|  |
| --- |
| 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: Величко В.Д. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ 4](#_30j0zll)

[2.1 Формулировка задачи (В списке №10, Вариант 10) 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 7](#_tyjcwt)

[2.2.2.1 Вставка узла. 7](#_3dy6vkm)

[2.2.2.2 Удаление узла. 10](#_1t3h5sf)

[2.2.2.3 Вывод элементов списка на экран. 13](#_4d34og8)

[2.2.2.4 Поиск узла с заданным значением. 15](#_2s8eyo1)

[2.2.2.5 Упорядочить созданный список из n узлов по стране изготовителю. 17](#_17dp8vu)

[2.2.2.6 Установить дату продажи автомобиля. 20](#_3rdcrjn)

[2.2.2.7 Удалить все узлы по проданным автомобилям. 21](#_26in1rg)

[2.2.2.8 Вставка узла в начале списка. 25](#_lnxbz9)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 29](#_35nkun2)

[2.3.1 Реализация на языке программирования C++ 29](#_1ksv4uv)

[2.3.2 Тестирование 41](#_44sinio)

[2.4 Вывод по заданию 51](#_2jxsxqh)

[3 ВЫВОДЫ 52](#_3j2qqm3)

[4 ЛИТЕРАТУРА 53](#_1y810tw)

# **1 ЦЕЛЬ**

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

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи (В списке №5, Вариант 5)**

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 счета в банке (20- значное число), дата, вид операции (приход или расход), сумма вклада

Дополнительные операции:

Вставка нового узла перед первым узлом.

Удаление сведений по счету (всех узлов), у которого общая сумма вклада равна нулю ( сумма по приходу, минус сумма по расходу).

Создать новый список из исходного, которого будет содержать остаток по всем видам операций одного счета, указав вид операции – приход, и текущую дату.

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

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

Определим структуру узла для двусвязного списка, который предназначен для хранения информации о банковских операциях. accountNumber (номер счета): строковая переменная, которая содержит номер счета в банке, date (дата операции): строковая переменная, хранящая дату выполнения операции, operationType (вид операции): строковая переменная, обозначающая тип операции, например, "приход" или "расход", amount (сумма вклада): переменная типа double, представляющая сумму операции, next (указатель на следующий узел): указатель на следующий узел списка. Этот указатель используется для перемещения по списку вперед, prev (указатель на предыдущий узел): указатель на предыдущий узел списка. Этот указатель используется для перемещения по списку назад.

Как правило, такая структура используется для создания списка операций, где каждый узел представляет собой одну операцию, а указатели next и prev обеспечивают связь между этими узлами, образуя цепочку.

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

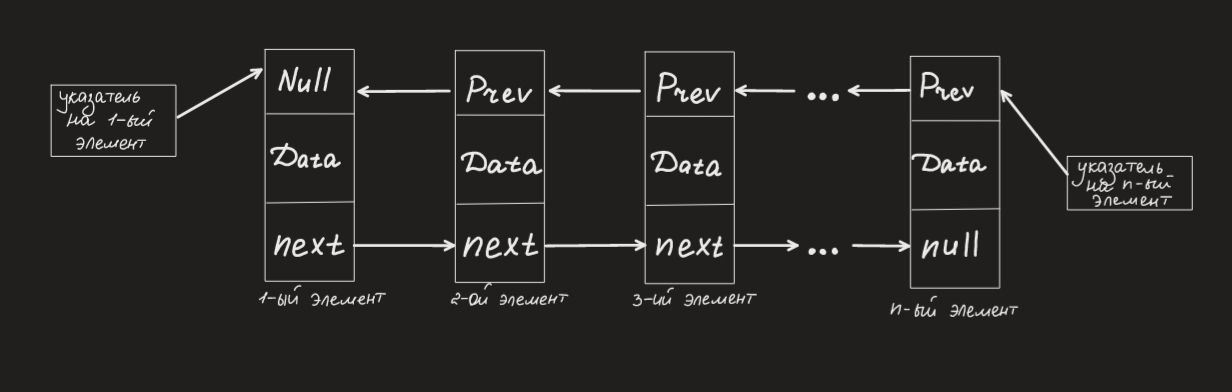


Рисунок 1 - Изображение структуры данных

| struct Node {  string accountNumber; // Номер счета в банке  string date; // Дата операции  string operationType; // Вид операции (приход или расход)  double amount; // Сумма вклада  Node\* next; // Указатель на следующий узел  Node\* prev; // Указатель на предыдущий узел }; |
| --- |

Блок кода 1 - Структура

### **2.2.2 Процесс выполнения операций**

#### 2.2.2.1 Вставка узла в конец списка.

Отобразим выполнение данной операции на рисунке 2.

