**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3353 |  | Шинкарь К. Д. |
| Преподаватель |  | Калмычков В.А. |

**КУРСОВАЯ РАБОТА**

**по дисциплине «Программирование»**

**Тема: Информационная система**

Санкт-Петербург

2024

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)**

Студент Шинкарь К. Д.

Группа 3353

Тема работы: Информационная система

Исходные данные: необходимо создать информационную систему, основываясь на односвязных списках.

|  |  |  |
| --- | --- | --- |
| Студент |  | Шинкарь К. Д. |
|  |  |  |
| Преподаватель |  | Калмычков В.А. |

Содержание пояснительной записки:

Предполагаемый объём пояснительной записки:

Не менее \_\_\_\_ страниц.

Дата выдачи задания: 20.02.2024

Дата сдачи реферата:

Дата защиты реферата:

**АННОТАЦИЯ**

В ходе выполнения курсовой работы была создана программа для отслеживания расписания пригородных автобусов Ленобласти. Программа позволяет хранить информацию о номерах автобусов, дате и времени отправки, начальном и конечном пунктах, а также позволяет пользователю получать необходимую информацию по одному из трёх запросов. Для реализации программы были использованы методы, изученные в курсе дисциплины «Программирования», таких как использование структур, классов, методов, односвязных списков.

**SUMMARY**

During the development of the course work there was created a program, which is able to follow the timetable of the buses of Saint Petersburgs’ area. The program allows to keep the information of buses’ numbers, dates and departure time, points of departure and destination. Moreover, it lets the user to get necessary information about one of the three provided requests. The methods, which were learned during the course of “Programming” discipline, such as using of structures, classes, methods of classes, singly linked lists, were used for the realization of the program.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1. Исходная формулировка 5

2. Анализ задачи 6

2.1 Описание списка 7

3 Контрольный пример 8

4. Особенности реализации на компьютере 9

5. Описание классов/структур 10

6. Формат хранения данных 20

7. Организация диалога с пользователем 25

8. Функции 27

9. Блок-схемы 33

10. Текст программы 56

11. Результаты работы программы 84

12. Вывод 85

ВВЕДЕНИЕ

Данная курсовая работа предполагает написание программы, которая будет способна хранить данные, изначально представленные в виде таблицы, в эффективном формате хранения, а также выполнять поисковые операции и выводить данные согласно пользовательским запросам, предусмотренным изначально.

Требуется разработать программу, которая будет хранить данные о расписании рейсовых автобусов Ленинградской области, которая будет соответствовать вышеописанным требованиям.

            Метод решения: при выполнении курсовой работы необходимо использовать иерархические списки, позволяющие логично структурировать данные, добавлять методы, нужные для удобного обращения к необходимой информации, а также учитывать эффективный расход памяти, поскольку необходимая информация хранится в единичном экземпляре в некоторой области памяти. В ходе работы используются указатели для передачи данных из различных областей памяти, что не противоречит требованиям к реализации программы.

1. Исходная формулировка задания

7.16. Служба автоперевозок имеет сведения, размещенные в эффективном формате хранения: названия начальных и конечных пунктов, номера автобусов с привязкой их маршрутов (начальный – конечный пункты), день и время отправления. Найти:

- все автобусы, отправившиеся до определенного дня и/или времени, выдача по названиям пунктов отправления по алфавиту,

- всю информацию о конкретном автобусе по его номеру,

- все маршруты, которые начинаются и/или заканчиваются в конкретном пункте, выдача с упорядочиванием по номеру автобуса.

2. Анализ задачи

Для полного и структурированного анализа предложенного задания необходимо определить на начальном этапе каким образом предпочтительнее эффективно хранить данные о расписании рейсовых автобусов Ленинградской области. Это данные о номерах автобусов, дате и времени их отправления, начальном и конечном пунктах. Для каждого наименования логично выделить по одной графе таблицы.

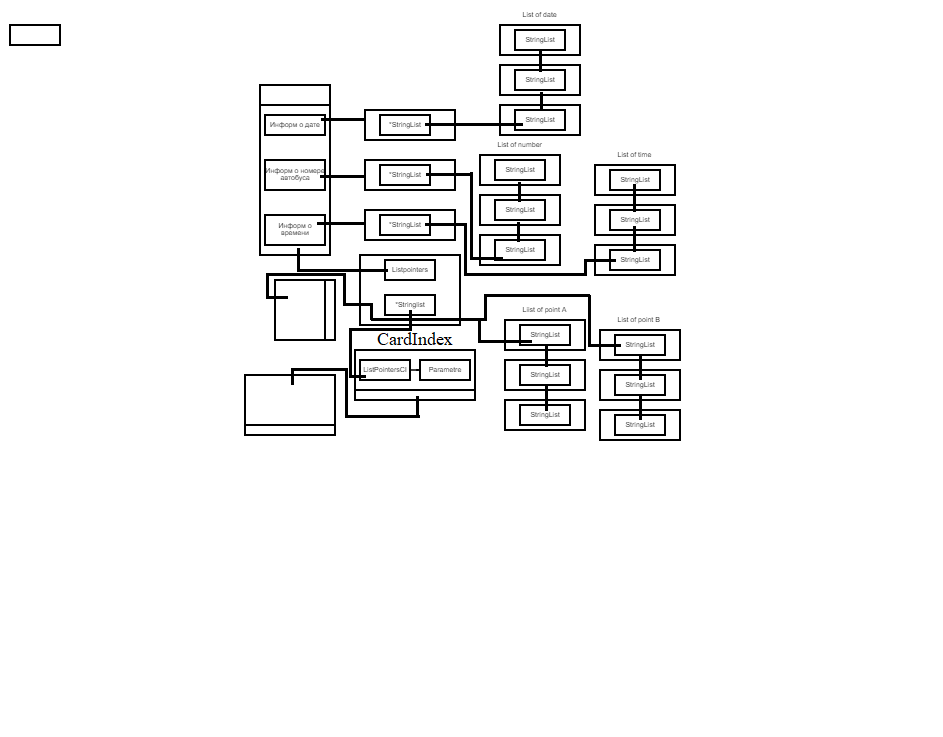
Необходимо организовать механизм работы программы, способный качественно работать с предоставленными данными. Он должен включать в себя возможность получения всей информации о конкретном автобусе по его номеру, быть способным выводить номера всех автобусов, отправившихся до определенного дня и/или времени, выдача должна быть по названиям пунктов отправления по алфавиту, а также подбирать все маршруты, которые начинаются и/или заканчиваются в конкретном пункте, выдача которых происходит с упорядочиванием по номеру автобуса.

Все это требует разработки структуры данных и алгоритмов, обеспечивающих эффективное выполнение требуемых операций с данными.

Информация должна храниться в иерархических односвязных списках, что позволит удобно ориентироваться в ней и эффективно расходовать память.

2.1 Описание списка

Для хранения данных необходима картотека. В элементе списка указателей хранится информация о пункте назначения и пункте отправления, а также и указатель на список указателей.



CardIndex – основная картотека, которая хранит информацию о указателях, в элементе ListPointersCI хранятся ещё два указателся, один на список строк и на список указателей ListPointers. В ListPointers хранятся указатели, необходимые нам (о дате, времени, номере автобуса).

3 Контрольный пример

Ввод:



Информация во входном файле расположена следующим образом:

1 графа – номер автобуса;

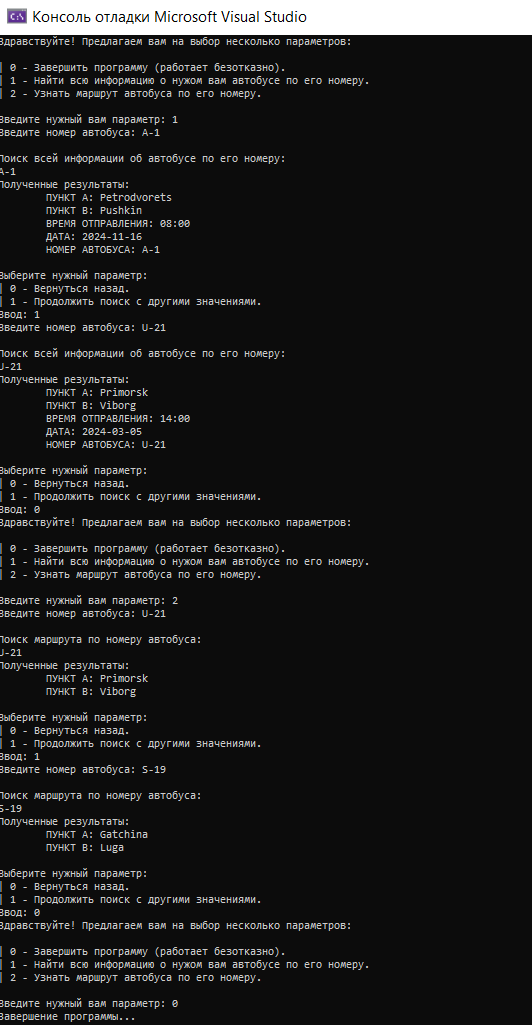
2 графа – дата отправления (может встречаться неоднократно);

3 графа – время отправления (может встречаться неоднократно);

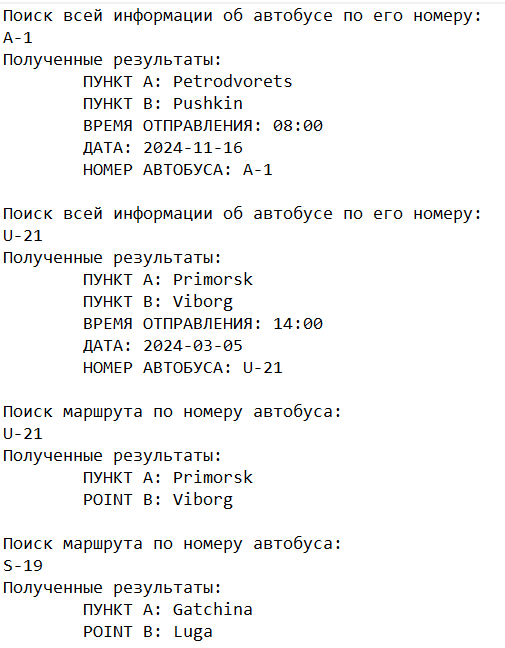
4 графа – пункт прибытия (может встречаться неоднократно);

5 графа – пункт отправления (может встречаться неоднократно);

Вывод в консоль:



Вывод в файл:



**Особенности реализации на компьютере**

В файле dataFile.txt дается табличка из данных. Программа создает 5 списков 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 создаёт картотеку. Далее запускается меню, где с помощью команд пользователя из консоли программа выполняет определенные действия.

Для полного и сокращенного поиска по номеру автобуса программа также проходится по картотеке и находит нужную нам информацию.

**Описание структур**

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **StringNode** | **String element** | Хранит строковый элемент |
| **StringNode\* pNext** | Указатель на следующий узел |
| **StringNode(String edElem, StringNode\* n)** | Конструктор: инициализирует элемент и указатель на следующий узел |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **StringList** | **StringNode\* pHead** | Указатель на первый элемент списка |
| **StringNode\* pTail** | Указатель на последний элемент списка |
| **StringNode\* pCur** | Указатель на текущий элемент списка |
| **StringList()** | Конструктор: инициализирует пустой список |
| **bool is\_empty()** | Проверяет, пуст ли список |
| **void pushFront(String newInfo)** | Добавляет новый элемент в начало списка |
| **void pushBack(String newInfo)** | Добавляет новый элемент в конец списка |
| **void insertAfter(StringNode\* elem, String newInfo)** | Вставляет новый элемент после указанного узла |
| **void popFront()** | Удаляет первый элемент списка |
| **void eraseAfter(StringNode\* elem)** | Удаляет элемент списка после указанного узла |
| **void readFromFile(ifstream& dataFile, int& spaces)** | Читает данные из файла и заполняет список |
| **int lengthList()** | Возвращает длину списка |
| **bool compareStringList(StringList S1, StringList S2)** | Сравнивает два списка строк |
| **void converToDate(StringList S, int\* date)** | Преобразует список строк в дату |
| **void littleConvertToDate(StringList S, int\* date)** | Преобразует список строк в сокращенную дату |
| **bool compareDates(StringList date1, StringList date2)** | Сравнивает две даты в виде списков строк |
| **bool comparebiba(StringList date1, StringList date2)** | Сравнивает две даты (особая логика) |
| **bool compareQuarters(StringList date1, StringList date2)** | Сравнивает кварталы дат |
| **bool isTrueDate(int\* date)** | Проверяет корректность даты |
|  | **void clearStringList()** | Очищает список строк |
| **void outData(ofstream& outData)** | Выводит данные списка в файл |
| **void outDataOnlyFile(ofstream& outData)** | Выводит данные списка в файл без дополнительных данных |
| **void outDataOnlyFileWithoutEnter(ofstream& outData)** | Выводит данные списка в файл без перевода строки |
| **int typeOfDate()** | Определяет тип даты |
| **void enterUserData()** | Ввод данных пользователем |
| **void fillWithInfo(char\* string)** | Заполняет список информацией |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **String** | **char string[N]** | Хранит строку |
| **int stringLenght = 0** | Длина строки |
| **int digit\_count = 0** | Количество цифр в строке |
| **void clearString()** | Очищает строку |
| **void addData(int index, char tmp\_symbol)** | Добавляет символ в строку по индексу |
| **char getData(int index)** | Возвращает символ по индексу |
| **void readString(ifstream& dataFile, int& spaces)** | Читает строку из файла |
| **void readStringV2(ifstream& dataFile, int& spaces, bool& is\_end)** | Читает строку из файла (вторая версия) |
| **void outData(ofstream& outFile)** | Выводит строку в файл |
| **void outDataWithoutEnter(ofstream& outFile)** | Выводит строку в файл без перевода строки |
| **void outDataOnlyFile(ofstream& outFile)** | Выводит строку в файл без дополнительных данных |
| **void outDataOnlyFileWithoutEnter(ofstream& outFile)** | Выводит строку в файл без перевода строки |
| **bool compareString(String S1, String S2)** | Сравнивает две строки |
| **bool compareFirst(String S1, String S2)** | Сравнивает первые символы двух строк |
| **void getStringLength()** | Возвращает длину строки |
| **void enterUserData()** | Ввод данных пользователем |
| **int typeOfDate()** | Определяет тип даты |
| **void converToDate(String S, int\* date)** | Преобразует строку в дату |
| **void littleConvertToDate(String S, int\* date)** | Преобразует строку в сокращенную дату |
| **bool isTrueDate(int\* date)** | Проверяет корректность даты |
|  | **bool inTheRightPeriodOfTime(String date1, String S2, String date2)** | Проверяет, находится ли дата в заданном периоде |
| **bool compareDates(String date1, String date2)** | Сравнивает две даты |
| **bool comparebiba(String date1, String date2)** | Сравнивает две даты (особая логика) |
| **bool compareQuarters(String date1, String date2)** | Сравнивает кварталы дат |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **NodePointersCI** | **StringList\* necessaryData** | Указатель на необходимые данные (список строк) |
| **ListPointers\* otherData** | Указатель на другие данные (список указателей) |
| **NodePointersCI\* pNext** | Указатель на следующий узел |
| **NodePointersCI(StringList\* editedData, ListPointers\* otherEdData, NodePointersCI\* n)** | Конструктор: инициализирует данные и указатель на следующий узел |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **NodePointers** | **StringList\* otherDataNUM** | Указатель на данные NUM (список строк) |
| **StringList\* otherDataTM** | Указатель на данные TM (список строк) |
| **StringList\* otherDataD** | Указатель на данные D (список строк) |
| **StringList\* otherDataPB** | Указатель на данные PB (список строк) |
| **NodePointers\* pNext** | Указатель на следующий узел |
| **NodePointers(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB, NodePointers\* n)** | Конструктор: инициализирует данные и указатель на следующий узел |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **NodeInfo** | **String data** | Хранит строковые данные |
| **NodeInfo(String edS)** | Конструктор: инициализирует строковые данные |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **Node** | **StringList data** | Хранит данные списка строк |
| **ListPointers otherData** | Хранит другие данные (список указателей) |
| **Node\* pNext** | Указатель на следующий узел |
| **Node(StringList editedInfo, Node\* n)** | Конструктор: инициализирует данные и указатель на следующий узел |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **ListPointersCI** | **NodePointersCI\* pHead** | Указатель на первый элемент списка указателей |
| **NodePointersCI\* pTail** | Указатель на последний элемент списка указателей |
| **NodePointersCI\* pCur** | Указатель на текущий элемент списка указателей |
| **ListPointersCI()** | Конструктор: инициализирует пустой список указателей |
| **bool is\_empty()** | Проверяет, пуст ли список |
| **void pushFront(StringList\* editedData, ListPointers\* otherEdData)** | Добавляет новый элемент в начало списка |
| **void pushBack(StringList\* editedData, ListPointers\* otherEdData)** | Добавляет новый элемент в конец списка |
| **void insertAfter(NodePointersCI\* elem, StringList\* editedData, ListPointers\* otherEdData)** | Вставляет новый элемент после указанного узла |
| **void popFront()** | Удаляет первый элемент списка |
| **void eraseAfter(NodePointersCI\* elem)** | Удаляет элемент списка после указанного узла |
| **void outDataListCI(ofstream& logs)** | Выводит данные списка указателей в файл |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **ListPointers** | **NodePointers\* pHead** | Указатель на первый элемент списка указателей |
| **NodePointers\* pTail** | Указатель на последний элемент списка указателей |
| **NodePointers\* pCur** | Указатель на текущий элемент списка указателей |
| **ListPointers()** | Конструктор: инициализирует пустой список указателей |
| **bool is\_empty()** | Проверяет, пуст ли список |
| **void pushFront(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB)** | Добавляет новый элемент в начало списка |
| **void pushBack(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB)** | Добавляет новый элемент в конец списка |
| **void insertAfter(NodePointers\* elem, StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB)** | Вставляет новый элемент после указанного узла |
| **void popFront()** | Удаляет первый элемент списка |
| **void eraseAfter(NodePointers\* elem)** | Удаляет элемент списка после указанного узла |
| **void outAllData(ofstream& logs)** | Выводит все данные списка указателей в файл |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **List** | **Node\* pHead** | Указатель на первый элемент списка |
| **Node\* pTail** | Указатель на последний элемент списка |
| **Node\* pCur** | Указатель на текущий элемент списка |
| **List()** | Конструктор: инициализирует пустой список |
| **bool is\_empty()** | Проверяет, пуст ли список |
| **void pushFront(StringList newInfo)** | Добавляет новый элемент в начало списка |
| **void pushBack(StringList newInfo)** | Добавляет новый элемент в конец списка |
| **void insertAfter(Node\* elem, StringList newInfo)** | Вставляет новый элемент после указанного узла |
| **void popFront()** | Удаляет первый элемент списка |
| **void eraseAfter(Node\* elem)** | Удаляет элемент списка после указанного узла |
| **bool isUnique(List& tmp, StringList S)** | Проверяет, является ли элемент уникальным в списке |
| **void readFromFile(ifstream& dataFile)** | Читает данные из файла и заполняет список |
| **void fillingWithData(List& tmp, StringList S)** | Заполняет массив данных |
| **void outAllDataList(ofstream& logs)** | Выводит все данные списка в файл |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **CardIndexNode** | **ListPointersCI necessaryDataAboutBus** | Список указателей с данными об автобусах |
| **char boba** | Дополнительный параметр |
| **CardIndexNode\* pNext** | Указатель на следующий узел |
| **CardIndexNode(ListPointersCI addedData, CardIndexNode\* n, char edPar)** | Конструктор: инициализирует данные и указатель на следующий узел |

