МИНОБРНАУКИ РОСИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №1.6

Тема:

Двунаправленные динамические списки

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Павлов Н.С.

Группа: ИКБО-30-23

Москва 2024

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Получение знаний и практических навыков управления двунаправленным списком в программах на языке C++.

2. ВЫПОЛНЕНИЕ ЗАДАНИЯ

2.1 ФОРМУЛИРОВКА ЗАДАЧИ

Разработать многомодульную программу, которая демонстрирует выполнение всех операций, определенных вариантом, над линейным двунаправленным динамическим списком.

Требования к разработке.

1. Разработать структуру узла списка, структура информационной части узла определена вариантом. Для определения структуры узла списка, используйте тип struct или class. Сохраните определение структуры узла и прототипы функций в заголовочном файле.

2. Разработайте функции для выполнения операции над линейным двунаправленным динамическим списком:

- создание списка;
- вставку узла;
- удаление узла;
- вывод списка в двух направлениях (слева направо и справа налево);

3. поиск узла с заданным значением (операция должна возвращать указатель на узел с заданным значением).

4. Дополнительные операции над списком, указанные вариантом, оформите в виде функций и включите в отдельный файл с расширением cpp. Подключите к этому файлу заголовочный файл с определением структуры узла.

5. Разработайте программу, управляемую текстовым меню, и включите в меню демонстрацию выполнения всех операций задания и варианта.

Персональный вариант: Структура узла списка содержит ссылку на следующий и предыдущий узел, а также информационные поля: Номер железнодорожного билета (буквенно-цифровой), станция назначения, номер поезда, номер вагона, номер места, стоимость проезда, дата продажи, дата отправления поезда, время в пути. Требуется дополнительно разработать функции с следующим функционалом:

1. Вставить новый узел в список, упорядоченный по номеру вагона.

2. Удалить сведения о билетах, пассажиры которых добрались уже до места (оценка по текущей дате).

3. Сформировать новый список из узлов исходного, включив в него сведения о тех проданных билетах, по которым пассажиры поедут в указанную дату.

2.2 СПИСОК ОПЕРАЦИЙ НАД СПИСКОМ

2.2.1 Структура узла

Узел содержит 11 полей:

Поля целочисленного типа:

- train – номер поезда;
- cart – номер вагона;
- seat – номер сиденья;
- cost – стоимость билета.

Поля строкового типа:

- bilet – номер билета (буквенно-цифровой);
- station – станция назначения;
- date_sale – дата продажи;
- date_leave – дата отправления поезда;
- time – время в пути.

Поля типа указателя на узел:

- next – указатель на следующий узел в списке;
- prev – указатель на предыдущий узел списка.

2.2.2 Схемы операций над списком

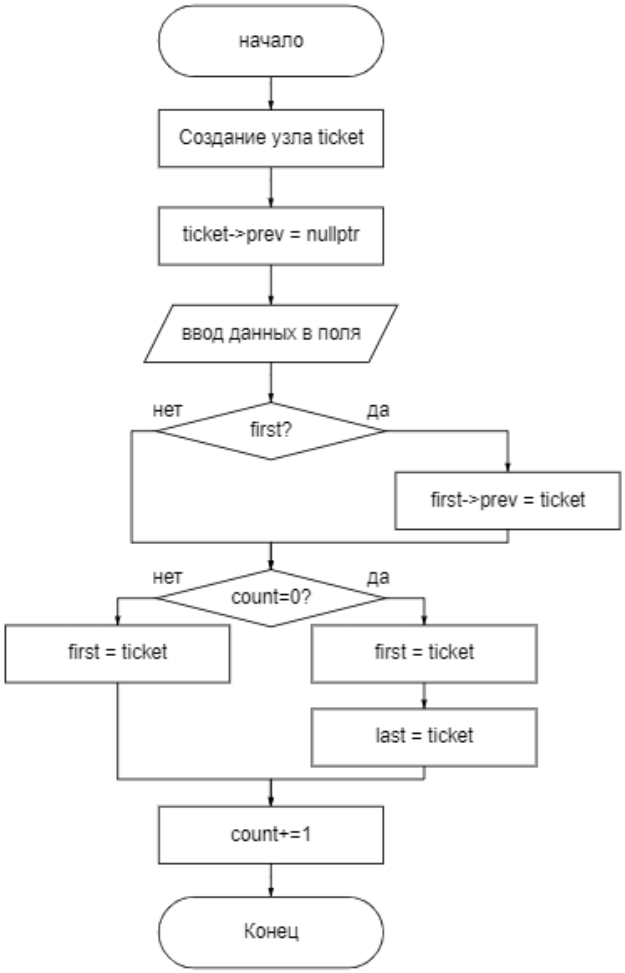

Рисунок 1 – Блок-схема функции вставки узла в начало списка.