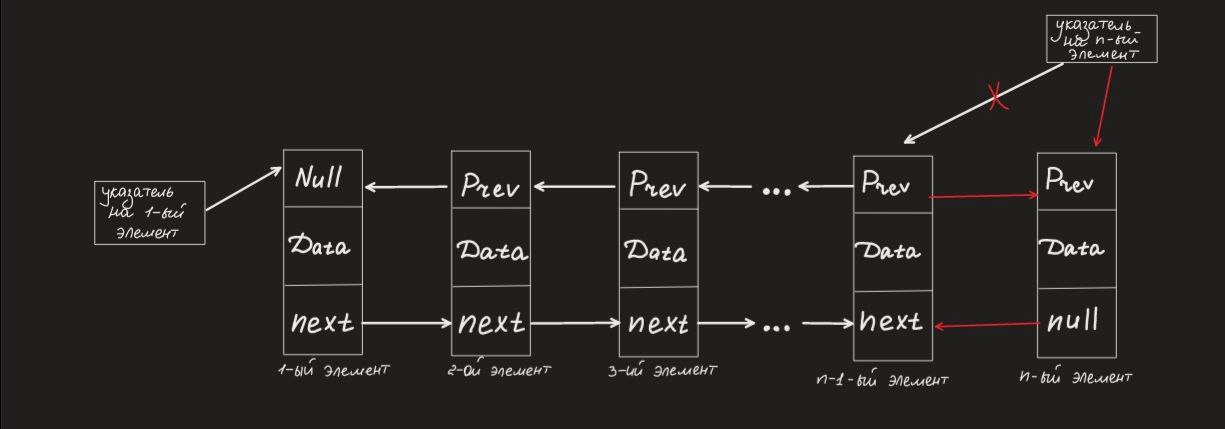


Рисунок 2 - Изображение добавление нового узла

Создается новый узел newNode с заданными данными: номером счета, датой, типом операции, суммой. Узел инициализируется с указателями prev и next, которые изначально устанавливаются в nullptr. Проверяется, если головной узел списка равен nullptr, что означает, что список пустой. Если это так, то новый узел newNode устанавливается как и головной, и хвостовой узлы списка. В противном случае, если список не пустой, новый узел newNode добавляется в конец списка. Устанавливается указатель next предыдущего последнего узла на новый узел newNode. Устанавливается указатель prev нового узла newNode на предыдущий последний узел. Обновляется указатель tail, чтобы он указывал на новый узел newNode, который становится новым последним узлом списка.

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 |  |  |  |
| 2 |  |  |  |

#### 2.2.2.2 Удаление узла.

Отобразим выполнение данной операции на рисунке 3.