| **Название** | **Входящие переменные/методы** | **Назначение переменных/методов** |
| --- | --- | --- |
| **CardIndex** | **CardIndexNode\* pHead** | Указатель на первый элемент списка |
| **CardIndexNode\* pTail** | Указатель на последний элемент списка |
| **CardIndexNode\* pCur** | Указатель на текущий элемент списка |
| **CardIndex()** | Конструктор: инициализирует пустой список |
| **bool is\_empty()** | Проверяет, пуст ли список |
| **void pushFront(ListPointersCI necessaryData, char edPar)** | Добавляет новый элемент в начало списка |
| **void pushBack(ListPointersCI necessaryData, char edPar)** | Добавляет новый элемент в конец списка |
| **void insertAfter(CardIndexNode\* elem, ListPointersCI necessaryData, char edPar)** | Вставляет новый элемент после указанного узла |
| **void popFront()** | Удаляет первый элемент списка |
| **void eraseAfter(CardIndexNode\* elem)** | Удаляет элемент списка после указанного узла |
| **void createEmptyList()** | Создает пустой список |
| **void readingFromFile(ifstream& dataFile, List& BusType, List& date, List& time, List& pointA, List& pointB, ofstream& logs)** | Читает данные из файла и заполняет списки |
| **void compareWithboba(List pointA)** | Сравнивает данные |
| **bool itWas(List pointA)** | Проверяет, были ли данные ранее |
| **void searchBybusType(ofstream& outData, ofstream& logs)** | Поиск по типу автобуса |
| **void searchByRoute(ofstream& outData, ofstream& logs)** | Поиск по маршруту |
| **void log(ofstream& log)** | Записывает данные в журнал |
| **void outToLog(ofstream& logs)** | Выводит данные в журнал |
| **void deleteAll(List& busType, List& date, List& time, List& pointA, ofstream& logs)** | Удаляет все данные |

**Формат хранения данных**

int main():

| **Переменная/Метод** | **Тип данных** | **Назначение** |
| --- | --- | --- |
| **main** | int | Основная функция программы |
| pointA | List | Объект для хранения данных о точке A |
| number | List | Объект для хранения данных о номере |
| time | List | Объект для хранения данных о времени |
| date | List | Объект для хранения данных о дате |
| pointB | List | Объект для хранения данных о точке B |
| alphabet | CardIndex | Объект для работы с картотекой |
| dataFile | ifstream | Файл для чтения данных |
| outFile | ofstream | Файл для записи выходных данных |
| logs | ofstream | Файл для записи логов |
| answerF | int | Переменная для хранения выбора пользователя в меню |
| answerSF | int | Переменная для хранения выбора в подменю |
| answerSS | int | Переменная для хранения выбора в подменю |
| setlocale | функция | Устанавливает локаль для корректного отображения русского текста |
| cout | объект | Вывод текста в консоль |
| cin | объект | Ввод данных с консоли |

CardIndex.cpp

| **Переменная** | **Тип данных** | **Назначение переменных** |
| --- | --- | --- |
| pHead | CardIndexNode\* | Указатель на первый элемент списка. |
| pTail | CardIndexNode\* | Указатель на последний элемент списка. |
| pCur | CardIndexNode\* | Текущий указатель для итерации по списку. |
| currentTime | time\_t | Переменная для хранения текущего времени. |
| tm\_time | tm | Структура для хранения локального времени. |
| timeStr | char[20] | Массив для хранения строки времени в формате "ГГГГ-ММ-ДД ЧЧ:ММ:СС". |
| necessaryInfo | ListPointersCI | Структура, содержащая информацию, необходимую для создания нового узла списка. |
| edPar | char | Переменная, представляющая дополнительный параметр для узла списка. |
| nextHead | CardIndexNode\* | Указатель на следующий элемент после головы, используемый при удалении первого элемента списка. |
| elem | CardIndexNode\* | Указатель на элемент списка, после которого будет вставлен новый узел. |
| pointA | List | Список, с которым будет произведено сравнение при итерации по текущему списку. |
| dataFile | ifstream | Входной файловый поток для чтения данных из файла. |
| BusType | List | Список типов автобусов, используемый при чтении данных из файла и заполнении списка. |
| date | List | Список дат, используемый при чтении данных из файла и заполнении списка. |
| time | List | Список времени, используемый при чтении данных из файла и заполнении списка. |
| pointB | List | Список пунктов назначения, используемый при чтении данных из файла и заполнении списка. |
| logs | ofstream | Выходной файловый поток для записи логов. |
| spaces | int | Счетчик пробелов, используемый для определения текущей позиции в строке файла при чтении данных. |
| s | StringList\* | Указатель на строковый список, используемый для временного хранения данных при чтении из файла. |
| cur | CardIndexNode\* | Текущий узел списка, используемый при поиске информации. |
| answer | StringList | Список строк, представляющий введенное пользователем значение для поиска информации. |

7. Организация диалога с пользователем

**Организация диалога с пользователем**

|  |  |
| --- | --- |
| Варианты ввода | Для файла:  |Номер автобуса|Дата|Время|Пункт назначения|Пункт отправления|  Для консоли:  Номер автобуса |
| Варианты вывода | Меню принимает разные формы:  1) Основные:  Здравствуйте, выберите нужный параметр:  | 0 - Завершить программу.  | 1 - Найти всю информацию о нужом вам автобусе по его номеру.  | 2 - Узнать маршрут автобуса по его номеру.  2) Для выбора типа самолёта:  Enter BusType:  3) Для выбора маршрута автобуса:  Enter BusType:  4) При завершении программы, она указывает об этом  5) Вся важная информация выводится в файл протокола и результирующий файл |

**Средства обеспечения ввода/вывода**

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

8. Функции

| **Функция + название переменных** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| main | int | Входные: Нет | int (код завершения) | Установка локали, чтение/запись из файлов, вызов остальных функций |
|  |  | Транзитные: Нет |  |  |
|  |  | Модифицируемые: pointA, number, time, date, pointB, alphabet, dataFile, outFile, logs, answerF, answerSF, answerSS |  |  |
| pointA | List | Модифицируемые |  |  |
| number | List | Модифицируемые |  |  |
| time | List | Модифицируемые |  |  |
| date | List | Модифицируемые |  |  |
| pointB | List | Модифицируемые |  |  |
| alphabet | CardIndex | Модифицируемые |  |  |
| dataFile | ifstream | Входные: const char \*filename, ios\_base::openmode mode |  | Открытие файла для чтения |
| outFile | ofstream | Входные: const char \*filename, ios\_base::openmode mode |  | Открытие файла для записи |
| logs | ofstream | Входные: const char \*filename, ios\_base::openmode mode |  | Открытие файла для записи логов |
| answerF | int | Модифицируемые |  |  |
| answerSF | int | Модифицируемые |  |  |
| answerSS | int | Модифицируемые |  |  |

**CardIndex::CardIndex()**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| CardIndex | Constructor | Входные: нет | Нет |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: pHead, pTail, pCur |  | Инициализация данных |

**CardIndex::log(ofstream& log)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| log | void | Входные: ofstream& log | Нет | Запись логов в файл |
|  |  | Транзитные: time\_t currentTime, tm tm\_time, char timeStr[20] |  |  |
|  |  | Модифицируемые: нет |  |  |

**CardIndex::is\_empty()**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| is\_empty | bool | Входные: нет | bool |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: нет |  |  |

**CardIndex::pushFront(ListPointersCI necessaryInfo, char edPar)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| pushFront | void | Входные: ListPointersCI necessaryInfo, char edPar | Нет |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: pHead, pTail |  | Изменение списка |

**CardIndex::pushBack(ListPointersCI necessaryInfo, char edPar)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| pushBack | void | Входные: ListPointersCI necessaryInfo, char edPar | Нет |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: pHead, pTail |  | Изменение списка |

**CardIndex::popFront()**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| popFront | void | Входные: нет | Нет |  |
|  |  | Транзитные: CardIndexNode\* nextHead |  |  |
|  |  | Модифицируемые: pHead, pTail |  | Изменение списка |

**CardIndex::insertAfter(CardIndexNode\* elem, ListPointersCI necessaryInfo, char edPar)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| insertAfter | void | Входные: CardIndexNode\* elem, ListPointersCI necessaryInfo, char edPar | Нет |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: pTail |  | Изменение списка |

**CardIndex::eraseAfter(CardIndexNode\* elem)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| eraseAfter | void | Входные: CardIndexNode\* elem | Нет |  |
|  |  | Транзитные: CardIndexNode\* nextElem |  |  |
|  |  | Модифицируемые: pTail |  | Изменение списка |

**CardIndex::createEmptyList()**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| createEmptyList | void | Входные: нет | Нет |  |
|  |  | Транзитные: char boba |  |  |
|  |  | Модифицируемые: нет |  | Создание пустого списка |

**CardIndex::compareWithboba(List pointA)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| compareWithboba | void | Входные: List pointA | Нет |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: pCur |  | Изменение списка |

**CardIndex::readingFromFile(ifstream& dataFile, List& BusType, List& date, List& time, List& pointA, List& pointB, ofstream& logs)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| readingFromFile | void | Входные: ifstream& dataFile, List& BusType, List& date, List& time, List& pointA, List& pointB, ofstream& logs | Нет |  |
|  |  | Транзитные: int spaces, StringList\* s |  |  |
|  |  | Модифицируемые: нет |  | Чтение данных из файла |

**CardIndex::searchBybusType(ofstream& outData, ofstream& logs)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| searchBybusType | void | Входные: ofstream& outData, ofstream& logs | Нет |  |
|  |  | Транзитные: StringList answer, CardIndexNode\* cur |  |  |
|  |  | Модифицируемые: pCur |  | Вывод информации |

**CardIndex::searchByRoute(ofstream& outData, ofstream& logs)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| searchByRoute | void | Входные: ofstream& outData, ofstream& logs | Нет |  |
|  |  | Транзитные: StringList answer, CardIndexNode\* cur |  |  |
|  |  | Модифицируемые: pCur |  | Вывод информации |

**CardIndex::deleteAll(List& busType, List& date, List& time, List& pointA, ofstream& logs)**

| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| deleteAll | void | Входные: List& busType, List& date, List& time, List& pointA, ofstream& logs | Нет |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: pHead, pTail |  | Удаление элементов |

**CardIndex::listModification(List& busType, List& date, List& time, List& pointA, ofstream& logs)**

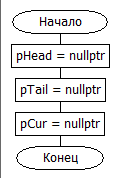
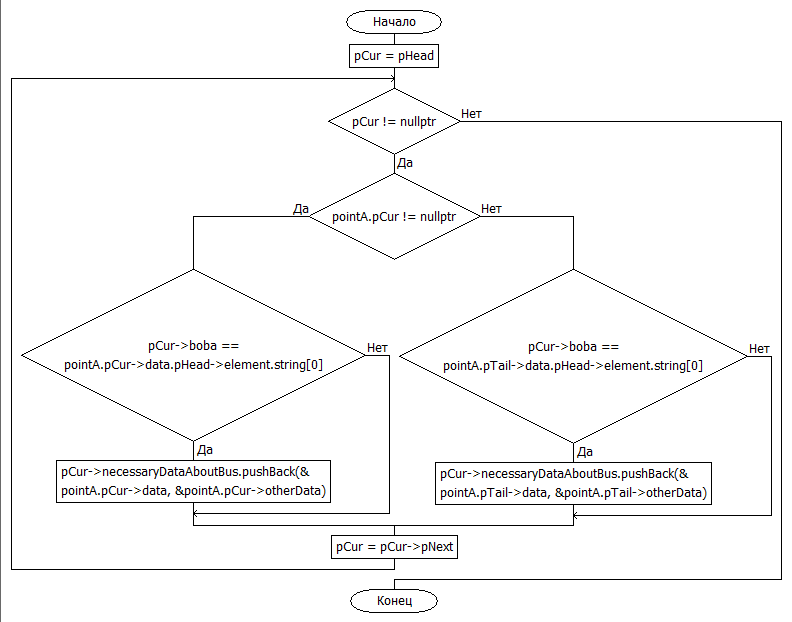
| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| listModification | void | Входные: List& busType, List& date, List& time, List& pointA, ofstream& logs | Нет |  |
|  |  | Транзитные: StringList tmpAnswer, CardIndexNode\* tmpCur |  |  |
|  |  | Модифицируемые: pCur, pHead |  | Изменение списка |

**CardIndex::~CardIndex()**

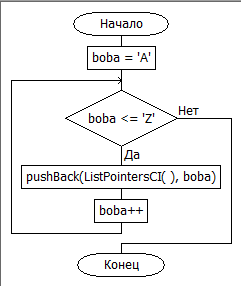
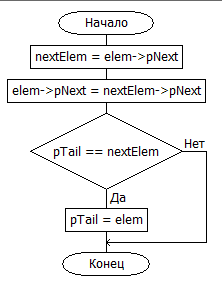
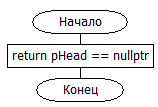
| **Имя функции** | **Тип данных** | **Параметры: входные, транзитные, модифицируемые** | **Возвращаемое значение** | **Внешние эффекты** |
| --- | --- | --- | --- | --- |
| ~CardIndex | Destructor | Входные: нет | Нет |  |
|  |  | Транзитные: нет |  |  |
|  |  | Модифицируемые: pHead, pTail |  | Удаление элементов |

9. Блок-схемы

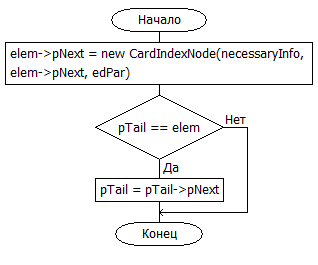
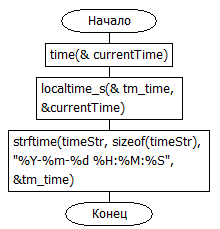
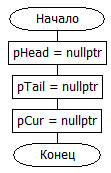
CardIndex CompareWith

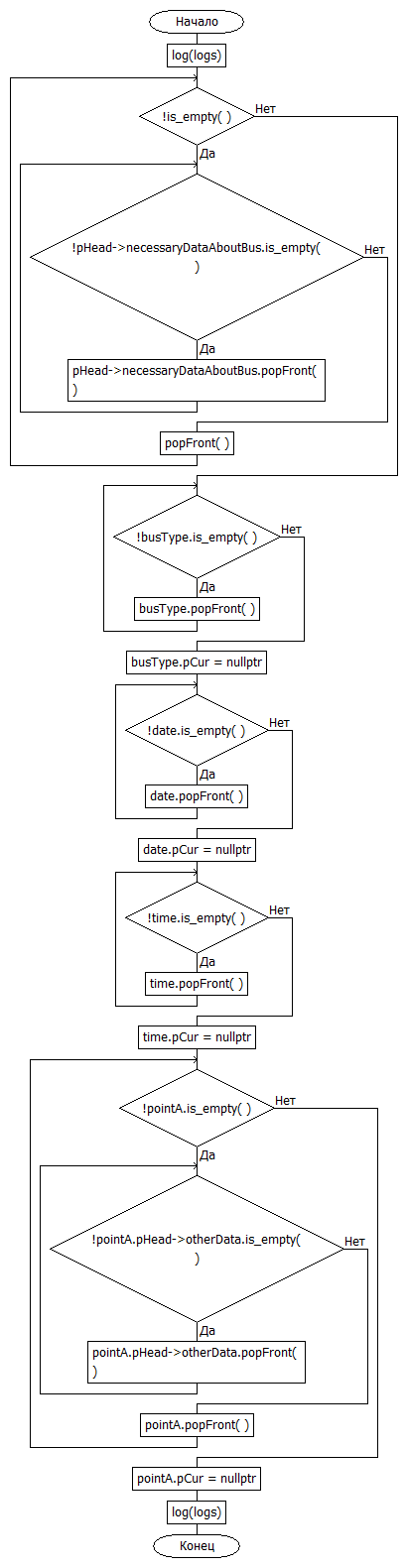
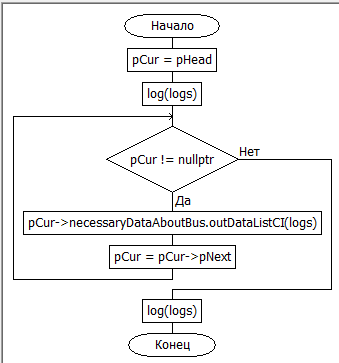
CreateEmptyList EraseAfter IsEmpty

