|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

| **Отчет по выполнению практического задания № 6** | |
| --- | --- |
| **Тема:** | |
| **«Двунаправленные динамические списки»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Гендриксон А.А. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ 4](#_30j0zll)

[2.1 Формулировка задачи (В списке №6, Вариант 6) 4](#_1fob9te)

[2.2 Определение списка и описание операций над списком 5](#_3znysh7)

[2.2.1 Определение структуры узла двунаправленного списка 5](#_2et92p0)

[2.2.2. Вставка узла в конец списка. 6](#_3dy6vkm)

[2.2.3 Удаление узла. 7](#_1t3h5sf)

[2.2.4 Вывод элементов списка на экран. 8](#_4d34og8)

[2.2.5 Поиск узла по номеру маршрута 9](#_2s8eyo1)

[2.2.6 Подсчитать количество раз, когда автобус выходил на маршрут за день 10](#_17dp8vu)

[2.2.7 Удалить все узлы с заданным номером шины 10](#_26in1rg)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 12](#_35nkun2)

[2.3.1 Реализация на языке программирования C++ 12](#_1ksv4uv)

[2.3.2 Тестирование 20](#_44sinio)

[2.4 Вывод по заданию 22](#_2jxsxqh)

[3 ВЫВОДЫ 23](#_3j2qqm3)

[4 ЛИТЕРАТУРА 24](#_1y810tw)

# **1 ЦЕЛЬ**

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

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи (В списке №6, Вариант 6)**

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);

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

3. 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.

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

5. Проведите тестирование операций.

• Оцените сложность алгоритма первой дополнительной операции.

6. Оформите отчет по разработке программы в соответствии с требованиями задания по однонаправленному списку.

Примечание: в определении информационной части узла варианта, подчеркнутое поле считать полем ключа.

Индивидуальный вариант. Тип информационной части узла: Номер автобусного маршрута, время отправления (целое число), номер автобуса (формат госномера автотранспорта в России), стоимость одной поездки, дата отправления.

Дополнительные операции:

1. Вставить новый узел после последнего узла с заданным номером автобуса.
2. Удалить все узлы заданного автобуса.
3. Подсчитать, сколько раз автобус выходил на маршрут в течении заданного дня.

## **2.2 Определение списка и описание операций над списком**

### **2.2.1 Определение структуры узла двунаправленного списка**

Структура BusNode представляет собой узел списка, который содержит информацию о конкретном автобусном маршруте. Вот описание каждого поля структуры:

routeNumber: Целочисленное значение, представляющее номер автобусного маршрута.

departureTime: Целочисленное значение, обозначающее время отправления автобуса.

busNumber: Строка, представляющая номер автобуса в формате государственного номера транспортного средства в России.

ticketCost: Дробное число, обозначающее стоимость одной поездки по данному маршруту.

departureDate: Строка, указывающая дату отправления автобуса.

prev: Указатель на предыдущий узел в двунаправленном списке.

next: Указатель на следующий узел в двунаправленном списке.

Данное описание структуры представлено в виде кода на C++ в блоке кода 1, а его изображение на рисунке 1.

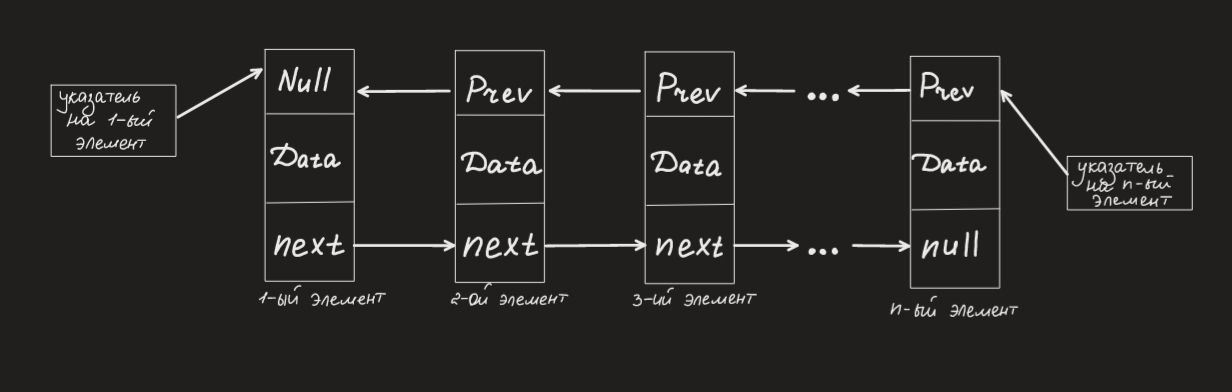


Рисунок 1 - Изображение структуры данных

| struct BusNode {  int routeNumber;  int departureTime;  string busNumber;  double ticketCost;  string departureDate;   BusNode\* prev;  BusNode\* next; }; |
| --- |

Блок кода 1 - Структура

### 2.2.2. Вставка узла в конец списка.

Отобразим выполнение данной операции на рисунке 2.

