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

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ 4](#_30j0zll)

[2.1 Формулировка задачи 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. 9](#)

[2.2.2.3 Вывод элементов списка на экран. 12](#_1t3h5sf)

[2.2.2.4 Поиск узла с заданным значением. 14](#_4d34og8)

[2.2.2.5 Упорядочение списка по специальному правилу. 16](#_2s8eyo1)

[2.2.2.6 Переформирование списка. 19](#_17dp8vu)

[2.2.2.7 Удаление товара, который продан до указанной даты. 21](#_3rdcrjn)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 23](#_lnxbz9)

[2.3.1 Реализация на языке программирования C++ 23](#_35nkun2)

[2.3.2 Тестирование 35](#_1ksv4uv)

[2.4 Вывод по заданию 48](#_44sinio)

[3 ВЫВОДЫ 49](#)

[4 ЛИТЕРАТУРА 50](#_z337ya)

# **1 ЦЕЛЬ**

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

# **2 ЗАДАНИЕ**

## **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);

• 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 (по списку №29; вариант 12): код товара (буквенно-цифровой), дата продажи, цена, отметка о возврате, дата возврата.

Дополнительные операции:

1. Упорядочить список по правилу: сначала проданные, но не возвращенные, а затем возвращенные.
2. Переформировать список следующим образом: часть списка, содержащая возвращенный товар перенести в начало исходного списка.
3. Удалить товар, который продан до указанной даты.

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

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

Определим структуру узла двунаправленного списка согласно варианту.

Структура Node представляет собой элемент двунаправленного списка товаров. Она содержит следующие поля: переменную productCode типа string для хранения буквенно-цифрового кода товара, переменную saleDate типа string для хранения даты продажи товара, переменную price типа double для хранения цены товара, переменную returned типа bool, которая указывает на возврат товара (значение true означает, что товар был возвращен, false — не был возвращен), переменную returnDate типа string для хранения даты возврата товара (если товар был возвращен), указатель prev типа Node, указывающий на предыдущий элемент в списке, и указатель next типа Node, указывающий на следующий элемент в списке. Конструктор Node принимает значения всех полей структуры и инициализирует соответствующие переменные. Указателям prev и next при создании элемента присваивается значение nullptr.

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

| struct Node {  string productCode; // Код товара (буквенноцифровой)  string saleDate; // Дата продажи  double price; // Цена  bool returned; // Отметка о возврате  string returnDate; // Дата возврата    Node\* prev;  Node\* next;    // Конструктор по умолчанию  Node() : productCode(""), saleDate(""), price(0.0), returned(false), returnDate(""), prev(nullptr), next(nullptr) {} }; |
| --- |

Блок кода 1 - Структура

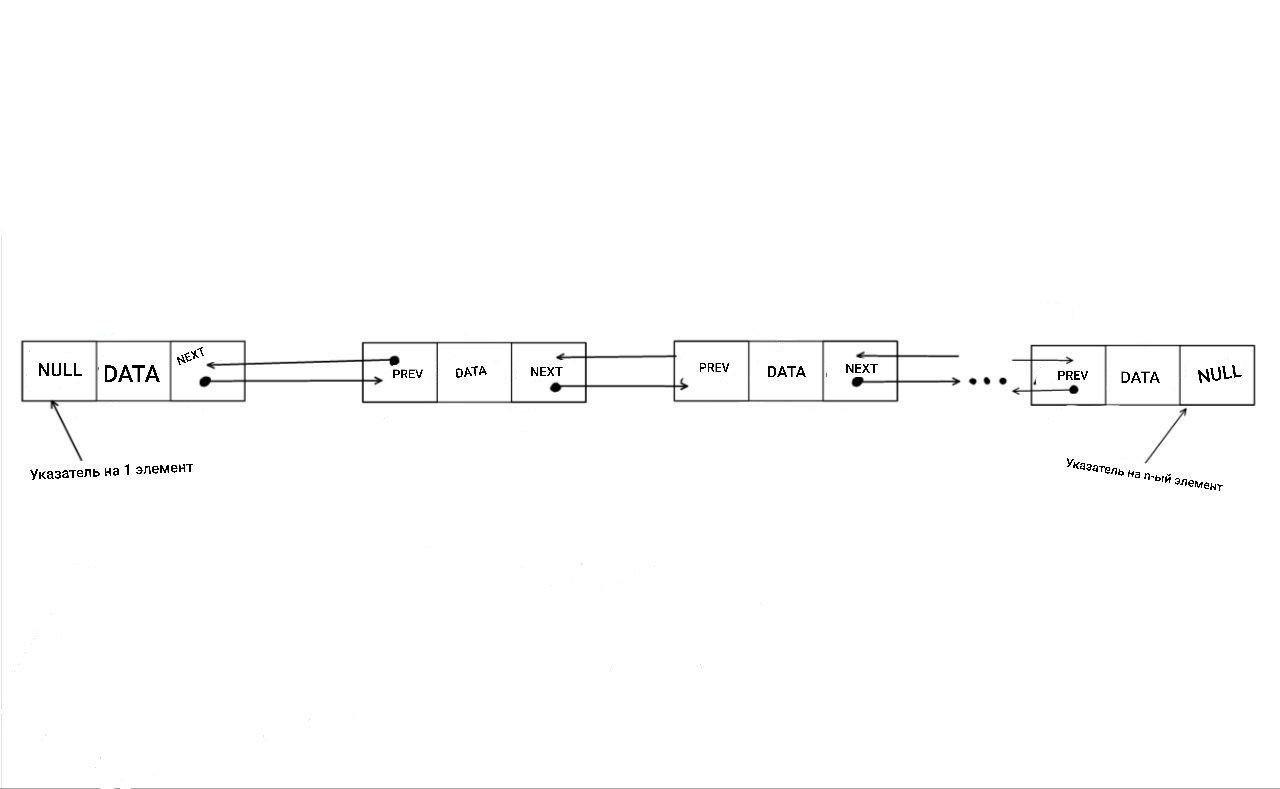


Рисунок 1 - Изображение структуры данных

### **2.2.2 Процесс выполнения операций**

На основании индивидуального варианта можно выделить следующие операции: вставка узла, удаление узла, вывод списка в двух направлениях (слева направо и справа налево), поиск узла с заданным значением (операция возвращает указатель на узел с заданным значением), упорядочение списка по следующему правилу: сначала проданные, но не возвращенные товары, затем возвращенные товары, и переформирование списка следующим образом: часть списка, содержащая возвращенные товары, переносится в начало исходного списка. Также осуществляется удаление товаров, проданных до указанной даты. Рассмотрим алгоритм этих операций, реализуем его и предоставим таблицу с данными для тестирования.

#### 2.2.2.1 Вставка узла.

Отобразим выполнение данной операции на рисунке 2.

