|  |
| --- |
| 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: Данюков К.А. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ 4](#_30j0zll)

[2.1 Формулировка задачи (В списке №7, Вариант 7) 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 Процесс выполнения операций 6](#_tyjcwt)

[2.2.2.1 Вставка узла. 7](#_3dy6vkm)

[2.2.2.2 Удаление узла. 8](#_1t3h5sf)

[2.2.2.3 Вывод элементов списка на экран. 11](#_4d34og8)

[2.2.2.4 Поиск узла с заданным значением. 13](#_2s8eyo1)

[2.2.2.5 Отсортировать список, располагая элементы в хронологическом порядке. 14](#_17dp8vu)

[2.2.2.6 Удалить все узлы по заданному товару, проданному в указанную дату. 16](#_3rdcrjn)

[2.2.2.7 Сформировать новый список, из узлов исходного, по тем товарам, по которым имеется возврат. 18](#_26in1rg)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 19](#_lnxbz9)

[2.3.1 Реализация на языке программирования C++ 19](#_35nkun2)

[2.3.2 Тестирование 26](#_1ksv4uv)

[2.4 Вывод по заданию 35](#_44sinio)

[3 ВЫВОДЫ 37](#_z337ya)

[4 ЛИТЕРАТУРА 38](#_3j2qqm3)

# **1 ЦЕЛЬ**

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

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи (В списке №7, Вариант 7)**

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: Код товара (буквенно-цифровой), дата продажи, цена, отметка о возврате.

Дополнительные операции:

Отсортировать список, располагая элементы в хронологическом порядке.

Удалить все узлы по заданному товару, проданному в указанную дату. .

Сформировать новый список, из узлов исходного, по тем товарам, по которым имеется возврат

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

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

Двунаправленный список (или двусвязный список) -это структура данных, включающая узлы с двумя ссылками: на предыдущий и следующий узел. Это позволяет эффективно перемещаться вперед и назад по списку. В отличие от однонаправленного списка, где можно перемещаться только в одном направлении, двусвязный список обеспечивает быстрый доступ к началу и концу списка, а также вставку и удаление элементов за постоянное время. Это делает его полезным для различных задач, где требуется такая гибкость.

Определим структуру узла двунаправленного списка согласно варианту.

Структура Node представляет собой узел двусвязного списка, который содержит информацию о продаже продукта. Каждый узел содержит следующие поля:

1. productCode - строка, содержащая код продукта

2. saleDate - строка, содержащая дату продажи

3. price - число с плавающей точкой, представляющее цену продукта

4. isReturned - логическая переменная, указывающая, был ли продукт возвращен

5. prev - указатель на предыдущий узел в списке

6. next - указатель на следующий узел в списке

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

| struct Node {  string productCode; //код продукта  string saleDate; //дата продажи  double price; //цена  bool isReturned; //возврат  Node\* prev;  Node\* next; }; |
| --- |

Блок кода 1 - Структура

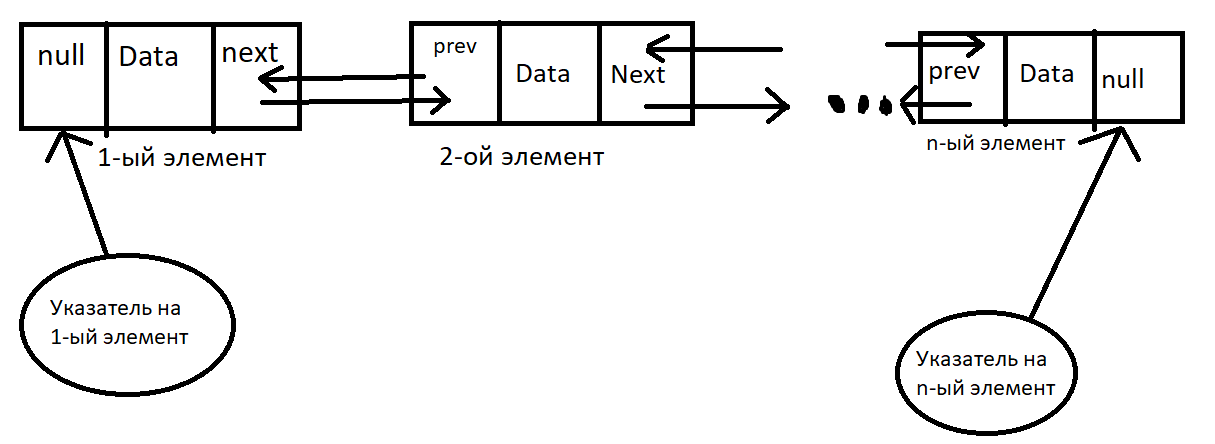


Рисунок 1 - Изображение структуры данных

### **2.2.2 Процесс выполнения операций**

На основании индивидуального варианта, можно утверждать, что применяются следующие операции: вставка 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), отсортировать список, располагая элементы в хронологическом порядке, удалить все узлы по заданному товару, проданному в указанную дату, сформировать новый список, из узлов исходного, по тем товарам, по которым имеется возврат.

Изобразим, рассмотрим алгоритм, реализуем и предоставим таблицу с данными для тестирования данных операций.

#### 2.2.2.1 Вставка узла.

Отобразим выполнение данной операции на рисунке 2.