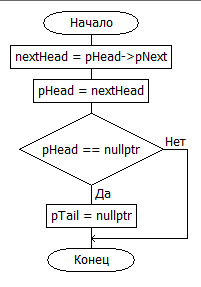
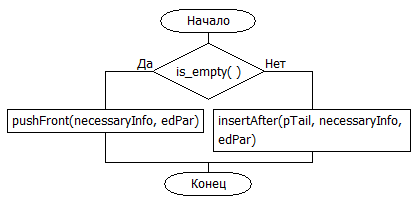
InsertAfter Log List

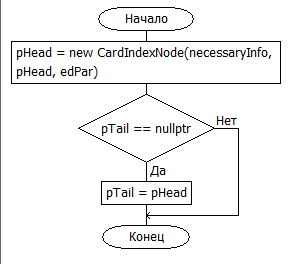
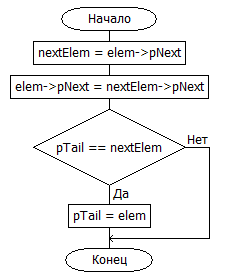
deleteAll OutToLog

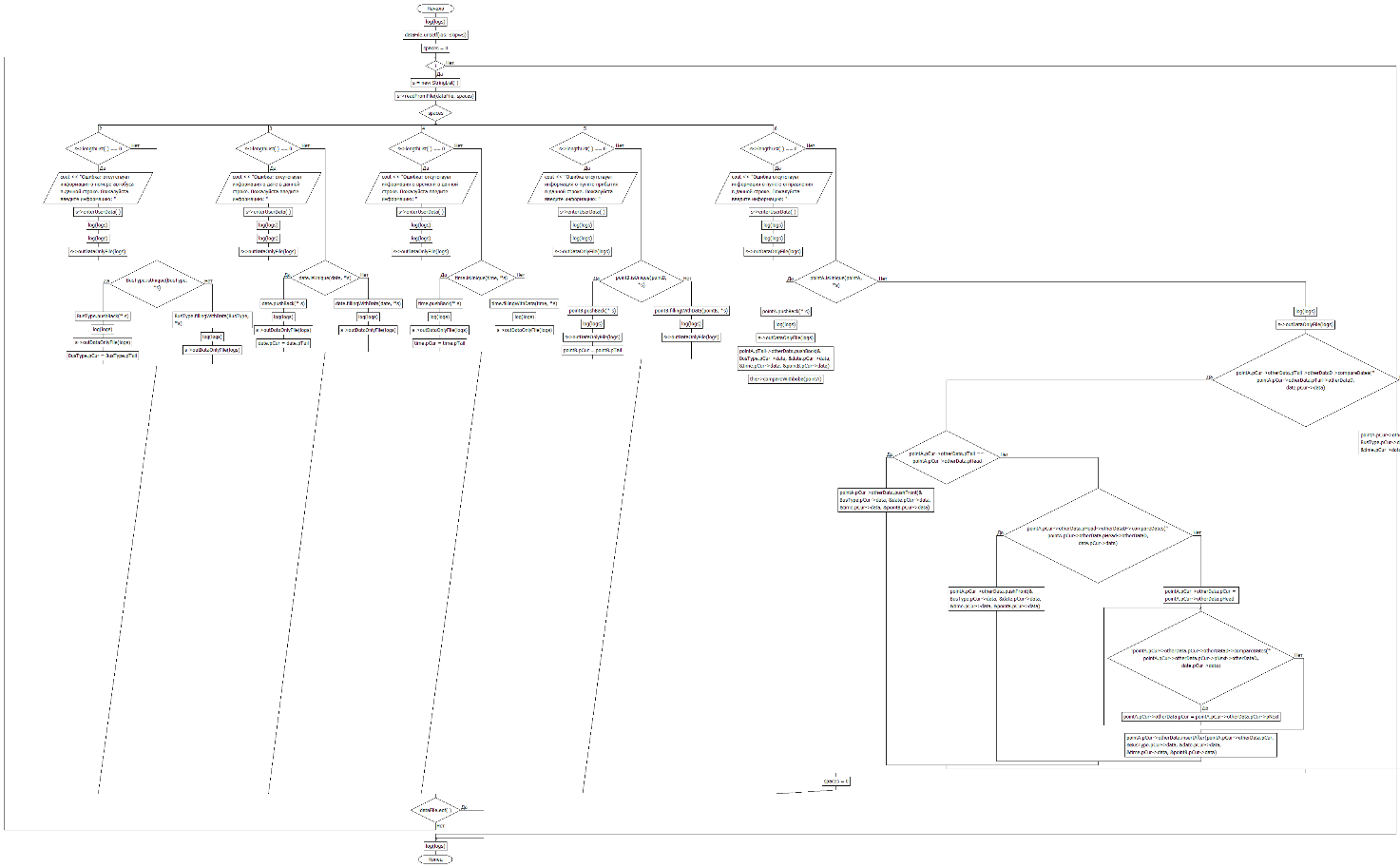
 

PopFront PushBack

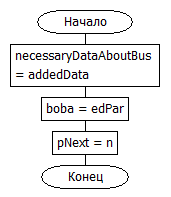
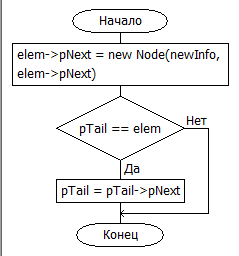
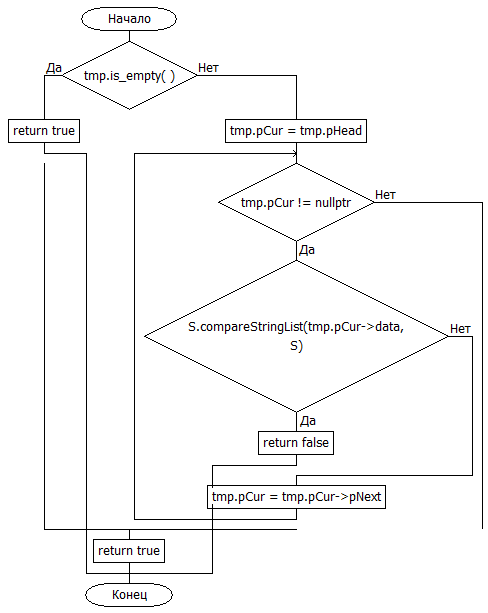
 

PushFront eraseAfter

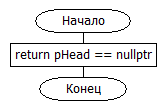
 

ReadingFromFile 

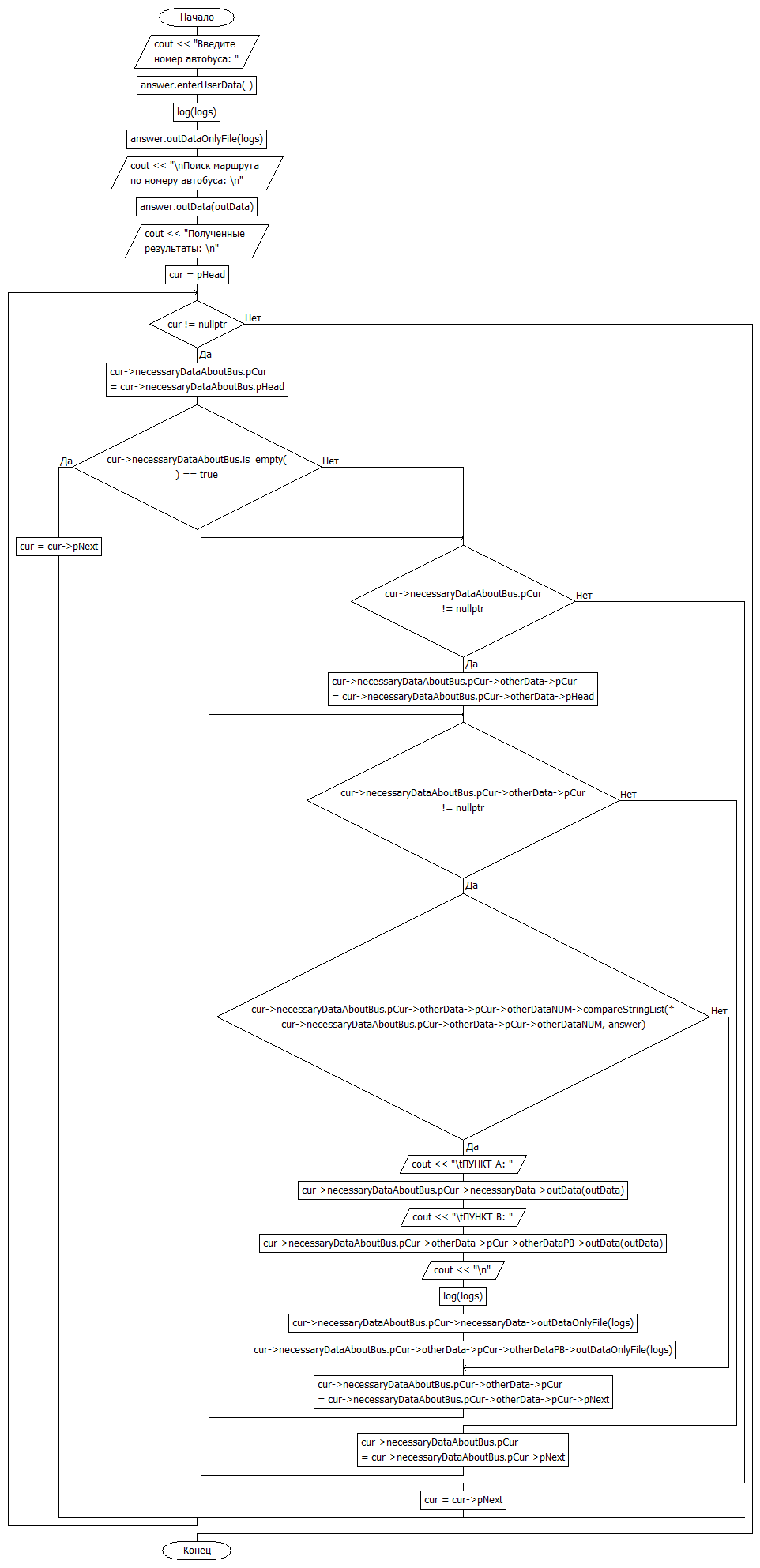
CardIndexNode InsertAfter IsUnique

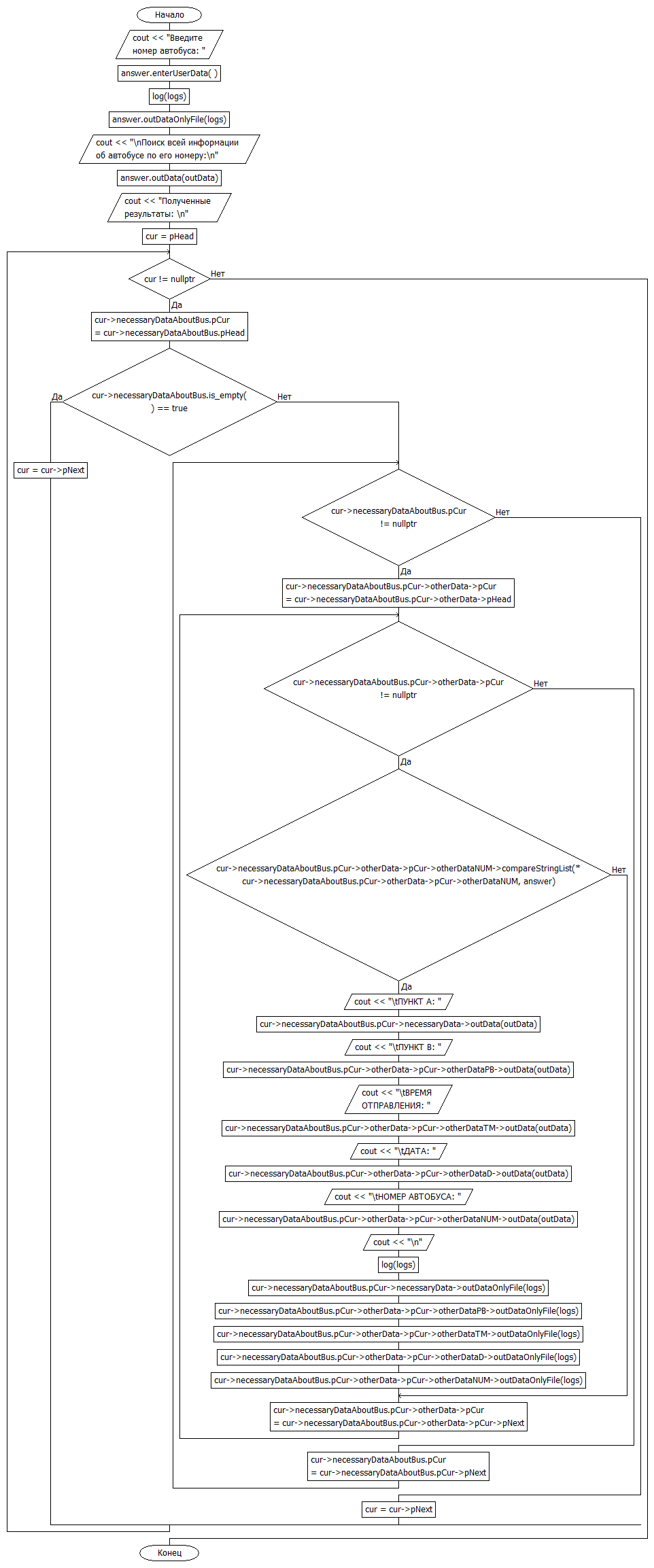
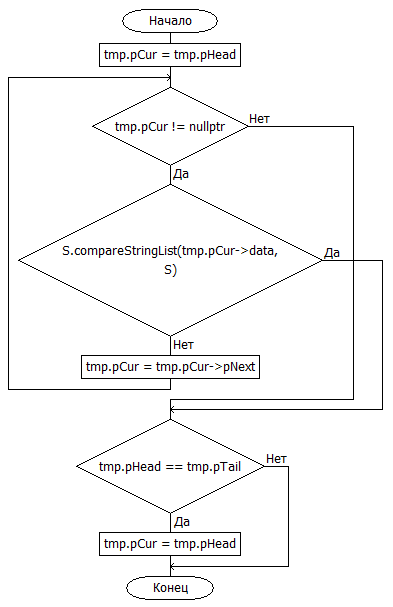
IsEmpty



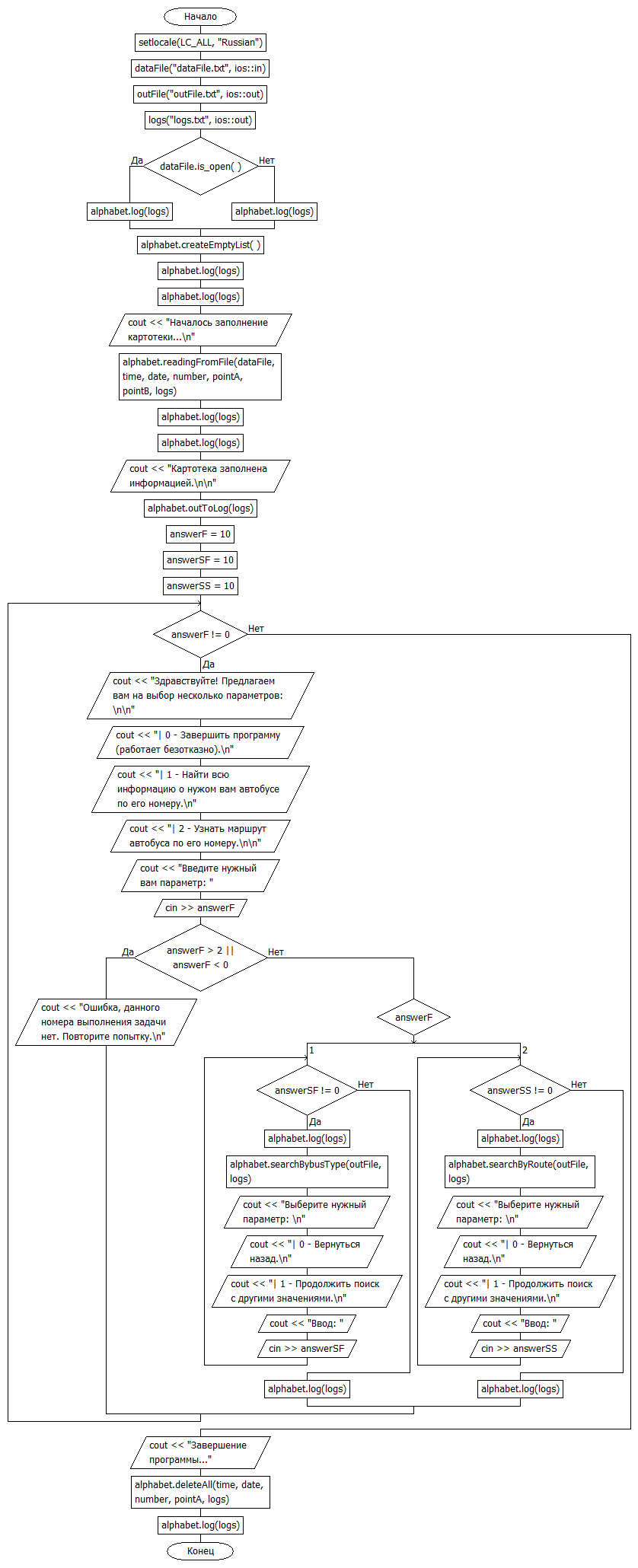
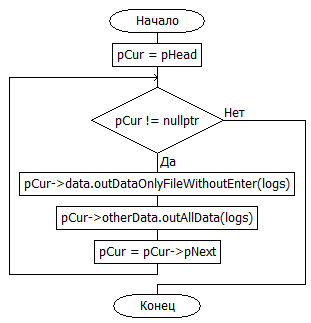
SearchByRoute