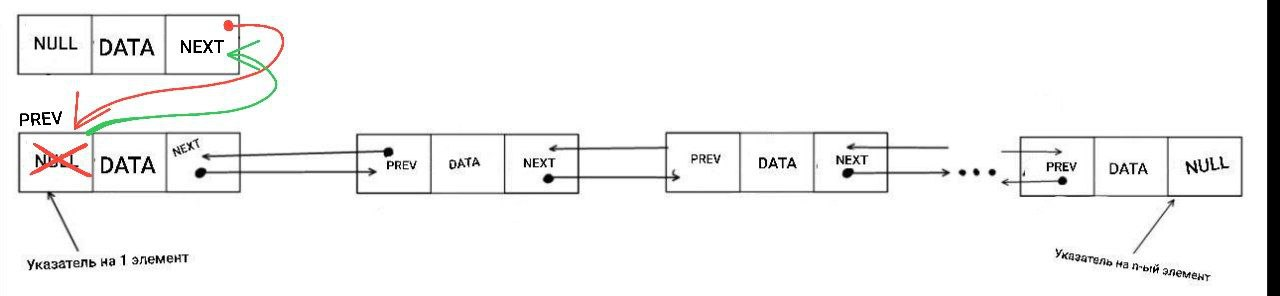


Рисунок 2 - Изображение добавление нового узла

Метод insertNode предназначен для вставки нового узла в двунаправленный список. Сначала создается новый узел newNode с помощью оператора new, в который копируются значения переданных аргументов. Затем проверяется, пуст ли список (проверка через head). Если список пуст (то есть head == nullptr), то новый узел становится единственным элементом списка, и head устанавливается равным newNode, а указатели prev и next нового узла устанавливаются равными nullptr, так как он единственный в списке. Если список не пуст, то новый узел вставляется в начало списка:

* newNode->next устанавливается равным текущему head, чтобы новый узел указывал на предыдущий "первый" элемент списка.
* Указатель prev предыдущего "первого" узла (head->prev) устанавливается равным newNode, чтобы обеспечить связь между новым узлом и предыдущим "первым" узлом.
* head обновляется, чтобы указывать на новый узел, который становится первым элементом списка.
* Указатель prev нового узла устанавливается равным nullptr, так как он теперь первый в списке.

Реализация данного алгоритма представлена в блоке кода 2. Данные для тестирования будут приведены в таблице 1.

| void LinkedList::insertNode(const string& productCode, const string& saleDate, double price, bool returned, const string& returnDate) {  Node\* newNode = new Node;  newNode->productCode = productCode;  newNode->saleDate = saleDate;  newNode->price = price;  newNode->returned = returned;  newNode->returnDate = returnDate;    if (head == nullptr) {  head = newNode;  newNode->prev = nullptr;  newNode->next = nullptr;  }  else {  newNode->next = head;  head->prev = newNode;  head = newNode;  newNode->prev = nullptr;  } } |
| --- |

Блок кода 2 - Добавление нового узла списка

Таблица 1 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date:  Product Code: U-239, Date of Sale: 2015-05-03, Price: 8669.37, Return: 0, Return date: | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date: | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date:  Product Code: U-239, Date of Sale: 2015-05-03, Price: 8669.37, Return: 0, Return date: |
| 2 | Product Code: M-987, Date of Sale: 2021-05-10, Price: 67.89, Return: 1, Return date: 2021-05-15  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05  Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: | Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05 | Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05  Product Code: M-987, Date of Sale: 2021-05-10, Price: 67.89, Return: 1, Return date: 2021-05-15  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05  Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: |

#### 2.2.2.2 Удаление узла.

Отобразим выполнение данной операции на рисунке 3.

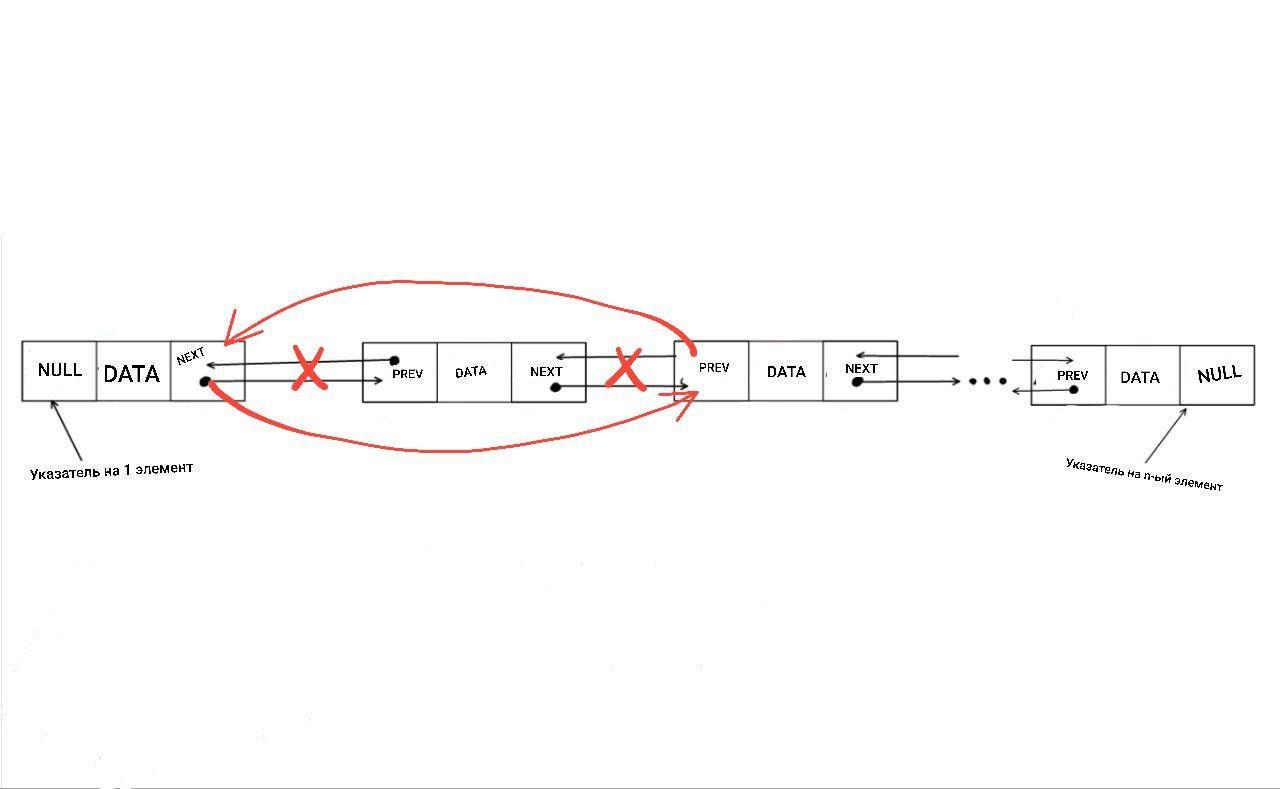


Рисунок 3 - Изображение удаления узла

Метод deleteNode предназначен для удаления узла из двунаправленного списка. Сначала проверяется, передан ли указатель node на узел, который нужно удалить. Если указатель равен nullptr, это означает, что нет узла для удаления, и метод завершает свою работу. Если узел, который нужно удалить (node), имеет предыдущий узел (node->prev != nullptr), то устанавливается связь между предыдущим узлом и следующим узлом после node. В противном случае, если node был первым элементом списка (его предыдущий указатель node->prev равен nullptr), то обновляется указатель head, чтобы он указывал на следующий за node элемент. Затем проверяется, есть ли следующий узел после node (node->next != nullptr). Если такой узел существует, его указатель prev обновляется, чтобы он указывал на предыдущий узел перед node. После обновления связей происходит освобождение памяти, занимаемой удаляемым узлом, с помощью оператора delete.