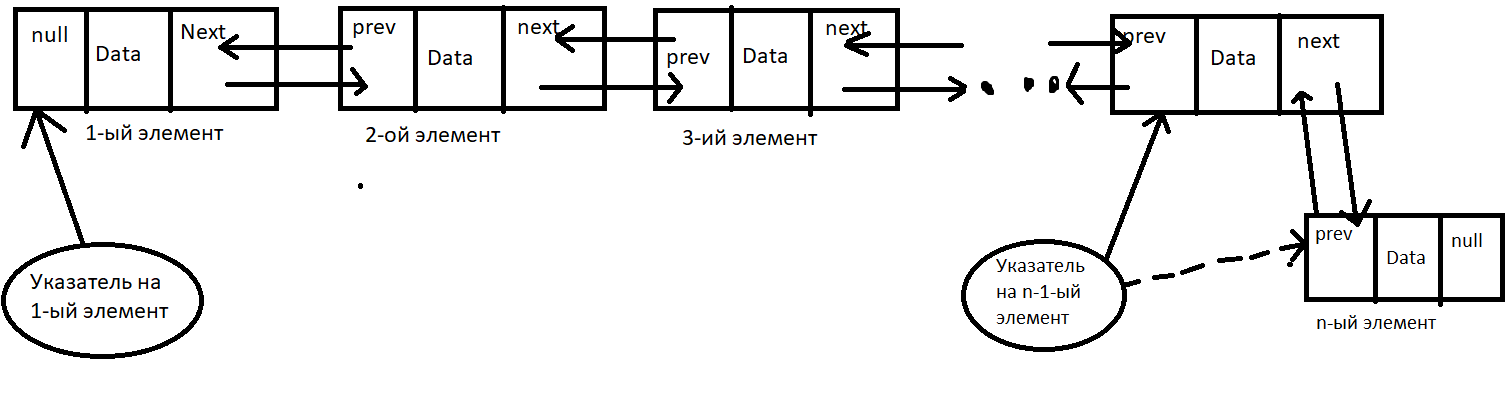


Рисунок 2 - Изображение добавление нового узла

Метод insertNode класса List, который используется для добавления нового узла в двунаправленный список. Он принимает четыре параметра: productCode (код продукта), saleDate (дата продажи), price (цена) и isReturned (флаг возврата). Сначала создается новый узел с переданными значениями и указателями на предыдущий и следующий узлы, которые инициализируются как nullptr. Затем происходит проверка, пуст ли список (head не указывает на ни один узел). Если список пуст, то указатель head присваивается значение нового узла. Если список не пуст, то создается указатель current, который указывает на первый узел списка. Затем происходит перемещение указателя current в конец списка, пока не достигнут последний элемент (у узла, на который указывает current, есть указатель на следующий узел). После этого новый узел добавляется в конец списка, указатель next на последнем узле списка указывает на новый узел, а указатель prev на новом узле указывает на последний узел.

Реализация данного алгоритма представлена в блок коде 2. Данные для тестирования будут приведены в таблице 1.

| *// Создание (добавление) узла void List::insertNode(string productCode, string saleDate, double price, bool isReturned) {  Node\* newNode = new Node{productCode, saleDate, price, isReturned, nullptr, nullptr};  if (!head) {  head = newNode;  } else {  Node\* current = head;  while (current->next) {  current = current->next;  }  current->next = newNode;  newNode->prev = current;  } }* |
| --- |

Блок кода 2 - Добавление нового узла в конце списка

Таблица 1 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0 | 004 2024-05-02 43.75 1 | 001 2023-09-15 50.25 Нет возврата  002 2022-02-22 23.50 Возврат  003 2023-08-21 12.00 Нет возврата  004 2024-05-02 43.75 Возврат |
| 2 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0 | 005 2024-01-23 33.25 0 | 001 2023-09-15 50.25 Нет возврата  002 2022-02-22 23.50 Возврат  003 2023-08-21 12.00 Нет возврата  005 2024-01-23 33.25 Нет возврата |

#### 2.2.2.2 Удаление узла.

Реализуем два вида удаления, так как один вид дан по основному условию задания, второй дан в дополнительных операциях. В данном случае реализуем удаление, данное по основному условию задачи.

Отобразим выполнение данной операции на рисунке 3.

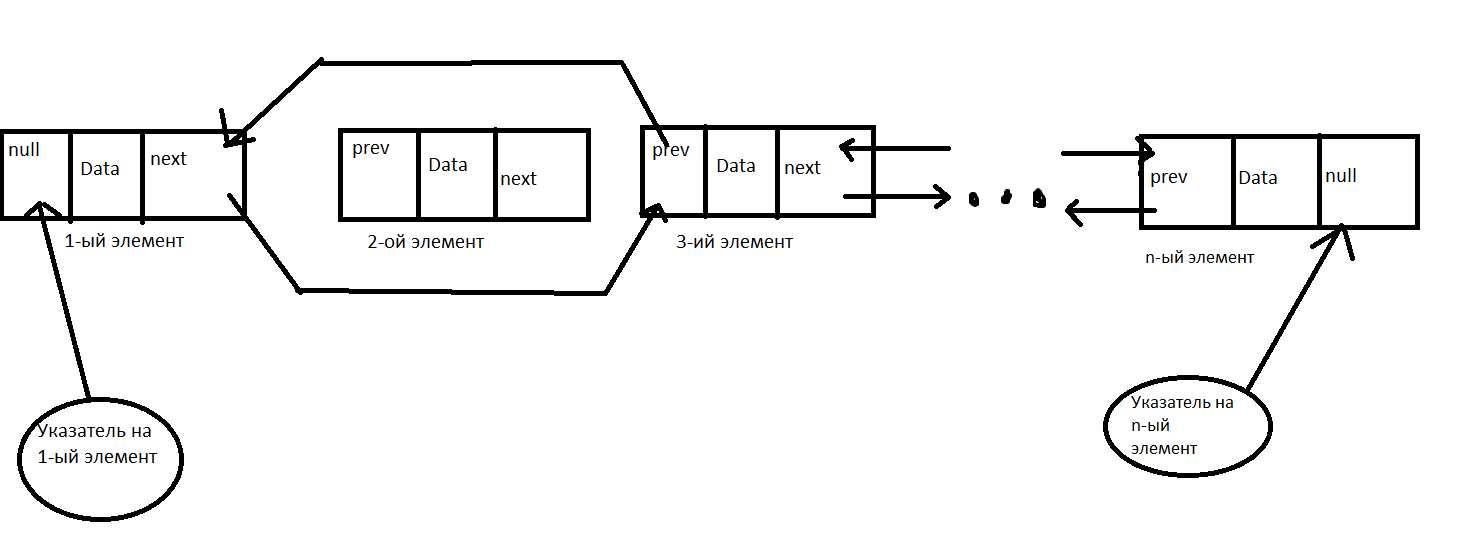


Рисунок 3 - Изображение удаления узла

Метод deleteNode класса List, который используется для удаления узла из двунаправленного списка по заданному productCode (коду продукта).

