2 || add == 0)

{

if (add == 0)

{

last\_added = added.get\_prt();

last = getList(i);

}

break;

}

}

switch (add)

{

case 1:

protocol << "Строка добавляется в конец списка\n";

last\_added = line.get\_prt();

push\_back(line);

break;

case 2:

protocol << "Строка добавляется внутрь списка\n";

last\_added = line.get\_prt();

push\_in(line, added);

break;

default:

protocol << "Строка уже лежит в списке, данные удаляются\n";

line.del();

break;

}

}

void List::connect(Node\* to, Block\_node\* dir\_or\_date, Block\_node\* type, Block\_node\* num)

{

to->connected\_info.add(dir\_or\_date, type, num);

}

Node\* List::getList(int k)

{

Node\* tmp = head;

for (int i = 0; i < k; i++) {

tmp = tmp->pnext;

}

return tmp;

}

void List::output(ofstream& res, int t)

{

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

{

if (t == 0)

{

cout << i + 1 << ") ";

res << i + 1 << ") ";

}

if (t == 0 || t == 1)

getList(i)->info.print\_data(res, t);

if (t == 2)

find\_for\_date(res, i, 1);

if (t == 0)

cout << endl;

if (t != 0 && i != size - 1)

{

if (t == 1)

res << "\n\\|/";

else

res << "\\|/";

}

res << endl;

}

}

void List::find\_info(ofstream& res, Block\_node\* to\_find)

{

bool f = 0, f1;

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

{

Node\* tmp = getList(i);

f1 = tmp->connected\_info.search(res, tmp->info.get\_prt(), to\_find);

if (f1)

f = 1;

}

if (!f)

{

cout << "По данному запросу ничего не найдено" << endl;

res << "По данному запросу ничего не найдено" << endl;

}

cout << "\n\n";

res << "\n\n";

}

void List::find\_for\_date(ofstream& res, int k, int t)

{

Node\* tmp = getList(k);

tmp->connected\_info.out(res, tmp->info.get\_prt(), t);

if (t == 0)

{

cout << "\n\n";

res << "\n\n";

}

}

void List::find\_for\_m\_q(ofstream& res, int month, int m2, int m3)

{

Node\* tmp;

bool f = 0;

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

{

int act = 0;

tmp = getList(i);

act = tmp->info.is\_date(tmp->info, month, m2, m3);

if (act)

{

tmp->connected\_info.out(res, tmp->info.get\_prt());

f = 1;

}

}

if (!f)

{

cout << "По данному запросу ничего не найдено" << endl;

res << "По данному запросу ничего не найдено" << endl;

}

cout << "\n\n";

res << "\n\n";

}

void List::del()

{

while (head)

{

Node\* current = head;

head->info.del();

head->connected\_info.del();

head = head->pnext;

delete current;

}

size = 0;

}

Node.h

#pragma once

#include "Blocks.h"

#include "List\_pointers.h"

struct Node

{

Node\* pnext;

Blocks info;

List\_pointers connected\_info;

Node(Blocks info, Node\* pnext = nullptr);

};

Node.cpp

#include "Node.h"

Node::Node(Blocks info, Node\* pnext)

{

this->info = info;

this->pnext = pnext;

}

Blocks.h

#pragma once

#include "Block\_node.h"

#include <fstream>

using namespace std;

class Blocks

{

Block\_node\* head;

public:

Blocks();

Block\_node\* get\_prt();

int size = 0;

void push\_back(Data info);

void get\_str(ifstream& departure);

int compare(Blocks l1, Blocks l2, int type);

int is\_date(Blocks l, int m, int m1 = 0, int m2 = 0);

void print\_data(ofstream& res, int type=0);

Block\_node\* getList(int k);

void del();

};

Blocks.cpp

#include "Blocks.h"

#include <iostream>

#include <fstream>

using namespace std;

Blocks::Blocks()

{

size = 0;

head = nullptr;

}

Block\_node\* Blocks::get\_prt()

{

return head;

}

void Blocks::push\_back(Data info)

{

if (head == nullptr)