Реализация данного алгоритма представлена в блоке кода 3. Данные для тестирования будут приведены в таблице 2.

| void LinkedList::deleteNode(Node\* node) {  if (node == nullptr)  return;    if (node->prev != nullptr)  node->prev->next = node->next;  else  head = node->next;    if (node->next != nullptr)  node->next->prev = node->prev;    delete node; } |
| --- |

Блок кода 3 - Удаление узла по ключу

Таблица 2 - Данные для тестирования

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date:  Product Code: U-239, Date of Sale: 2015-05-03, Price: 8669.37, Return: 0, Return date: | U-239 | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date: |
| 2 | Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05  Product Code: M-987, Date of Sale: 2021-05-10, Price: 67.89, Return: 1, Return date: 2021-05-15  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05  Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: | M-987 | Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05  Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: |

#### 2.2.2.3 Вывод элементов списка на экран.

Для вывода элементов списка на экран можно использовать два направления: вперед и назад. Рассмотрим оба.

Для вывода элементов списка слева направо используется метод printForward. В этом методе устанавливается указатель current равным head, начиная с начала списка. Затем в цикле проходится по каждому узлу списка, пока current не станет равным nullptr. На каждой итерации выводятся данные текущего узла, такие как код продукта, дата продажи, цена, отметка о возврате и дата возврата. Указатель current обновляется, чтобы он указывал на следующий узел в списке (т.е. current = current->next), переходя к следующему узлу.

Для вывода элементов списка справа налево используется метод printBackward. В этом методе устанавливается указатель current равным head, начиная с начала списка. Затем в цикле перемещается до последнего узла списка, пока current->next не станет равным nullptr. После выхода из первого цикла current будет указывать на последний узел списка. Затем во втором цикле проходится по каждому узлу списка в обратном порядке, начиная с последнего узла и двигаясь к началу списка. На каждой итерации выводятся данные текущего узла, такие как код продукта, дата продажи, цена, отметка о возврате и дата возврата. Указатель current обновляется, чтобы он указывал на предыдущий узел в списке (т.е. current = current->prev), переходя к предыдущему узлу.

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

| // Вывод списка слева направо void LinkedList::printForward() {  Node\* current = head;  while (current != nullptr) {  cout << "Product Code: " << current->productCode << ", date of sale: " << current->saleDate << ", Price: " << current->price;  cout << ", Return: " << current->returned << ", Return date: " << current->returnDate << endl;  current = current->next;  } }   // Вывод списка справа налево void LinkedList::printBackward() {  Node\* current = head;  while (current->next != nullptr) {  current = current->next;  }    while (current != nullptr) {  cout << "Product Code: " << current->productCode << ", date of sale: " << current->saleDate << ", Price: " << current->price;  cout << ", Return: " << current->returned << ", Return date: " << current->returnDate << endl;  current = current->prev;  } } |
| --- |

Блок кода 4 - Вывод элементов на экран

#### 2.2.2.4 Поиск узла с заданным значением.

Отобразим выполнение данной операции на рисунке 4.

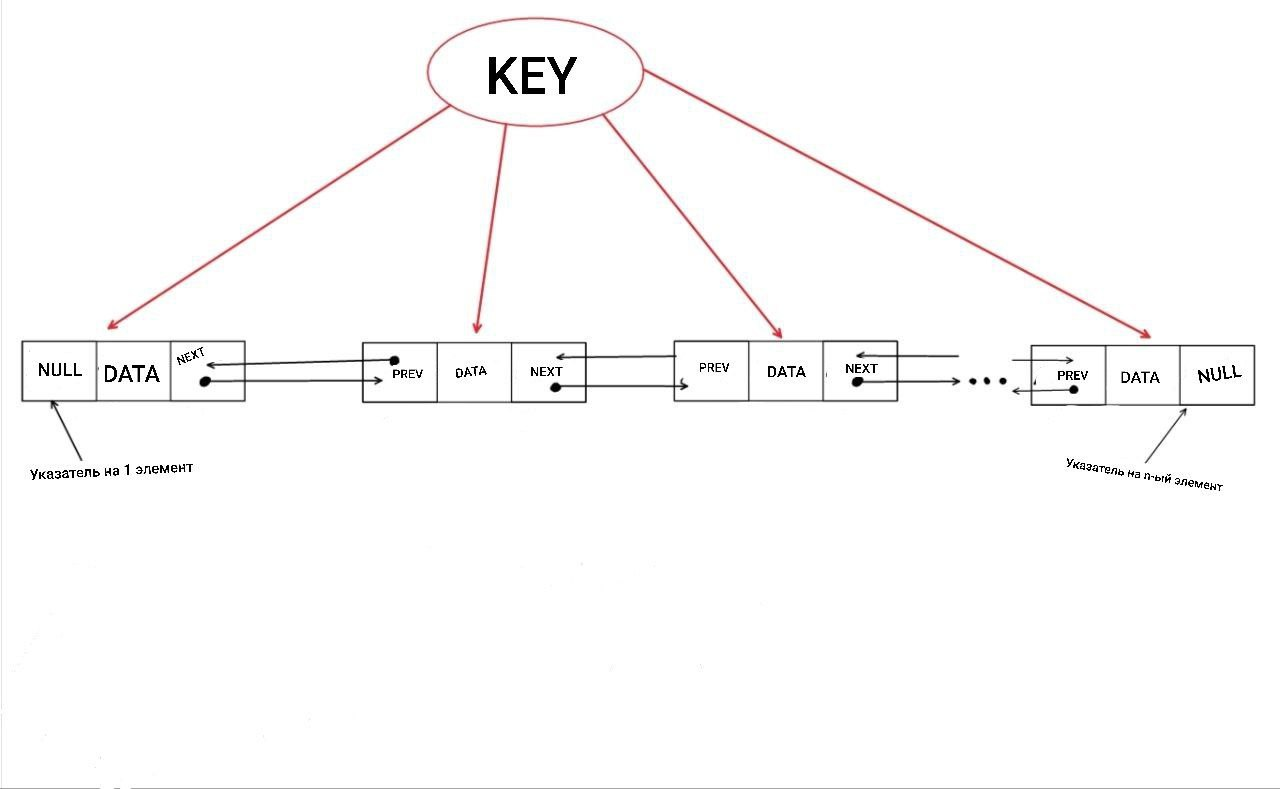


Рисунок 4 - Поиск узла с заданным значением

Метод searchNode предназначен для поиска узла в списке по заданному коду товара. Для этого устанавливается указатель current равным head, начиная с начала списка. Затем в цикле проходится по каждому узлу списка, пока current не станет равным nullptr. На каждой итерации проверяется, равен ли код продукта текущего узла (current->productCode) заданному коду поиска. Если код продукта текущего узла соответствует заданному коду поиска, возвращается указатель на текущий узел (return current). Если узел с заданным кодом не найден после прохождения по всем узлам списка, метод возвращает nullptr, указывая на то, что узел с заданным кодом не был найден в списке.