Сначала создается указатель current, который указывает на начало списка (на первый узел). Затем запускается цикл while, который будет выполняться до тех пор, пока указатель current не станет равен nullptr (концу списка). Внутри цикла происходит проверка, совпадает ли productCode текущего узла с заданным productCode для удаления. Если совпадение найдено, то выполняются следующие действия:

1. Проверяется, есть ли предыдущий узел у текущего. Если есть, то указатель next предыдущего узла присваивается указателю next текущего узла. Это связывает предыдущий узел с следующим узлом после удаляемого.

2. В случае, если удаляемый узел первый в списке (т.е. у него нет предыдущего), то указатель head присваивается указателю next текущего узла. Таким образом корень списка (head) изменяется на следующий узел после удаляемого.

3. Проверяется, есть ли следующий узел после текущего. Если есть, то указатель prev следующего узла присваивается указателю prev текущего узла. Это связывает следующий узел с предыдущим узлом перед удаляемым.

4. Удаляется текущий узел при помощи оператора delete.

5. Возвращение из метода.

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

| *//Удаление узла void List::deleteNode(string productCode) {  Node\* current = head;  while (current) {  if (current->productCode == productCode) {  if (current->prev) {  current->prev->next = current->next;  } else {  head = current->next;  }  if (current->next) {  current->next->prev = current->prev;  }  delete current;  return;  }  current = current->next;  } }* |
| --- |

Блок кода 3 - Удаление узла по ключу

Таблица 2 - Данные для тестирования

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0  004 2024-05-02 43.75 1 | 004 2024-05-02 43.75 1 | 001 2023-09-15 50.25 Нет возврата  002 2022-02-22 23.50 Возврат  003 2023-08-21 12.00 Нет возврата |
| 2 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0  005 2024-01-23 33.25 0 | 005 2024-01-23 33.25 0 | 001 2022-09-15 50.25 Нет возврата  002 2023-05-12 23.50 Возврат  003 2023-08-21 12.00 Нет возврата |

#### 2.2.2.3 Вывод элементов списка на экран.

Вывод элементов списка на экран может быть осуществлен в двух направлениях (вперед и назад). Рассмотрим оба направления.

Вывод элементов списка слева направо. Сначала создается указатель current, который указывает на начало списка (на первый узел). Затем запускается цикл while, который будет выполняться до тех пор, пока указатель current не станет равен nullptr (концу списка). Внутри цикла происходит вывод информации о текущем узле на экран с помощью объекта cout. Каждый узел хранит информацию о productCode (код продукта), saleDate (дата продажи), price (цена продукта) и isReturned (флаг, указывающий на то, был ли продукт возвращен). Выводится строка, содержащая значения этих полей текущего узла, разделенные пробелами. Если флаг isReturned установлен в true, то выводится "Возврат", в противном случае - "Нет возврата". После вывода информации о текущем узле, указатель current переходит к следующему узлу с помощью current = current->next.

Выводим элементы списка справа налево. Сначала создается указатель current, который указывает на начало списка (на первый узел). Затем запускается первый цикл while, который будет выполняться до тех пор, пока указатель current не станет равен nullptr (концу списка) или до тех пор, пока current->next не станет равен nullptr. Внутри этого цикла указатель current перемещается вперед на один узел, пока не достигнет последнего узла. После выполнения первого цикла указатель current указывает на последний узел списка. Далее запускается второй цикл while, который будет выполняться до тех пор, пока указатель current не станет равен nullptr. Внутри цикла выводится информация о текущем узле аналогично методу printListForward: productCode, saleDate, price и информация о возврате продукта. Затем указатель current перемещается к предыдущему узлу с помощью current = current->prev.

Реализация данного алгоритма представлена в блок коде 4. Данные для тестирования слева направо будут приведены в таблице 3, а для справа налево в таблице 4.

| //Вывод вперед void List::printListForward() {  Node\* current = head;  while (current) {  cout << current->productCode << " " << current->saleDate << " " << current->price << " " << (current->isReturned ? "Возврат" : "Нет возврата") << endl;  current = current->next;  } } //Вывод назад void List::printListBackward() {  Node\* current = head;  while (current && current->next) {  current = current->next;  }  while (current) {  cout << current->productCode << " " << current->saleDate << " " << current->price << " " << (current->isReturned ? "Возврат" : "Нет возврата") << endl;  current = current->prev;  } } |
| --- |

Блок кода 4 - Вывод элементов на экран

Таблица 3 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0 | 001 2023-09-15 50.25 Нет возврата  002 2022-02-22 23.50 Возврат  003 2023-08-21 12.00 Нет возврата |

Таблица 4 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0 | 003 2023-08-21 12.00 Нет возврата  002 2022-02-22 23.50 Возврат  001 2023-09-15 50.25 Нет возврата |

#### 2.2.2.4 Поиск узла с заданным значением.

Отобразим выполнение данной операции на рисунке 4.