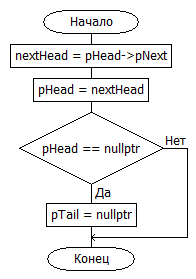
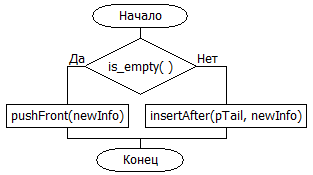
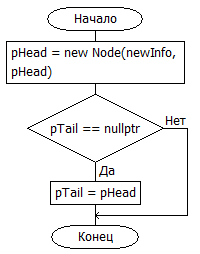
SearchByBusType fillingWithData

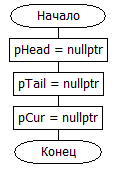
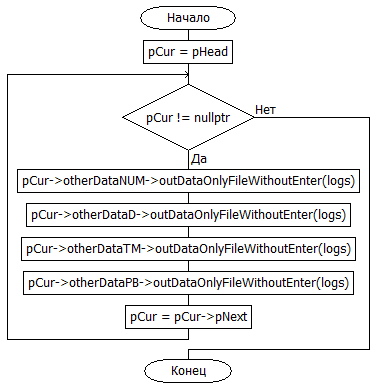
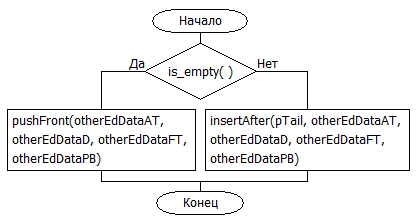
Int main(): OutAllDataList

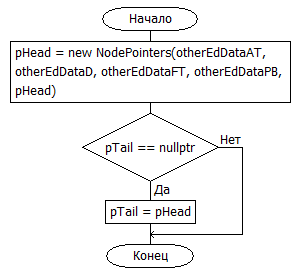
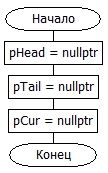
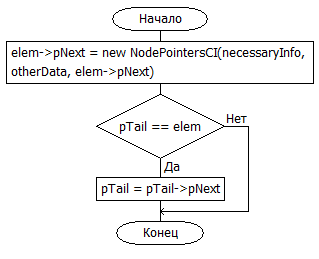
PopFront PushBack PushFront

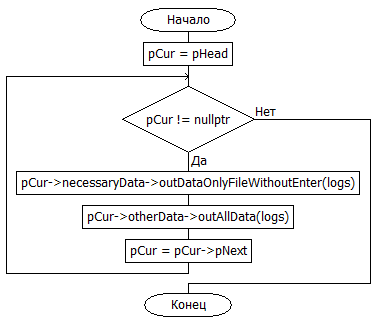
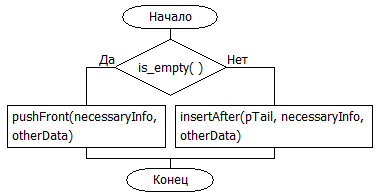
ListPointers OutAllData PushBack

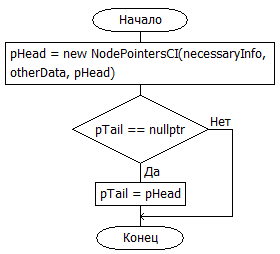
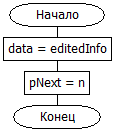
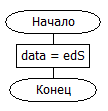
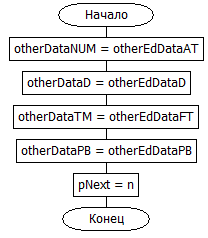
PushFront ListPointersCI InsertAfter

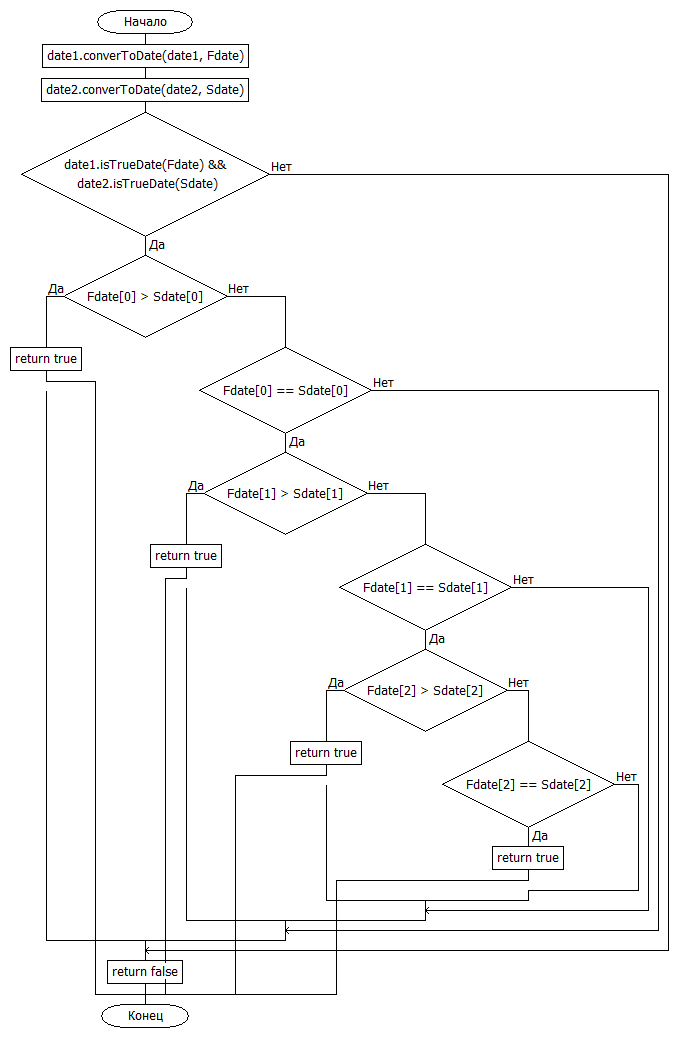
OutDataListCI PushBack

 ****

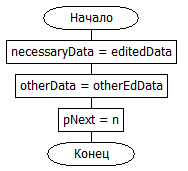
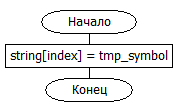
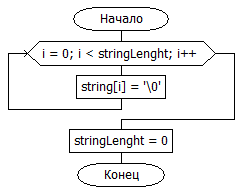
PushFront Node NodeInfo NodePointers

**** ****  

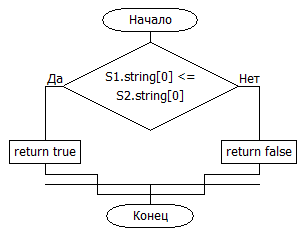
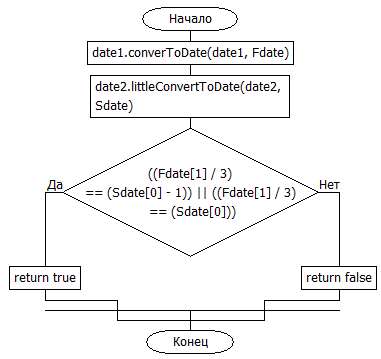
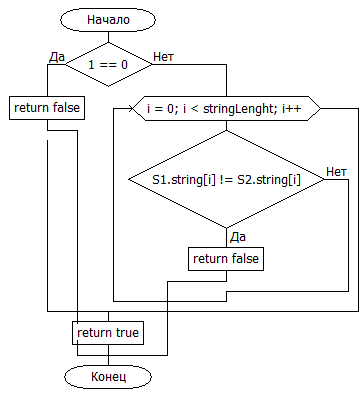
CompareDates

****

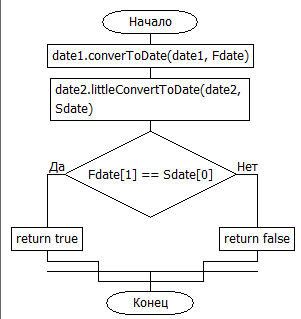
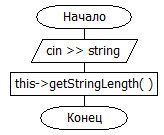
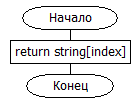
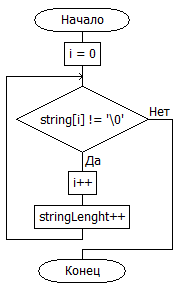
NodePointersCl addData ClearString

**** **** 

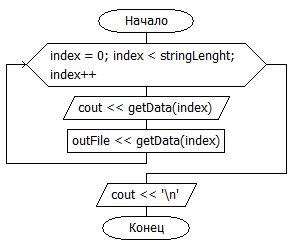
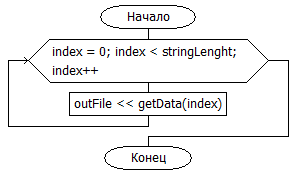
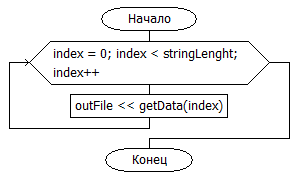
CompareFirst CompareQuarters CompareString

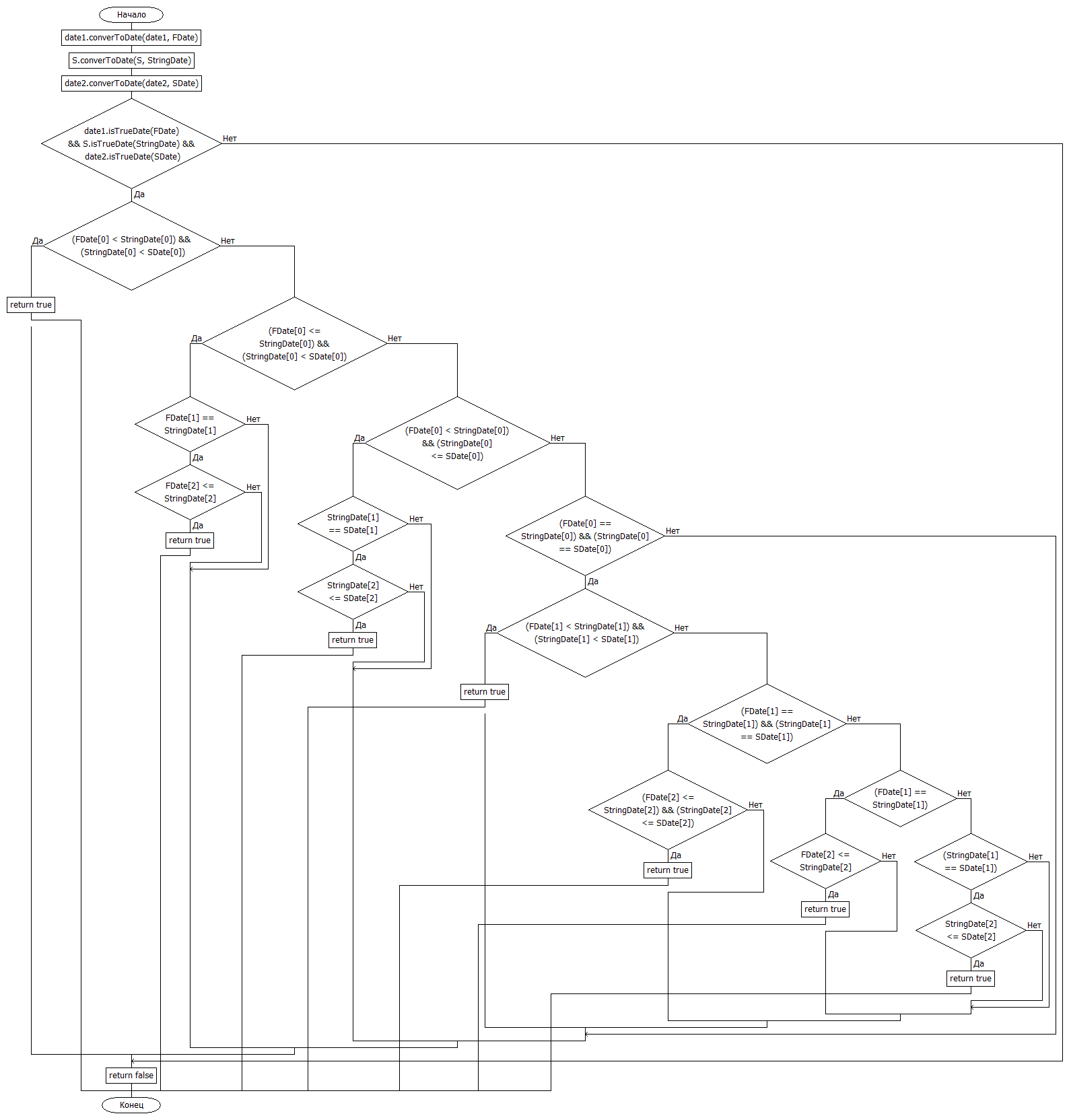
Comparebiba EnterUserDate GetData GetStringLength

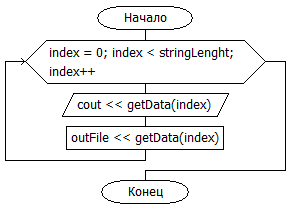
OutData OutDataOnlyFile OutDataOnlyFileWithoutEnter

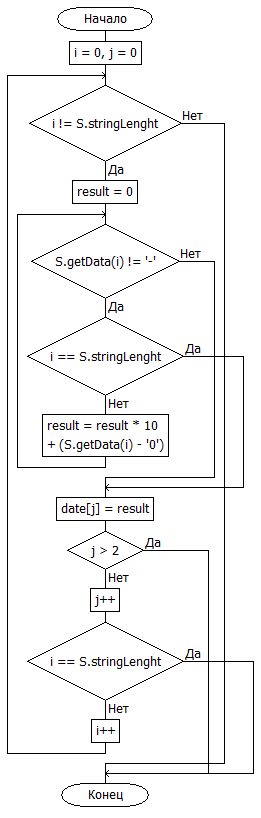
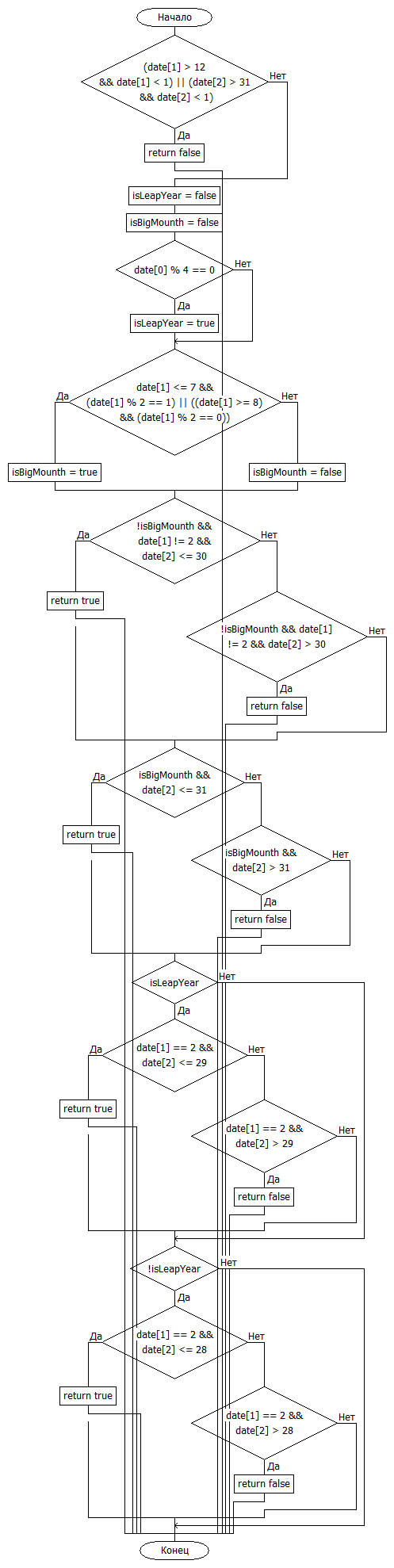
InTheRightPeriodOfTime