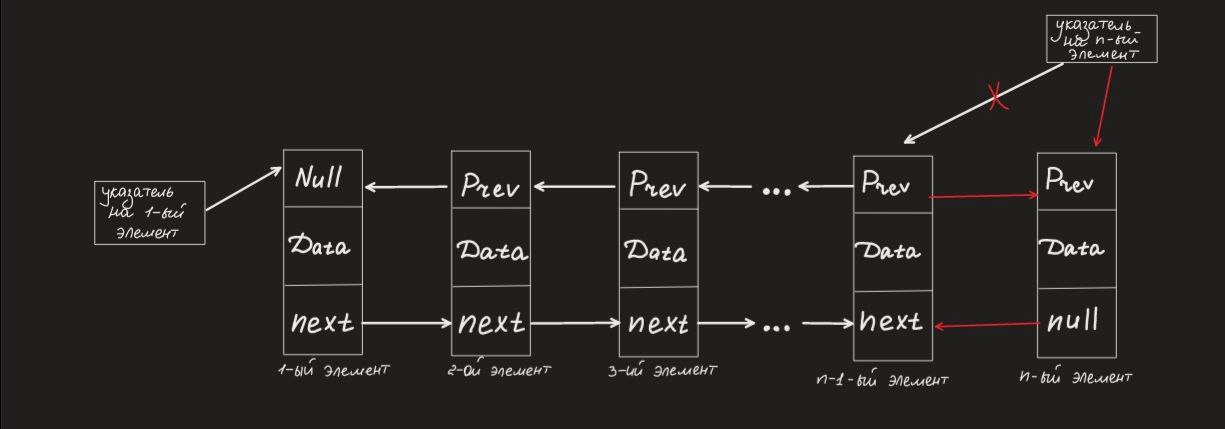


Рисунок 2 - Изображение добавление нового узла

Создается новый объект типа BusNode, который будет вставлен в список. Значения аргументов передаются в поля нового узла newNode. Устанавливаются указатели prev и next нового узла на nullptr, поскольку он будет добавлен в конец списка и не имеет соседних узлов.Если список пуст (head == nullptr), то head и tail указывают на новый узел newNode. Иначе новый узел добавляется в конец списка путем установки указателя next последнего узла на новый узел, а указатель prev нового узла на последний узел, после чего tail обновляется, чтобы указывать на новый узел.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 1.

Таблица 1 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 1, "8:00", "A123BC", 50.0, "2024-05-14"  2, "9:00", "B456DE", 60.0, "2024-05-15"  3, "10:00", "C789FG", 70.0, "2024-05-16"  4, "11:00", "D012HI", 80.0, "2024-05-17"  1, "12:00", "A123BC", 50.0, "2024-05-14" | 3, "8:30", "A126BC", 53.0, "2024-06-14" | 1, "8:00", "A123BC", 50.0, "2024-05-14"  2, "9:00", "B456DE", 60.0, "2024-05-15"  3, "10:00", "C789FG", 70.0, "2024-05-16"  4, "11:00", "D012HI", 80.0, "2024-05-17"  1, "12:00", "A123BC", 50.0, "2024-05-14"  3, "8:30", "A126BC", 53.0, "2024-06-14" |

### 2.2.3 Удаление узла.

Отобразим выполнение данной операции на рисунке 3.