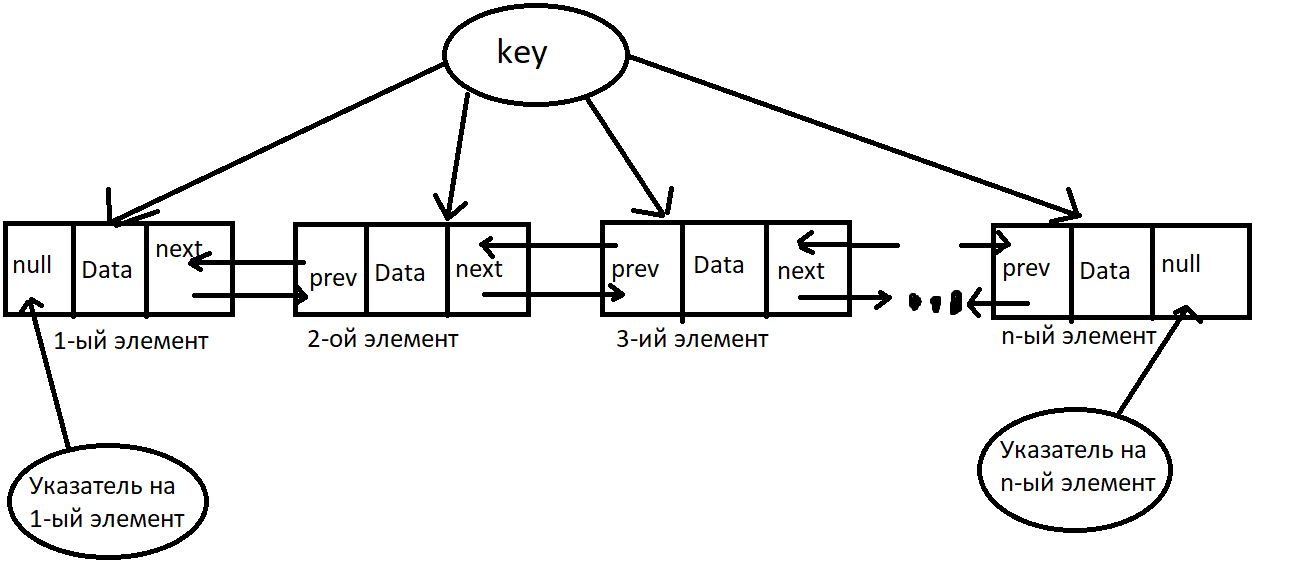


Рисунок 4 - Поиск узла с заданным значением

Сначала создается указатель current, который указывает на начало списка (на первый узел). Затем запускается цикл while, который будет выполняться до тех пор, пока указатель current не станет равен nullptr (концу списка). Внутри цикла происходит проверка, совпадает ли значение productCode текущего узла с заданным значением productCode. Если значения совпадают, то метод возвращает указатель на текущий узел и завершает свое выполнение. Если значение productCode текущего узла не совпадает с заданным значением, указатель current перемещается к следующему узлу с помощью current = current->next. Если метод не найдет узел с заданным значением productCode в списке, то будет возвращено значение nullptr, что означает, что узел с заданным значением не найден в списке.

Реализация данного алгоритма представлена в блок коде 5. Данные для тестирования будут приведены в таблице 6.

| *Node\* List::searchNode(string productCode) {  Node\* current = head;  while (current) {  if (current->productCode == productCode) {  return current;  }  current = current->next;  }  return nullptr; // Узел с заданным значением не найден }* |
| --- |

Блок кода 5 - Поиск узла с заданным значением

#### 2.2.2.5 Отсортировать список, располагая элементы в хронологическом порядке.

Начинаем с внешнего цикла, который будет проходить по всем элементам списка, начиная с первого элемента (head) и двигаясь до конца списка (пока i не станет равным nullptr). Для каждого элемента i во внешнем цикле запускается внутренний цикл, который также проходит по всем элементам списка, начиная с первого элемента (head) и двигаясь до предпоследнего элемента (последний элемент j не имеет смысла сравнивать с его следующим, так что проверяем j->next != nullptr). Внутри внутреннего цикла происходит сравнение даты продажи текущего элемента j с датой продажи следующего элемента j->next. Если дата продажи текущего элемента больше даты продажи следующего элемента, то выполняются операции обмена значений полей у этих двух элементов: productCode, saleDate, price и isReturned. Обмен значений осуществляется с помощью функции swap(), которая меняет местами значения переменных. После прохода внутреннего цикла текущий элемент i во внешнем цикле сдвигается на следующий элемент (i = i->next), и процесс снова повторяется до тех пор, пока весь список не будет отсортирован по возрастанию даты продажи.

Оценим сложность данного алгоритма. Алгоритм сортировки списка по коду продукта, реализованный в функции sortListByDate(), использует сортировку пузырьком. В общем случае, если список содержит n элементов, сложность данного алгоритма будет O(n2).

Это происходит потому, что внешний цикл проходит по всем элементам списка (n раз), а внутренний цикл проходит от первого элемента до предпоследнего элемента (n-1 раз), делая сравнение и при необходимости выполнение операции обмена.

Таким образом, при использовании этого алгоритма сортировки для больших списков с большим количеством элементов, время выполнения может значительно увеличиться.

Реализация данного алгоритма представлена в блок коде 6. Данные для тестирования будут приведены в таблице 5.

| *// Сортировка списка по дате void List::sortListByDate() {  for (Node\* i = head; i != nullptr; i = i->next) {  for (Node\* j = head; j->next != nullptr; j = j->next) {  if (j->saleDate > j->next->saleDate) {  swap(j->productCode, j->next->productCode);  swap(j->saleDate, j->next->saleDate);  swap(j->price, j->next->price);  swap(j->isReturned, j->next->isReturned);  }  }  } }* |
| --- |

Блок кода 6 - Отсортировать список, располагая элементы в хронологическом порядке.

Таблица 5 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0 | 002 2022-02-22 23.50 Возврат  003 2023-08-21 12.00 Нет возврата  001 2023-09-15 50.25 Нет возврата |

#### 2.2.2.6 Удалить все узлы по заданному товару, проданному в указанную дату.

Создается указатель на текущий элемент списка (Node\* current), который инициализируется указателем на начало списка (head). Запускается цикл while, который будет выполняться, пока текущий элемент не достигнет конца списка (current != nullptr). Внутри цикла происходит проверка условий: если код продукта (productCode) и дата продажи (saleDate) текущего элемента совпадают с заданными, то выполняются следующие действия:

- Создается временный указатель temp на текущий элемент.

- Происходит изменение ссылок узлов перед текущим и после текущего элемента, чтобы "перескочить" текущий элемент и удалить его из списка.

- Удаляется узел текущего элемента, используя оператор delete.

- Переход к следующему элементу списка (current = current->next).

Если условия не выполнены (текущий элемент не соответствует заданным критериям), то просто происходит переход к следующему элементу списка (current = current->next).