Реализация данного алгоритма представлена в блок коде 5. Данные для тестирования будут приведены в таблице 3.

| // Поиск узла с заданным кодом товара Node\* LinkedList::searchNode(const string& code) {  Node\* current = head;  while (current != nullptr) {  if (current->productCode == code) {  return current;  }  current = current->next;  }  return nullptr; // Если узел с заданным кодом не найден } |
| --- |

Блок кода 5 - Поиск узла с заданным значением

Таблица 3 - Данные для тестирования

| № | Входные данные | Найти | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date: | F-8960 | The node with the product code F-8960 was not found. |
| 2 | Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05  Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: | T-502 | The node with the product code T-502 found. |

#### 2.2.2.5 Упорядочение списка по специальному правилу.

Метод orderListByRule предназначен для упорядочивания списка заказов в соответствии с правилом: сначала проданные, но не возвращенные товары, затем возвращенные товары.

Для этого метод создает два новых списка: soldList для проданных товаров и returnedList для возвращенных товаров. Затем происходит проход по всем элементам текущего списка в цикле:

Если товар не был возвращен (!current->returned), он добавляется в список soldList.

Если товар был возвращен, он добавляется в список returnedList.

После этого все элементы из текущего списка удаляются в цикле while, чтобы очистить его. Затем элементы из списка returnedList добавляются в текущий список, что позволяет добавить возвращенные товары в начало списка. Затем элементы из списка soldList добавляются в текущий список, добавляя проданные товары после возвращенных.

Сложность данного алгоритма оценивается как O(n), где n — количество элементов в исходном списке. Алгоритм эффективен и обрабатывает список заказов, сохраняя порядок согласно заданному правилу.

Реализация данного алгоритма представлена в блок коде 6. Данные для тестирования будут приведены в таблице 4.

| void LinkedList::orderListByRule() {  // Сначала создадим два списка - один для проданных товаров, другой для возвращенных  LinkedList soldList;  LinkedList returnedList;    // Переберем все элементы в текущем списке  Node\* current = head;  while (current != nullptr) {  // Если товар продан, но не возвращен, добавляем его в список проданных  if (!current->returned) {  soldList.insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  }  // Если товар возвращен, добавляем его в список возвращенных  else {  returnedList.insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  }  current = current->next;  }    // Удалим все элементы из текущего списка  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }    // Добавим элементы из списка возвращенных  current = returnedList.head;  while (current != nullptr) {  insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  current = current->next;  }    // Добавим элементы из списка проданных  current = soldList.head;  while (current != nullptr) {  insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  current = current->next;  } } |
| --- |

Блок кода 6 - Упорядочение списка по специальному правилу

Таблица 4 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date: | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date:  Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01 |
| 2 | Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05  Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: | Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date:  Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05 |

#### 2.2.2.6 Переформирование списка.

Метод restructureList предназначен для переформирования списка заказов, перемещая часть списка, содержащую возвращенный товар, в начало исходного списка.

Проход по всем элементам списка начинается в цикле для поиска последнего проданного товара (lastSoldNode). Если в списке нет возвращенных товаров, то lastSoldNode останется равным nullptr.

После нахождения последнего проданного товара проверяется, не является ли он начальным элементом списка. Если это так, то список уже находится в нужном порядке, и метод завершает свою работу.

Если последний проданный товар не является начальным элементом списка, то сохраняется указатель на начало исходного списка (originalHead). Затем происходит удаление всех элементов списка, начиная с элемента после последнего проданного товара.

После удаления элементов список перестраивается таким образом, чтобы удаленные элементы были вставлены в начало списка, сохраняя порядок. Обновляется указатель head, чтобы он указывал на новое начало списка, которое сформировано из удаленных элементов.

Реализация данного алгоритма представлена в блоке кода 7. Данные для тестирования будут приведены в таблице 5.

| void LinkedList::restructureList() {  Node\* current = head;  Node\* lastSoldNode = nullptr;    // Найдем последний проданный товар  while (current != nullptr) {  if (!current->returned) {  lastSoldNode = current;  }  current = current->next;  }    // Если нет возвращенных товаров или список пуст, ничего не делаем  if (lastSoldNode == nullptr || lastSoldNode == head) {  return;  }    // Сохраняем указатель на начало списка  Node\* originalHead = head;    // Удаляем все проданные товары после последнего возвращенного из текущего списка  Node\* temp = lastSoldNode->next;  while (temp != nullptr) {  Node\* nextNode = temp->next;  deleteNode(temp);  temp = nextNode;  }    // Теперь вставим удаленные элементы в начало списка  current = lastSoldNode->next;  while (current != nullptr) {  Node\* nextNode = current->next;  current->next = originalHead;  current->prev = nullptr;  originalHead->prev = current;  originalHead = current;  current = nextNode;  }  // Обновляем указатель head  head = originalHead; } |
| --- |

Блок кода 7 - Переформирование списка

Таблица 5 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date:  Product Code: IO-35, Date of Sale: 1985-12-06, Price: 20.56, Return: 1, Return date: 1987-09-07  Product Code: A-78, Date of Sale: 2000-08-29, Price: 654.25, Return: 1, Return date: 2000-09-01 | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date: |
| 2 | Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date:  Product Code: X-001, Date of Sale: 2024-04-29, Price: 999.99, Return: 1, Return date: 2026-12-05  Product Code: L-123, Date of Sale: 2022-07-01, Price: 789, Return: 1, Return date: 2022-07-05 | Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: |

#### 2.2.2.7 Удаление товара, который продан до указанной даты.

Отобразим выполнение данной операции на рисунке 7.