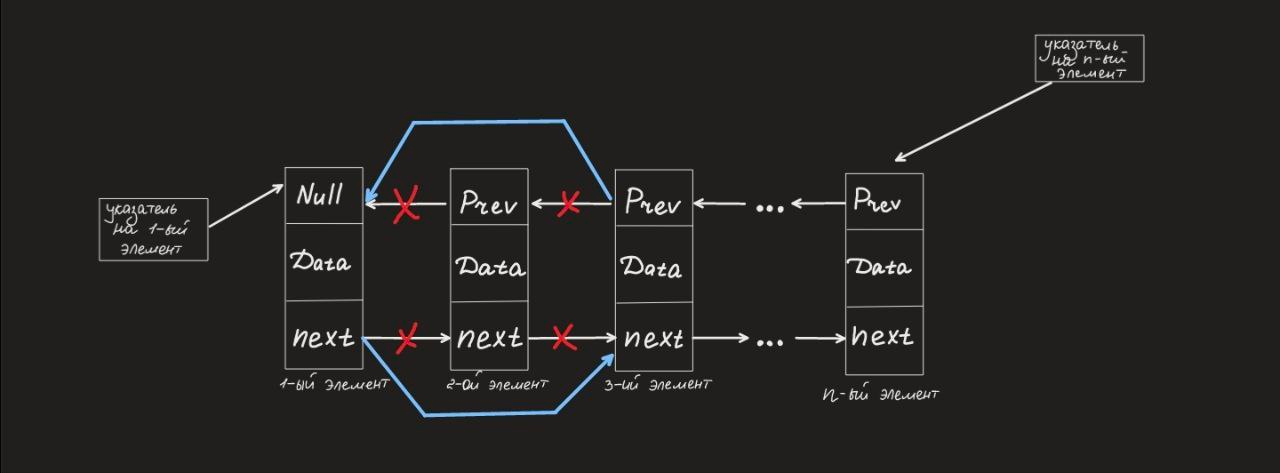


Рисунок 3 - Изображение удаления узла

Удалим узел из списка по номеру счета. Сначала объявляется указатель current, который инициализируется значением head, указывающим на начало списка. Затем начинается цикл while, который продолжается до тех пор, пока current не станет равным nullptr, то есть до тех пор, пока мы не дойдем до конца списка. Внутри цикла проверяется, совпадает ли номер счета текущего узла (current->accountNumber) с номером счета, который мы ищем для удаления. Если номер счета совпадает, то проверяется несколько условий:

- Если текущий узел единственный в списке (когда он и head и tail), обнуляются указатели head и tail.

- Если текущий узел является head, обновляется head на следующий узел, а указатель prev этого узла устанавливается в nullptr.

- Если текущий узел является tail, обновляется tail на предыдущий узел, а указатель next этого узла устанавливается в nullptr.

- Во всех остальных случаях (когда текущий узел находится где-то посередине списка), обновляются указатели предыдущего и следующего узлов таким образом, чтобы обойти текущий узел.

После выполнения операций по удалению узла освобождается память, выделенная для этого узла, с помощью оператора delete.

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 |  |  |  |
| 2 |  |  |  |

#### 2.2.2.3 Вывод элементов списка на экран.

Этот код представляет собой два метода printForward и printBackward, которые выводят содержимое списка в прямом и обратном порядке соответственно.

printForward:

Этот метод начинает с начала списка и перебирает каждый узел, пока не достигнет конца списка. Для каждого узла выводится информация о счете , дате операции, виде операции и сумме. Затем указатель current перемещается на следующий узел с помощью current = current->next.

printBackward:

Этот метод начинает с конца списка и перебирает каждый узел, пока не достигнет начала списка. Для каждого узла выводится информация о счете, дате операции, виде операции и сумме. Затем указатель current перемещается на предыдущий узел с помощью current = current->prev.

Реализация данного алгоритма представлена в блок коде 3.

#### 2.2.2.4 Поиск узла по номеру счёта

Отобразим выполнение данной операции на рисунке 4.