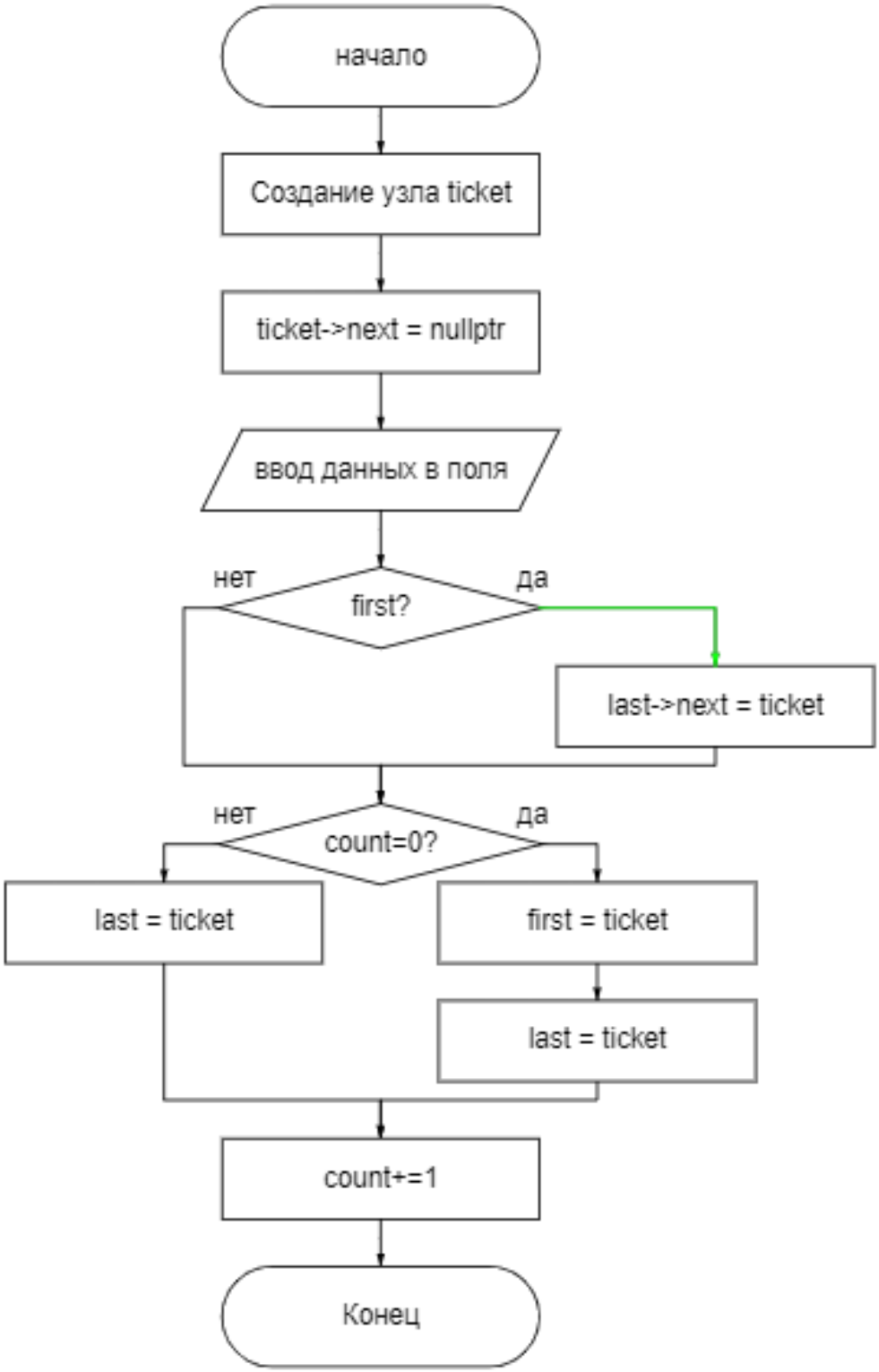

Рисунок 2 – Блок-схема функции вставки узла в конец списка.

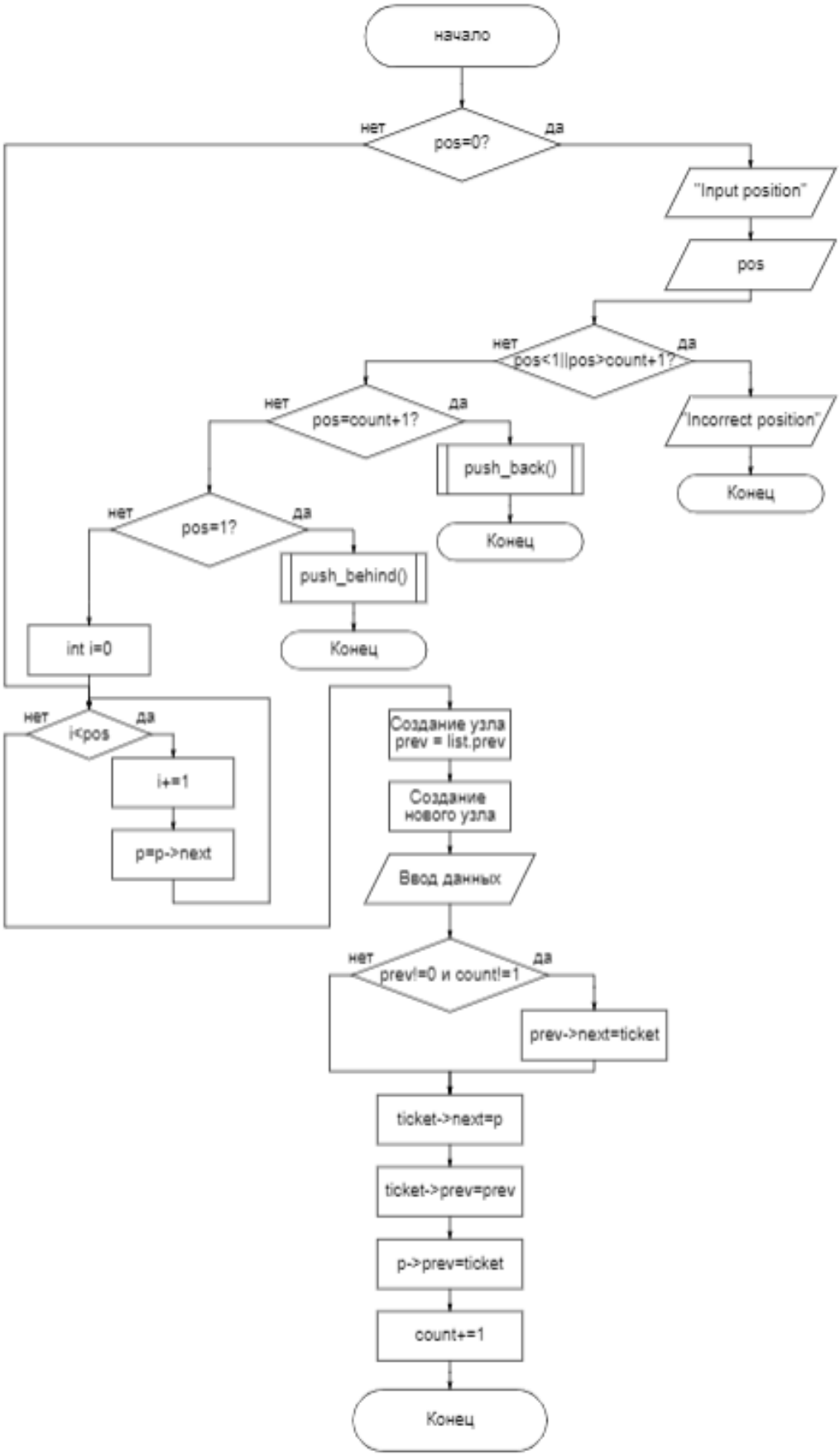

Рисунок 3 – Блок-схема функции вставки узла на указанное место.