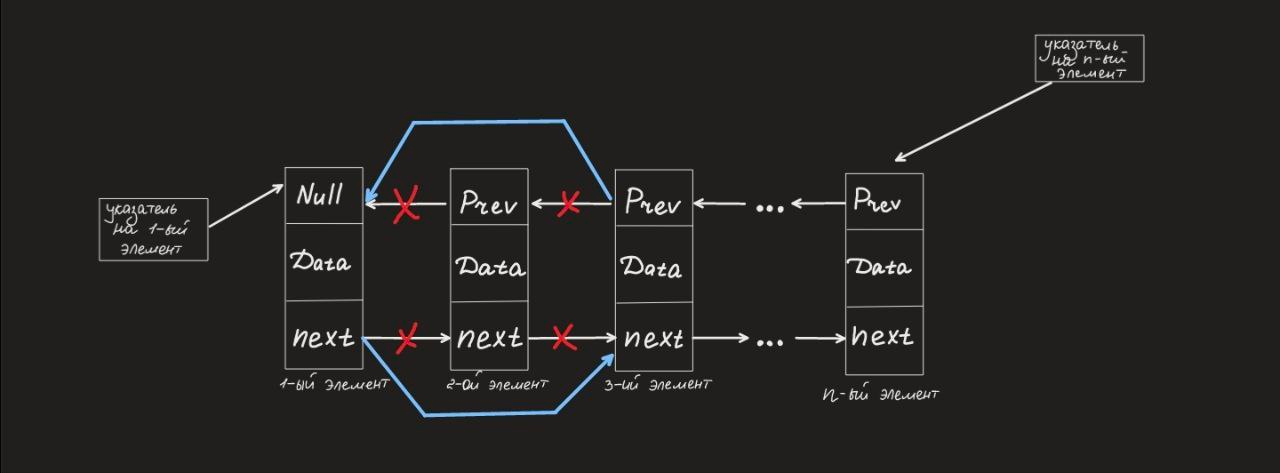


Рисунок 3 - Изображение удаления узла

Это определение метода deleteNode, который принимает номер маршрута в качестве аргумента и удаляет узел с соответствующим номером маршрута из списка. Создается указатель current, который начинает свой обход списка с головы. Начинается цикл обхода списка. Цикл продолжается, пока указатель current не станет равным nullptr, что означает достижение конца списка. Проверяется, соответствует ли номер маршрута текущего узла искомому номеру маршрута для удаления. Проверяется, является ли узел единственным в списке. Если да, то указатели head и tail устанавливаются в nullptr, чтобы очистить список. Проверяется, является ли узел первым в списке. Если да, то указатель head обновляется на следующий узел, а указатель prev следующего узла устанавливается в nullptr. Проверяется, является ли узел последним в списке. Если да, то указатель tail обновляется на предыдущий узел, а указатель next предыдущего узла устанавливается в nullptr. Если узел находится в середине списка, то указатели prev и next предыдущего и следующего узлов обновляются, чтобы пропустить текущий узел. Удаляется текущий узел. Метод завершается, так как узел уже удален. Переходим к следующему узлу в списке для продолжения обхода.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 2.

Таблица 2 - Данные для тестирования

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 1, "8:00", "A123BC", 50.0, "2024-05-14"  2, "9:00", "B456DE", 60.0, "2024-05-15"  3, "10:00", "C789FG", 70.0, "2024-05-16"  4, "11:00", "D012HI", 80.0, "2024-05-17"  1, "12:00", "A123BC", 50.0, "2024-05-14"  3, "8:30", "A126BC", 53.0, "2024-06-14" | Маршрут №2 | 1, "8:00", "A123BC", 50.0, "2024-05-14"  3, "10:00", "C789FG", 70.0, "2024-05-16"  4, "11:00", "D012HI", 80.0, "2024-05-17"  1, "12:00", "A123BC", 50.0, "2024-05-14"  3, "8:30", "A126BC", 53.0, "2024-06-14" |

### 2.2.4 Вывод элементов списка на экран.

Вывод списка слева направо: Создается указатель current, который начинает обход списка с головы. Начинается цикл обхода списка. Цикл продолжается, пока указатель current не станет равным nullptr, что означает достижение конца списка. Выводится информация о текущем узле списка. Переходим к следующему узлу в списке для продолжения обхода.

Вывод списка справа налево: Создается указатель current, который начинает обход списка с хвоста. Начинается цикл обхода списка. Цикл продолжается, пока указатель current не станет равным nullptr, что означает достижение начала списка. Выводится информация о текущем узле списка. Переходим к предыдущему узлу в списке для продолжения обхода в обратном направлении.

Реализация данного алгоритма представлена в блок коде 3.

### 2.2.5 Поиск узла по номеру маршрута

Отобразим выполнение данной операции на рисунке 4.

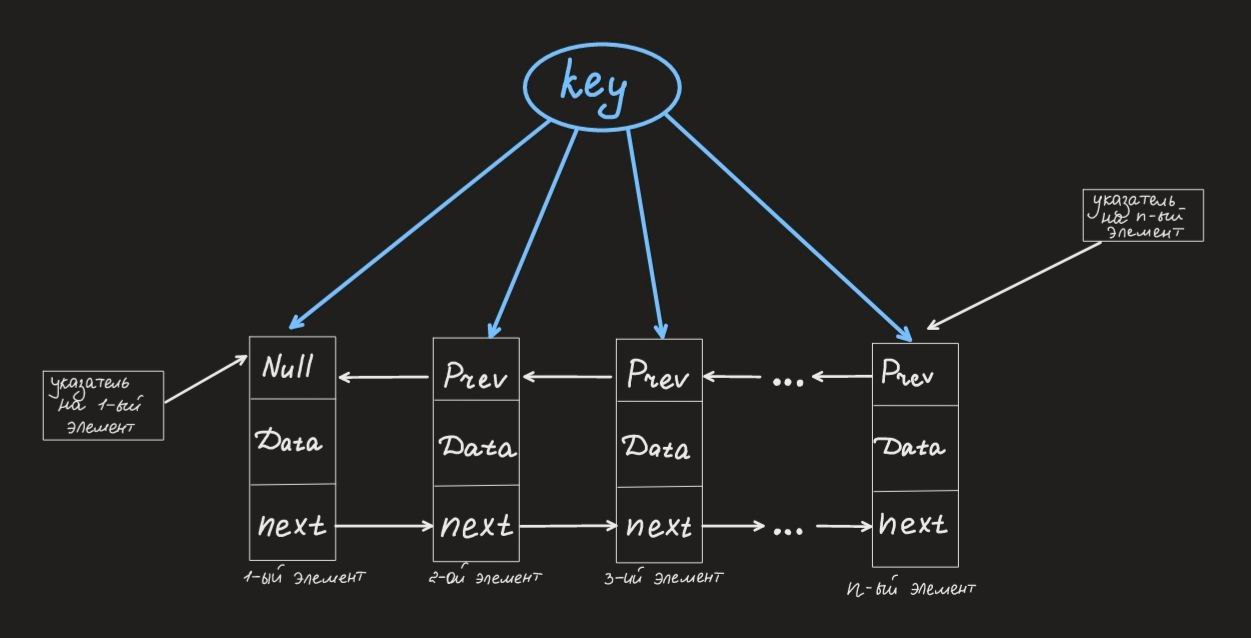


Рисунок 4 - Поиск узла с заданным значением

Это определение метода findNode, который принимает номер маршрута в качестве аргумента и возвращает указатель на узел с соответствующим номером маршрута. Создается указатель current, который начинает свой обход списка с головы. Начинается цикл обхода списка. Цикл продолжается, пока указатель current не станет равным nullptr, что означает достижение конца списка. Проверяется, соответствует ли номер маршрута текущего узла искомому номеру маршрута. Если найден узел с заданным номером маршрута, возвращается указатель на этот узел. Если узел не найден, переходим к следующему узлу в списке для продолжения поиска.Если узел с указанным номером маршрута не найден во всем списке, возвращается nullptr, чтобы указать на отсутствие результатов поиска.