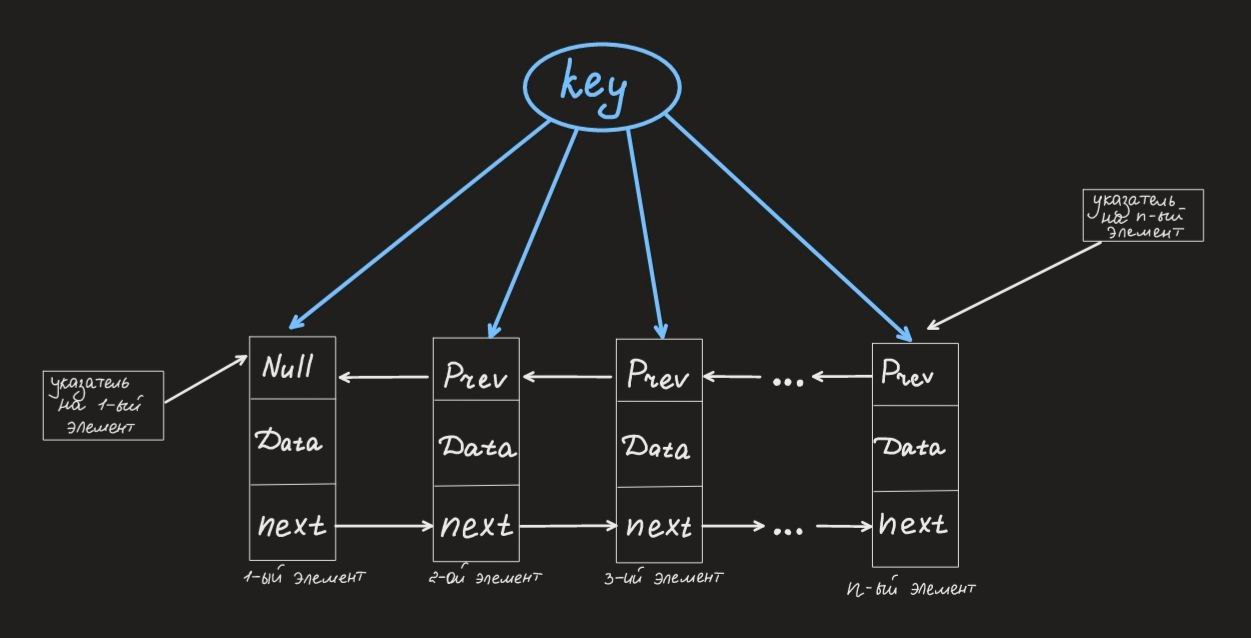


Рисунок 4 - Поиск узла с заданным значением

Осуществим поиск узла в списке по номеру счета. Сначала объявляется указатель current, который инициализируется значением head, указывающим на начало списка. Затем начинается цикл while, который продолжается до тех пор, пока current не станет равным nullptr, то есть до тех пор, пока мы не дойдем до конца списка. Внутри цикла проверяется, совпадает ли номер счета текущего узла с искомым номером счета. Если номер счета совпадает, возвращается указатель на текущий узел. Если номер счета не совпадает, указатель current перемещается на следующий узел списка, чтобы продолжить поиск. Если цикл завершается и узел с искомым номером счета не найден, метод возвращает nullptr, чтобы указать, что такой узел не обнаружен в списке.

Реализация данного алгоритма представлена в блок коде 3.

#### 2.2.2.5 Вставка узла в начале списка.

Отобразим выполнение данной операции на рисунке 5.

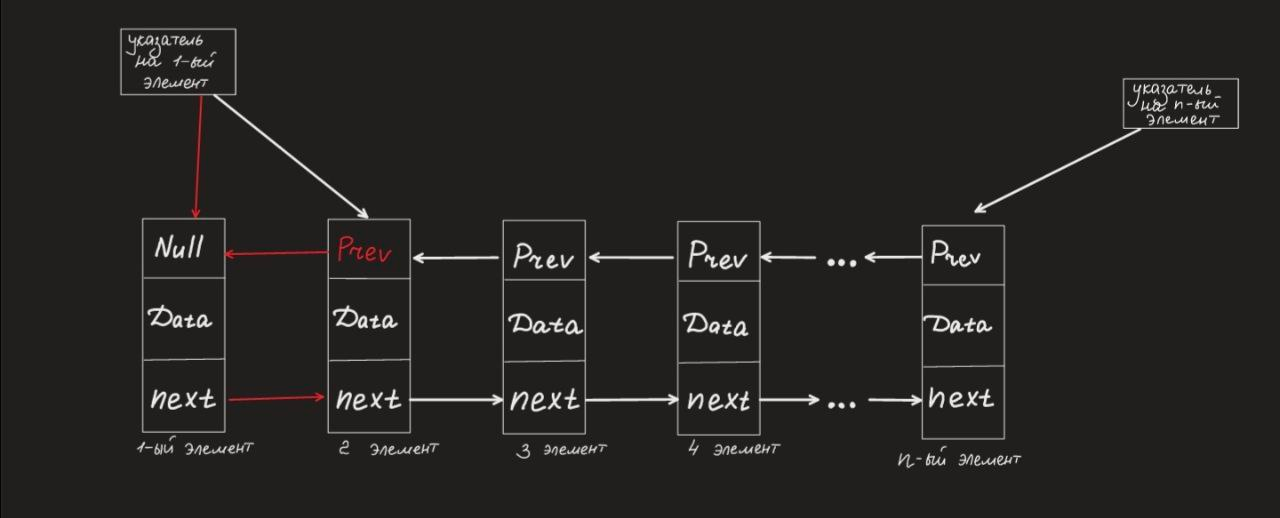


Рисунок 5 - Изображение добавление нового узла

Реализуем два вида вставки, так как одна дана по условию задания, вторая дана в дополнительных функциях. В данном случае рассматривается тот метод, где вставка происходит в начале списка.