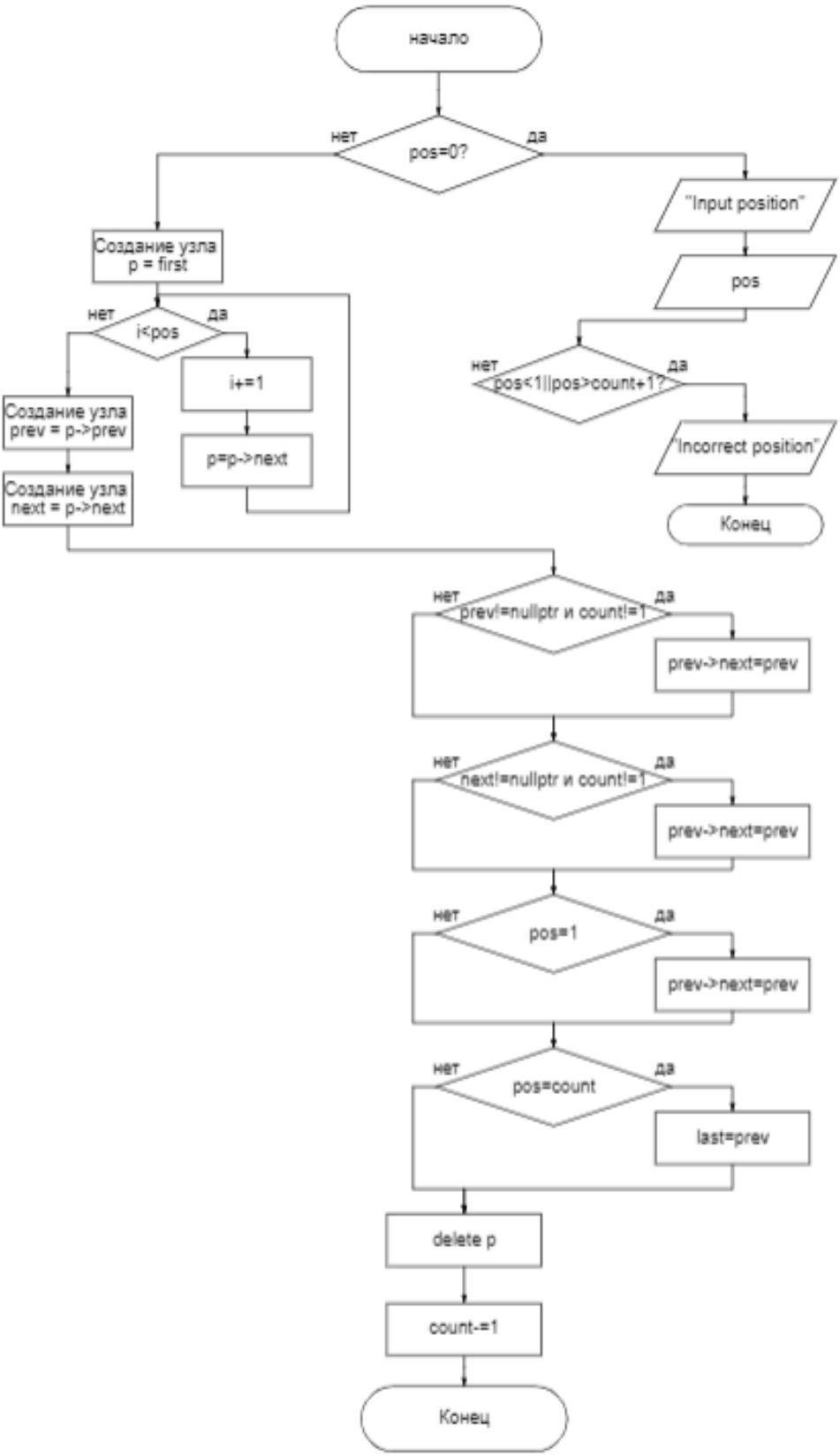

Рисунок 4 – Блок-схема функции удаления узла на указанной позиции.

Рисунок 5 – Блок-схема функции вывода списка.

Рисунок 6 – Блок-схема функции вывода списка в обратном порядке.