Реализация данного алгоритма представлена в блок коде 3.

### 2.2.6 Подсчитать количество раз, когда автобус выходил на маршрут за день

Создается переменная count, которая будет использоваться для подсчета количества выездов автобуса на маршрут в указанный день. Изначально она устанавливается в 0. Создается указатель current, который начинает обход списка с головы. Начинается цикл обхода списка. Цикл продолжается, пока указатель current не станет равным nullptr, что означает достижение конца списка. Проверяется, соответствует ли номер автобуса и дата отправления текущего узла переданным значениям. Если условие выполняется, то увеличиваем счетчик count на 1. Переходим к следующему узлу в списке для продолжения обхода. По завершении обхода списка возвращается значение переменной count, которое представляет собой количество выездов автобуса на маршрут в указанный день.

Реализация данного алгоритма представлена в блок коде 3.

### 2.2.7 Удалить все узлы с заданным номером шины

Создается указатель current, который начинает обход списка с головы. Начинается цикл обхода списка. Цикл продолжается, пока указатель current не станет равным nullptr, что означает достижение конца списка. Проверяется, соответствует ли номер автобуса текущего узла заданному номеру автобуса. Создается временный указатель temp, который указывает на текущий узел. Это делается для того, чтобы мы могли безопасно удалить текущий узел после обновления связей. Если текущий узел является единственным в списке (current == head && current == tail), то указатели head и tail устанавливаются в nullptr, чтобы очистить список. Если текущий узел является первым в списке (current == head), то указатель head обновляется на следующий узел, а указатель prev следующего узла устанавливается в nullptr. Если текущий узел является последним в списке (current == tail), то указатель tail обновляется на предыдущий узел, а указатель next предыдущего узла устанавливается в nullptr. Если текущий узел находится в середине списка, то указатели prev и next предыдущего и следующего узлов обновляются, чтобы пропустить текущий узел. Переходим к следующему узлу в списке для продолжения обхода. Удаляем временный узел, который содержит указатель на удаляемый узел. В случае, если текущий узел не соответствует заданному номеру автобуса, мы просто переходим к следующему узлу в списке, чтобы продолжить поиск.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 3.

Таблица 3 - Данные для тестирования

| № | Входные данные | Номер шины | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 1, "8:00", "A123BC", 50.0, "2024-05-14"  2, "9:00", "B456DE", 60.0, "2024-05-15"  3, "10:00", "C789FG", 70.0, "2024-05-16"  4, "11:00", "D012HI", 80.0, "2024-05-17"  1, "12:00", "A123BC", 50.0, "2024-05-14"  3, "8:30", "A126BC", 53.0, "2024-06-14" | A123BC | 2, "9:00", "B456DE", 60.0, "2024-05-15"  3, "10:00", "C789FG", 70.0, "2024-05-16"  4, "11:00", "D012HI", 80.0, "2024-05-17"  3, "8:30", "A126BC", 53.0, "2024-06-14" |

## **2.3 Реализация алгоритма на языке C++ и проведение тестирования**

### **2.3.1 Реализация на языке программирования C++**

Реализуем данный алгоритм на языке C++. Для реализации понадобятся такие библиотеки, как iostream, string.