Вставим новый узел перед первым узлом в списке. Сначала создается новый узел newNode, инициализирующийся переданными в метод значениями accountNumber, date, operationType и amount. next и prev устанавливаются в nullptr, так как этот узел будет первым в списке. После этого проверяется, пуст ли список. Если head равен nullptr, это означает, что список пустой, и новый узел становится как head, так и tail списка. Если список не пустой, то новый узел добавляется перед первым узлом. Для этого:

- Устанавливается указатель next нового узла на текущий head, чтобы он указывал на первый узел.

- Устанавливается указатель prev текущего head на новый узел newNode, чтобы связать его с новым узлом.

- head обновляется, чтобы указывать на новый узел newNode, который теперь стал первым в списке.

Этот метод эффективно вставляет новый узел перед первым узлом в списке, обновляя указатели next и prev для правильной связи между узлами.

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

Таблица 3 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 |  |  |  |
| 2 |  |  |  |

#### 2.2.2.6 Создание нового списка, содержащиц остаток по всем видам операций одного счета

Создается новый пустой список balanceList, который будет содержать только операции "Приход" для заданного номера счета. Инициализируется указатель current на головной узел списка head. Запускается цикл, который будет выполняться до тех пор, пока current не станет равным nullptr, то есть пока мы не достигнем конца списка. Проверяется, если номер счета текущего узла (current->accountNumber) совпадает с заданным номером счета, и если тип операции текущего узла равен "Приход". Если условие выполняется, то создается новый узел в списке balanceList, куда копируются данные о текущей операции: номер счета, дата, тип операции и сумма. Перемещается указатель current на следующий узел в списке head. По завершении цикла возвращается указатель на новый список balanceList, который содержит только операции "Приход" для заданного номера счета.

Сложность данного алгоритма можно оценить, исходя из его структуры: Проход по всему исходному списку: O(n), где n - количество узлов в исходном списке. Для каждого узла в исходном списке выполняется проверка условия и, возможно, создание нового узла в новом списке. Это операции выполняются за константное время O(1), но их количество зависит от размера исходного списка и фильтрации по условию. Таким образом, общая сложность алгоритма будет O(n), где n - количество узлов в исходном списке. Это потому, что мы выполняем одну операцию (проверку условия и, возможно, создание нового узла) для каждого узла в исходном списке.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 4.

Таблица 4 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 |  |  |
| 2 |  |  |

#### 2.2.2.7 Удаление сведений по счету, у которого общая сумма вклада равна нулю

Отобразим выполнение данной операции на рисунке 6.