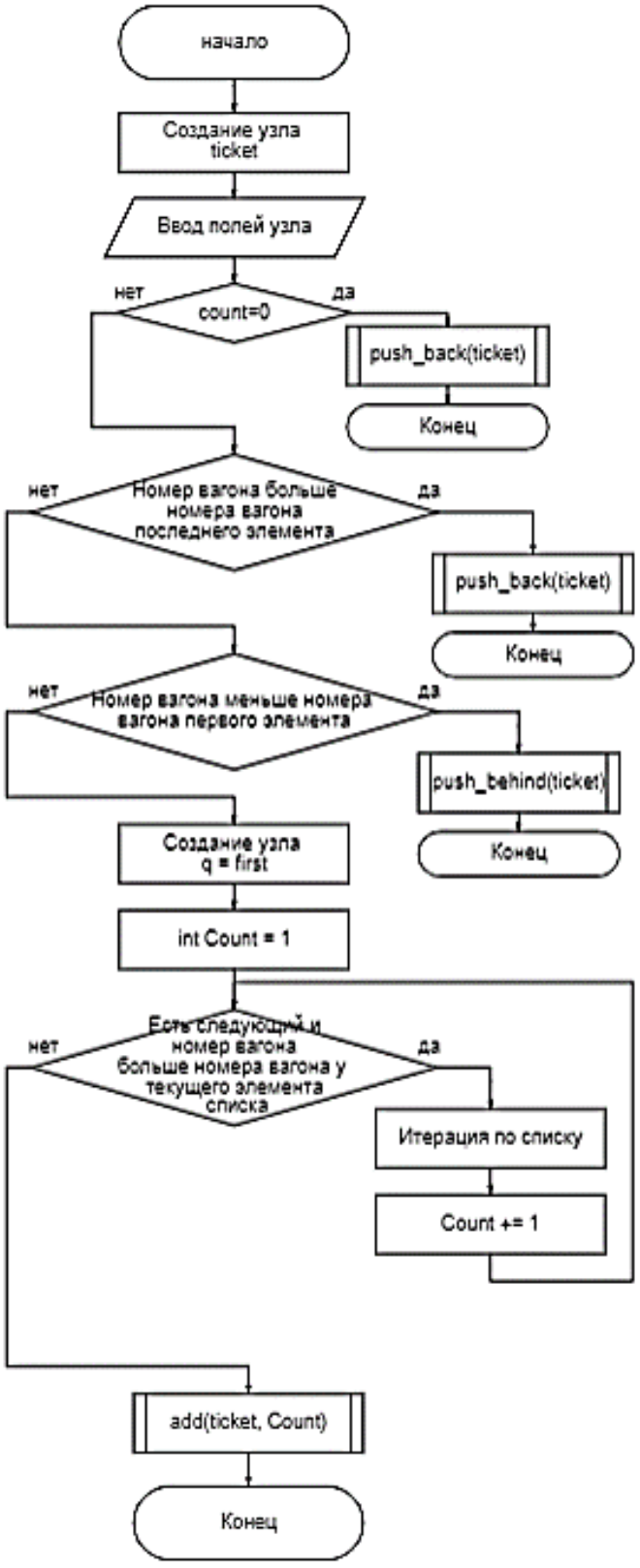

Рисунок 7 – Блок-схема функции вставки узла в список, упорядоченный по номеру вагона.

В функции удаления данных, основываясь на дате, дата вычисляется по формуле year*365+mon*30+day, где year – год, month - месяц, day – день, для удобочитаемости схемы используется обозначение date (дата в текущем билете) и now (текущая дата).

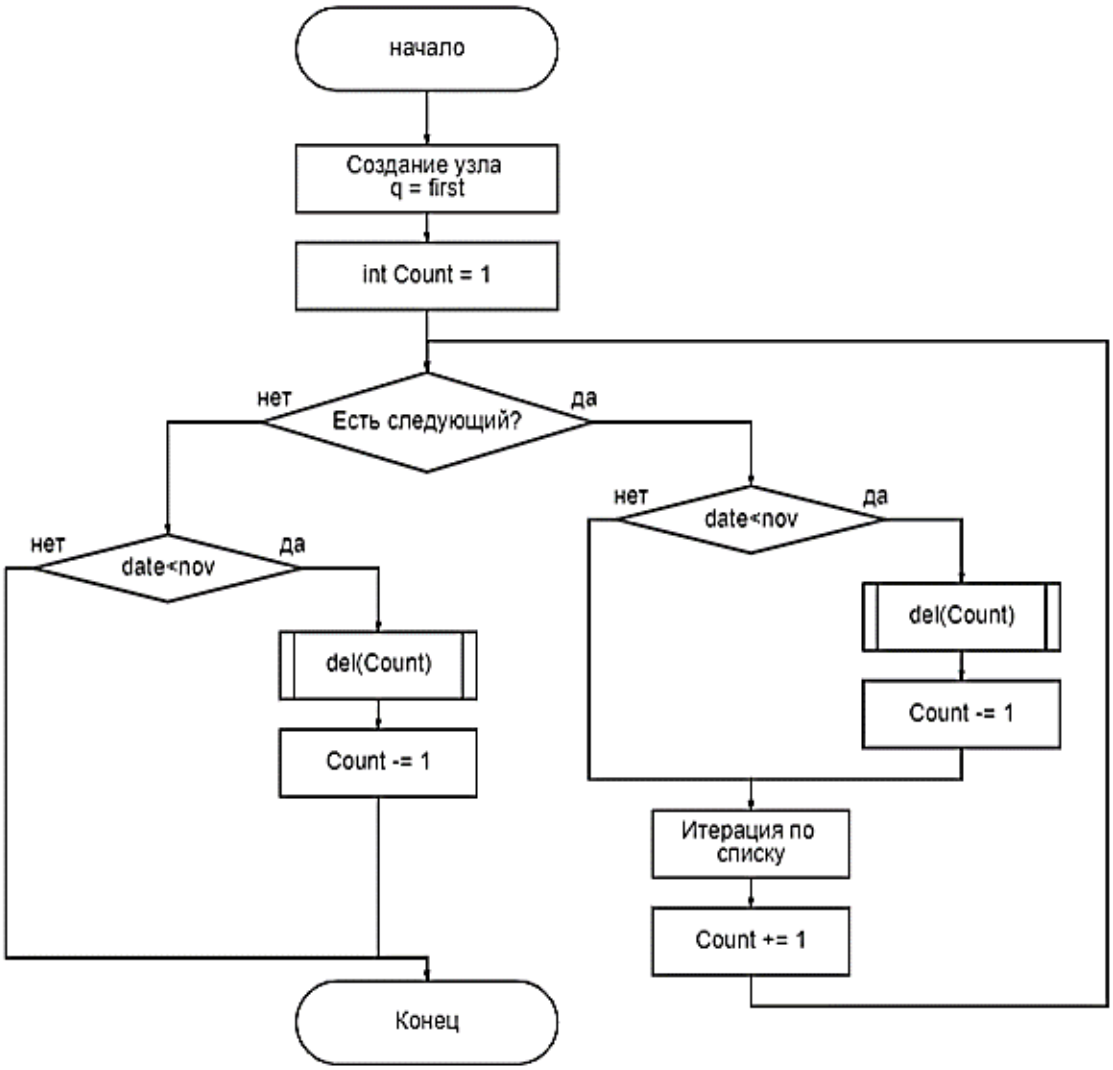

Рисунок 8 – Блок-схема функции удаления билетов, которые прибыли к месту назначения.