| #include "doubly\_linked\_list.h"  DoublyLinkedList::DoublyLinkedList() : head(nullptr), tail(nullptr) {}  DoublyLinkedList::~DoublyLinkedList() {  while (head) {  BusNode\* temp = head;  head = head->next;  delete temp;  } }  void DoublyLinkedList::insertNode(int routeNumber, const char\* departureTime, const string& busNumber, double ticketCost, const string& departureDate) {  BusNode\* newNode = new BusNode;  newNode->routeNumber = routeNumber;  newNode->departureTime = stoi(departureTime);  newNode->busNumber = busNumber;  newNode->ticketCost = ticketCost;  newNode->departureDate = departureDate;  newNode->prev = nullptr;  newNode->next = nullptr;   if (!head) {  head = tail = newNode;  } else {  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  } }  void DoublyLinkedList::deleteNode(int routeNumber) {  BusNode\* current = head;  while (current) {  if (current->routeNumber == routeNumber) {  if (current == head && current == tail) {  head = tail = nullptr;  } else if (current == head) {  head = head->next;  head->prev = nullptr;  } else if (current == tail) {  tail = tail->prev;  tail->next = nullptr;  } else {  current->prev->next = current->next;  current->next->prev = current->prev;  }  delete current;  return;  }  current = current->next;  } }  void DoublyLinkedList::deleteAllNodesWithBusNumber(const string& busNumber) {  BusNode\* current = head;  while (current) {  if (current->busNumber == busNumber) {  BusNode\* temp = current;  if (current == head && current == tail) {  head = tail = nullptr;  } else if (current == head) {  head = head->next;  head->prev = nullptr;  } else if (current == tail) {  tail = tail->prev;  tail->next = nullptr;  } else {  current->prev->next = current->next;  current->next->prev = current->prev;  }  current = current->next;  delete temp;  } else {  current = current->next;  }  } }  BusNode\* DoublyLinkedList::findNode(int routeNumber) {  BusNode\* current = head;  while (current) {  if (current->routeNumber == routeNumber) {  return current;  }  current = current->next;  }  return nullptr; }  int DoublyLinkedList::countBusRouteOnDay(const string& busNumber, const string& date) {  int count = 0;  BusNode\* current = head;  while (current) {  if (current->busNumber == busNumber && current->departureDate == date) {  count++;  }  current = current->next;  }  return count; }  void DoublyLinkedList::printListLeftToRight() {  BusNode\* current = head;  while (current) {  cout << "Номер маршрута: " << current->routeNumber << ", Время отъезда: " << current->departureTime << ", Номер автобуса: " << current->busNumber << ", Стоимость билета: " << current->ticketCost << ", Дата отъезда: " << current->departureDate << endl;  current = current->next;  } }  void DoublyLinkedList::printListRightToLeft() {  BusNode\* current = tail;  while (current) {  cout << "Номер маршрута: " << current->routeNumber << ", Дата отъезда: " << current->departureTime << ", Номер автобуса: " << current->busNumber << ", Стоимость билета: " << current->ticketCost << ", Дата отъезда: " << current->departureDate << endl;  current = current->prev;  } } |
| --- |

Блок кода 2 – doubly\_linked\_list.h

| #include "doubly\_linked\_list.h"  DoublyLinkedList::DoublyLinkedList() : head(nullptr), tail(nullptr) {}  DoublyLinkedList::~DoublyLinkedList() {  while (head) {  BusNode\* temp = head;  head = head->next;  delete temp;  } }  void DoublyLinkedList::insertNode(int routeNumber, const char\* departureTime, const string& busNumber, double ticketCost, const string& departureDate) {  BusNode\* newNode = new BusNode;  newNode->routeNumber = routeNumber;  newNode->departureTime = stoi(departureTime);  newNode->busNumber = busNumber;  newNode->ticketCost = ticketCost;  newNode->departureDate = departureDate;  newNode->prev = nullptr;  newNode->next = nullptr;   if (!head) {  head = tail = newNode;  } else {  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  } }  void DoublyLinkedList::deleteNode(int routeNumber) {  BusNode\* current = head;  while (current) {  if (current->routeNumber == routeNumber) {  if (current == head && current == tail) {  head = tail = nullptr;  } else if (current == head) {  head = head->next;  head->prev = nullptr;  } else if (current == tail) {  tail = tail->prev;  tail->next = nullptr;  } else {  current->prev->next = current->next;  current->next->prev = current->prev;  }  delete current;  return;  }  current = current->next;  } }  void DoublyLinkedList::deleteAllNodesWithBusNumber(const string& busNumber) {  BusNode\* current = head;  while (current) {  if (current->busNumber == busNumber) {  BusNode\* temp = current;  if (current == head && current == tail) {  head = tail = nullptr;  } else if (current == head) {  head = head->next;  head->prev = nullptr;  } else if (current == tail) {  tail = tail->prev;  tail->next = nullptr;  } else {  current->prev->next = current->next;  current->next->prev = current->prev;  }  current = current->next;  delete temp;  } else {  current = current->next;  }  } }  BusNode\* DoublyLinkedList::findNode(int routeNumber) {  BusNode\* current = head;  while (current) {  if (current->routeNumber == routeNumber) {  return current;  }  current = current->next;  }  return nullptr; }  int DoublyLinkedList::countBusRouteOnDay(const string& busNumber, const string& date) {  int count = 0;  BusNode\* current = head;  while (current) {  if (current->busNumber == busNumber && current->departureDate == date) {  count++;  }  current = current->next;  }  return count; }  void DoublyLinkedList::printListLeftToRight() {  BusNode\* current = head;  while (current) {  cout << "Номер маршрута: " << current->routeNumber << ", Время отъезда: " << current->departureTime << ", Номер автобуса: " << current->busNumber << ", Стоимость билета: " << current->ticketCost << ", Дата отъезда: " << current->departureDate << endl;  current = current->next;  } }  void DoublyLinkedList::printListRightToLeft() {  BusNode\* current = tail;  while (current) {  cout << "Номер маршрута: " << current->routeNumber << ", Дата отъезда: " << current->departureTime << ", Номер автобуса: " << current->busNumber << ", Стоимость билета: " << current->ticketCost << ", Дата отъезда: " << current->departureDate << endl;  current = current->prev;  } } |
| --- |