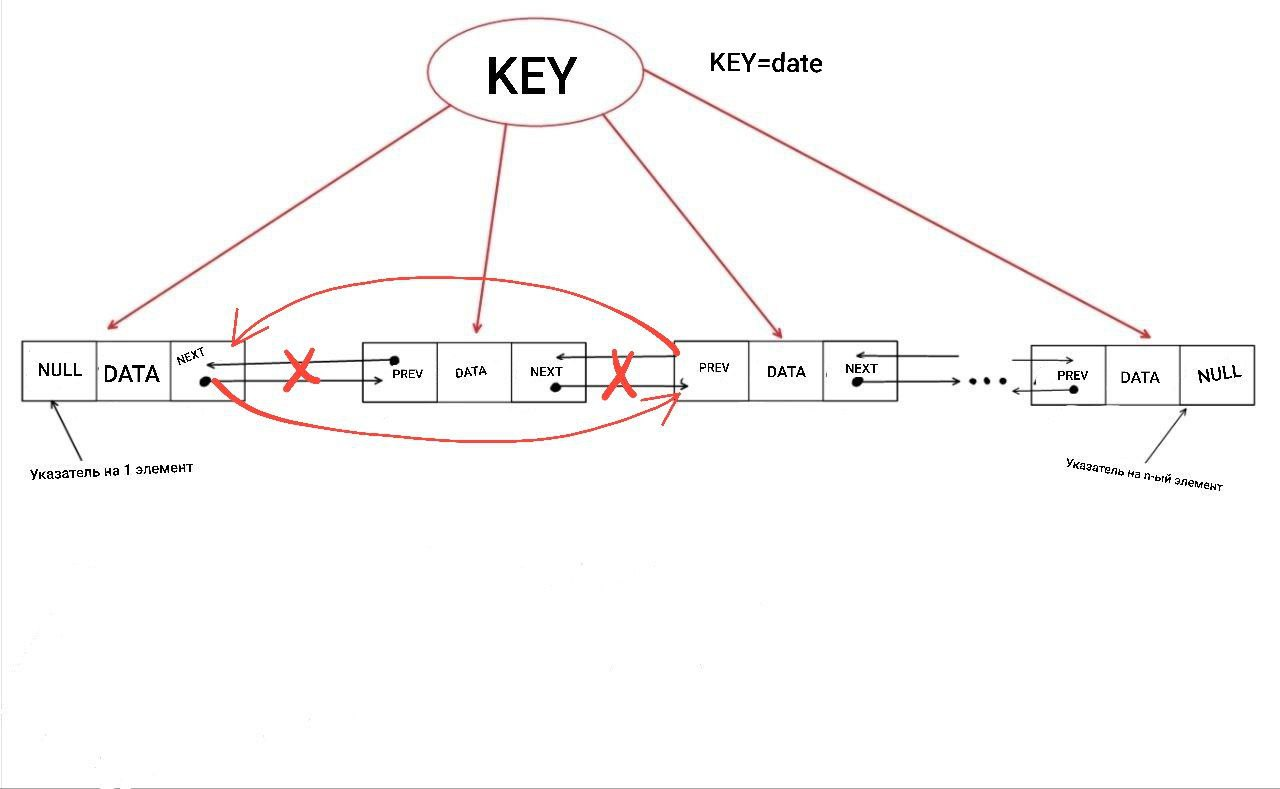


Рисунок 7 - Удаление товара, который продан до указанной даты

Метод deleteItemsBeforeDate предназначен для удаления товаров из списка, которые были проданы до указанной даты.

Проход по всем элементам списка начинается в цикле. Для каждого элемента списка проверяется, была ли дата продажи (значение current->saleDate) перед указанной датой (date).

Если дата продажи элемента меньше указанной даты, то этот элемент удаляется из списка с помощью вызова метода deleteNode. После удаления элемента продолжается проход по списку.

Если дата продажи элемента не меньше указанной даты, то происходит переход к следующему элементу списка.

Процесс продолжается до тех пор, пока не будут проверены все элементы списка.

Реализация данного алгоритма представлена в блоке кода 8. Данные для тестирования будут приведены в таблице 6.

| void LinkedList::deleteItemsBeforeDate(const string& date) {  Node\* current = head;  while (current != nullptr) {  if (current->saleDate < date) {  Node\* temp = current;  current = current->next;  deleteNode(temp);  }  else {  current = current->next;  }  } } |
| --- |

Блок кода 8 - Удаление товара, который продан до указанной дата

Таблица 6 - Данные для тестирования

| № | Входные данные | Дата | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | Product Code: FA-90, Date of Sale: 1990-11-15, Price: 753.15, Return: 0, Return date:  Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date: | 1991-11-15 | Product Code: F-89, Date of Sale: 2012-04-10, Price: 65.02, Return: 0, Return date: |
| 2 | Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date:  Product Code: R-740, Date of Sale: 2023-11-15, Price: 399.99, Return: 0, Return date: | 2023-11-16 | Product Code: T-502, Date of Sale: 2024-02-20, Price: 123.45, Return: 0, Return date: |

## **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.

| #ifndef LINKEDLIST\_H #define LINKEDLIST\_H   #include <string>   using namespace std;    struct Node {  string productCode; // Код товара (буквенноцифровой)  string saleDate; // Дата продажи  double price; // Цена  bool returned; // Отметка о возврате  string returnDate; // Дата возврата    Node\* prev;  Node\* next;    // Конструктор по умолчанию  Node() : productCode(""), saleDate(""), price(0.0), returned(false), returnDate(""), prev(nullptr), next(nullptr) {} };   class LinkedList {  private:  Node\* head;  Node\* tail;    public:  LinkedList();  ~LinkedList();    void insertNode(const string& code, const string& date, double price, bool returned, const string& returnDate);  void deleteNode(Node\* node);  void printForward();  void printBackward();  Node\* searchNode(const string& code);    // Дополнительные функции  void orderListByRule();  void restructureList();  void deleteItemsBeforeDate(const string& date); };   #endif |
| --- |

Блок кода 9 – LinkedList.h

| #include <iostream> #include "LinkedList.h"   using namespace std;    // Конструктор LinkedList::LinkedList() {  head = nullptr; }   // Деструктор LinkedList::~LinkedList() {  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  } }   // Вставка узла void LinkedList::insertNode(const string& productCode, const string& saleDate, double price, bool returned, const string& returnDate) {  Node\* newNode = new Node;  newNode->productCode = productCode;  newNode->saleDate = saleDate;  newNode->price = price;  newNode->returned = returned;  newNode->returnDate = returnDate;    if (head == nullptr) {  head = newNode;  newNode->prev = nullptr;  newNode->next = nullptr;  }  else {  newNode->next = head;  head->prev = newNode;  head = newNode;  newNode->prev = nullptr;  } }   // Удаление узла void LinkedList::deleteNode(Node\* node) {  if (node == nullptr)  return;    if (node->prev != nullptr)  node->prev->next = node->next;  else  head = node->next;    if (node->next != nullptr)  node->next->prev = node->prev;    delete node; }   // Вывод списка слева направо void LinkedList::printForward() {  Node\* current = head;  while (current != nullptr) {  cout << "Product Code: " << current->productCode << ", Date of Sale: " << current->saleDate << ", Price: " << current->price;  cout << ", Return: " << current->returned << ", Return date: " << current->returnDate << endl;  current = current->next;  } }   // Вывод списка справа налево void LinkedList::printBackward() {  Node\* current = head;  while (current->next != nullptr) {  current = current->next;  }    while (current != nullptr) {  cout << "Product Code: " << current->productCode << ", Date of Sale: " << current->saleDate << ", Price: " << current->price;  cout << ", Return: " << current->returned << ", Return date: " << current->returnDate << endl;  current = current->prev;  } }   // Поиск узла с заданным кодом товара Node\* LinkedList::searchNode(const string& code) {  Node\* current = head;  while (current != nullptr) {  if (current->productCode == code) {  return current;  }  current = current->next;  }  return nullptr; // Если узел с заданным кодом не найден }   // Дополнительные функции   // Упорядочить список заказов по правилу void LinkedList::orderListByRule() {  // Сначала создадим два списка - один для проданных товаров, другой для возвращенных  LinkedList soldList;  LinkedList returnedList;    // Переберем все элементы в текущем списке  Node\* current = head;  while (current != nullptr) {  // Если товар продан, но не возвращен, добавляем его в список проданных  if (!current->returned) {  soldList.insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  }  // Если товар возвращен, добавляем его в список возвращенных  else {  returnedList.insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  }  current = current->next;  }    // Удалим все элементы из текущего списка  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }    // Добавим элементы из списка возвращенных  current = returnedList.head;  while (current != nullptr) {  insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  current = current->next;  }    // Добавим элементы из списка проданных  current = soldList.head;  while (current != nullptr) {  insertNode(current->productCode, current->saleDate, current->price, current->returned, current->returnDate);  current = current->next;  } }   // Переформировать список void LinkedList::restructureList() {  Node\* current = head;  Node\* lastSoldNode = nullptr;    // Найдем последний проданный товар  while (current != nullptr) {  if (!current->returned) {  lastSoldNode = current;  }  current = current->next;  }    // Если нет возвращенных товаров или список пуст, ничего не делаем  if (lastSoldNode == nullptr || lastSoldNode == head) {  return;  }    // Сохраняем указатель на начало списка  Node\* originalHead = head;    // Удаляем все проданные товары после последнего возвращенного из текущего списка  Node\* temp = lastSoldNode->next;  while (temp != nullptr) {  Node\* nextNode = temp->next;  deleteNode(temp);  temp = nextNode;  }    // Теперь вставим удаленные элементы в начало списка  current = lastSoldNode->next;  while (current != nullptr) {  Node\* nextNode = current->next;  current->next = originalHead;  current->prev = nullptr;  originalHead->prev = current;  originalHead = current;  current = nextNode;  }    // Обновляем указатель head  head = originalHead; }   // Удалить товар, который продан до указанной даты. void LinkedList::deleteItemsBeforeDate(const string& date) {  Node\* current = head;  while (current != nullptr) {  if (current->saleDate < date) {  Node\* temp = current;  current = current->next;  deleteNode(temp);  }  else {  current = current->next;  }  } } |
| --- |