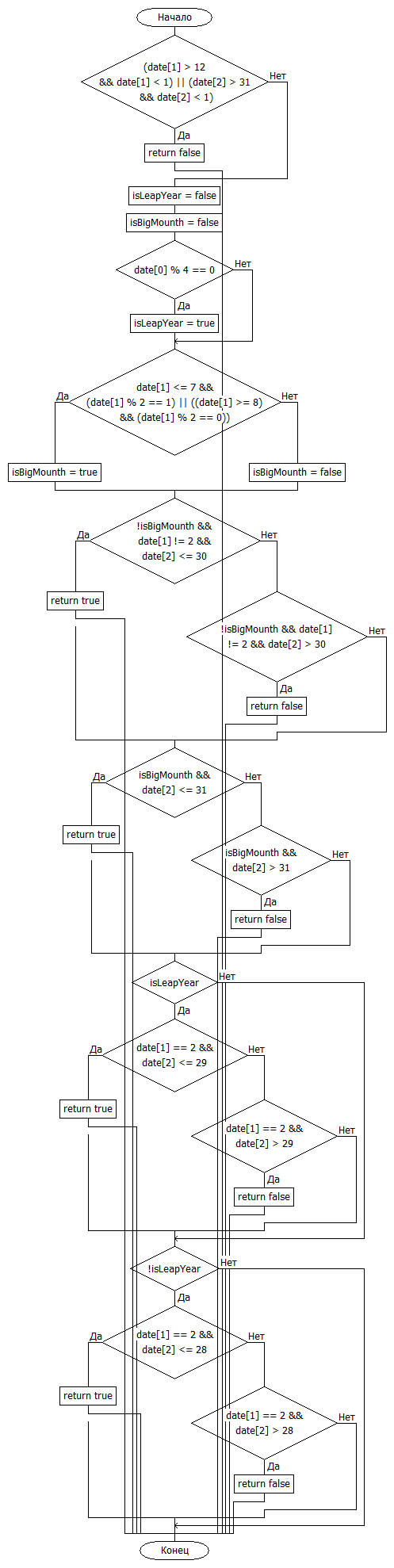
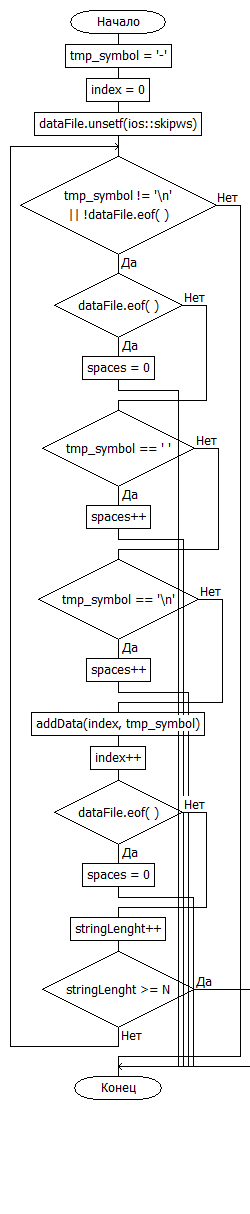
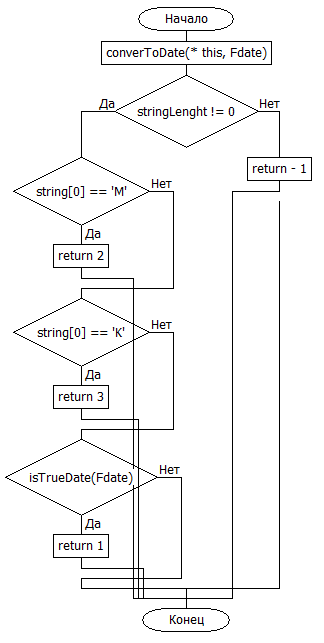
OutDataWithoutEnter



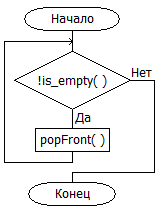
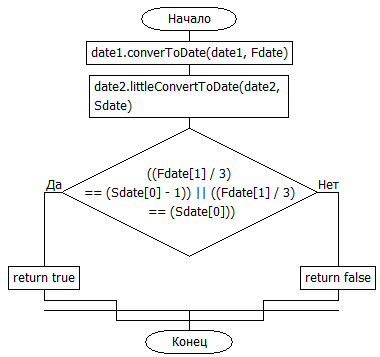
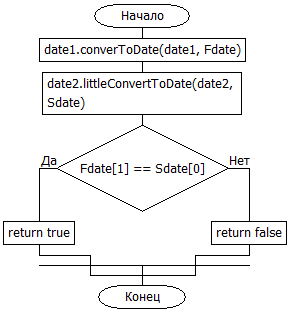
ConverToDate IsTrueDate

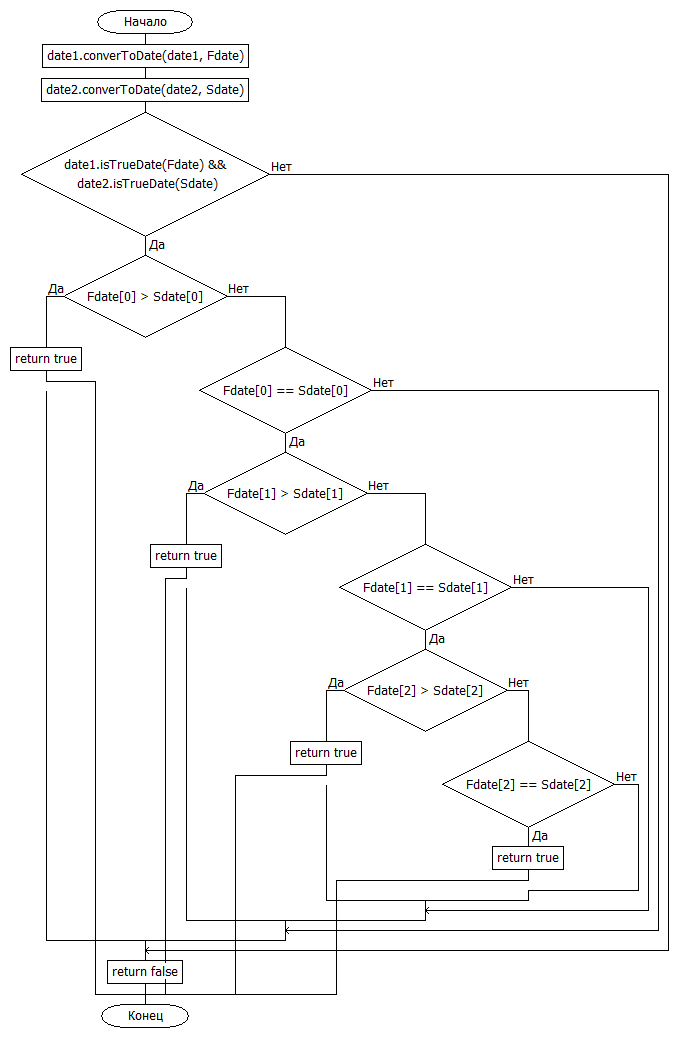
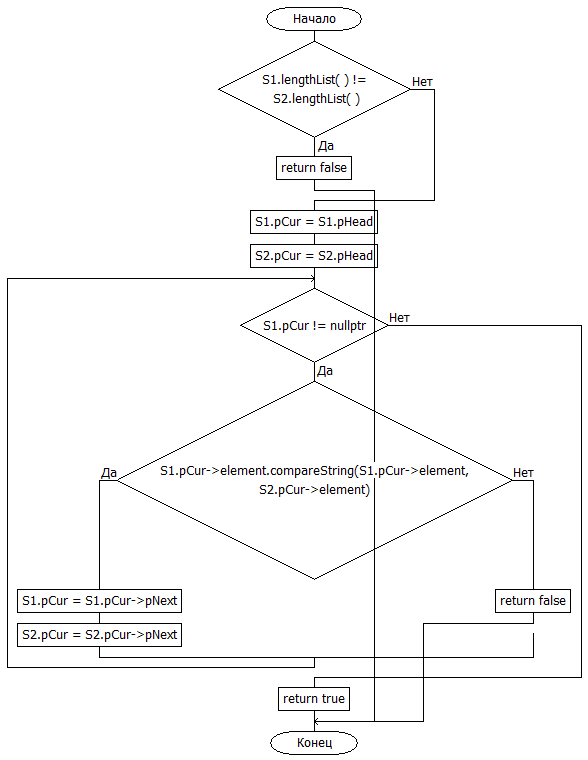
ReadString ReadStringV2 typeOfDate

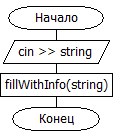
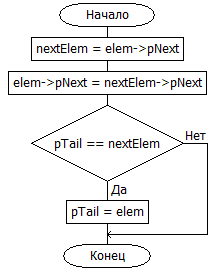
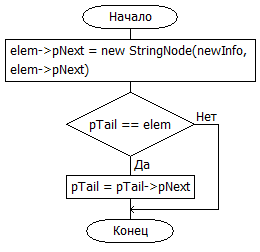
ClearStringList CompareQuarters Comparebiba

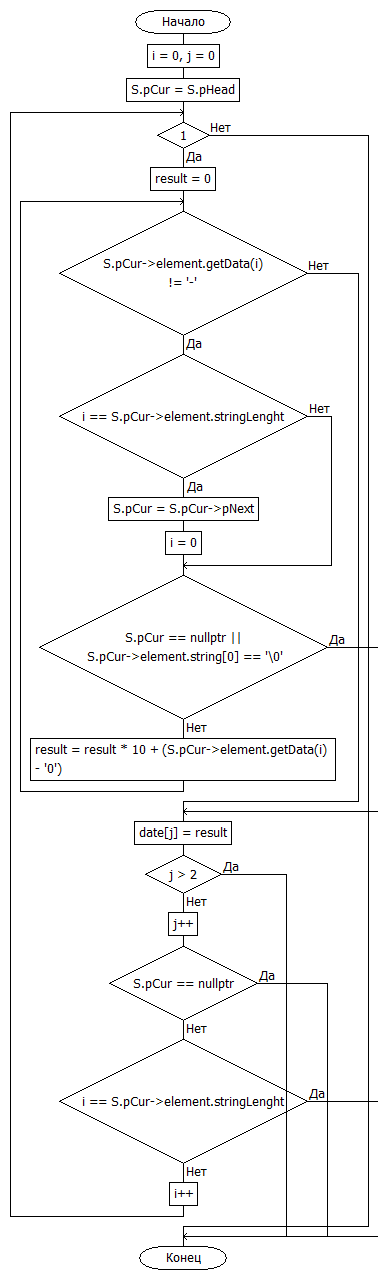
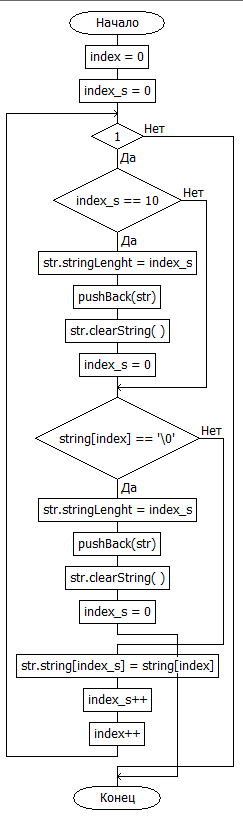
CompareDates CompareStringList

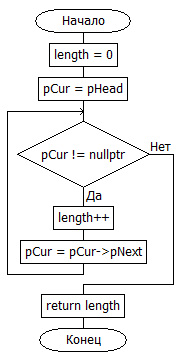
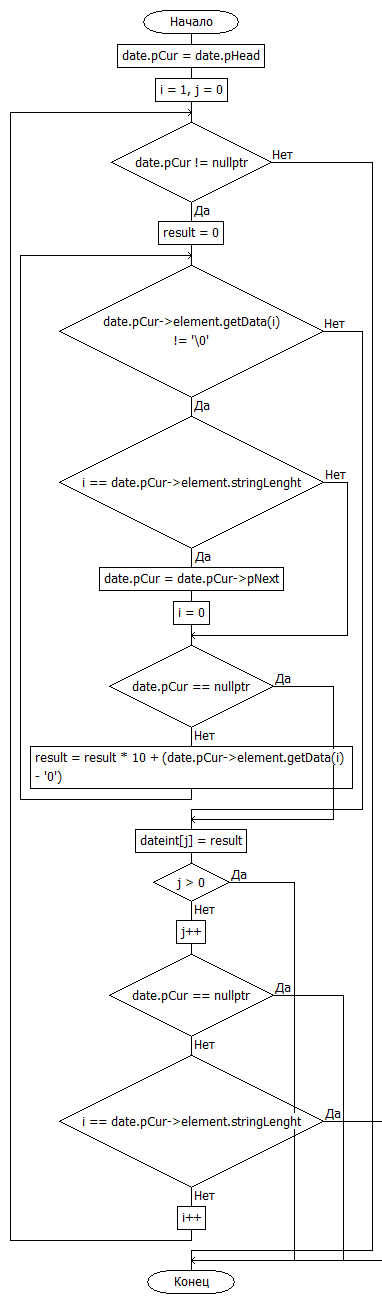
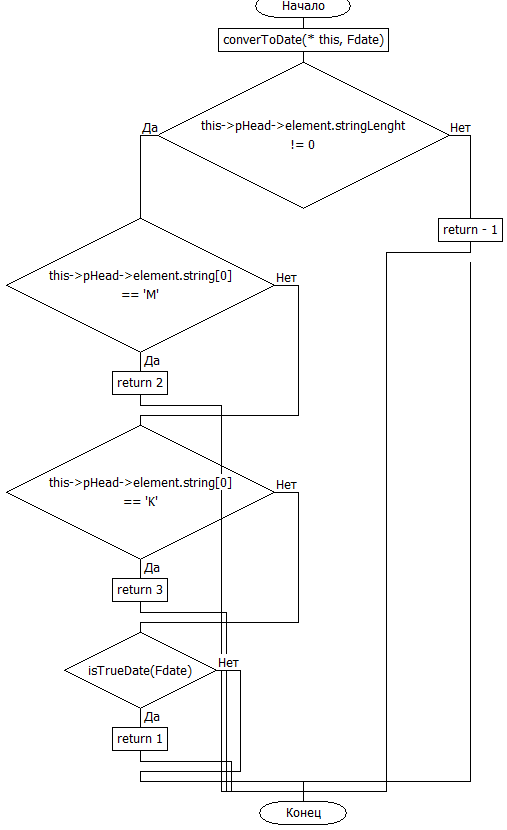
EnterUserData EraseAfter InsertAfter

ConverToDate FillWithInfo

LengthList LittleConvertToDate TypeOfType

10. Текст программы

CardIndex.cpp

#include "CardIndex.h"

#include <iostream>

#include <fstream>

#include <chrono>

using namespace std;

CardIndex::CardIndex()

{

pHead = nullptr;

pTail = nullptr;

pCur = nullptr;

}

void CardIndex::log(ofstream& log)

{

time\_t currentTime;

time(&currentTime);

tm tm\_time;

localtime\_s(&tm\_time, &currentTime);

char timeStr[20];

strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", &tm\_time);

log << timeStr << ' ';

}

bool CardIndex::is\_empty()

{

return pHead == nullptr;

}

void CardIndex::pushFront(ListPointersCI necessaryInfo, char edPar)

{

pHead = new CardIndexNode(necessaryInfo, pHead, edPar);

if (pTail == nullptr)

pTail = pHead;

}

void CardIndex::pushBack(ListPointersCI necessaryInfo, char edPar)

{

if (is\_empty())

{

pushFront(necessaryInfo, edPar);

}

else

insertAfter(pTail, necessaryInfo, edPar);

}

void CardIndex::popFront()

{

CardIndexNode\* nextHead = pHead->pNext;

delete pHead;

pHead = nextHead;

if (pHead == nullptr)

pTail = nullptr;

}

void CardIndex::insertAfter(CardIndexNode\* elem, ListPointersCI necessaryInfo, char edPar)

{

elem->pNext = new CardIndexNode(necessaryInfo, elem->pNext, edPar);

if (pTail == elem)

pTail = pTail->pNext;

}

void CardIndex::eraseAfter(CardIndexNode\* elem)

{

CardIndexNode\* nextElem = elem->pNext;

elem->pNext = nextElem->pNext;

delete nextElem;

if (pTail == nextElem)

pTail = elem;

}

void CardIndex::createEmptyList()

{

char boba = 'A';

while (boba <= 'Z')

{

pushBack(ListPointersCI(), boba);

boba++;

}

}

void CardIndex::compareWithboba(List pointA)

{

pCur = pHead;

while (pCur != nullptr)

{

if (pointA.pCur != nullptr)

{

if (pCur->boba == pointA.pCur->data.pHead->element.string[0])

{

pCur->necessaryDataAboutBus.pushBack(&pointA.pCur->data, &pointA.pCur->otherData);

}

}

else

{

if (pCur->boba == pointA.pTail->data.pHead->element.string[0])

{

pCur->necessaryDataAboutBus.pushBack(&pointA.pTail->data, &pointA.pTail->otherData);

}

}

pCur = pCur->pNext;

}

}

void CardIndex::readingFromFile(ifstream& dataFile, List& BusType, List& date, List& time, List& pointA, List& pointB, ofstream& logs)

{

log(logs); logs << "Началось считывание с файла.\n.";

dataFile.unsetf(ios::skipws);

int spaces = 0;

while (1)

{

StringList\* s = new StringList();

s->readFromFile(dataFile, spaces); //Если тут V2 значит меняем case

switch (spaces)

{

case 2:

if (s->lengthList() == 0)

{

cout << "Ошибка: отсутствует информация о номере автобуса в данной строке. Пожалуйста введите информацию: "; s->enterUserData();

log(logs); logs << "ОШИБКА: В строке не хватает информации.\n";

log(logs); logs << "Пользователь добавил элемент: "; s->outDataOnlyFile(logs);

}

if (BusType.isUnique(BusType, \*s))

{

BusType.pushBack(\*s);

log(logs); logs << "Добавлен элемент 1: "; s->outDataOnlyFile(logs);

BusType.pCur = BusType.pTail;

}

else

{

BusType.fillingWithData(BusType, \*s);

log(logs); logs << "Элемент 1 уже существует: "; s->outDataOnlyFile(logs);