Реализация данного алгоритма представлена в блок коде 7. Данные для тестирования будут приведены в таблице 6.

| *//Супер удаление по коду и дате void List::deleteNodesByProductAndDate(string productCode, string saleDate) {  Node\* current = head;  while (current) {  if (current->productCode == productCode && current->saleDate == saleDate) {  Node\* temp = current;  if (current->prev) {  current->prev->next = current->next;  } else {  head = current->next;  }  if (current->next) {  current->next->prev = current->prev;  }  current = current->next;  delete temp;  } else {  current = current->next;  }  } }* |
| --- |

Блок кода 7 - Удалить все узлы по заданному товару, проданному в указанную дату.

| № | Входные данные | Удалить | Выходные данные |
| --- | --- | --- | --- |
| 1 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0 | 001  2023-09-15 | 002 2022-02-22 23.50 Возврат  003 2023-08-21 12.00 Нет возврата |

#### 2.2.2.7 Сформировать новый список, из узлов исходного, по тем товарам, по которым имеется возврат.

Создается указатель на новый список returnList, который инициализируется как пустой список, созданный с помощью оператора new. Создается указатель на текущий элемент списка (Node\* current), который инициализируется указателем на начало исходного списка (head). Запускается цикл while, который будет выполняться, пока текущий элемент не достигнет конца списка (current != nullptr). Внутри цикла происходит проверка условия: если текущий элемент отмечен как возвращенный (current->isReturned = true), то происходит следующее:

- Вызывается метод insertNode для вставки нового узла в returnList с данными текущего элемента (productCode, saleDate, price, isReturned).

После обработки текущего элемента, происходит переход к следующему элементу списка (current = current->next). По завершении цикла возвращается указатель на созданный список returnList.

Реализация данного алгоритма представлена в блок коде 8. Данные для тестирования будут приведены в таблице 6.

| *//NEW СПИСОК ВОЗВРАЩЕНОК List\* List::createReturnList() {  List\* returnList = new List();  Node\* current = head;  while (current) {  if (current->isReturned) {  returnList->insertNode(current->productCode, current->saleDate, current->price, current->isReturned);  }  current = current->next;  }  return returnList; }* |
| --- |

Блок кода 8 - Сформировать новый список, из узлов исходного, по тем товарам, по которым имеется возврат.

Таблица 6 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 001 2023-09-15 50.25 0  002 2022-02-22 23.50 1  003 2023-08-21 12.00 0 | 002 2022-02-22 23.50 Возврат |

## **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 и algorithm.

iostream: библиотека в C++, представляющая собой заголовочный файл, содержащий классы, функции и переменные для обеспечения взаимодействия с консолью через стандартные потоки ввода и вывода (cout и cin).

string: класс в языке C++, доступный через заголовочный файл <string>, обеспечивает удобные операции над последовательностью символов. Он позволяет работать с текстовыми данными, создавать, хранить и модифицировать строки.

algoritm: библиотека в С++, которая предоставляет набор стандартных алгоритмов для работы с данными. В неё включены функции для сортировки, поиска, объединения, трансформации и других манипуляций с данными.

В программе также будут задействованы классы (в C++ — это абстракция, описывающая методы и свойства ещё не существующих объектов) и объекты (это конкретное представление абстракции, имеющее свои свойства и методы. Созданные объекты на основе одного класса называются экземплярами этого класса.)

В программе также будут использоваться структура (форма данных, отражающая определенную сущность и имеющую схожие характеристики с классами).

Используем заголовочный файл. Заголовочный файл list.h (блок кода 10) определяет структуру данных двунаправленного списка товаров и класс List, который предоставляет интерфейс для работы с этим списком. Используем файл list.cpp(блок кода 11), который будет содержать определения методов класса List, описанных в заголовочном файле list.h и выше в отчёте. Файл main.cpp(блок кода 12) является точкой входа в программу. В нем содержится функция main(), которая запускается при старте программы. В данном случае, main.cpp используется для демонстрации функциональности класса CarList, который определен в файлах list.h и list.cpp.

| #ifndef LIST\_H #define LIST\_H #include <string> using namespace std;  struct Node {  string productCode; // Код товара  string saleDate; // Дата продажи  double price; // Цена  bool isReturned; // Отметка о возврате  Node\* prev; // Указатель на предыдущий узел  Node\* next; // Указатель на следующий узел };  class List {  public:  List(); // Конструктор  ~List(); // Деструктор    void insertNode(string productCode, string saleDate, double price, bool isReturned);  void deleteNode(string productCode); // Удаление узла  void printListForward(); // Вывод списка вперед  void printListBackward(); // Вывод списка назад  Node\* searchNode(string productCode); // Поиск узла по коду товара    // Дополнительные функции  void sortListByDate(); // Сортировка списка по дате  void deleteNodesByProductAndDate(string productCode, string saleDate); //Супер удаление по коду и дате  List\* createReturnList(); //NEW СПИСОК ВОЗВРАЩЕНОК    private:  Node\* head; // Указатель на начало списка };  #endif |
| --- |