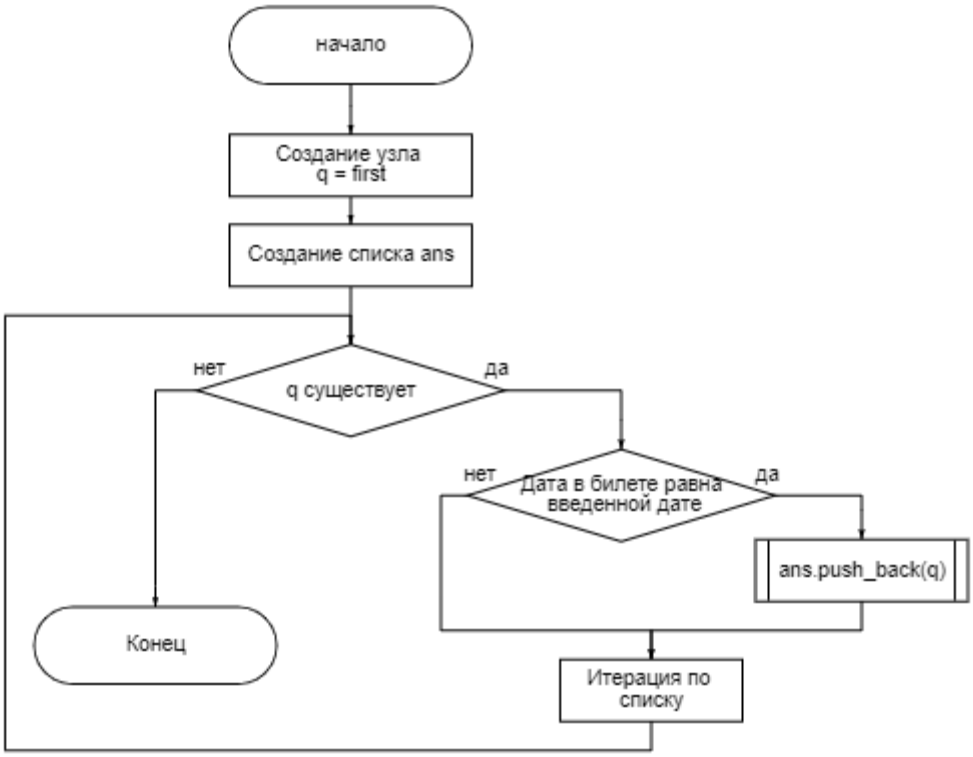

Рисунок 9 – Блок-схема функции создания списка с билетами с указанной датой.

2.2.3 Структура данных

Ниже представлен схематичный рисунок используемой структуры

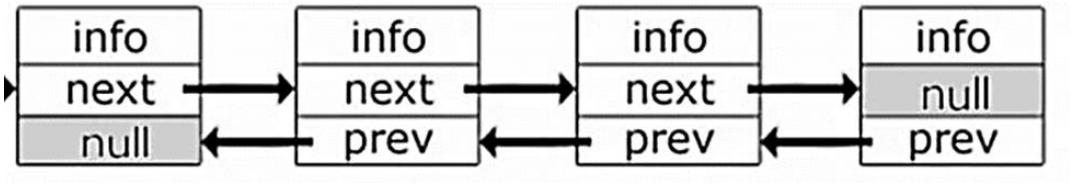

Рисунок 10 – Структура данных

2.2.4 Алгоритм выполнения операций

1. push_behind(). Функция создает узел, устанавливает предшествующему ему элементу значение nullptr, а следующему – значение первого элемента списка. Если первый элемент не существует, предшествующему ему элементу присваивается значение нашего узла. Если список пустой, то первому элементу и последнему элементу присваивается

значение нашего узла, если нет, то первому элементу присваивается значение нашего узла. Счётчик элементов увеличивается на 1.

2. push_back(). Функция создает узел, устанавливает следующему за ним элементу значение nullptr, а предыдущему – значение последнего элемента списка. Если последний элемент не существует, предшествующему ему элементу присваивается значение нашего узла. Если список пустой, то первому элементу и последнему элементу присваивается значение нашего узла, если нет, то последнему элементу присваивается значение нашего узла. Счётчик элементов увеличивается на 1.

3. add(int pos). Если позиция равна 0, запрашивается ввод позиции. Если позиция некорректна, выводится соответствующее сообщение. Если позиция на 1 больше количества элементов списка, вызывается функция добавления узла в конец списка, если позиция равна 1, вызывается функция добавления элемента в начало списка. Если ни одно условие не выполнено, создаем узел равный первому узлу списка и переменную i = 1 и, пока она меньше введенной позиции, итерируемся по списку, каждый раз увеличивая i на 1. Создаем вспомогательный узел для настройки связей. Создаем новый узел равный первому узлу списка и вводим данные с клавиатуры, после чего настраиваем связи. Счетчик элементов увеличивается на 1.

4. del(int pos). Если позиция равна 0, запрашивается ввод позиции. Если позиция некорректна, выводится соответствующее сообщение. Создаем узел равный первому узлу списка и переменную i = 1 и, пока она меньше введенной позиции, итерируемся по списку, каждый раз увеличивая i на 1. Настраиваем связи, как бы пропуская элемент, который нужно удалить и все его поля. В итоге ничего не ссылается на поля удаляемого элемента. Удаляем элемент и уменьшаем счётчик элементов на 1.

5. print(). Создаем узел равный первому узлу списка, пока он существует, выводим данные из него и переходим к следующему узлу. После цикла выводим перенос на следующую строку.

6. printback(). Создаем узел равный последнему узлу списка, пока он существует, выводим данные из него и переходим к предыдущему узлу. После цикла выводим перенос на следующую строку.

7. cartsorted(). Создаем узел и вводим значения его полей. Если список пуст или если номер вагона очередного билета больше номера вагона последнего элемента списка, вызываем функцию push_back(). Иначе, если номер вагона очередного билета больше номера вагона последнего элемента списка меньше номера вагона первого элемента списка, вызываем функцию push_behind(). Если ничего из этого не выполнено, создаем узел со значением первого элемента списка. Инициализируем переменную Count = 1. Пока есть следующий элемент и номер вагона больше номера вагона в текущем элементе, Итерируемся по списку и увеличиваем Count на 1 каждый раз. После цикла вызываем функцию add([Наш узел], Count).

8. del_reach(string date). Создаем узел со значением первого элемента списка. Инициализируем переменную Count = 1. Пока есть следующий элемент: 1) Если дата в билете раньше, чем текущая, вызывается функция del(Count) и Count уменьшается на 1. 2) Итерируемся по списку. 3) Count увеличивается на 1. Аналогичная операция повторяется для последнего элемента.