}

//s->clearStringList();

break;

case 3:

if (s->lengthList() == 0)

{

cout << "Ошибка: отсутствует информация о дате в данной строке. Пожалуйста введите информацию: "; s->enterUserData();

log(logs); logs << "ОШИБКА: В строке не хватает информации\n.";

log(logs); logs << "Пользователь добавил элемент: "; s->outDataOnlyFile(logs);

}

if (date.isUnique(date, \*s))

{

date.pushBack(\*s);

log(logs); logs << "Добавлен элемент 2: "; s->outDataOnlyFile(logs);

date.pCur = date.pTail;

}

else

{

date.fillingWithData(date, \*s);

log(logs); logs << "Элемент 2 уже существует: "; s->outDataOnlyFile(logs);

}

//s->clearStringList();

break;

case 4:

if (s->lengthList() == 0)

{

cout << "Ошибка: отсутствует информация о времени в данной строке. Пожалуйста введите информацию: "; s->enterUserData();

log(logs); logs << "ОШИБКА: В строке не хватает информации\n.";

log(logs); logs << "Пользователь добавил элемент: "; s->outDataOnlyFile(logs);

}

if (time.isUnique(time, \*s))

{

time.pushBack(\*s);

log(logs); logs << "Добавлен элемент 3: "; s->outDataOnlyFile(logs);

time.pCur = time.pTail;

}

else

{

time.fillingWithData(time, \*s);

log(logs); logs << "Элемент 3 уже существует: "; s->outDataOnlyFile(logs);

}

//s->clearStringList();

break;

case 5:

if (s->lengthList() == 0)

{

cout << "Ошибка отсутствует информация о пункте прибытия в данной строке. Пожалуйста введите информацию: "; s->enterUserData();

log(logs); logs << "ОШИБКА: В строке не хватает информации\n.";

log(logs); logs << "Пользователь добавил элемент: "; s->outDataOnlyFile(logs);

}

if (pointB.isUnique(pointB, \*s))

{

pointB.pushBack(\*s);

log(logs); logs << "Добавлен элемент 4: "; s->outDataOnlyFile(logs);

pointB.pCur = pointB.pTail;

}

else

{

pointB.fillingWithData(pointB, \*s);

log(logs); logs << "Элемент 4 уже существует: "; s->outDataOnlyFile(logs);

}

//s->clearStringList();

break;

case 6:

if (s->lengthList() == 0)

{

cout << "Ошибка отсутствует информация о пункте отправления в данной строке. Пожалуйста введите информацию: "; s->enterUserData();

log(logs); logs << "ОШИБКА: В строке не хватает информации.\n";

log(logs); logs << "Пользователь добавил элемент: "; s->outDataOnlyFile(logs);

}

if (pointA.isUnique(pointA, \*s))

{

pointA.pushBack(\*s);

log(logs); logs << "Добавлен элемент 5: "; s->outDataOnlyFile(logs);

pointA.pTail->otherData.pushBack(&BusType.pCur->data, &date.pCur->data, &time.pCur->data, &pointB.pCur->data);

this->compareWithboba(pointA);

}

else

{

log(logs); logs << "Элемент 5 уже существует: "; s->outDataOnlyFile(logs);

//Что-то по сортировке даты //УАААААА ОШИБКА ОШИБКА ОШИБКА

if (pointA.pCur->otherData.pTail->otherDataD->compareDates(\*pointA.pCur->otherData.pTail->otherDataD, date.pCur->data))

{

if (pointA.pCur->otherData.pTail == pointA.pCur->otherData.pHead)

pointA.pCur->otherData.pushFront(&BusType.pCur->data, &date.pCur->data, &time.pCur->data, &pointB.pCur->data);

else if (pointA.pCur->otherData.pHead->otherDataD->compareDates(\*pointA.pCur->otherData.pHead->otherDataD, date.pCur->data))

{

pointA.pCur->otherData.pushFront(&BusType.pCur->data, &date.pCur->data, &time.pCur->data, &pointB.pCur->data);

}

else

{

pointA.pCur->otherData.pCur = pointA.pCur->otherData.pHead;

while (!pointA.pCur->otherData.pCur->otherDataD->compareDates(\*pointA.pCur->otherData.pCur->pNext->otherDataD, date.pCur->data))

{

pointA.pCur->otherData.pCur = pointA.pCur->otherData.pCur->pNext;

}

pointA.pCur->otherData.insertAfter(pointA.pCur->otherData.pCur, &BusType.pCur->data, &date.pCur->data, &time.pCur->data, &pointB.pCur->data);

}

}

//Конец

else pointA.pCur->otherData.pushBack(&BusType.pCur->data, &date.pCur->data, &time.pCur->data, &pointB.pCur->data);

}

//s->clearStringList();

spaces = 0;

break;

}

if (dataFile.eof()) break;

}

log(logs); logs << "Считывание и заполнение данными закончено.\n";

}

void CardIndex::searchBybusType(ofstream& outData, ofstream& logs)

{

StringList answer;

cout << "Введите номер автобуса: ";

answer.enterUserData();

log(logs); logs << "Пользователь ввёл значение: "; answer.outDataOnlyFile(logs);

cout << "\nПоиск всей информации об автобусе по его номеру:\n"; outData << "Поиск всей информации об автобусе по его номеру:\n"; answer.outData(outData); cout << "Полученные результаты: \n"; outData << "Полученные результаты: \n";

CardIndexNode\* cur = pHead;

while (cur != nullptr)

{

cur->necessaryDataAboutBus.pCur = cur->necessaryDataAboutBus.pHead;

if (cur->necessaryDataAboutBus.is\_empty() == true) cur = cur->pNext;

else

{

while (cur->necessaryDataAboutBus.pCur != nullptr)

{

cur->necessaryDataAboutBus.pCur->otherData->pCur = cur->necessaryDataAboutBus.pCur->otherData->pHead;

while (cur->necessaryDataAboutBus.pCur->otherData->pCur != nullptr)

{

if (cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataNUM->compareStringList(\*cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataNUM, answer))

{

cout << "\tПУНКТ А: "; outData << "\tПУНКТ А: "; cur->necessaryDataAboutBus.pCur->necessaryData->outData(outData);

cout << "\tПУНКТ B: "; outData << "\tПУНКТ B: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataPB->outData(outData);

cout << "\tВРЕМЯ ОТПРАВЛЕНИЯ: "; outData << "\tВРЕМЯ ОТПРАВЛЕНИЯ: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataTM->outData(outData);

cout << "\tДАТА: "; outData << "\tДАТА: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataD->outData(outData);

cout << "\tНОМЕР АВТОБУСА: "; outData << "\tНОМЕР АВТОБУСА: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataNUM->outData(outData);

//cout << "\tPOINT B: "; outData << "\tPOINT B: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataPB->outData(outData);

cout << "\n"; outData << "\n";

log(logs); logs << "Были найдены данные значения:\n";

logs << "\t\t\t\tПУНКТ А: "; cur->necessaryDataAboutBus.pCur->necessaryData->outDataOnlyFile(logs);

logs << "\t\t\t\tПУНКТ B: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataPB->outDataOnlyFile(logs);

logs << "\t\t\t\tВРЕМЯ ОТПРАВЛЕНИЯ: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataTM->outDataOnlyFile(logs);

logs << "\t\t\t\tДАТА: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataD->outDataOnlyFile(logs);

logs << "\t\t\t\tНОМЕР АВТОБУСА: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataNUM->outDataOnlyFile(logs);

//logs << "\t\t\t\tPOINT B: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataPB->outDataOnlyFile(logs);

}

cur->necessaryDataAboutBus.pCur->otherData->pCur = cur->necessaryDataAboutBus.pCur->otherData->pCur->pNext;

}

cur->necessaryDataAboutBus.pCur = cur->necessaryDataAboutBus.pCur->pNext;

}

cur = cur->pNext;

}

}

}

void CardIndex::searchByRoute(ofstream& outData, ofstream& logs)

{

StringList answer;

cout << "Введите номер автобуса: ";

answer.enterUserData();

log(logs); logs << "Пользователь ввёл значение: "; answer.outDataOnlyFile(logs);

cout << "\nПоиск маршрута по номеру автобуса: \n"; outData << "Поиск маршрута по номеру автобуса: \n"; answer.outData(outData); cout << "Полученные результаты: \n"; outData << "Полученные результаты: \n";

CardIndexNode\* cur = pHead;

while (cur != nullptr)

{

cur->necessaryDataAboutBus.pCur = cur->necessaryDataAboutBus.pHead;

if (cur->necessaryDataAboutBus.is\_empty() == true) cur = cur->pNext;

else

{

while (cur->necessaryDataAboutBus.pCur != nullptr)

{

cur->necessaryDataAboutBus.pCur->otherData->pCur = cur->necessaryDataAboutBus.pCur->otherData->pHead;

while (cur->necessaryDataAboutBus.pCur->otherData->pCur != nullptr)

{

if (cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataNUM->compareStringList(\*cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataNUM, answer))

{

cout << "\tПУНКТ A: "; outData << "\tПУНКТ А: "; cur->necessaryDataAboutBus.pCur->necessaryData->outData(outData);

/\*cout << "\tNUMBER: "; outData << "\tNUMBER: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataTM->outData(outData);

cout << "\tDATE: "; outData << "\tDATE: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataD->outData(outData);

cout << "\ttime: "; outData << "\ttime: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataNUM->outData(outData);\*/

cout << "\tПУНКТ B: "; outData << "\tPOINT B: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataPB->outData(outData);

cout << "\n"; outData << "\n";

log(logs); logs << "Были найдены данные значения:\n";

logs << "\t\t\t\tПУНКТ А: "; cur->necessaryDataAboutBus.pCur->necessaryData->outDataOnlyFile(logs);

logs << "\t\t\t\tПУНКТ B: "; cur->necessaryDataAboutBus.pCur->otherData->pCur->otherDataPB->outDataOnlyFile(logs);

//logs << "\t\t\t\tВРЕМЯ ОТПРАВЛЕНИЯ: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataTM->outDataOnlyFile(logs);

//logs << "\t\t\t\tДАТА: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataD->outDataOnlyFile(logs);

//logs << "\t\t\t\tНОМЕР АВТОБУСА: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataNUM->outDataOnlyFile(logs);

//logs << "\t\t\t\tPOINT B: "; cur->necessaryDataAboutFlight.pCur->otherData->pCur->otherDataPB->outDataOnlyFile(logs);

}

cur->necessaryDataAboutBus.pCur->otherData->pCur = cur->necessaryDataAboutBus.pCur->otherData->pCur->pNext;

}

cur->necessaryDataAboutBus.pCur = cur->necessaryDataAboutBus.pCur->pNext;

}

cur = cur->pNext;

}

}

}

void CardIndex::deleteAll(List& busType, List& date, List& time, List& pointA, ofstream& logs)

{

log(logs); logs << "Начало очистки всех данных.\n";

while (!is\_empty())

{

while (!pHead->necessaryDataAboutBus.is\_empty()) pHead->necessaryDataAboutBus.popFront();

popFront();

}

while (!busType.is\_empty()) busType.popFront();

busType.pCur = nullptr;

while (!date.is\_empty()) date.popFront();

date.pCur = nullptr;

while (!time.is\_empty()) time.popFront();

time.pCur = nullptr;

while (!pointA.is\_empty())

{

while (!pointA.pHead->otherData.is\_empty()) pointA.pHead->otherData.popFront();

pointA.popFront();

}

pointA.pCur = nullptr;

log(logs); logs << "Очистка была произведена.\n";

}

void CardIndex::outToLog(ofstream& logs)

{

pCur = pHead;

log(logs); logs << "Начало вывода всей информации.\n";

while (pCur != nullptr)

{

logs << "Параметр: " << pCur->boba << "\n";

pCur->necessaryDataAboutBus.outDataListCI(logs);

logs << "|\n";

logs << "V\n";

pCur = pCur->pNext;

}

logs << "NULL\n";

log(logs); logs << "Вывод всей информации завершен.\n\n";

}

CardIndexNode.cpp

#include "CardIndexNode.h"

CardIndexNode::CardIndexNode(ListPointersCI addedData, CardIndexNode\* n, char edPar)

{

necessaryDataAboutBus = addedData;

boba = edPar;

pNext = n;

}

Lazurni\_avtobus.cpp

#include <iostream>

#include <fstream>

#include "List.h"

#include "StringList.h"

#include "locale"

#include "CardIndex.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

List pointA, number, time, date, pointB;

CardIndex alphabet;

ifstream dataFile("dataFile.txt", ios::in);

ofstream outFile("outFile.txt", ios::out);

ofstream logs("logs.txt", ios::out);

//ifstream A("A.txt", ios::in);

//ifstream B("B.txt", ios::in);

//int dates2[3];

//StringList ab, bob;

//ab.enterUserData();

//ab.converToDate(ab, dates2);

if (dataFile.is\_open())

{

alphabet.log(logs); logs << "Файл успешно открыт.\n";

}

else

{

alphabet.log(logs); logs << "ОШИБКА: Файл не открыт или отсутствует.\n";

}

alphabet.createEmptyList();

alphabet.log(logs); logs << "ШАГ 1: Создана пустая картотека.\n";

alphabet.log(logs); logs << "ШАГ 2: Началось заполнение картотеки...\n"; cout << "Началось заполнение картотеки...\n";

alphabet.readingFromFile(dataFile, time, date, number, pointA, pointB, logs);

alphabet.log(logs); logs << "ШАГ 3: Все списки заполнены информацией.\n";

alphabet.log(logs); logs << "ШАГ 4: Картотека заполнена информацией.\n\n"; cout << "Картотека заполнена информацией.\n\n";

alphabet.outToLog(logs);

//Проработка сценария:

int answerF = 10;

int answerSF = 10;

int answerSS = 10;

while (answerF != 0)

{

cout << "Здравствуйте! Предлагаем вам на выбор несколько параметров: \n\n";

cout << "| 0 - Завершить программу (работает безотказно).\n";

cout << "| 1 - Найти всю информацию о нужом вам автобусе по его номеру.\n";

cout << "| 2 - Узнать маршрут автобуса по его номеру.\n\n";

cout << "Введите нужный вам параметр: "; cin >> answerF;

if (answerF > 2 || answerF < 0) cout << "Ошибка, данного номера выполнения задачи нет. Повторите попытку.\n";

else

{

switch (answerF)

{

case 1:

while (answerSF != 0)

{

alphabet.log(logs); logs << "Пользователь выбрал поиск всей информации об автобусе по его номеру.\n";

alphabet.searchBybusType(outFile, logs);

cout << "Выберите нужный параметр: \n";

cout << "| 0 - Вернуться назад.\n";

cout << "| 1 - Продолжить поиск с другими значениями.\n";

cout << "Ввод: "; cin >> answerSF;

}

alphabet.log(logs); logs << "Поиск всей информации об автобусе по его номеру был завершен.\n";

break;

case 2:

while (answerSS != 0)

{

alphabet.log(logs); logs << "Пользователь выбрал поиск маршрута автобуса по его номеру.\n";

alphabet.searchByRoute(outFile, logs);

cout << "Выберите нужный параметр: \n";

cout << "| 0 - Вернуться назад.\n";

cout << "| 1 - Продолжить поиск с другими значениями.\n";

cout << "Ввод: "; cin >> answerSS;

}

alphabet.log(logs); logs << "Поиск маршрута автобуса по его номеру был завершен.\n";

break;

}

}

}

cout << "Завершение программы...";

alphabet.deleteAll(time, date, number, pointA, logs);

alphabet.log(logs); logs << "Завершение программы.\n";

}

List.cpp

#include "List.h"

#include <iostream>

#include <fstream>

using namespace std;

List::List()

{

pHead = nullptr;

pTail = nullptr;

pCur = nullptr;

}

bool List::is\_empty()

{

return pHead == nullptr;

}

void List::pushFront(StringList newInfo)

{

pHead = new Node(newInfo, pHead);

if (pTail == nullptr)

pTail = pHead;

}

void List::pushBack(StringList newInfo)

{

if (is\_empty())

{

pushFront(newInfo);

}

else

insertAfter(pTail, newInfo);

}

void List::popFront()

{

Node\* nextHead = pHead->pNext;

delete pHead;

pHead = nextHead;

if (pHead == nullptr)

pTail = nullptr;

}

void List::insertAfter(Node\* elem, StringList newInfo)

{

elem->pNext = new Node(newInfo, elem->pNext);

if (pTail == elem)

pTail = pTail->pNext;

}

void List::eraseAfter(Node\* elem)

{

Node\* nextElem = elem->pNext;

elem->pNext = nextElem->pNext;

delete nextElem;

if (pTail == nextElem)

pTail = elem;

}

/\*void List::readFromFile(ifstream& dataFile)

{

dataFile.unsetf(ios::skipws);

int spaces = 0;

while (1)

{

String S;

S.readString(dataFile, spaces);

pushBack(S);

if (dataFile.eof()) break;

}

}\*/

bool List::isUnique(List& tmp, StringList S)

{

if (tmp.is\_empty()) return true;

else

{

tmp.pCur = tmp.pHead;

while (tmp.pCur != nullptr)

{

if (S.compareStringList(tmp.pCur->data, S))

{

//tmp.cur = tmp.cur->pNext;

return false;

}

tmp.pCur = tmp.pCur->pNext;

}

}

return true;

}

void List::fillingWithData(List& tmp, StringList S)

{

tmp.pCur = tmp.pHead;

while (tmp.pCur != nullptr)

{

if (S.compareStringList(tmp.pCur->data, S)) break;

tmp.pCur = tmp.pCur->pNext;