Блок кода 10 – list.h

| #include "List.h" #include <iostream> #include <algorithm>  List::List() : head(nullptr) {}  List::~List() {  Node\* current = head;  while (current) {  Node\* temp = current;  current = current->next;  delete temp;  } }  // Создание (добавление) узла void List::insertNode(string productCode, string saleDate, double price, bool isReturned) {  Node\* newNode = new Node{productCode, saleDate, price, isReturned, nullptr, nullptr};  if (!head) {  head = newNode;  } else {  Node\* current = head;  while (current->next) {  current = current->next;  }  current->next = newNode;  newNode->prev = current;  } } //Удаление узла void List::deleteNode(string productCode) {  Node\* current = head;  while (current) {  if (current->productCode == productCode) {  if (current->prev) {  current->prev->next = current->next;  } else {  head = current->next;  }  if (current->next) {  current->next->prev = current->prev;  }  delete current;  return;  }  current = current->next;  } }  //Вывод вперед void List::printListForward() {  Node\* current = head;  while (current) {  cout << current->productCode << " " << current->saleDate << " " << current->price << " " << (current->isReturned ? "Возврат" : "Нет возврата") << endl;  current = current->next;  } } //Вывод назад void List::printListBackward() {  Node\* current = head;  while (current && current->next) {  current = current->next;  }  while (current) {  cout << current->productCode << " " << current->saleDate << " " << current->price << " " << (current->isReturned ? "Возврат" : "Нет возврата") << endl;  current = current->prev;  } }  Node\* List::searchNode(string productCode) {  Node\* current = head;  while (current) {  if (current->productCode == productCode) {  return current;  }  current = current->next;  }  return nullptr; // Узел с заданным значением не найден }  //Дополнительные функции // Сортировка списка по дате void List::sortListByDate() {  for (Node\* i = head; i != nullptr; i = i->next) {  for (Node\* j = head; j->next != nullptr; j = j->next) {  if (j->saleDate > j->next->saleDate) {  swap(j->productCode, j->next->productCode);  swap(j->saleDate, j->next->saleDate);  swap(j->price, j->next->price);  swap(j->isReturned, j->next->isReturned);  }  }  } } //Супер удаление по коду и дате void List::deleteNodesByProductAndDate(string productCode, string saleDate) {  Node\* current = head;  while (current) {  if (current->productCode == productCode && current->saleDate == saleDate) {  Node\* temp = current;  if (current->prev) {  current->prev->next = current->next;  } else {  head = current->next;  }  if (current->next) {  current->next->prev = current->prev;  }  current = current->next;  delete temp;  } else {  current = current->next;  }  } } //NEW СПИСОК ВОЗВРАЩЕНОК List\* List::createReturnList() {  List\* returnList = new List();  Node\* current = head;  while (current) {  if (current->isReturned) {  returnList->insertNode(current->productCode, current->saleDate, current->price, current->isReturned);  }  current = current->next;  }  return returnList; } |
| --- |

Блок кода 11 – list.cpp

| #include <iostream> #include "List.h"  using namespace std;  int main() {  // Создание объекта  List productList;  // Объявление переменной выбора в меню  int choice;  // Данные списка  string productCode, saleDate;  double price;  bool isReturned;   do {  cout << "Менюшка:" << endl;  cout << "1. Вставить узел";  cout << "\n2. Удалить узел";  cout << "\n3. Вывести список слева направо";  cout << "\n4. Вывести список справа налево";  cout << "\n5. Поиск узла по коду товара";  cout << "\n6. Сортировать список по дате";  cout << "\n7. Удалить узлы по коду товара и дате";  cout << "\n8. Создать новый список из узлов с возвратом";  cout << "\n0. Выход";  cout << "\nВыберите операцию: ";  cin >> choice;   switch(choice) {  case 1:  cout << "Введите код товара: ";  cin >> productCode;  cout << "Введите дату продажи (ГГГГ-ММ-ДД): ";  cin >> saleDate;  cout << "Введите цену: ";  cin >> price;  cout << "Был ли возврат? (1 - да, 0 - нет): ";  cin >> isReturned;  productList.insertNode(productCode, saleDate, price, isReturned);  break;  case 2:  cout << "Введите код товара для удаления: ";  cin >> productCode;  productList.deleteNode(productCode);  break;  case 3:  cout << "Список слева направо:";  productList.printListForward();  break;  case 4:  cout << "Список справа налево:";  productList.printListBackward();  break;  case 5:  cout << "Введите код товара для поиска: ";  cin >> productCode;  if (Node\* foundNode = productList.searchNode(productCode)) {  cout << "Товар найден: " << foundNode->productCode << " " << foundNode->saleDate << " " << foundNode->price << " " << (foundNode->isReturned ? "Возврат" : "Без возврата") << endl;  } else {  cout << "Товар не найден";  }  break;  case 6:  productList.sortListByDate();  cout << "Список отсортирован по дате";  break;  case 7:  cout << "Введите код товара для удаления: ";  cin >> productCode;  cout << "Введите дату продажи для удаления (ГГГГ-ММ-ДД): ";  cin >> saleDate;  productList.deleteNodesByProductAndDate(productCode, saleDate);  break;  case 8:  List\* returnList;  returnList = productList.createReturnList();  cout << "Новый список из узлов с возвратом:";  returnList->printListForward();  delete returnList;  break;  case 0:  cout << "Выход";  break;  default:  cout << "Ошибка";  break;  }  } while(choice != 0);   return 0; } |
| --- |

Блок кода 12 – main.cpp

### **2.3.2 Тестирование**

Проведем тестирование основываясь на данные из таблиц 1-7. Результаты тестирования №1 строки представлены на рисунках 5-10.