9. list_date(string date). Создаем новый список и узел равный первому узлу текущего списка. Пока он существует: 1) Проверяется, равна ли дата в билете введенной дате, и, если равна, этот билет добавляется в ранее созданный список. 2) Итерируемся по списку. После цикла возвращается созданный вначале список.

2.2.5 Контрольные тесты для операций

1. Вставка узла в список, отсортированный по номеру вагона (во вставляемом узле указан только номер вагона). Номер вагона – четвертое поле в выводе.

Входные данные	Выходные данные
1 1 3 4 5 9 12	1 1 3 4 5 8 9 12
1 1 1 1 2 4 8	1 1 1 1 1 2 4 8
1 2 2 3 5	1 2 2 3 4 5

2. Удаление билетов, которые уже добрались (в каждом из узлов указана только дата отъезда). Текущая дата 19.03.2024

Входные данные	Выходные данные
12.04.2022 14.04.2024 15.01.2024 29.02.2025	14.04.2023 29.02.2025
01.01.2023 02.04.2024 23.08.2024	02.04.2024 23.08.2024
12.04.2022 29.02.2023 01.01.2023 02.04.2023	[Пустой список]

3. Создание списка из билетов, по которым поедут в указанную дату (в каждом из узлов указана только дата отъезда). Первая строчка – необходимая дата

Входные данные	Выходные данные
13.04.2022 13.04.2022 13.04.2022 15.01.2023 29.02.2023 13.04.2022	13.04.2022 13.04.2022 13.04.2022
01.01.2023 01.01.2023 01.01.2023 23.08.2023	01.01.2023 01.01.2023
28.02.2023 12.04.2022 29.02.2023 01.01.2023 02.04.2023	[Пустой список]

2.3 РЕАЛИЗАЦИЯ АЛГОРИТМА

```
struct Node {
public:
    string bilet;
    string station;
    int train;
    int cart;
    int seat;
    int cost;
    string date_sale;
    string date_leave;
    string time;

    Node* next = nullptr;
    Node* prev = nullptr;
};
```

```
class List {
    Node* first = nullptr;
    Node* last = nullptr;
    int count;
public:
    List();
    void push_behind();
    void push_behind_node(Node* node);
    void push_back();
    void push_back_node(Node* node);
    void add(int pos);
    void del(int pos);
    void add_node(Node* node, int pos);

    void print();
    void printback();

    void cartsorted();
    void del_reach(string date);
    List list_date(string date);

    Node* clone_single(Node* t);
}
```

Рисунок 11-12 – Реализация header файла

```
List::List() {
    first = nullptr;
    last = nullptr;
    count = 0;
}

void List::push_behind() {
    Node* ticket = new Node;
    ticket->prev = nullptr;
    cin >> ticket->bilet >> ticket->station >> ticket->train >> ticket->cart >> ticket->seat
        >> ticket->cost >> ticket->date_sale >> ticket->date_leave >> ticket->time;
    ticket->next = first;
    if (first != nullptr)
        first->prev = ticket;
    if (count == 0) {
        first = ticket;
        last = ticket;
    }
    else
        first = ticket;
    count++;
}
```

Рисунок 13 – Реализация функций (Часть 1)

```
void List::push_behind_node(Node* ticket) {
    ticket->next = first;
    if (first != nullptr)
        first->prev = ticket;
    if (count == 0) {
        first = ticket;
        last = ticket;
    }
    else
        first = ticket;
    count++;
}

void List::push_back() {
    Node* ticket = new Node;
    ticket->next = nullptr;
    cin >> ticket->bilet >> ticket->station >> ticket->train >> ticket->cart >> ticket->seat
        >> ticket->cost >> ticket->date_sale >> ticket->date_leave >> ticket->time;
    ticket->prev = last;
    if (last != nullptr)
        last->next = ticket;
    if (count == 0) {
        first = ticket;
        last = ticket;
    }
    else
        last = ticket;
    count++;
}

void List::push_back_node(Node* ticket) {
    ticket->prev = last;
    if (last != nullptr)
        last->next = ticket;
    if (count == 0) {
        first = ticket;
        last = ticket;
    }
    else
        last = ticket;
    count++;
}
```

Рисунок 14 – Реализация функций (Часть 2)

```
void List::add(int pos) {
    if (pos == 0) {
        cout << "Input position ";
        cin >> pos;
    }
    if (pos<1 || pos> count + 1) {
        cout << "Incorrect position!";
        return;
    }
    if (pos == count + 1) {
        push_back();
        return;
    }
    else if (pos == 1) {
        push_behind();
        return;
    }
    int i = 1;
    Node* p = first;
    while (i < pos) {
        p = p->next;
        i++;
    }
    Node* prev = p->prev;
    Node* ticket = new Node;

    cin >> ticket->bilet >> ticket->station >> ticket->train >> ticket->cart >> ticket->seat
        >> ticket->cost >> ticket->date_sale >> ticket->date_leave >> ticket->time;

    if (prev != 0 && count != 1)
        prev->next = ticket;
    ticket->next = p;
    ticket->prev = prev;
    p->prev = ticket;

    count++;
}
```

Рисунок 15 – Реализация функций (Часть 3)

```
void List::add_node(Node* ticket, int pos) {
    if (pos == count + 1) {
        push_back_node(ticket);
        return;
    }
    else if (pos == 1) {
        push_behind_node(ticket);
        return;
    }
    int i = 1;
    Node* p = first;
    while (i < pos) {
        p = p->next;
        i++;
    }
    Node* prev = p->prev;

    if (prev != 0 && count != 1)
        prev->next = ticket;
    ticket->next = p;
    ticket->prev = prev;
    p->prev = ticket;

    count++;
}
```

Рисунок 16 – Реализация функций (Часть 4)

```
void List::del(int pos) {
    if (pos == 0) {
        cout << "Input position ";
        cin >> pos;
    }
    if (pos<1 || pos> count + 1) {
        cout << "Incorrect position!";
        return;
    }
    int i = 1;
    Node* p = first;
    while (i < pos) {
        p = p->next;
        i++;
    }
    Node* prev = p->prev;
    Node* next = p->next;
    if (prev != nullptr && count != 1)
        prev->next = next;
    if (next != nullptr && count != 1)
        next->prev = prev;
    if (pos == 1)
        first = next;
    if (pos == count)
        last = prev;
    delete p;
    count--;
}
```