Блок кода 3 – doubly\_linked\_list.cpp

| #include <iostream> #include "doubly\_linked\_list.h"  int main() {  // Создаем объект класса DoublyLinkedList  DoublyLinkedList busList;   // Вставляем несколько узлов в список  busList.insertNode(1, "8:00", "A123BC", 50.0, "2024-05-14");  busList.insertNode(2, "9:00", "B456DE", 60.0, "2024-05-15");  busList.insertNode(3, "10:00", "C789FG", 70.0, "2024-05-16");  busList.insertNode(4, "11:00", "D012HI", 80.0, "2024-05-17");  busList.insertNode(1, "12:00", "A123BC", 50.0, "2024-05-14");   int choice;  do {  // Выводим меню  cout << "Выберите операцию:" << endl;  cout << "1. Вставить узел после последнего узла" << endl;  cout << "2. Удалить узел по номеру маршрута" << endl;  cout << "3. Найти узел по номеру маршрута" << endl;  cout << "4. Подсчитать количество раз, когда автобус выходил на маршрут за день" << endl;  cout << "5. Удалить все узлы с заданным номером шины" << endl;  cout << "6. Вывести список слева направо" << endl;  cout << "7. Вывести список справо налево" << endl;  cout << "8. Выход" << endl;  cout << "Ваш выбор: ";  cin >> choice;   switch (choice) {  case 1: {  busList.insertNode(3, "8:30", "A126BC", 53.0, "2024-06-14");  break;  }  case 2: {  busList.deleteNode(2);  cout << "Удален узел с маршрутом № 2." << endl;  cout << endl;  cout << "Список после удаления узла с номером маршрута 2:" << endl;  busList.printListLeftToRight();  cout << endl;  break;  }  case 3: {  BusNode\* node = busList.findNode(3);  if (node) {  cout << "Найден узел с маршрутом № 3." << endl;  } else {  cout << "Узел с маршрутом № 3 не найден." << endl;  }  cout << endl;  break;  }  case 4: {  cout << "Сколько раз автобус A123BC отправлялся по маршруту 2024-05-14: " << busList.countBusRouteOnDay("A123BC", "2024-05-14") << endl;  break;  }  case 5: {  busList.deleteAllNodesWithBusNumber("A123BC");  cout << "Удалены все узлы с номером шины A123BC." << endl;  cout << "Список после удаления всех узлов с номером шины A123BC:" << endl;  busList.printListLeftToRight();   break;  }  case 6: {  cout << "Список слева направо:" << endl;  busList.printListLeftToRight();  cout << endl;  break;  }  case 7: {  cout << "Список справо налево:" << endl;  busList.printListRightToLeft();  cout << endl;  break;  }  case 8: {  cout << "Выход из программы." << endl;  break;  }  default:  cout << "Неверный выбор. Пожалуйста, выберите операцию из меню." << endl;  }  } while (choice != 8);   return 0; } |
| --- |

Блок кода 4 – main.cpp

### **2.3.2 Тестирование**

Проведем тестирование основываясь на данные из таблиц 1-3. Результаты тестирования представлены на рисунках 5-8.