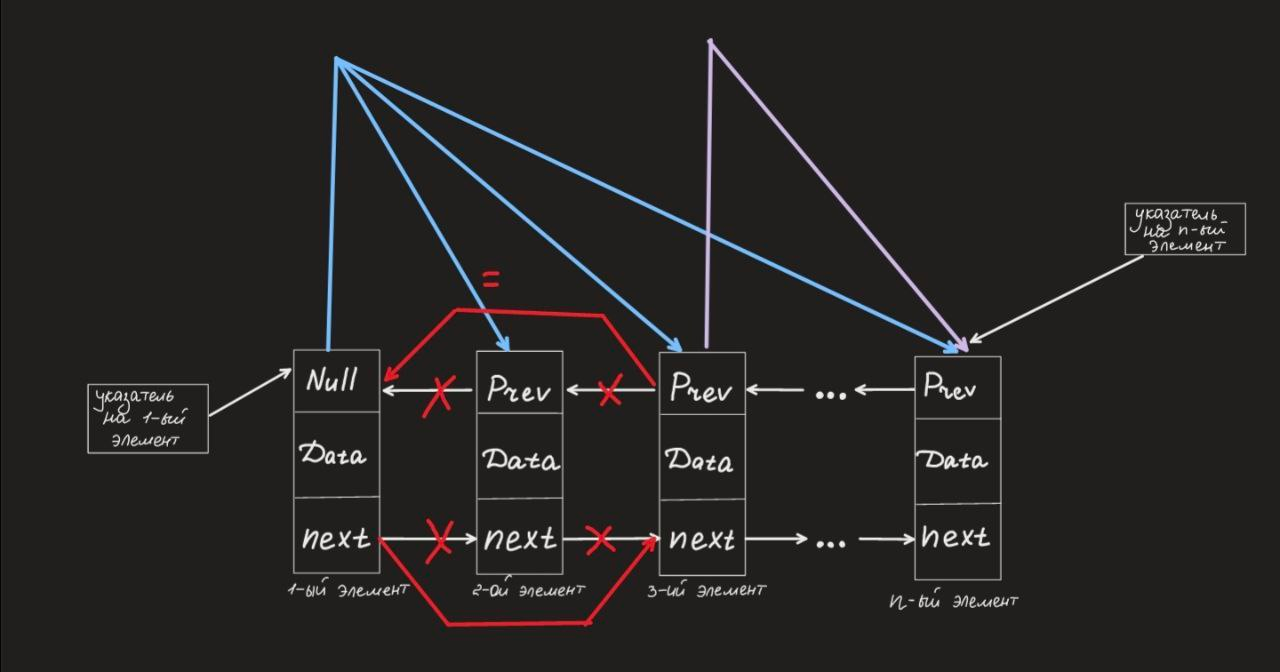


Рисунок 6 - Изобразим удаление сведений по счёту

Удалим сведения о счетах с нулевой суммой вклада. Сначала создается указатель current, который инициализируется значением head, указывающим на начало списка. Затем начинается цикл while, который продолжается до тех пор, пока current не станет равным nullptr, то есть до тех пор, пока мы не дойдем до конца списка. Внутри цикла для каждого узла current ищется сумма вклада по счету. Для этого:

- Создается указатель nextNode, который инициализируется значением current->next, указывающим на следующий узел списка.

- Создается переменная sum, в которой будет суммироваться вклад по текущему счету.

- Затем во внутреннем цикле while перебираются все следующие узлы, начиная с nextNode, и если номер счета совпадает с номером счета текущего узла current, то вклады (приходы и расходы) суммируются.

- После завершения внутреннего цикла, если сумма вклада равна нулю, вызывается метод deleteNode для удаления текущего узла current, содержащего информацию о счете с нулевой суммой вклада.

Указатель current перемещается на следующий узел списка, чтобы продолжить поиск и удаление сведений о счетах с нулевой суммой вклада.

Этот метод эффективно проходит по списку, ищет счета с нулевой суммой вклада и удаляет информацию о них.

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

Таблица 5 - Данные для тестирования

| № | Входные данные | Дата | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 |  |  |  |
| 2 |  |  |  |

## **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 LIST\_H #define LIST\_H #include <iostream> #include <string> using namespace std;  // Структура узла списка struct Node {  string accountNumber; // Номер счета в банке  string date; // Дата операции  string operationType; // Вид операции (приход или расход)  double amount; // Сумма вклада  Node\* next; // Указатель на следующий узел  Node\* prev; // Указатель на предыдущий узел };  // Класс двунаправленного динамического списка class List { public:  List(); // Конструктор  ~List(); // Деструктор   // Функции для выполнения операций над списком  void insertNode(string accountNumber, string date, string operationType, double amount); // Вставка узла  void deleteNode(string accountNumber); // Удаление узла  void printForward(); // Вывод списка в прямом порядке  void printBackward(); // Вывод списка в обратном порядке  Node\* searchNode(string accountNumber); // Поиск узла по номеру счета   // Дополнительные операции над списком  void insertBeforeFirst(string accountNumber, string date, string operationType, double amount); // Вставка нового узла перед первым узлом  void deleteZeroSumAccounts(); // Удаление сведений по счету с нулевой суммой вклада  List\* createBalanceList(string accountNumber); // Создание нового списка с остатком по всем видам операций одного счета  private:  Node\* head; // Указатель на начало списка  Node\* tail; // Указатель на конец списка };  #endif |
| --- |