Рисунок 17 – Реализация функций (Часть 5)

```
void List::print() {
    Node* ticket = first;
    while (ticket) {
        cout << ticket->bilet << " " << ticket->station << " " << ticket->train
            << " " << ticket->cart << " " << ticket->seat << " " << ticket->cost
            << " " << ticket->date_sale << " " << ticket->date_leave << " " << ticket->time << endl;
        ticket = ticket->next;
    }
    cout << endl;
}

void List::printback() {
    Node* ticket = last;
    while (ticket) {
        cout << ticket->bilet << " " << ticket->station << " " << ticket->train << " "
            << ticket->cart << " " << ticket->seat << " " << ticket->cost << " "
            << ticket->date_sale << " " << ticket->date_leave << " " << ticket->time << endl;
        ticket = ticket->prev;
    }
    cout << endl;
}
```

Рисунок 18 – Реализация функций (Часть 6)

```
void List::cartsorted() {
    Node* ticket = new Node;
    cin >> ticket->bilet >> ticket->station >> ticket->train >> ticket->cart >> ticket->seat
        >> ticket->cost >> ticket->date_sale >> ticket->date_leave >> ticket->time;
    if (count == 0) {
        push_back_node(ticket);
        return;
    }
    if (ticket->cart > last->cart) {
        push_back_node(ticket);
        return;
    }
    if (ticket->cart <= first->cart) {
        push_behind_node(ticket);
        return;
    }
    Node* q = first;
    int Count = 1;
    while (q->next && ticket->cart > q->cart) {
        Count += 1;
        q = q->next;
    }
    add_node(ticket, Count);
    return;
}
```

Рисунок 19 – Реализация функций (Часть 7)

```
void List::del_reach(string date) {
    int day1 = (date[0] - 48) * 10 + date[1] - 48;
    int mon1 = (date[3] - 48) * 10 + date[4] - 48;
    int year1 = (date[6] - 48) * 1000 + (date[7] - 48) * 100 + (date[8] - 48) * 10 + (date[9] - 48);
    int now = year1 * 365 + mon1 * 30 + day1;
    Node* q = first;
    int Count = 1;
    while (q->next) {
        int day = (q->date_leave[0] - 48) * 10 + q->date_leave[1] - 48;
        int mon = (q->date_leave[3] - 48) * 10 + q->date_leave[4] - 48;
        int year = (q->date_leave[6] - 48) * 1000 + (q->date_leave[7] - 48) * 100
            + (q->date_leave[8] - 48) * 10 + (q->date_leave[9] - 48);
        int leave = year * 365 + mon * 30 + day;
        if (leave < now) {
            Node* nextNode = q->next;
            del(Count);
            q = nextNode;
        }
        else {
            q = q->next;
            Count++;
        }
    }
    int day = (q->date_leave[0] - 48) * 10 + q->date_leave[1] - 48;
    int mon = (q->date_leave[3] - 48) * 10 + q->date_leave[4] - 48;
    int year = (q->date_leave[6] - 48) * 1000 + (q->date_leave[7] - 48) * 100
        + (q->date_leave[8] - 48) * 10 + (q->date_leave[9] - 48);
    int leave = year * 365 + mon * 30 + day;
    if (leave < now)
        del(Count);
}
```

Рисунок 20 – Реализация функций (Часть 8)

```
List List::list_date(string date) {
    List ans;
    Node* q = first;
    while (q) {
        if (q->date_leave == date) {
            ans.push_back_node(clone_single(q));
        }
        q = q->next;
    }
    return ans;
}

Node* List::clone_single(Node* t)
{
    Node* temp = new Node();
    temp->bilet = t->bilet;
    temp->cart = t->cart;
    temp->cost = t->cost;
    temp->date_leave = t->date_leave;
    temp->date_sale = t->date_sale;
    temp->seat = t->seat;
    temp->station = t->station;
    temp->time = t->time;
    temp->train = t->train;

    temp->next = nullptr;
    temp->prev = nullptr;

    return temp;
}
```

Рисунок 21 – Реализация функций (Часть 9)