}

if (tmp.pHead == tmp.pTail) tmp.pCur = tmp.pHead;

}

void List::outAllDataList(ofstream& logs)

{

pCur = pHead;

while (pCur != nullptr)

{

pCur->data.outDataOnlyFileWithoutEnter(logs);

logs << "-->";

pCur->otherData.outAllData(logs);

logs << "|\n"; logs << "\V\n";

pCur = pCur->pNext;

}

logs << "NULL";

}

ListPointers.cpp

#include "ListPointers.h"

#include <iostream>

#include <fstream>

using namespace std;

// Конструктор класса ListPointers

ListPointers::ListPointers()

{

// Инициализируем указатели на голову, хвост и текущий элемент списка как nullptr

pHead = nullptr;

pTail = nullptr;

pCur = nullptr;

}

// Метод проверки, пуст ли список

bool ListPointers::is\_empty()

{

// Возвращаем true, если pHead равен nullptr, что означает, что список пуст

return pHead == nullptr;

}

// Метод добавления элемента в начало списка

void ListPointers::pushFront(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB)

{

// Создаем новый узел и устанавливаем его в качестве головы списка

pHead = new NodePointers(otherEdDataAT, otherEdDataD, otherEdDataFT, otherEdDataPB, pHead);

// Если хвост был пуст, устанавливаем хвост на голову (единственный элемент в списке)

if (pTail == nullptr)

pTail = pHead;

}

// Метод добавления элемента в конец списка

void ListPointers::pushBack(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB)

{

// Если список пуст, используем метод pushFront для добавления элемента

if (is\_empty())

{

pushFront(otherEdDataAT, otherEdDataD, otherEdDataFT, otherEdDataPB);

}

else

// Иначе, добавляем элемент после текущего хвоста

{

insertAfter(pTail, otherEdDataAT, otherEdDataD, otherEdDataFT, otherEdDataPB);

}

}

// Метод удаления элемента из начала списка

void ListPointers::popFront()

{

// Сохраняем указатель на следующий элемент после головы

NodePointers\* nextHead = pHead->pNext;

// Удаляем текущую голову

delete pHead;

// Обновляем голову на следующий элемент

pHead = nextHead;

// Если новый голова пуст, также очищаем хвост

if (pHead == nullptr)

pTail = nullptr;

}

// Метод вставки элемента после заданного узла

void ListPointers::insertAfter(NodePointers\* elem, StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB)

{

// Создаем новый узел и вставляем его после указанного элемента

elem->pNext = new NodePointers(otherEdDataAT, otherEdDataD, otherEdDataFT, otherEdDataPB, elem->pNext);

// Если вставленный элемент стал последним (хвостом), обновляем указатель на хвост

if (pTail == elem)

pTail = pTail->pNext;

}

// Метод удаления элемента после заданного узла

void ListPointers::eraseAfter(NodePointers\* elem)

{

// Сохраняем указатель на элемент, который будет удален

NodePointers\* nextElem = elem->pNext;

// Обновляем указатель на следующий элемент после удаляемого

elem->pNext = nextElem->pNext;

// Удаляем узел

delete nextElem;

// Если удаленный элемент был хвостом, обновляем указатель на хвост

if (pTail == nextElem)

pTail = elem;

}

// Метод вывода всех данных списка в файл

void ListPointers::outAllData(ofstream& logs)

{

// Устанавливаем текущий элемент на голову списка

pCur = pHead;

// Печатаем начало строки в файл

logs << "|| ";

// Проходим по всем элементам списка

while (pCur != nullptr)

{

// Выводим данные из каждого узла в файл без перехода на новую строку

pCur->otherDataNUM->outDataOnlyFileWithoutEnter(logs); logs << "->";

pCur->otherDataD->outDataOnlyFileWithoutEnter(logs); logs << "->";

pCur->otherDataTM->outDataOnlyFileWithoutEnter(logs); logs << "->";

pCur->otherDataPB->outDataOnlyFileWithoutEnter(logs); logs << "->";

// Переходим к следующему элементу

pCur = pCur->pNext;

}

// Выводим конец строки в файл

logs << "NULL";

logs << " ||\n";

}

ListPointersCI.cpp

#include "ListPointersCI.h"

#include <iostream>

#include <fstream>

using namespace std;

ListPointersCI::ListPointersCI()

{

pHead = nullptr;

pTail = nullptr;

pCur = nullptr;

}

bool ListPointersCI::is\_empty()

{

return pHead == nullptr;

}

void ListPointersCI::pushFront(StringList\* necessaryInfo, ListPointers\* otherData)

{

pHead = new NodePointersCI(necessaryInfo, otherData, pHead);

if (pTail == nullptr)

pTail = pHead;

}

void ListPointersCI::pushBack(StringList\* necessaryInfo, ListPointers\* otherData)

{

if (is\_empty())

{

pushFront(necessaryInfo, otherData);

}

else

insertAfter(pTail, necessaryInfo, otherData);

}

void ListPointersCI::popFront()

{

NodePointersCI\* nextHead = pHead->pNext;

delete pHead;

pHead = nextHead;

if (pHead == nullptr)

pTail = nullptr;

}

void ListPointersCI::insertAfter(NodePointersCI\* elem, StringList\* necessaryInfo, ListPointers\* otherData)

{

elem->pNext = new NodePointersCI(necessaryInfo, otherData, elem->pNext);

if (pTail == elem)

pTail = pTail->pNext;

}

void ListPointersCI::eraseAfter(NodePointersCI\* elem)

{

NodePointersCI\* nextElem = elem->pNext;

elem->pNext = nextElem->pNext;

delete nextElem;

if (pTail == nextElem)

pTail = elem;

}

void ListPointersCI::outDataListCI(ofstream& logs)

{

pCur = pHead;

logs << "--------------------------------------\n";

while (pCur != nullptr)

{

pCur->necessaryData->outDataOnlyFileWithoutEnter(logs);

logs << "-->";

pCur->otherData->outAllData(logs);

logs << "|\n"; logs << "\V\n";

pCur = pCur->pNext;

}

logs << "NULL\n";

logs << "--------------------------------------\n";

}

Node.cpp

#include "Node.h"

using namespace std;

Node::Node(StringList editedInfo, Node\* n)

{

data = editedInfo;

pNext = n;

}

NodeInfo.cpp

#include "NodeInfo.h"

using namespace std;

NodeInfo::NodeInfo(String edS)

{

data = edS;

}

NodePointers.cpp

#include "NodePointers.h"

NodePointers::NodePointers(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB, NodePointers\* n)

{

otherDataNUM = otherEdDataAT;

otherDataD = otherEdDataD;

otherDataTM = otherEdDataFT;

otherDataPB = otherEdDataPB;

pNext = n;

}

NodePointersCI.cpp

#include "NodePointersCI.h"

NodePointersCI::NodePointersCI(StringList\* editedData, ListPointers\* otherEdData, NodePointersCI\* n)

{

necessaryData = editedData;

otherData = otherEdData;

pNext = n;

}

String.cpp

#include "String.h"

#include <fstream>

#include <iostream>

#include <chrono>

using namespace std;

void String::addData(int index, char tmp\_symbol)

{

string[index] = tmp\_symbol;

}

void String::clearString()

{

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

{

string[i] = '\0';

}

stringLenght = 0;

}

void String::getStringLength()

{

int i = 0;

while (string[i] != '\0')

{

i++;

stringLenght++;

}

}

char String::getData(int index)

{

return string[index];

}

void String::readString(ifstream& dataFile, int& spaces)

{

char tmp\_symbol = '-';

int index = 0;

dataFile.unsetf(ios::skipws);

while (tmp\_symbol != '\n' || !dataFile.eof())

{

if (dataFile.eof())

{

spaces = 0;

break;

}

dataFile >> tmp\_symbol;

if (tmp\_symbol == ' ')

{

spaces++;

break;

}

if (tmp\_symbol == '\n')

{

spaces++;

break;

}

addData(index, tmp\_symbol);

index++;

if (dataFile.eof())

{

spaces = 0;

break;

}

stringLenght++;

if (stringLenght >= N) break;

}

}

void String::readStringV2(ifstream& dataFile, int& spaces, bool& is\_end)

{

char tmp\_symbol = '-';

int index = 0;

dataFile.unsetf(ios::skipws);

while (tmp\_symbol != '\n' || !dataFile.eof())

{

if (dataFile.eof())

{

spaces = 0;

break;

}

dataFile >> tmp\_symbol;

if (tmp\_symbol == ' ')

{

while (tmp\_symbol == ' ') dataFile >> tmp\_symbol;

}

if (tmp\_symbol == '|')

{

spaces++;

is\_end = true;

break;

}

if (tmp\_symbol == '\n')

{

is\_end = true;

break;

}

addData(index, tmp\_symbol);

index++;

if (dataFile.eof())

{

is\_end = true;

spaces = 0;

break;

}

stringLenght++;

if (stringLenght >= N) break;

}

}

void String::outData(ofstream& outFile)

{

for (int index = 0; index < stringLenght; index++)

{

cout << getData(index);

outFile << getData(index);

}

cout << '\n';

outFile << '\n';

}

void String::outDataWithoutEnter(ofstream& outFile)

{

for (int index = 0; index < stringLenght; index++)

{

cout << getData(index);

outFile << getData(index);

}

}

void String::outDataOnlyFile(ofstream& outFile)

{

for (int index = 0; index < stringLenght; index++)

{

outFile << getData(index);

}

outFile << '\n';

}

void String::outDataOnlyFileWithoutEnter(ofstream& outFile)

{

for (int index = 0; index < stringLenght; index++)

{

outFile << getData(index);

}

}

bool String::compareString(String S1, String S2)

{

if (1 == 0) return false;

else

{

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

{

if (S1.string[i] != S2.string[i]) return false;

}

}

return true;

}

bool String::compareFirst(String S1, String S2)

{

if (S1.string[0] <= S2.string[0]) return true;

else return false;

}

void String::enterUserData()

{

cin >> string;

this->getStringLength();

}

void String::converToDate(String S, int\* date)

{

int i = 0, j = 0;

while (i != S.stringLenght)

{

int result = 0;

while (S.getData(i) != '-')

{

if (i == S.stringLenght) break;

result = result \* 10 + (S.getData(i) - '0');

++i;

}

date[j] = result;

if (j > 2) break;

j++;

if (i == S.stringLenght) break;

i++;

}

}

bool String::isTrueDate(int\* date)

{

if ((date[1] > 12 && date[1] < 1) || (date[2] > 31 && date[2] < 1)) return false;

bool isLeapYear = false;

bool isBigMounth = false;

if (date[0] % 4 == 0)

isLeapYear = true;

if (date[1] <= 7 && (date[1] % 2 == 1) || ((date[1] >= 8) && (date[1] % 2 == 0))) isBigMounth = true;

else isBigMounth = false;

if (!isBigMounth && date[1] != 2 && date[2] <= 30) return true;

else if (!isBigMounth && date[1] != 2 && date[2] > 30) return false;

if (isBigMounth && date[2] <= 31) return true;

else if (isBigMounth && date[2] > 31) return false;

if (isLeapYear)

{

if (date[1] == 2 && date[2] <= 29) return true;

else if (date[1] == 2 && date[2] > 29) return false;

}

if (!isLeapYear)

{

if (date[1] == 2 && date[2] <= 28) return true;

else if (date[1] == 2 && date[2] > 28) return false;

}

}

bool String::inTheRightPeriodOfTime(String date1, String S, String date2)

{

int FDate[3], StringDate[3], SDate[3];

date1.converToDate(date1, FDate);

S.converToDate(S, StringDate);

date2.converToDate(date2, SDate);

if (date1.isTrueDate(FDate) && S.isTrueDate(StringDate) && date2.isTrueDate(SDate))

{

if ((FDate[0] < StringDate[0]) && (StringDate[0] < SDate[0])) return true;

else if ((FDate[0] <= StringDate[0]) && (StringDate[0] < SDate[0]))

{

if (FDate[1] == StringDate[1])

{

if (FDate[2] <= StringDate[2]) return true;

}

}

else if ((FDate[0] < StringDate[0]) && (StringDate[0] <= SDate[0]))

{

if (StringDate[1] == SDate[1])

{

if (StringDate[2] <= SDate[2]) return true;

}

}

else if ((FDate[0] == StringDate[0]) && (StringDate[0] == SDate[0]))

{

if ((FDate[1] < StringDate[1]) && (StringDate[1] < SDate[1])) return true;

else if ((FDate[1] == StringDate[1]) && (StringDate[1] == SDate[1]))

{

if ((FDate[2] <= StringDate[2]) && (StringDate[2] <= SDate[2])) return true;

}

else if ((FDate[1] == StringDate[1]))

{

if (FDate[2] <= StringDate[2]) return true;

}

else if ((StringDate[1] == SDate[1]))

{

if (StringDate[2] <= SDate[2]) return true;

}

}

}

return false;

}

bool String::compareDates(String date1, String date2)

{

int Fdate[3], Sdate[3];

date1.converToDate(date1, Fdate);

date2.converToDate(date2, Sdate);

if (date1.isTrueDate(Fdate) && date2.isTrueDate(Sdate))

{

if (Fdate[0] > Sdate[0]) return true;

else if (Fdate[0] == Sdate[0])

{

if (Fdate[1] > Sdate[1])

return true;

else if (Fdate[1] == Sdate[1])

{

if (Fdate[2] > Sdate[2]) return true;

else if (Fdate[2] == Sdate[2]) return true;

}

}

}

return false;

}

int String::typeOfDate()

{

int Fdate[3];

converToDate(\*this, Fdate);

if (stringLenght != 0)

{

if (string[0] == 'M')

return 2;

if (string[0] == 'K')

return 3;

if (isTrueDate(Fdate))

return 1;

}

else return -1;

}

bool String::comparebiba(String date1, String date2)

{

int Fdate[3];

date1.converToDate(date1, Fdate);

int Sdate[3];

date2.littleConvertToDate(date2, Sdate);

if (Fdate[1] == Sdate[0]) return true;

else return false;

}

bool String::compareQuarters(String date1, String date2)

{

int Fdate[3];

date1.converToDate(date1, Fdate);

int Sdate[3];

date2.littleConvertToDate(date2, Sdate);

if (((Fdate[1] / 3) == (Sdate[0] - 1)) || ((Fdate[1] / 3) == (Sdate[0]))) return true;

else return false;

}

void String::littleConvertToDate(String date, int\* dateint)

{

int i = 1, j = 0;

while (i != date.stringLenght)

{

int result = 0;

while (date.getData(i) != '-')

{

if (i == date.stringLenght) break;

result = result \* 10 + (date.getData(i) - '0');

++i;

}

dateint[j] = result;

if (j > 0) break;

j++;

if (i == date.stringLenght) break;

i++;

}

}

Stringlist.cpp

#include "StringList.h"

#include <iostream>

#include <fstream>

using namespace std;

StringList::StringList()

{

pHead = nullptr;

pTail = nullptr;

pCur = nullptr;

}

bool StringList::is\_empty()

{

return pHead == nullptr;

}

void StringList::pushFront(String newInfo)

{

pHead = new StringNode(newInfo, pHead);

if (pTail == nullptr)

pTail = pHead;

}

void StringList::pushBack(String newInfo)

{

if (is\_empty())

{

pushFront(newInfo);

}

else

insertAfter(pTail, newInfo);

}

void StringList::popFront()

{

StringNode\* nextHead = pHead->pNext;

delete pHead;

pHead = nextHead;

if (pHead == nullptr)

pTail = nullptr;

}

void StringList::insertAfter(StringNode\* elem, String newInfo)

{

elem->pNext = new StringNode(newInfo, elem->pNext);

if (pTail == elem)

pTail = pTail->pNext;

}

void StringList::eraseAfter(StringNode\* elem)

{

StringNode\* nextElem = elem->pNext;

elem->pNext = nextElem->pNext;

delete nextElem;

if (pTail == nextElem)

pTail = elem;

}

void StringList::readFromFile(ifstream& dataFile, int& spaces)

{

dataFile.unsetf(ios::skipws);

String S;

bool is\_end = false;

while (!is\_end)

{

S.readStringV2(dataFile, spaces, is\_end);

pushBack(S);

S.clearString();

}

}

int StringList::lengthList()

{

int length = 0;

pCur = pHead;

while (pCur != nullptr)

{

length++;

pCur = pCur->pNext;

}

return length;

}

bool StringList::compareStringList(StringList S1, StringList S2)

{

if (S1.lengthList() != S2.lengthList()) return false;

S1.pCur = S1.pHead;

S2.pCur = S2.pHead;

while (S1.pCur != nullptr)

{

if (S1.pCur->element.compareString(S1.pCur->element, S2.pCur->element))

{

S1.pCur = S1.pCur->pNext;

S2.pCur = S2.pCur->pNext;

}

else return false;

}

return true;

}

void StringList::converToDate(StringList S, int\* date)

{

int i = 0, j = 0;

S.pCur = S.pHead;

while (1)

{

int result = 0;

while (S.pCur->element.getData(i) != '-')

{

if (i == S.pCur->element.stringLenght)

{

S.pCur = S.pCur->pNext;

i = 0;

}

if (S.pCur == nullptr || S.pCur->element.string[0] == '\0')

{

break;

}

result = result \* 10 + (S.pCur->element.getData(i) - '0');

++i;

}

date[j] = result;

if (j > 2) break;

j++;

if (S.pCur == nullptr) break;

if (i == S.pCur->element.stringLenght) break;

i++;

}

}

void StringList::littleConvertToDate(StringList date, int\* dateint)

{

date.pCur = date.pHead;

int i = 1, j = 0;

while (date.pCur != nullptr)

{

int result = 0;

while (date.pCur->element.getData(i) != '\0')