{

head = new Block\_node(info);

}

else

{

Block\_node\* current = head;

while (current->pnext != nullptr)

{

current = current->pnext;

}

current->pnext = new Block\_node(info);

}

size++;

}

void Blocks::get\_str(ifstream& departure)

{

departure.unsetf(ios::skipws);

char tmp = NULL;

int count = 0;

Data s;

while (1)

{

if (count < N)

{

departure >> tmp;

if (tmp == '|' || tmp == '\n' || departure.eof())

{

s.len = count;

push\_back(s);

break;

}

else

{

s.data[count] = tmp;

count++;

}

}

else

{

s.len = count;

push\_back(s);

count = 0;

Data s;

}

}

}

int Blocks::compare(Blocks l1, Blocks l2, int type)

{

int blc, last;

char tmp1 = NULL, tmp2 = NULL;

Data d1, d2;

blc = min(l1.size, l2.size);

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

{

last = i;

d1 = l1.getList(i)->material;

d2 = l2.getList(i)->material;

if (type == 2)

{

int d, m, y, dd,mm,yy;

d = (d1.data[0] - '0') \* 10 + (d1.data[1] - '0');

m = (d1.data[3] - '0') \* 10 + (d1.data[4] - '0');

y = (d1.data[6] - '0') \* 1000 + (d1.data[7] - '0') \* 100 + (d1.data[8] - '0') \* 10 + (d1.data[9] - '0');

dd = (d2.data[0] - '0') \* 10 + (d2.data[1] - '0');

mm = (d2.data[3] - '0') \* 10 + (d2.data[4] - '0');

yy = (d2.data[6] - '0') \* 1000 + (d2.data[7] - '0') \* 100 + (d2.data[8] - '0') \* 10 + (d2.data[9] - '0');

if (y == yy)

{

if (m == mm)

{

if (d == dd)

return 0;

else

{

if (d < dd)

return 1;

return 2;

}

}

else

{

if (m < mm)

return 1;

return 2;

}

}

else

{

if (y < yy)

return 1;

return 2;

}

}

else

{

int dt;

dt = min(d1.len, d2.len);

for (int j = 0; j < dt; j++)

{

tmp1 = d1.data[j];

tmp2 = d2.data[j];

if (tmp1 < tmp2)

return 1;

if (tmp1 > tmp2)

return 2;

}

}

}

if (l1.size < l2.size || l1.getList(last)->material.len < l2.getList(last)->material.len)

return 1;

if (l1.size > l2.size || l1.getList(last)->material.len > l2.getList(last)->material.len)

return 2;

return 0;

}

int Blocks::is\_date(Blocks l, int m, int m1, int m2)

{

Data dt;

dt = l.getList(0)->material;

int month = (dt.data[3] - '0') \* 10 + (dt.data[4] - '0');

if (month == m || month == m1 || month == m2)

return 1;

return 0;

}

void Blocks::print\_data(ofstream& res, int type)

{

Data n;

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

{

n = getList(i)->material;

for (int j = 0; j < n.len; j++)

{

if (type == 0)

cout << n.data[j];

res << n.data[j];

}

if (n.len == N)

res << "->";

}

}

Block\_node\* Blocks::getList(int k)

{

Block\_node\* tmp = head;

for (int i = 0; i < k; i++) {

tmp = tmp->pnext;

}

return tmp;

}

void Blocks::del()

{

while (head)

{

Block\_node\* current = head;

head = head->pnext;

delete current;

}

size = 0;

}

Block\_node.h

#pragma once

#include "Data.h"

struct Block\_node

{

Block\_node\* pnext;

Data material;

Block\_node(Data material, Block\_node\* pnext = nullptr);

};

Block\_node.cpp

#include "Block\_node.h"

Block\_node::Block\_node(Data material, Block\_node\* pnext)

{

this->material = material;

this->pnext = pnext;

}

Data.h

#pragma once

const unsigned N = 30;

struct Data

{

char data[N];

int len;

};

List\_pointer.h

#pragma once

#include "Node\_pointer.h"