2.4 ТЕСТИРОВАНИЕ АЛГОРИТМА

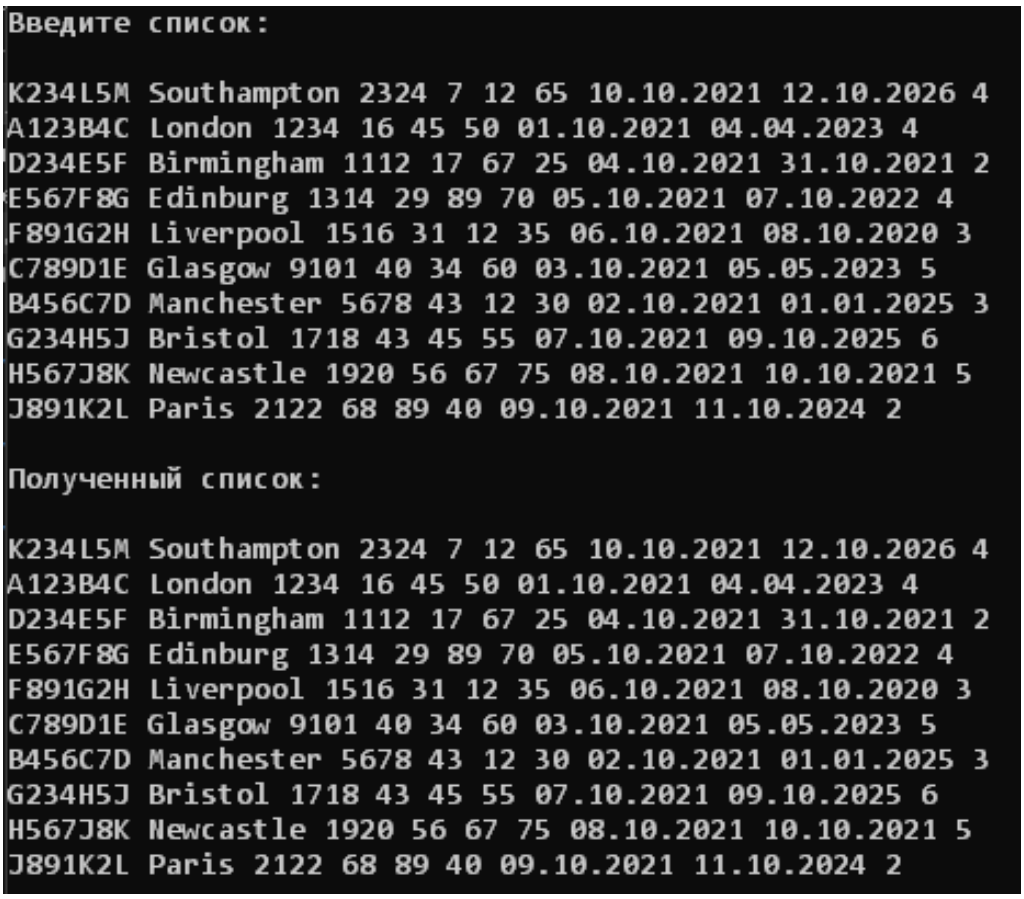

Рисунок 22 – Результат создания списка

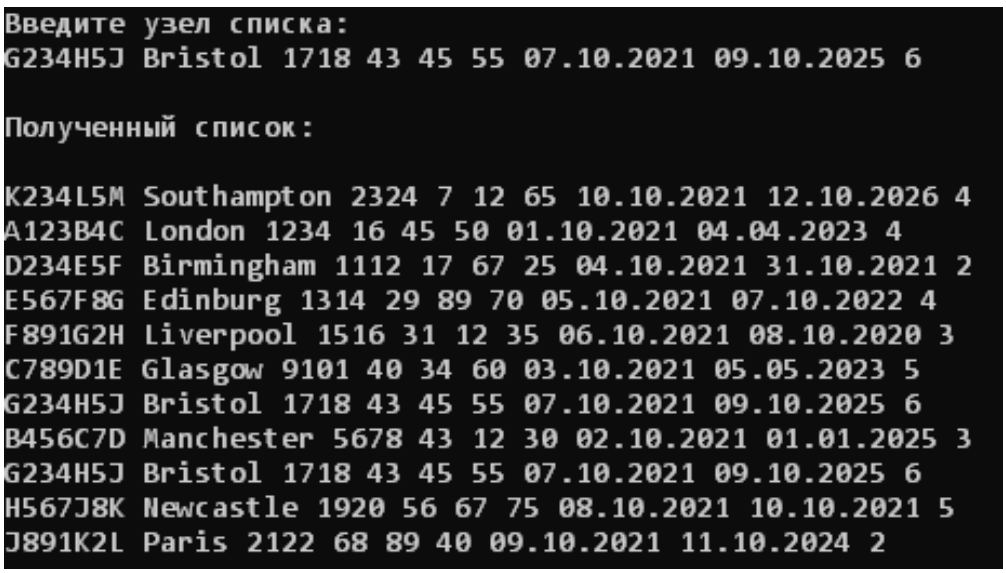

Рисунок 23 – Результат вставки узла в упорядоченный список

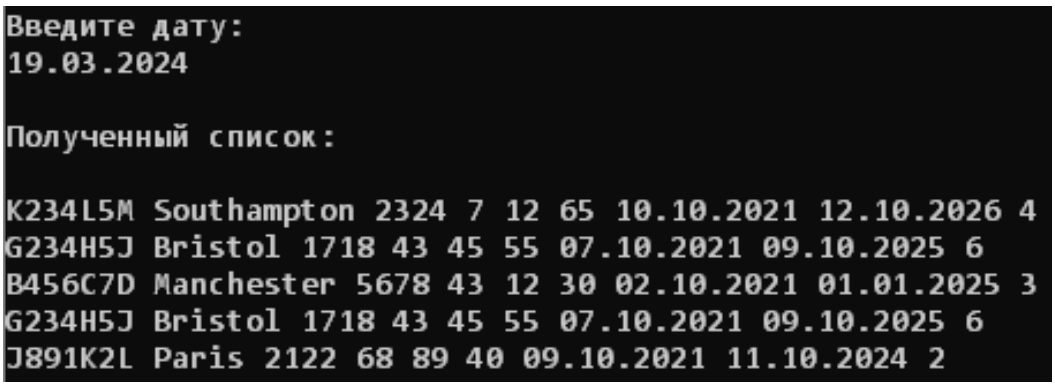

Рисунок 24 – Удаление сведений о билетах добравшихся до места пассажирах

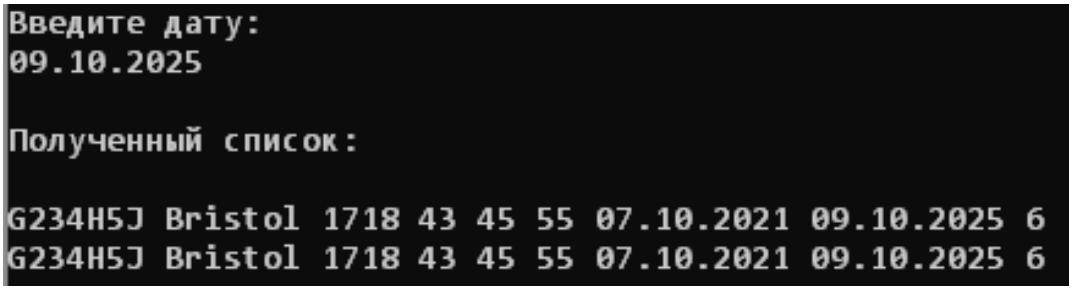

Рисунок 25 – Создание списка по указанной дате отправления

3 ВЫВОД

Изучена такая структура как односвязный список, а также работа с ней. Реализовано несколько функций для работы с этой структурой. Проверена корректность их работы.

СПИСОК ИНФОРМАЦТОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных. Часть 1. Сложность алгоритмов. Сортировки. Линейные структуры данных. Поиск в таблице. : учеб. пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова ; МИРЭА – Российский технологический университет, 2022. – 109 с. – ISBN 978-5-7339-1612-5.

2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01-01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с.