Блок кода 10 – LinkedList.cpp

| #include <iostream> #include "LinkedList.h" #include "LinkedList.cpp"   using namespace std;   void printMenu() {  cout << "\nOperation selection menu:" << endl;  cout << "1. Insert a node" << endl;  cout << "2. Delete a node" << endl;  cout << "3. Searching for a node with a given value" << endl;  cout << "4. Delete nodes before the specified date" << endl;  cout << "5. Arrange the list according to the rule" << endl;  cout << "6. To reformulate the list" << endl;  cout << "7. Bring the list forward" << endl;  cout << "8. Bring the list backward" << endl;  cout << "9. Exit" << endl;  cout << "Select an action: "; }   int main() {  LinkedList list;  int choice;    do {  printMenu();  cin >> choice;  cin.ignore(); // Очистим буфер ввода от символа новой строки    switch (choice) {  case 1: {  string code, date, returnDate;  double price;  bool returned;    cout << "Enter the product code: ";  getline(cin, code);  cout << "Enter the date of sale (yyyy-mm-dd): ";  getline(cin, date);  cout << "Enter the price: ";  cin >> price;  cout << "Has the item been returned (1 - yes, 0 - no): ";  cin >> returned;  cin.ignore(); // Очистим буфер ввода от символа новой строки  if (returned) {  cout << "Enter the return date (yyyy-mm-dd): ";  getline(cin, returnDate);  } else {  returnDate = "";  }    list.insertNode(code, date, price, returned, returnDate);  break;  }  case 2: {  string codeToDelete;  cout << "Enter the product code to delete: ";  cin >> codeToDelete;  Node\* nodeToDelete = list.searchNode(codeToDelete);  if (nodeToDelete != nullptr) {  list.deleteNode(nodeToDelete);  cout << "The node with the product code " << codeToDelete << " successfully deleted." << endl;  }  else {  cout << "The node with the product code " << codeToDelete << " was not found." << endl;  }  break;  }  case 3: {  string searchCode;  cout << "Enter the product code to search for: ";  cin >> searchCode;  Node\* foundNode = list.searchNode(searchCode);  if (foundNode != nullptr) {  cout << "The node with the product code " << searchCode << " found." << endl;  }  else {  cout << "The node with the product code " << searchCode << " was not found." << endl;  }  break;  }  case 4: {  string date;  cout << "Enter the date: ";  getline(cin, date);  list.deleteItemsBeforeDate(date);  cout << "Nodes sold before " << date << ", deleted." << endl;  break;  }  case 5: {  list.orderListByRule();  cout << "The list is ordered according to the rule." << endl;  break;  }  case 6: {  list.restructureList();  cout << "The list has been reorganized." << endl;  break;  }  case 7: {  cout << "Bringing the list forward:" << endl;  list.printForward();  break;  }  case 8: {  cout << "Bringing the list backward:" << endl;  list.printBackward();  break;  }  case 9: {  cout << "The program is completed." << endl;  break;  }  default: {  cout << "Wrong choice. Try again." << endl;  break;  }  }  } while (choice != 9);    return 0; } |
| --- |

Блок кода 11 – main.cpp

### **2.3.2 Тестирование**

Проведем тестирование, основываясь на данных из таблиц 1-7. Результаты тестирования 1-ой строки представлены на рисунках 7-12.