{

if (i == date.pCur->element.stringLenght)

{

date.pCur = date.pCur->pNext;

i = 0;

}

if (date.pCur == nullptr) break;

result = result \* 10 + (date.pCur->element.getData(i) - '0');

++i;

}

dateint[j] = result;

if (j > 0) break;

j++;

if (date.pCur == nullptr) break;

if (i == date.pCur->element.stringLenght) break;

i++;

}

}

bool StringList::compareDates(StringList date1, StringList date2)

{

int Fdate[3], Sdate[3];

date1.converToDate(date1, Fdate);

date2.converToDate(date2, Sdate);

if (date1.isTrueDate(Fdate) && date2.isTrueDate(Sdate))

{

if (Fdate[0] > Sdate[0]) return true;

else if (Fdate[0] == Sdate[0])

{

if (Fdate[1] > Sdate[1])

return true;

else if (Fdate[1] == Sdate[1])

{

if (Fdate[2] > Sdate[2]) return true;

else if (Fdate[2] == Sdate[2]) return true;

}

}

}

return false;

}

bool StringList::comparebiba(StringList date1, StringList date2)

{

int Fdate[3];

date1.converToDate(date1, Fdate);

int Sdate[3];

date2.littleConvertToDate(date2, Sdate);

if (Fdate[1] == Sdate[0]) return true;

else return false;

}

bool StringList::compareQuarters(StringList date1, StringList date2)

{

int Fdate[3];

date1.converToDate(date1, Fdate);

int Sdate[3];

date2.littleConvertToDate(date2, Sdate);

if (((Fdate[1] / 3) == (Sdate[0] - 1)) || ((Fdate[1] / 3) == (Sdate[0]))) return true;

else return false;

}

bool StringList::isTrueDate(int\* date)

{

if ((date[1] > 12 && date[1] < 1) || (date[2] > 31 && date[2] < 1)) return false;

bool isLeapYear = false;

bool isBigMounth = false;

if (date[0] % 4 == 0)

isLeapYear = true;

if (date[1] <= 7 && (date[1] % 2 == 1) || ((date[1] >= 8) && (date[1] % 2 == 0))) isBigMounth = true;

else isBigMounth = false;

if (!isBigMounth && date[1] != 2 && date[2] <= 30) return true;

else if (!isBigMounth && date[1] != 2 && date[2] > 30) return false;

if (isBigMounth && date[2] <= 31) return true;

else if (isBigMounth && date[2] > 31) return false;

if (isLeapYear)

{

if (date[1] == 2 && date[2] <= 29) return true;

else if (date[1] == 2 && date[2] > 29) return false;

}

if (!isLeapYear)

{

if (date[1] == 2 && date[2] <= 28) return true;

else if (date[1] == 2 && date[2] > 28) return false;

}

}

void StringList::clearStringList()

{

while (!is\_empty()) popFront();

}

void StringList::outData(ofstream& outData)

{

pCur = pHead;

while (pCur != nullptr)

{

pCur->element.outDataWithoutEnter(outData);

pCur = pCur->pNext;

}

cout << '\n';

outData << '\n';

}

void StringList::outDataOnlyFile(ofstream& outData)

{

pCur = pHead;

while (pCur != nullptr)

{

pCur->element.outDataOnlyFileWithoutEnter(outData);

pCur = pCur->pNext;

}

outData << '\n';

}

void StringList::outDataOnlyFileWithoutEnter(ofstream& outData)

{

pCur = pHead;

outData << "| ";

while (pCur != nullptr)

{

pCur->element.outDataOnlyFileWithoutEnter(outData);

outData << "->";

pCur = pCur->pNext;

}

outData << "NULL |";

}

int StringList::typeOfDate()

{

int Fdate[3];

converToDate(\*this, Fdate);

if (this->pHead->element.stringLenght != 0)

{

if (this->pHead->element.string[0] == 'M')

return 2;

if (this->pHead->element.string[0] == 'K')

return 3;

if (isTrueDate(Fdate))

return 1;

}

else return -1;

}

void StringList::enterUserData()

{

char string[100];

cin >> string;

fillWithInfo(string);

}

void StringList::fillWithInfo(char\* string)

{

int index = 0;

int index\_s = 0;

String str;

while (1)

{

if (index\_s == 10)

{

str.stringLenght = index\_s;

pushBack(str);

str.clearString();

index\_s = 0;

}

if (string[index] == '\0')

{

//if (index\_s == 0) break;

str.stringLenght = index\_s;

pushBack(str);

str.clearString();

index\_s = 0;

break;

}

str.string[index\_s] = string[index];

index\_s++;

index++;

}

}

StringNode.cpp

#include "StringNode.h"

StringNode::StringNode(String edElem, StringNode\* n)

{

element = edElem;

pNext = n;

}

CardIndex.h

#pragma once

#include "CardIndexNode.h"

#include "List.h"

#include <fstream>

using namespace std;

struct CardIndex

{

CardIndexNode\* pHead, \* pTail, \* pCur;

CardIndex();

bool is\_empty();

void pushFront(ListPointersCI necessaryData, char edPar);

void pushBack(ListPointersCI necessaryData, char edPar);

void insertAfter(CardIndexNode\* elem, ListPointersCI necessaryData, char edPar);

void popFront();

void eraseAfter(CardIndexNode\* elem);

void createEmptyList();

void readingFromFile(ifstream& dataFile, List& BusType, List& date, List& time, List& pointA, List& pointB, ofstream& logs);

void compareWithboba(List pointA);

bool itWas(List pointA);

void searchBybusType(ofstream& outData, ofstream& logs);

void searchByRoute(ofstream& outData, ofstream& logs);

//void searchByDateAndpointA(ofstream& outData, ofstream& logs);

void log(ofstream& log);

void outToLog(ofstream& logs);

//Удаление всех данных

void deleteAll(List& busType, List& date, List& time, List& pointA, ofstream& logs);

};

CardIndexNode.h

#pragma once

#include "ListPointersCI.h"

struct CardIndexNode

{

ListPointersCI necessaryDataAboutBus;

char boba;

CardIndexNode\* pNext;

CardIndexNode(ListPointersCI addedData, CardIndexNode\* n, char edPar);

};

List.h

#pragma once

#include "Node.h"

#include "fstream"

using namespace std;

struct List

{

Node\* pHead, \* pTail, \* pCur;

List();

bool is\_empty();

void pushFront(StringList newInfo);

void pushBack(StringList newInfo);

void insertAfter(Node\* elem, StringList newInfo);

void popFront();

void eraseAfter(Node\* elem);

bool isUnique(List& tmp, StringList S);

void readFromFile(ifstream& dataFile);

//Новый тип сравнения и заполнения массивов

void fillingWithData(List& tmp, StringList S);

void outAllDataList(ofstream& logs);

//void printList(ofstream& outFile);

//void readFromFile(ifstream& dataFile, List& busType, List& date, List& time, List& pointA);

};

ListPointers.h

#pragma once

#include "NodePointers.h"

struct ListPointers

{

NodePointers\* pHead, \* pTail, \* pCur;

ListPointers();

bool is\_empty();

void pushFront(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB);

void pushBack(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB);

void insertAfter(NodePointers\* elem, StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB);

void popFront();

void eraseAfter(NodePointers\* elem);

void outAllData(ofstream& logs);

};

ListPointersCI.h

#pragma once

#include "NodePointersCI.h"

struct ListPointersCI

{

NodePointersCI\* pHead, \* pTail, \* pCur;

ListPointersCI();

bool is\_empty();

void pushFront(StringList\* editedData, ListPointers\* otherEdData);

void pushBack(StringList\* editedData, ListPointers\* otherEdData);

void insertAfter(NodePointersCI\* elem, StringList\* editedData, ListPointers\* otherEdData);

void popFront();

void eraseAfter(NodePointersCI\* elem);

void outDataListCI(ofstream& logs);

};

Node.h

#pragma once

#include "ListPointers.h"

#include "StringList.h"

struct Node

{

StringList data;

ListPointers otherData;

Node\* pNext;

Node(StringList editedInfo, Node\* n);

};

NodeInfo.h

#pragma once

#include "String.h"

struct NodeInfo

{

String data;

NodeInfo(String edS);

};

NodePointers.h

#pragma once

#include "NodeInfo.h"

#include "StringList.h"

struct NodePointers

{

StringList\* otherDataNUM;

StringList\* otherDataTM;

StringList\* otherDataD;

StringList\* otherDataPB;

NodePointers\* pNext;

NodePointers(StringList\* otherEdDataAT, StringList\* otherEdDataD, StringList\* otherEdDataFT, StringList\* otherEdDataPB, NodePointers\* n);

};

NodePointersCI.h

#pragma once

#include "Node.h"

#include "ListPointers.h"

struct NodePointersCI

{

StringList\* necessaryData;

ListPointers\* otherData;

NodePointersCI\* pNext;

NodePointersCI(StringList\* editedData, ListPointers\* otherEdData, NodePointersCI\* n);

};

String.h

#pragma once

#include <fstream>

using namespace std;

const unsigned N = 10;

struct String

{

char string[N];

int stringLenght = 0;

int digit\_count = 0;

void clearString();

void addData(int index, char tmp\_symbol);

char getData(int index);

void readString(ifstream& dataFile, int& spaces);

void readStringV2(ifstream& dataFile, int& spaces, bool& is\_end);

void outData(ofstream& outFile);

void outDataWithoutEnter(ofstream& outFile);

void outDataOnlyFile(ofstream& outFile);

void outDataOnlyFileWithoutEnter(ofstream& outFile);

bool compareString(String S1, String S2);

bool compareFirst(String S1, String S2);

//Функции созданные для задачи

void getStringLength();

void enterUserData();

//Всё так же функции для задачи, но для работы с датами

int typeOfDate();

void converToDate(String S, int\* date);

void littleConvertToDate(String S, int\* date);

bool isTrueDate(int\* date);

bool inTheRightPeriodOfTime(String date1, String S2, String date2);

bool compareDates(String date1, String date2);

bool comparebiba(String date1, String date2);

bool compareQuarters(String date1, String date2);

};

StringList.h

#pragma once

#include "StringNode.h"

#include "fstream"

using namespace std;

struct StringList

{

StringNode\* pHead, \* pTail, \* pCur;

StringList();

bool is\_empty();

void pushFront(String newInfo);

void pushBack(String newInfo);

void insertAfter(StringNode\* elem, String newInfo);

void popFront();

void eraseAfter(StringNode\* elem);

void readFromFile(ifstream& dataFilem, int& spaces);

int lengthList();

bool compareStringList(StringList S1, StringList S2);

void converToDate(StringList S, int\* date);

void littleConvertToDate(StringList S, int\* date);

bool compareDates(StringList date1, StringList date2);

bool comparebiba(StringList date1, StringList date2);

bool compareQuarters(StringList date1, StringList date2);

bool isTrueDate(int\* date);

void clearStringList();

void outData(ofstream& outData);

void outDataOnlyFile(ofstream& outData);

void outDataOnlyFileWithoutEnter(ofstream& outData);

int typeOfDate();

void enterUserData();

void fillWithInfo(char\* string);

};

#pragma once

#include "String.h"

struct StringNode

{

String element;

StringNode\* pNext;

StringNode(String edElem, StringNode\* n);

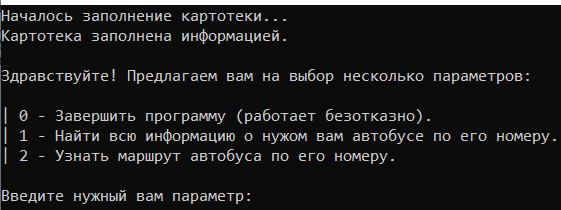
};

11. Результаты работы программы

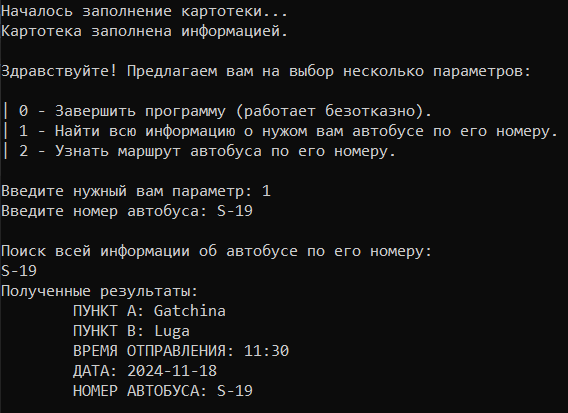
Входные данные:

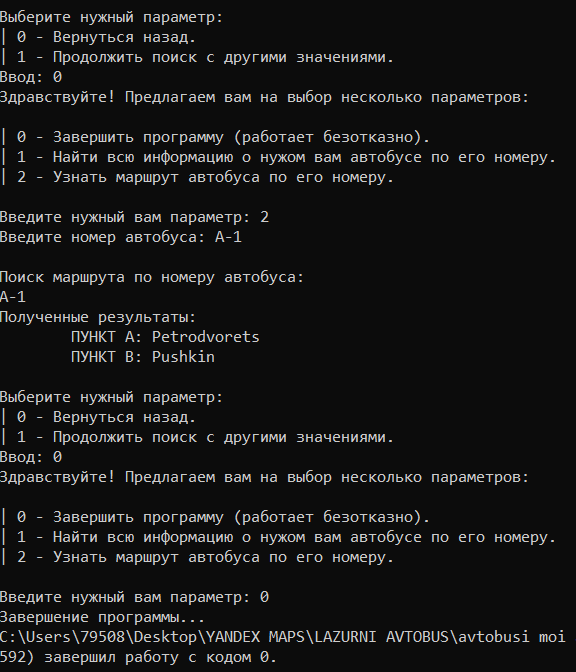


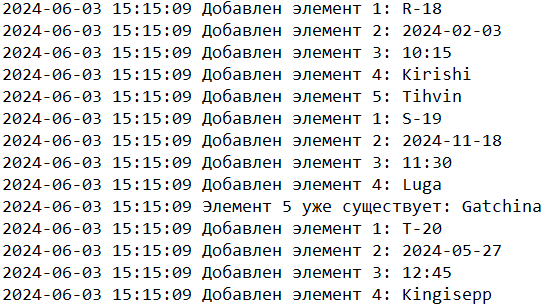
Начало работы в консоли:

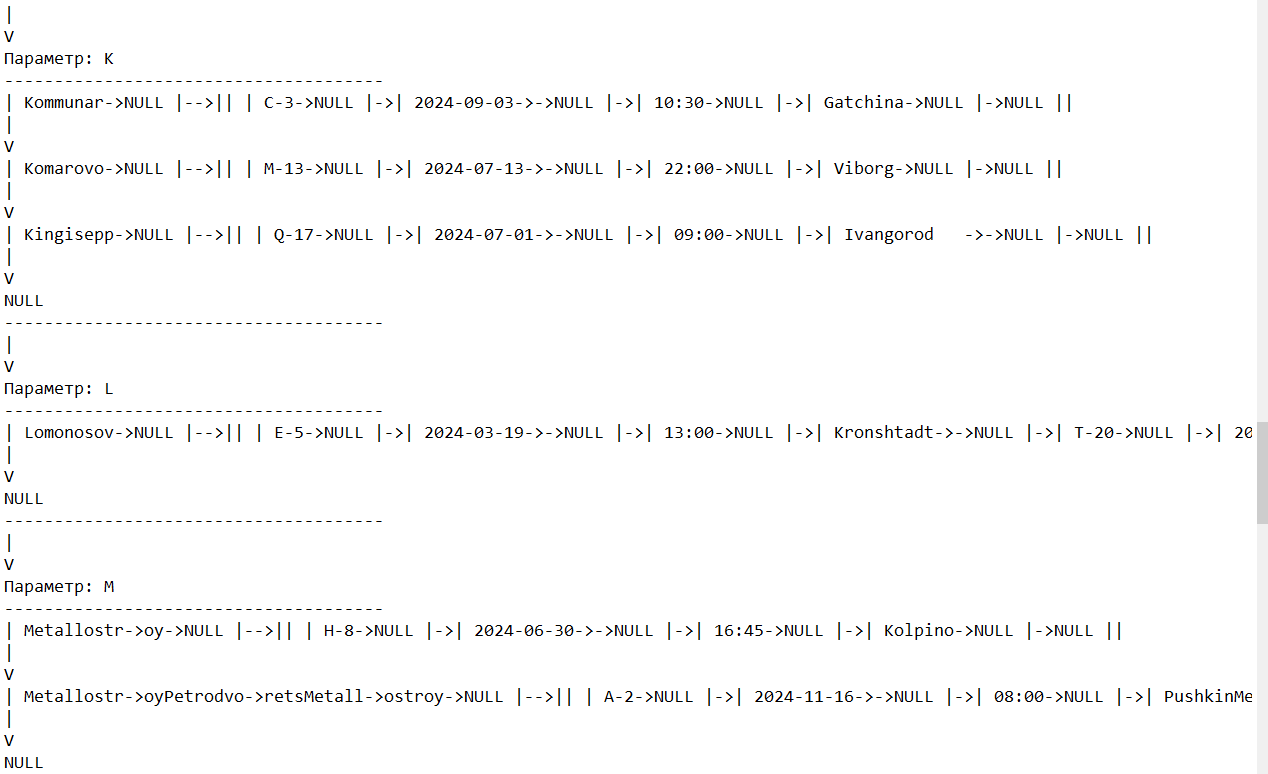


Работа с запросами:





Файл log (частично):



12. Вывод

В ходе выполнения данной курсовой работы были отработаны навыки работы с иерархическими односвязными списками в языке C++. Написана программа, способная хранить информацию в списках, а также отвечать на запросы пользователя.