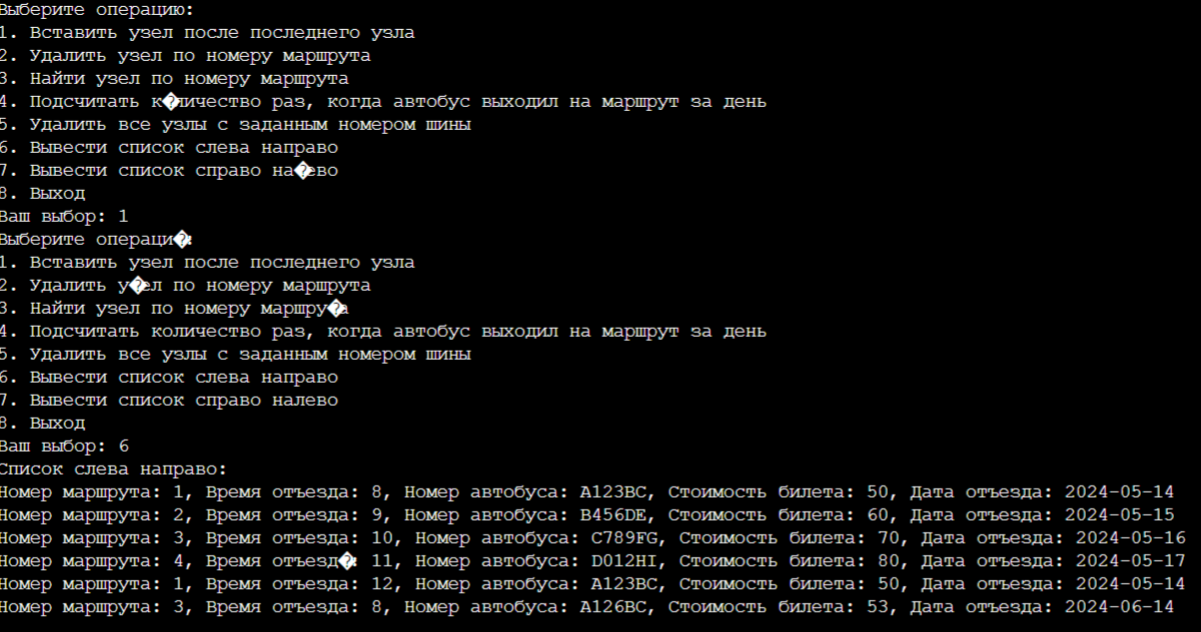


Рисунок 5 - Тестирование программы

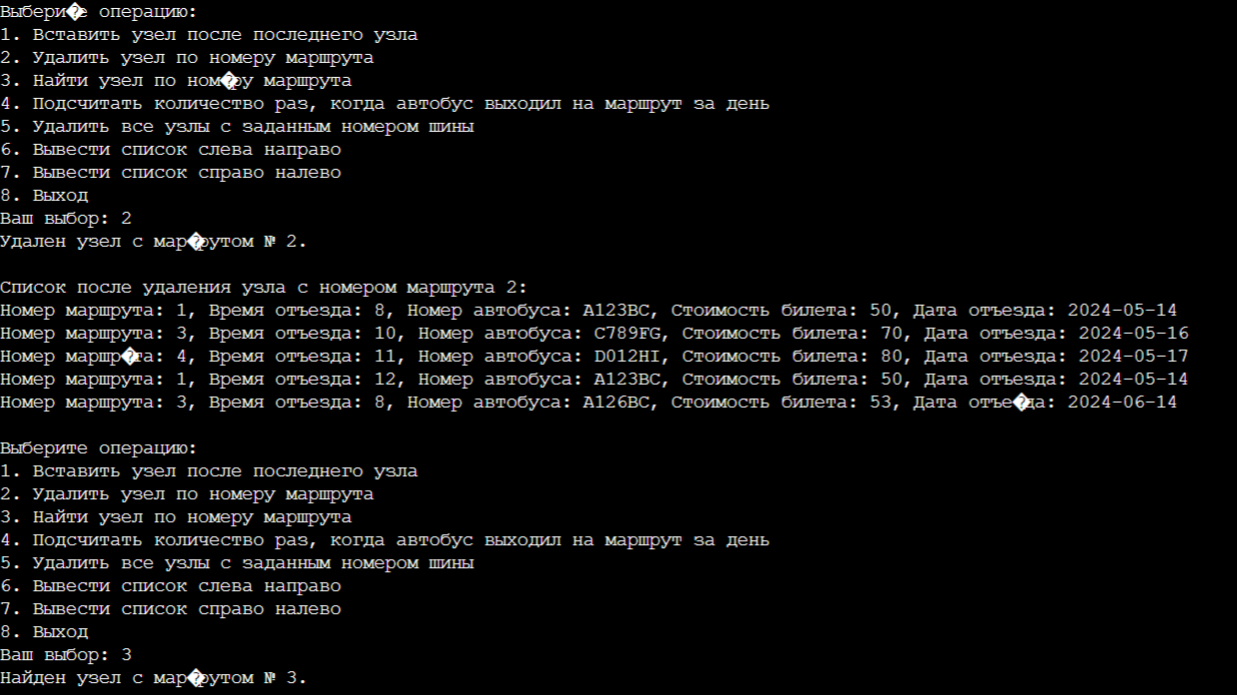


Рисунок 6 - Тестирование программы

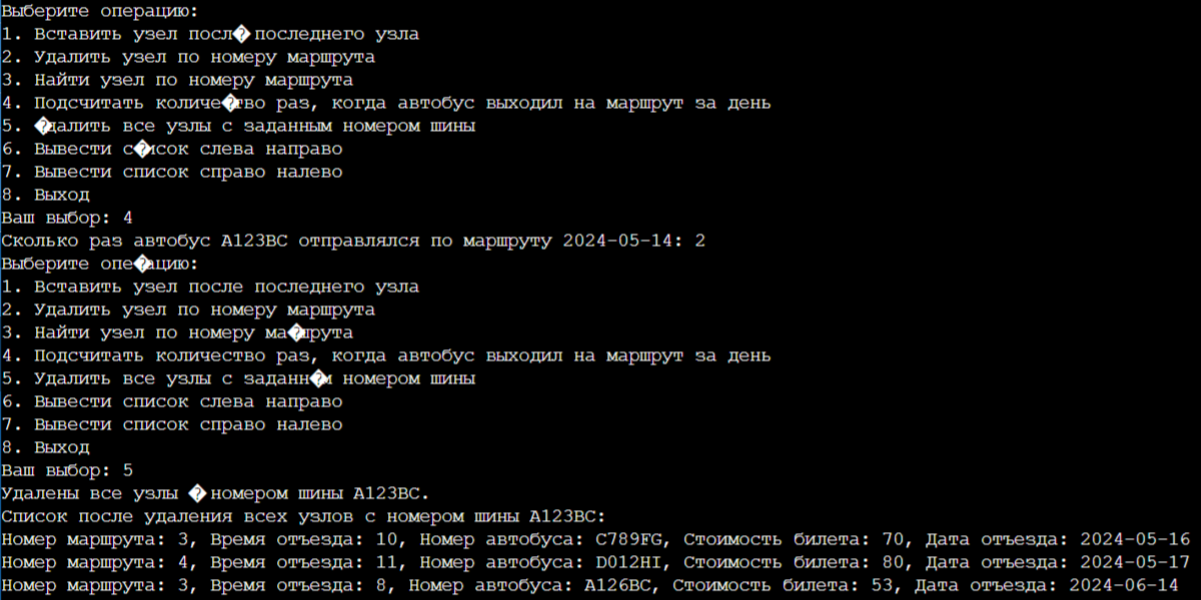


Рисунок 7 - Тестирование программы

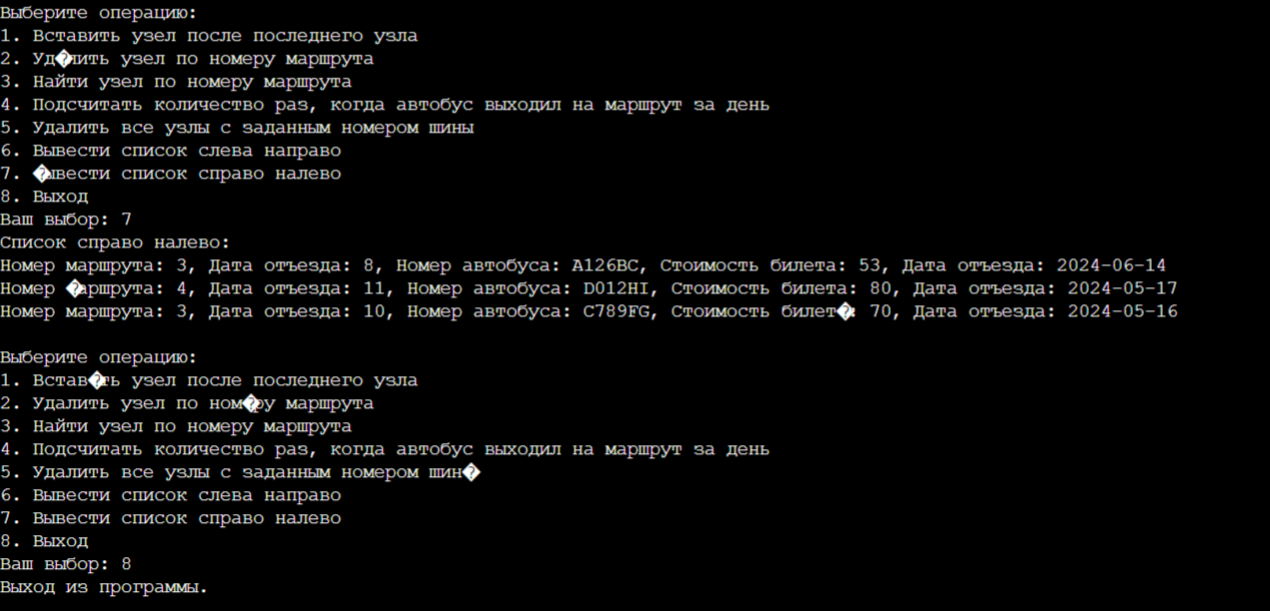


Рисунок 8 - Тестирование программы

Проведение тестирования на основе таблиц 1-3, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

## **2.4 Вывод по заданию**

Двунаправленный список представляет собой структуру данных, в которой каждый элемент содержит не только ссылку на следующий элемент, но и ссылку на предыдущий элемент. Это обеспечивает возможность двунаправленного перемещения по списку, что делает его удобным для различных операций, таких как вставка, удаление и поиск элементов.

Двунаправленные списки обычно обеспечивают эффективность при операциях вставки и удаления элементов, так как они не требуют переупорядочивания всего списка при выполнении этих операций. Это происходит за счет того, что операции вставки и удаления могут быть выполнены за время O(1), если известны узлы, перед и после которых нужно вставить или удалить элемент. Однако, при использовании алгоритма сортировки вставками сложность упорядочивания списка по стране изготовителю становится O(n2) в худшем случае, где n - количество элементов в списке.

# **3 ВЫВОДЫ**

В ходе практической работы были выполнены следующие задачи:

- Получены знания по управления двунаправленным списком в программах на языке С++;

- Получены практические навыки управления двунаправленным списком в программах на языке С++;

- Проведён анализ структуры в соответствии с индивидуальным вариантом;