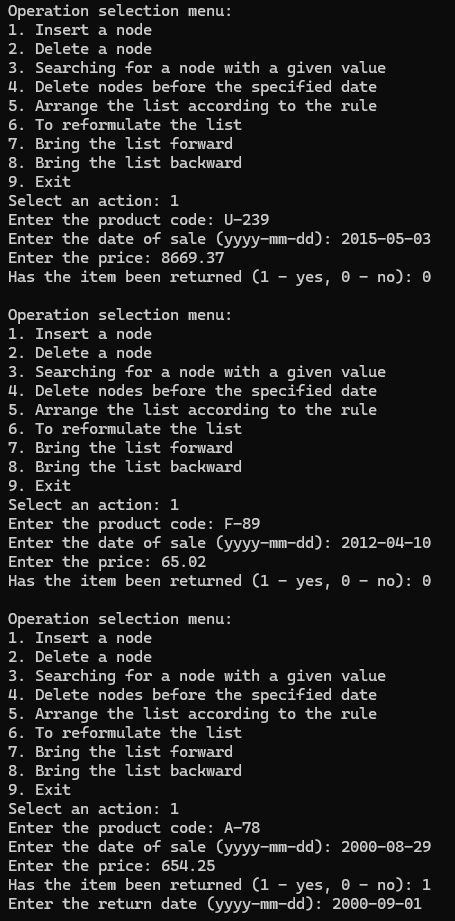


Рисунок 7 - Тестирование программы на данных 1-ой строки



Рисунок 8 - Тестирование программы на данных 1-ой строки



Рисунок 9 - Тестирование программы на данных 1-ой строки



Рисунок 10 - Тестирование программы на данных 1-ой строки

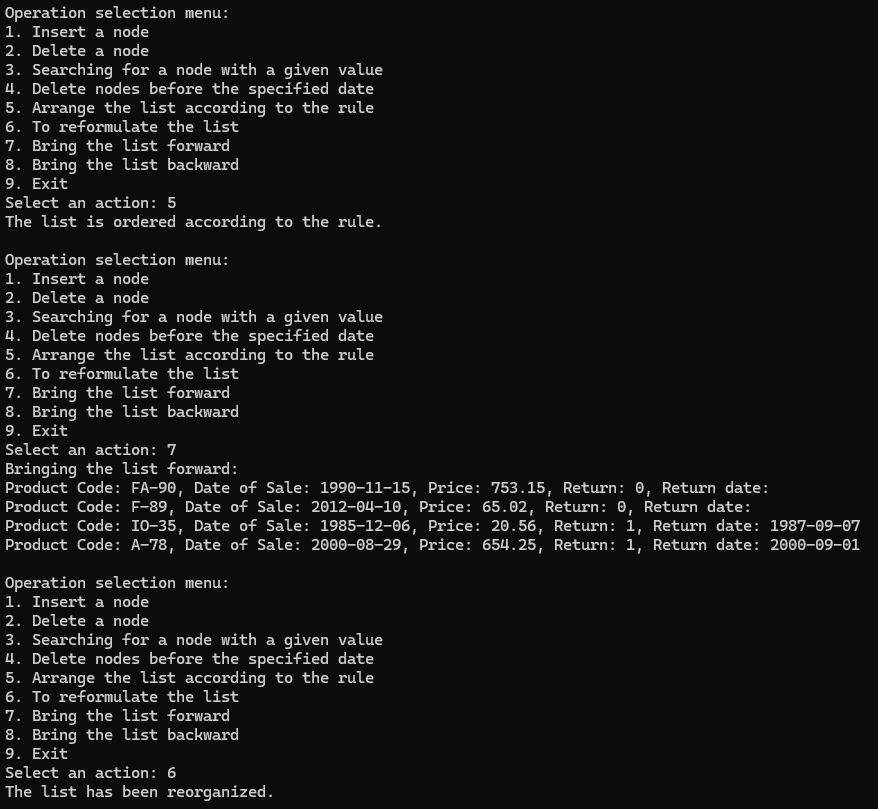


Рисунок 11 - Тестирование программы на данных 1-ой строки



Рисунок 12 - Тестирование программы на данных 1-ой строки

Проведение тестирования на основе 1-ой строки таблиц 1-6, показало, что программа отработала верно, так как совпала с ожидаемым результатом.

Результаты тестирования 2-ой строки представлены на рисунках 13-19.