#include <fstream>

using namespace std;

class List\_pointers

{

Node\_pointer\* head;

int size;

public:

List\_pointers();

void push\_back(Ptr lnk);

void add(Block\_node\* inf1, Block\_node\* inf2, Block\_node\* inf3);

bool search(ofstream& res, Block\_node\* given, Block\_node\* obj);

void out(ofstream& res, Block\_node\* given, int t=0);

void print(ofstream& res, Block\_node\* given, Node\_pointer\* tmp, int var, int t=0);

void print\_help(ofstream& res, Block\_node\* data, int t=0);

Node\_pointer\* getList(int k);

void del();

};

List\_pointer.cpp

#include "List\_pointers.h"

#include <iostream>

#include <fstream>

using namespace std;

List\_pointers::List\_pointers()

{

head = nullptr;

size = 0;

}

void List\_pointers::push\_back(Ptr lnk)

{

if (head == nullptr)

{

head = new Node\_pointer(lnk);

}

else

{

Node\_pointer\* current = head;

while (current->pnext != nullptr)

{

current = current->pnext;

}

current->pnext = new Node\_pointer(lnk);

}

size++;

}

void List\_pointers::add(Block\_node\* inf1, Block\_node\* inf2, Block\_node\* inf3)

{

Ptr a;

a.dir\_or\_date = inf1;

a.type = inf2;

a.num = inf3;

push\_back(a);

}

bool List\_pointers::search(ofstream& res, Block\_node\* given, Block\_node\* obj)

{

Node\_pointer\* tmp;

bool f = 0;

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

{

tmp = getList(i);

if (tmp->inf.dir\_or\_date == obj)

{

print(res, given, tmp, 1);

f = 1;

}

if (tmp->inf.type == obj)

{

print(res, given, tmp, 2);

f = 1;

}

}

return f;

}

void List\_pointers::out(ofstream& res, Block\_node\* given, int t)

{

Node\_pointer\* tmp;

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

{

tmp = getList(i);

print(res, given, tmp, 3, t);

}

}

void List\_pointers::print(ofstream& res, Block\_node\* given, Node\_pointer\* tmp, int var, int t)

{

Block\_node\* a;

Block\_node\* b;

Block\_node\* c;

a = tmp->inf.dir\_or\_date;

b = tmp->inf.type;

c = tmp->inf.num;

if (var == 1)

{

print\_help(res, a, t);

if (t == 0)

cout << " - ";

res << " - ";

print\_help(res, given, t);

if (t == 0)

cout << "|";

res << "|";

print\_help(res, b, t);

if (t == 0)

cout << "|";

res << "|";

print\_help(res, c, t);

if (t == 0)

cout << endl;

res << endl;

}

if (var == 2)

{

print\_help(res, b, t);

if (t == 0)

cout << " - ";

res << " - ";

print\_help(res, given, t);

if (t == 0)

cout << "|";

res << "|";

print\_help(res, a, t);

if (t == 0)

cout << "|";

res << "|";

print\_help(res, c, t);

if (t == 0)

cout << endl;

res << endl;

}

if (var == 3)

{

print\_help(res, given, t);

if (t == 0)

cout << " - ";

res << " - ";

print\_help(res, a, t);

if (t == 0)

cout << "|";

res << "|";

print\_help(res, b, t);

if (t == 0)

cout << "|";

res << "|";

print\_help(res, c, t);

if (t == 0)

cout << endl;

res << endl;

}

}

void List\_pointers::print\_help(ofstream& res, Block\_node\* data, int t)

{

while (1)

{

for (int i = 0; i < data->material.len; i++)

{

if (t == 0)

cout << data->material.data[i];

res << data->material.data[i];

}

if (data->pnext == nullptr)

break;

else

{

data = data->pnext;

if (t != 0)

res << "->";

}

}

}

Node\_pointer\* List\_pointers::getList(int k)

{

Node\_pointer\* tmp = head;

for (int i = 0; i < k; i++) {

tmp = tmp->pnext;

}

return tmp;

}

void List\_pointers::del()

{

while (head)