- Проведён анализ операций в соответствии с индивидуальным вариантом;

- Была реализована программа для структуры в соответствии с индивидуальным вариантом;

- Были реализованы программы для операций в соответствии с индивидуальным вариантом;

- Было реализовано меню выбора способа ввода и реализации функций;

- Было оценена сложность первой дополнительной операции;

- Проведено тестирование программы с различными данными и способами ввода;

Таким образом, главную цель практической работы, а именно получение знаний и практических навыков управления двунаправленным списком в программах на языке С++, можно считать выполненной.

# **4 ЛИТЕРАТУРА**

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.

2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.

3. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.

4. Кораблин Ю.П. Структуры и алгоритмы обработки данных: учебно-методическое пособие / Ю.П. Кораблин, В.П. Сыромятников, Л.А. Скворцова. – М.: РТУ МИРЭА, 2020. — 219 с.

5. Кормен Т.Х. и др. Алгоритмы: построение и анализ, 3-е изд. – М.: ООО «И.Д.Вильямс», 2013. – 1328 с.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., - М.: Техносфера, 2018. – 416 с.

7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

8. Скиена С. Алгоритмы. Руководство по разработке, - 2-е изд. – СПб: БХВ-Петербург, 2011. – 720 с.

9. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд. – СПб: ООО «Альфа-книга», 2017. – 432 с.

10. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL: http://algolist.manual.ru/ (дата обращения 15.03.2022).

11. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 15.03.2022).