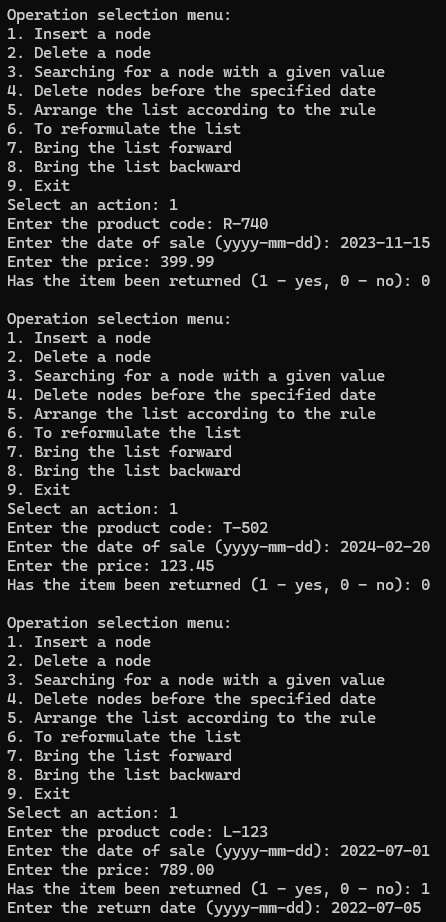


Рисунок 13 - Тестирование программы на данных 2-ой строки

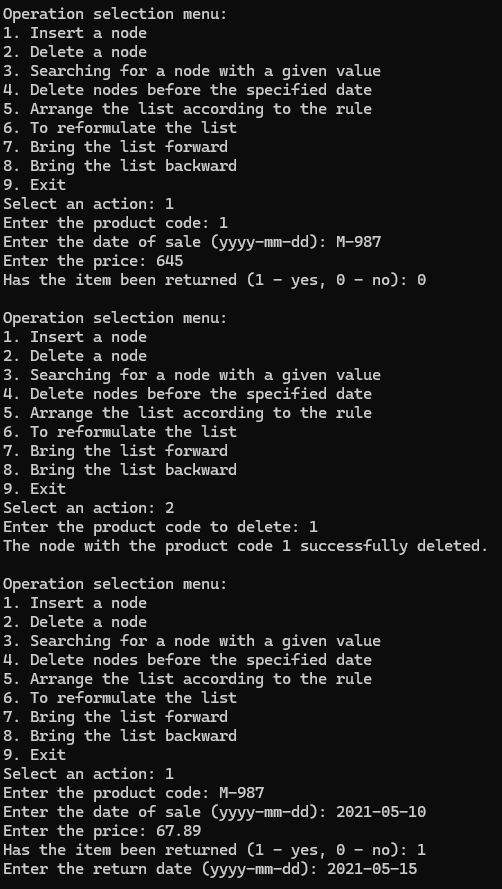


Рисунок 14 - Тестирование программы на данных 2-ой строки

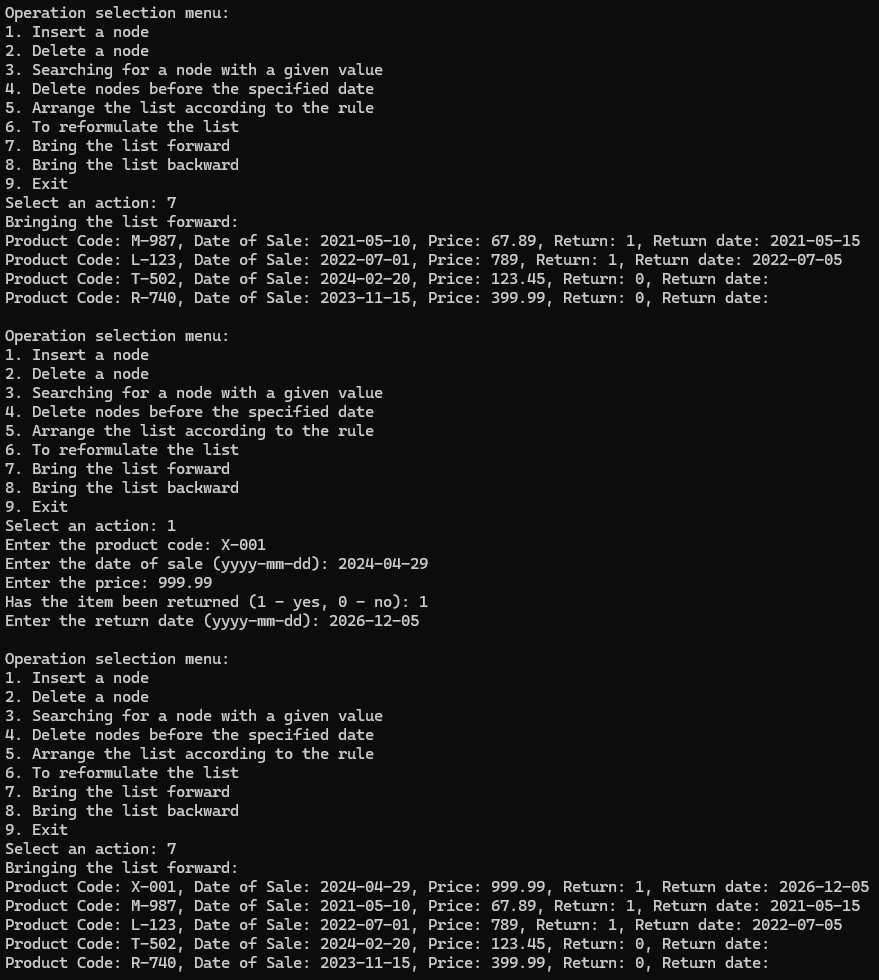


Рисунок 15 - Тестирование программы на данных 2-ой строки

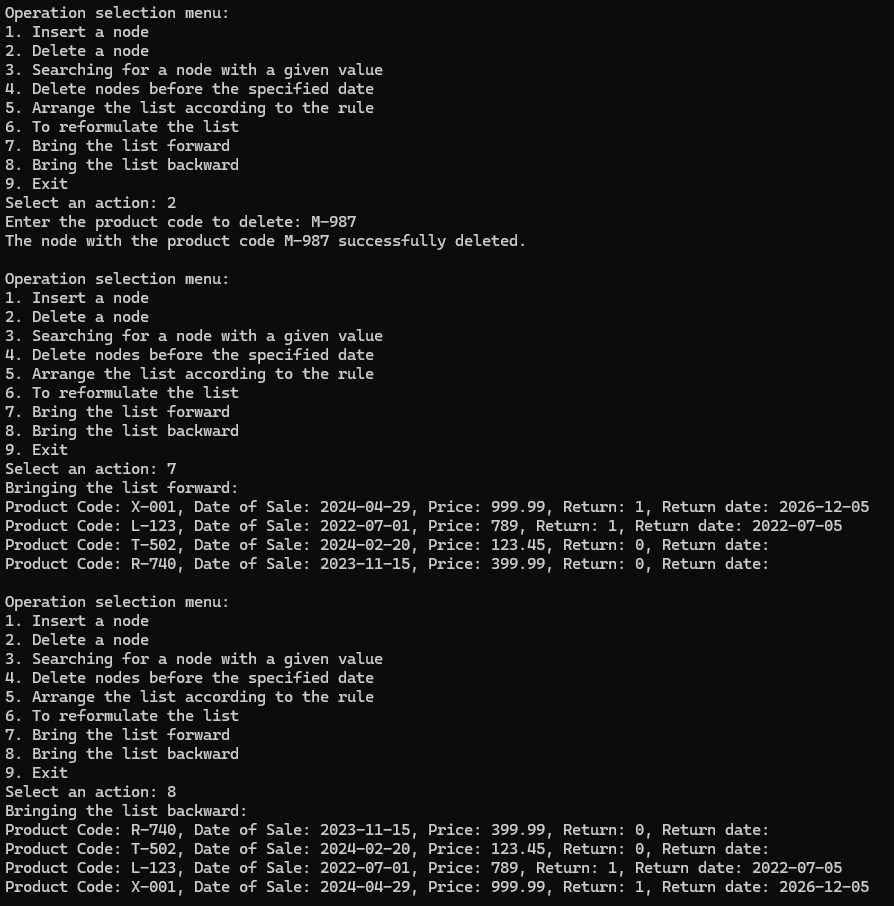


Рисунок 16 - Тестирование программы на данных 2-ой строки

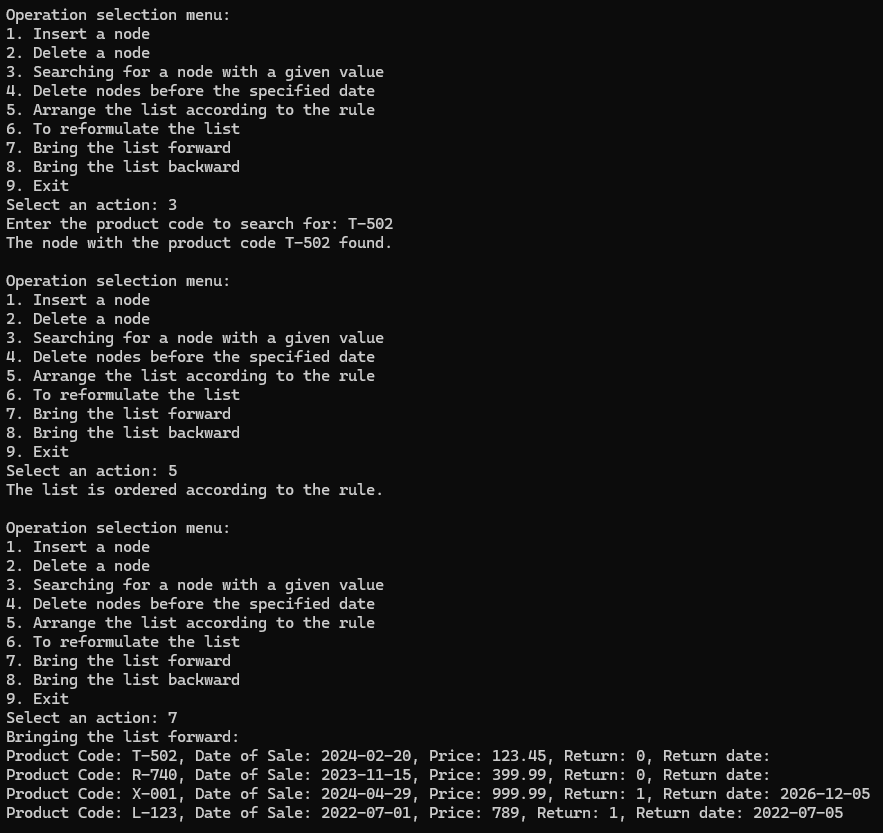


Рисунок 17 - Тестирование программы на данных 2-ой строки



Рисунок 18 - Тестирование программы на данных 2-ой строки

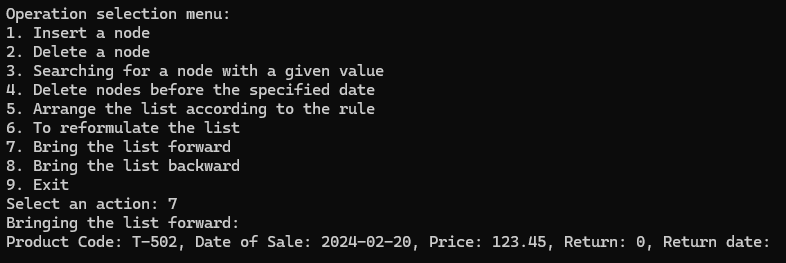


Рисунок 19 - Тестирование программы на данных 2-ой строки

Проведение тестирования на основе 2-ой строки таблиц 1-6, показало, что программа отработала верно, так как совпала с ожидаемым результатом.

## **2.4 Вывод по заданию**

Двунаправленный список, или двусвязный список, представляет собой структуру данных, в которой каждый узел содержит данные и два указателя: на предыдущий и следующий узлы. Эта структура обеспечивает возможность эффективной вставки и удаления элементов в любом месте списка, а также позволяет перебирать элементы как в прямом, так и в обратном направлении.

При использовании двунаправленного списка важно учитывать следующие аспекты:

При добавлении и удалении элементов необходимо обновлять указатели на предыдущий и следующий узлы.

Важно корректно обрабатывать краевые случаи, такие как вставка в начало или конец списка.

При использовании итераторов необходимо учитывать направление движения при доступе к элементам списка.

Таким образом, двунаправленный список предоставляет удобную структуру данных для операций вставки, удаления и обращения к элементам как в прямом, так и в обратном направлении. Это особенно полезно, когда требуется доступ к элементам в обратном порядке или необходимо часто добавлять и удалять элементы из середины списка. Однако, если требуется частый поиск элементов в списке, другие структуры данных, такие как хэш-таблицы или деревья, могут быть более эффективность.

# **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).