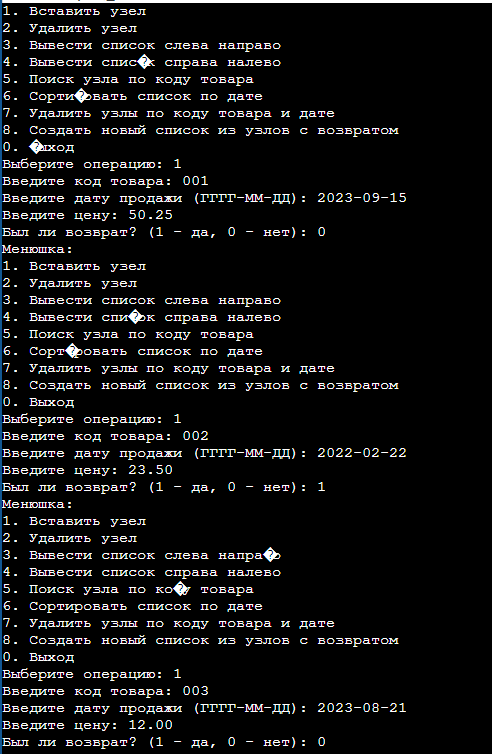


Рисунок 5 - Тестирование программы на данных №1 строки

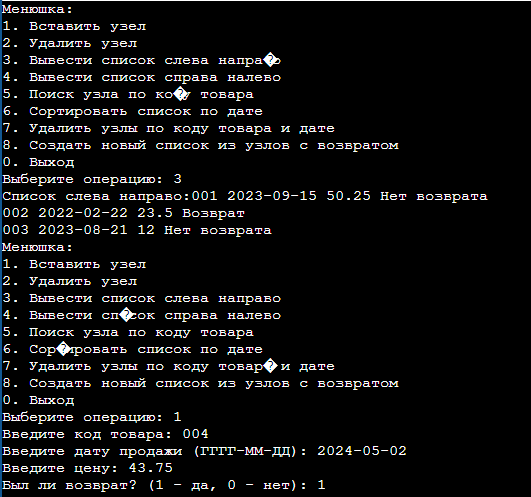


Рисунок 6 - Тестирование программы на данных №1 строки

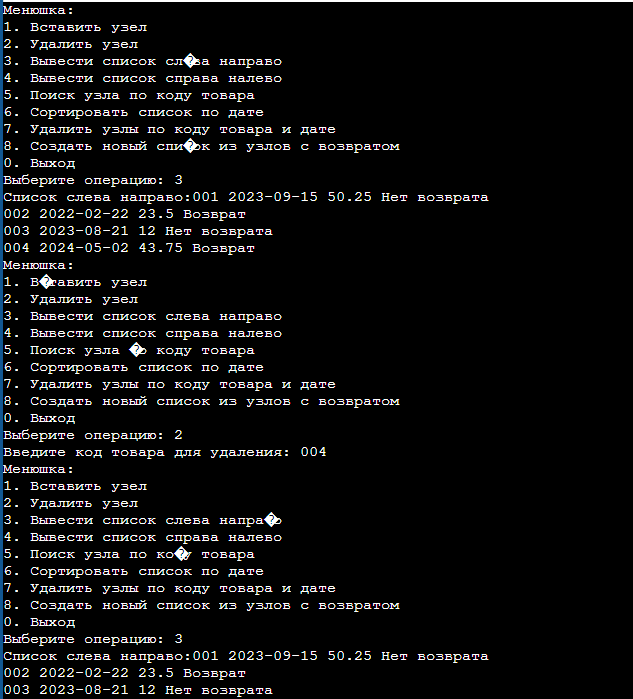


Рисунок 7 - Тестирование программы на данных №1 строки

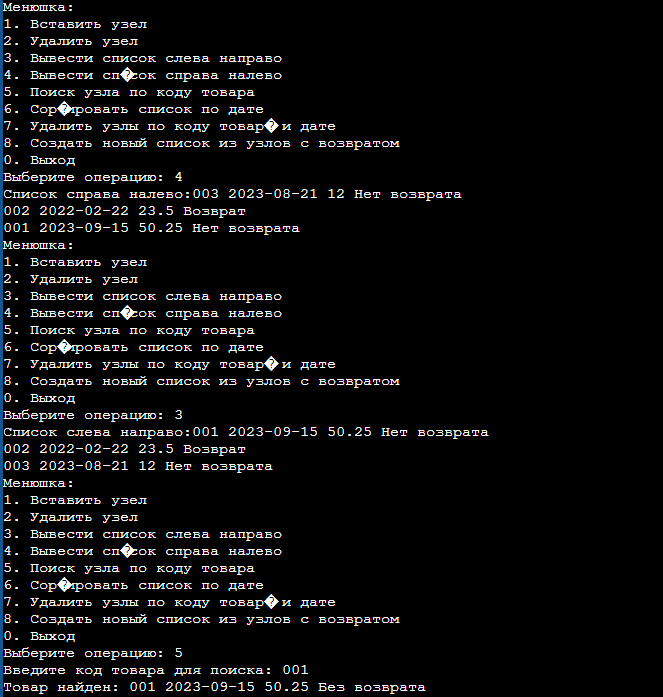


Рисунок 8 - Тестирование программы на данных №1 строки

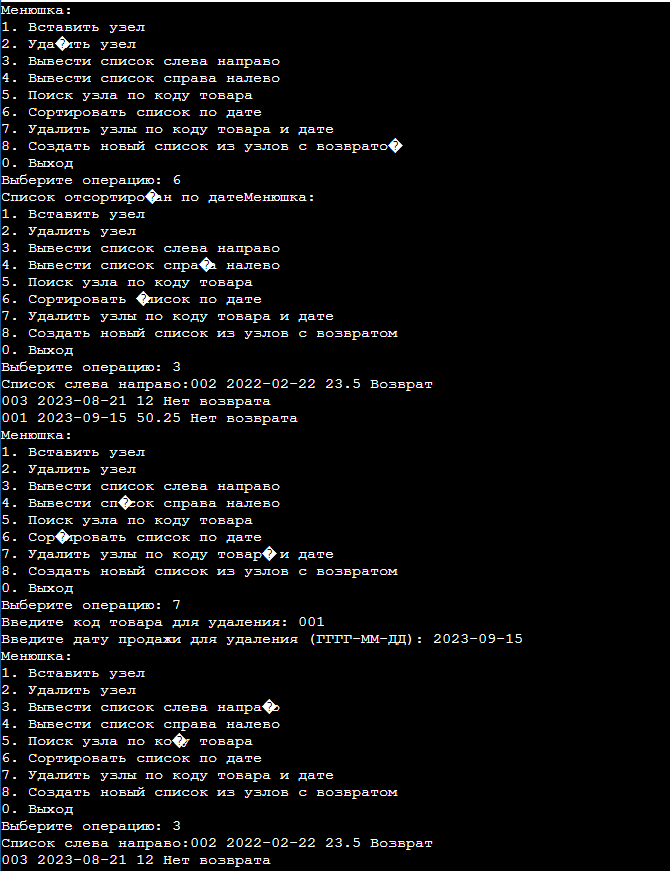


Рисунок 9 - Тестирование программы на данных №1 строки

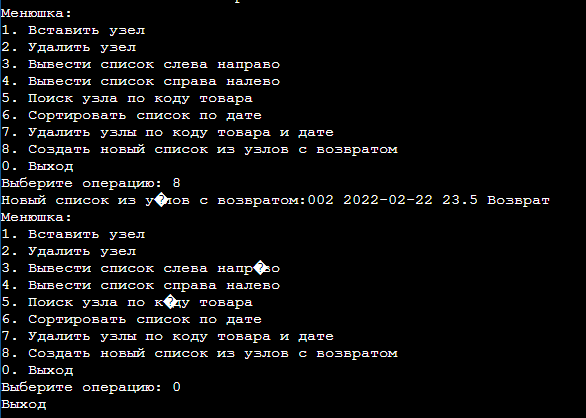


Рисунок 10 - Тестирование программы на данных №1 строки

Проведение тестирования на основе №1 строки таблиц 1-7, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

Результаты тестирования №2 строки представлены на рисунках 11-13.