{

Node\_pointer\* current = head;

head = head->pnext;

delete current;

}

size = 0;

}

Node\_pointer.h

#pragma once

#include "Ptr.h"

struct Node\_pointer

{

Node\_pointer\* pnext;

Ptr inf;

Node\_pointer(Ptr inf, Node\_pointer\* pnext = nullptr);

};

Node\_pointer.cpp

#include "Node\_pointer.h"

Node\_pointer::Node\_pointer(Ptr inf, Node\_pointer\* pnext)

{

this->inf = inf;

this->pnext = pnext;

}

Prt.h

#pragma once

#include "Block\_node.h"

struct Ptr

{

Block\_node\* dir\_or\_date;

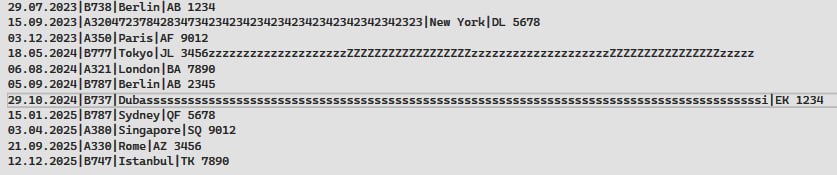
Block\_node\* type;

Block\_node\* num;

};

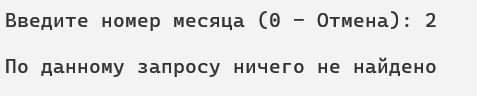
Результат работы программы.

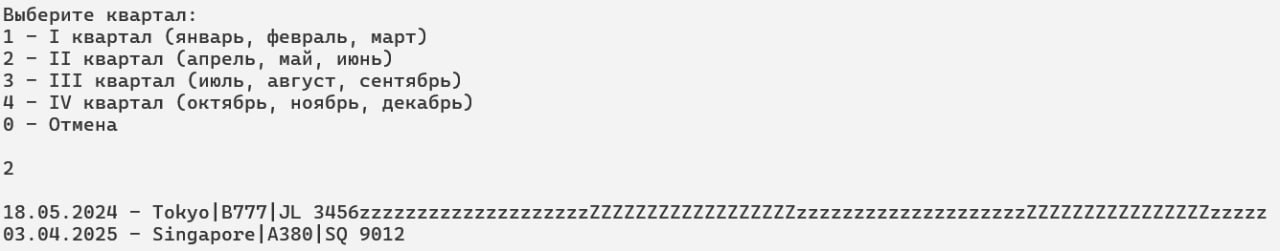
Входные данные

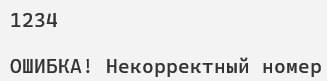


Работа программы (вывод в файл и в консоль одинаковый)

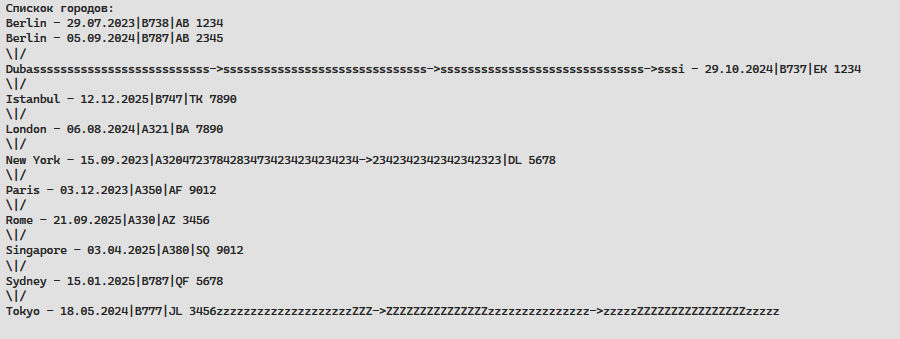
 



Пример иллюстрации списка в протоколе



Вывод.

В ходе выполнения данной курсовой работы изучены мультисписки и взаимодействие с ними. Написана программа, способная хранить информацию в эффективном формате хранения, а также отвечать на запросы пользователя.