Блок кода 2 – List.h

| #include "List.h"  // Конструктор списка List::List() : head(nullptr), tail(nullptr) {}  // Деструктор списка List::~List() {  // Освобождение памяти, выделенной под узлы списка  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  } }  // Вставка нового узла в конец списка void List::insertNode(string accountNumber, string date, string operationType, double amount) {  Node\* newNode = new Node{accountNumber, date, operationType, amount, nullptr, nullptr};  if (head == nullptr) {  head = tail = newNode;  } else {  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  } }  // Удаление узла из списка по номеру счета void List::deleteNode(string accountNumber) {  Node\* current = head;  while (current != nullptr) {  if (current->accountNumber == accountNumber) {  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;  } }  // Вывод списка в прямом порядке void List::printForward() {  Node\* current = head;  while (current != nullptr) {  cout << "Account: " << current->accountNumber << ", Date: " << current->date   << ", Operation: " << current->operationType << ", Amount: " << current->amount << endl;  current = current->next;  } }  // Вывод списка в обратном порядке void List::printBackward() {  Node\* current = tail;  while (current != nullptr) {  cout << "Account: " << current->accountNumber << ", Date: " << current->date   << ", Operation: " << current->operationType << ", Amount: " << current->amount << endl;  current = current->prev;  } }  // Поиск узла по номеру счета Node\* List::searchNode(string accountNumber) {  Node\* current = head;  while (current != nullptr) {  if (current->accountNumber == accountNumber) {  return current;  }  current = current->next;  }  return nullptr; }  // Дополнительные операции над списком  // Вставка нового узла перед первым узлом void List::insertBeforeFirst(string accountNumber, string date, string operationType, double amount) {  Node\* newNode = new Node{accountNumber, date, operationType, amount, nullptr, nullptr};  if (head == nullptr) {  head = tail = newNode;  } else {  newNode->next = head;  head->prev = newNode;  head = newNode;  } }  // Удаление сведений по счету с нулевой суммой вклада void List::deleteZeroSumAccounts() {  Node\* current = head;  while (current != nullptr) {  Node\* nextNode = current->next;  double sum = 0.0;  while (nextNode != nullptr) {  if (nextNode->accountNumber == current->accountNumber) {  if (nextNode->operationType == "Приход") {  sum += nextNode->amount;  } else if (nextNode->operationType == "Расход") {  sum -= nextNode->amount;  }  }  nextNode = nextNode->next;  }  if (sum == 0.0) {  deleteNode(current->accountNumber);  }  current = current->next;  } }    // Создание нового списка с остатком по всем видам операций одного счета List\* List::createBalanceList(string accountNumber) {  List\* balanceList = new List();  Node\* current = head;  while (current != nullptr) {  if (current->accountNumber == accountNumber && current->operationType == "Приход") {  balanceList->insertNode(current->accountNumber, current->date, current->operationType, current->amount);  }  current = current->next;  }  return balanceList; } |
| --- |