Рисунок 11 - Тестирование программы на данных №2 строки

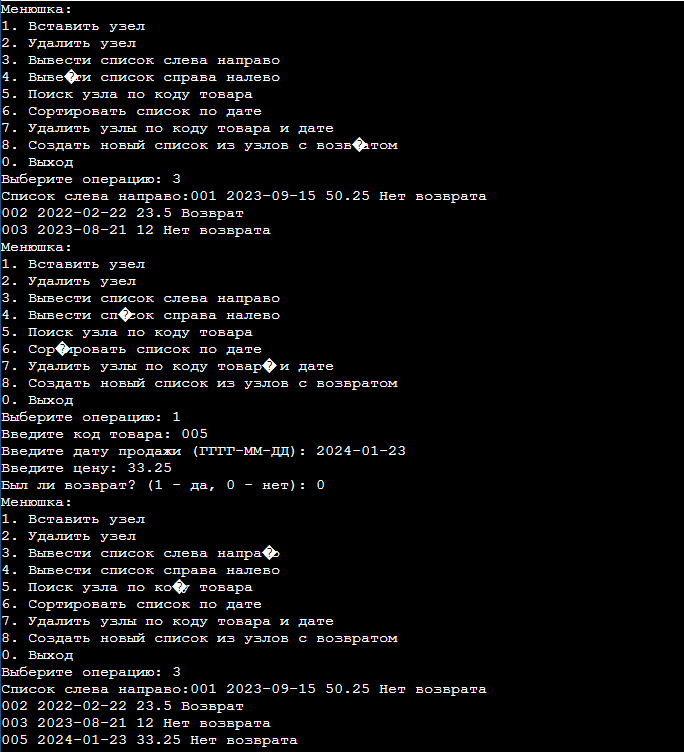


Рисунок 12 - Тестирование программы на данных №2 строки

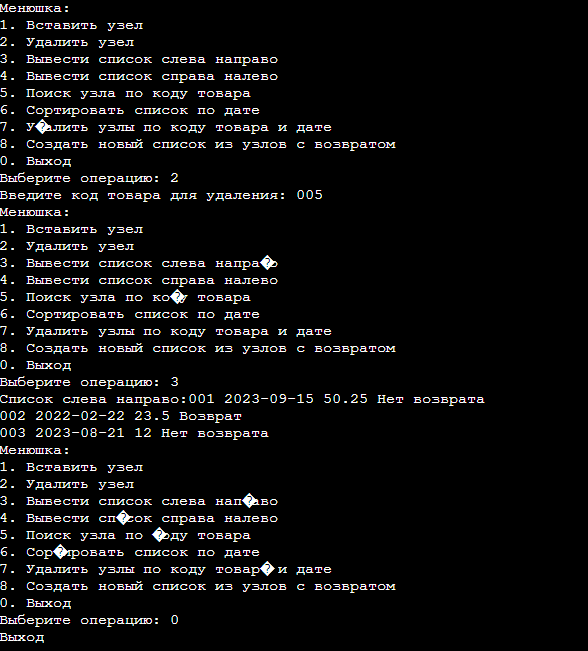


Рисунок 13 - Тестирование программы на данных №2 строки

Проведение тестирования на основе №2 строки таблиц 1-7, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

## **2.4 Вывод по заданию**

Двунаправленный список (или двусвязный список) - это форма данных, включающая в себя элементы, каждый из которых содержит информацию и два указателя: на предыдущий и следующий элементы. Эта форма данных облегчает добавление и удаление элементов в любой части списка, а также дает возможность проходиться по элементам как вперед, так и назад.

Для эффективного использования двунаправленного списка необходимо обновлять указатели при добавлении и удалении элементов, обрабатывать особые ситуации при вставке в начало или конец списка, и использовать итераторы с учетом направления движения по элементам.

Сложность алгоритмов для двунаправленного списка:

Вставка и удаление элемента: O(1), если известны узлы, перед которыми или за которыми нужно вставить или удалить элемент.

Поиск элемента: O(n), где n - количество элементов в списке.

Обращение к элементам: O(n).

Двунаправленный список облегчает выполнение операций вставки, удаления и доступа к элементам как в прямом, так и в обратном порядке. Это особенно полезно, если нужен доступ к элементам в обратном порядке или много операций добавления и удаления из середины списка. Однако, если требуется частый поиск элементов в списке, другие структуры данных, такие как хэш-таблицы или деревья, могут оказаться более эффективными.

# 

# **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++;

- Был проведен анализ организации согласно индивидуальному варианту.

- Был проведён анализ операций в соответствии с индивидуальным вариантом;

- Была выполнена разработка программы для структуры в соответствии с индивидуальным вариантом;

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

- Были разработаны программы для выполнения операций в соответствии с индивидуальным вариантом;

- Была проведена оценка сложности первой дополнительной операции;

- Проведено тестирование программы с различными данными и способами ввода;

Таким образом, достижение цели практической работы, а именно овладение знаниями и умениями управления двунаправленным списком в программировании на языке C++, считается завершенным.

# **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).

12. НОУ ИНТУИТ | Технопарк Mail.ru Group: Алгоритмы и структуры данных [Электронный ресурс]. URL: https://intuit.ru/studies/courses/3496/738/info (дата обращения 15.03.2022).