Блок кода 3 – List.cpp

| #include <iostream> #include "List.h" using namespace std;  int main() {  int choice, choose;  List list; // Declare an instance of the List class   cout << "Выберите способ ввода данных:" << endl;  cout << "1. Ввести данные с клавиатуры" << endl;  cout << "2. Использовать уже существующие данные" << endl;  cout << "Введите номер способа (1 или 2): ";  cin >> choose;   if (choose == 1) {  string accountNumber, date, operationType;  double amount;  cout << "Введите данные для узла списка:" << endl;  cout << "Номер счета в банке: ";  cin >> accountNumber;  cout << "Дата операции: ";  cin >> date;  cout << "Вид операции (Приход/Расход): ";  cin >> operationType;  cout << "Сумма вклада: ";  cin >> amount;  list.insertNode(accountNumber, date, operationType, amount);  } else if (choose == 2) {  list.insertNode("12345678901234567890", "2024-04-25", "Приход", 1000.0);  list.insertNode("12345678901234567891", "2024-04-26", "Расход", 500.0);  list.insertNode("12345678901234567890", "2024-04-27", "Приход", 1500.0);  list.insertNode("12345678901234567892", "2024-04-28", "Приход", 2000.0);  } else {  cout << "Неверный ввод. Завершение программы." << endl;  return 0;  }   do {  // Вывод текстового меню  cout << "\nМеню:" << endl;  cout << "1. Вывод списка вперед на экран" << endl;  cout << "2. Вывод списка назад на экран" << endl;  cout << "3. Вставка нового узла в конец подсписка" << endl;  cout << "4. Вставка нового узла в начало подсписка" << endl;  cout << "5. Удаление узла" << endl;  cout << "6. Поиск узла с заданным значением" << endl;  cout << "7. Удаление сведений по счету, у которого общая сумма вклада равна нулю" << endl;  cout << "8. Создать новый список из исходного, содержащий остаток по всем видам операций одного счета" << endl;  cout << "9. Выход из программы" << endl;  cout << "Введите пункт от 1 до 9: ";  cin >> choice;     switch(choice) {  case 1:  cout << "\nВывод списка вперед на экран:" << endl;  list.printForward();  break;  case 2:  cout << "\nВывод списка назад на экран:" << endl;  list.printBackward();  break;  case 3:  {  string accountNumber, date, operationType;  double amount;  cout << "Введите номер счета: ";  cin >> accountNumber;  cout << "Введите дату: ";  cin >> date;  cout << "Введите вид операции (Приход/Расход): ";  cin >> operationType;  cout << "Введите сумму: ";  cin >> amount;  list.insertNode(accountNumber, date, operationType, amount);  break;  }  case 4:  {  string accountNumber, date, operationType;  double amount;  cout << "Введите номер счета: ";  cin >> accountNumber;  cout << "Введите дату: ";  cin >> date;  cout << "Введите вид операции (Приход/Расход): ";  cin >> operationType;  cout << "Введите сумму: ";  cin >> amount;  list.insertBeforeFirst(accountNumber, date, operationType, amount);  break;  }  case 5:  {  string accountNumber;  cout << "Введите номер счета, который хотите удалить: ";  cin >> accountNumber;  list.deleteNode(accountNumber);  break;  }  case 6:  {  string accountNumber;  cout << "Введите номер счета, который хотите найти: ";  cin >> accountNumber;  Node\* node = list.searchNode(accountNumber);  if (node != nullptr) {  cout << "\nНайденный узел: Счёт: " << node->accountNumber << ", Дата: " << node->date << ", Вид операции: " << node->operationType << ", Сумма счёта: " << node->amount << endl;  } else {  cout << "\nУзел не найден." << endl;  }  break;  }  case 7:  cout << "\nУдаление сведений по счету, у которого общая сумма вклада равна нулю:" << endl;  list.deleteZeroSumAccounts();  cout << "Сведения удалены." << endl;  break;  case 8:  {  string accountNumber;  cout << "Введите номер счета для создания балансовой ведомости: ";  cin >> accountNumber;  List\* balanceList = list.createBalanceList(accountNumber);  cout << "\nБалансовая ведомость для счёта " << accountNumber << ":" << endl;  balanceList->printForward();  delete balanceList;  break;  }  case 9:  cout << "Выход из программы." << endl;  break;  default:  cout << "Неверный ввод. Попробуйте еще раз." << endl;  }  } while (choice != 9);   return 0; } |
| --- |

Блок кода 4 – main.cpp

### **2.3.2 Тестирование**

Проведем тестирование основываясь на данные из таблиц 1-7. Результаты тестирования №1 строки представлены на рисунках 9-13.

Рисунок 9 - Тестирование программы на данных №1 строки

Рисунок 10 - Тестирование программы на данных №1 строки

Рисунок 11 - Тестирование программы на данных №1 строки

Рисунок 12 - Тестирование программы на данных №1 строки

Рисунок 13 - 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 13-17.

Рисунок 14 - Тестирование программы на данных №2 строки

Рисунок 15 - Тестирование программы на данных №2 строки

Рисунок 16 - Тестирование программы на данных №2 строки

Рисунок 17 - 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 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).