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

| **Отчет по выполнению практического задания № 5** | |
| --- | --- |
| **Тема:** | |
| **«Однонаправленный динамический список»** | |
| 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 4](#_1fob9te)

[2.2 Определение списка и описание операций над списком 5](#_2et92p0)

[2.2.1 Определение структуры узла однонаправленного списка 5](#_9wlltpb353rl)

[2.2.2 Процесс выполнения операций 6](#_3dy6vkm)

[2.2.2.1 Добавление узла в начале списка 6](#_q1lqrbkjc7s7)

[2.2.2.2 Удаление узла по ключу. 7](#_4zvnqh1lwco1)

[2.2.2.3 Вывод элементов списка на экран. 8](#_2ulva7p85a0g)

[2.2.2.4 Проверка на равенство списков L1 и L2 10](#_vmzjybkbzn5f)

[2.2.2.5 Вставление в список L1 последний элемент списка L2 11](#_nnk2khvvtj82)

[2.2.2.6 Удаление из списка L2, узлы, содержащие цифровые значения 12](#_kzf4118xeyo7)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 14](#_bubrpb7ydlxv)

[2.3.1 Реализация на языке программирования C++ 14](#_4d34og8)

[2.3.2 Тестирование 20](#_2s8eyo1)

[2.4 Вывод по заданию 27](#_lnxbz9)

[3 КОНТРОЛЬНЫЕ ВОПРОСЫ 28](#_49x2ik5)

[4 ВЫВОДЫ 37](#_n59xv9kokqs4)

[5 ЛИТЕРАТУРА 38](#_2p2csry)

# **1 ЦЕЛЬ**

Получить знания и практические навыки управления динамическим однонаправленным списком.

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи**

В списке 5, Вариант 5

Реализуйте программу решения задачи варианта по использованию линейного однонаправленного списка.

Требования

1. Информационная часть узла определена вариантом

2. Разработать функции вставки нового узла перед первым узлом и удаления узла по ключу.

3. Реализуйте возможность а) создания нового списка вручную, а также б) использования уже готового списка для тестирования заданий индивидуального варианта.

4. Разработать функцию вывода списка в консоль.

5. Разработать функции согласно индивидуальному варианту. При необходимости можно добавлять вспомогательные функции, декомпозируя задачу.

6. Реализуйте текстовое пользовательское меню.

7. В основной программе выполните тестирование каждой функции.

8. Составить отчет по выполненному заданию.

Индивидуальный вариант. Тип информационной части узла: char.

Дополнительные операции:

Даны два линейных однонаправленных списка L1 и L2 с головным элементом.

1. Разработать функцию, которая проверяет на равенство списки L1 и L2.

2. Разработать функцию, которая вставляет в список L1 последний элемент списка L2.

3. Разработать функцию, которая удаляет из списка L2, узлы, содержащие цифровые значения.

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

### **2.2.1 Определение структуры узла однонаправленного списка**

Однонаправленный список - это структура данных, которая представляет собой последовательность элементов, называемых узлами, связанных между собой однонаправленными ссылками. Каждый узел содержит данные и указатель на следующий узел в списке. В таком списке можно двигаться только в одном направлении - от начала к концу, начиная с головы списка. Конец списка обычно обозначается нулевым указателем или как nullptr в языке C++. Однонаправленные списки часто используются в программировании для реализации различных алгоритмов, таких как стеки, очереди и другие структуры данных.

Структура Node представляет собой элемент однонаправленного списка. Она содержит переменную data типа char, которая хранит значение элемента, и указатель на следующий элемент списка next. Конструктор Node принимает значение типа char и инициализирует переменную data этим значением, а указатель next устанавливается в nullptr.

| struct Node {  char data;  Node\* next;  Node(char d) : data(d), next(nullptr) {} }; |
| --- |

Таким образом, каждый узел списка имеет доступ к своему значению и указателю на следующий узел, обеспечивая эффективное представление данных в однонаправленном списке.

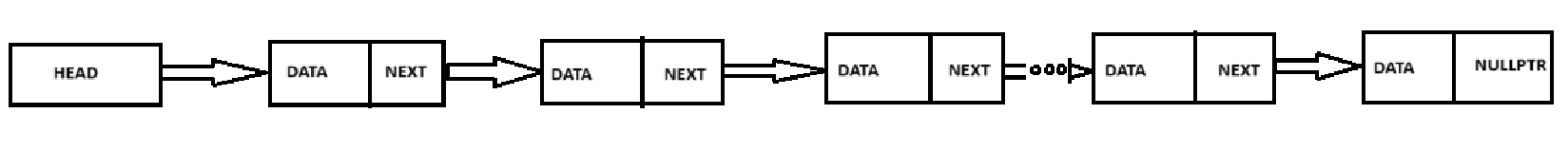


Рисунок 1 - Изображение структуры данных

### **2.2.2 Процесс выполнения операций**

На основании индивидуального варианта, можно утверждать, что выполняются следующие операции над однонаправленным списком: добавление нового узла в начало списка, удаление узла по ключу, вывод элементов списка на экран, поиск самых длинных последовательностей одинаковых узлов, оставление по одному элементу из каждой группы одинаковых элементов и создание нового списка с уникальными цифровыми значениями.

#### 2.2.2.1 Добавление узла в начале списка

Узел содержит два поля данных: data (переменная типа char, которая хранит информацию (значение) узла), next (указатель на следующий узел в списке. Тип указателя Node\*).

Конструктор Node принимает аргумент char d, который инициализирует поле data узла. Поле next инициализируется значением nullptr, указывая на то, что изначально узел не имеет следующего узла в списке. Рассмотрим выполнение операции добавления нового узла в начало списка, алгоритм которой представлен на рисунке 2. Реализация кода предаставлено в блок коде 2. Для проверки работы кода приведены данные в таблице 1.

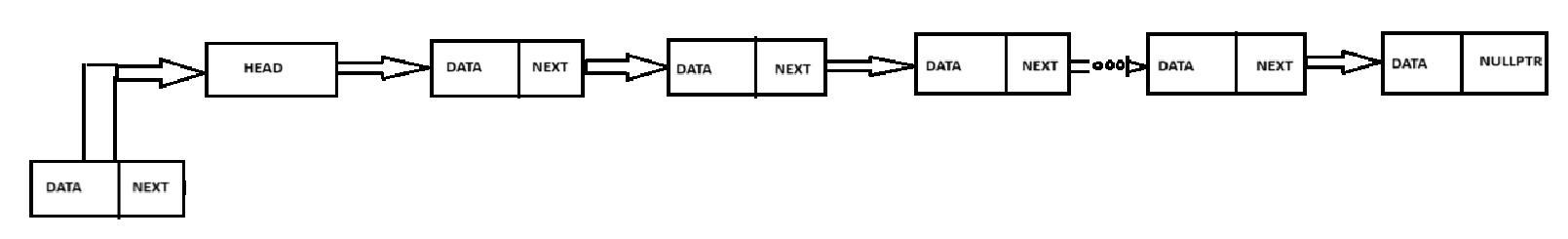


Рисунок 2 - Изображение добавление нового узла

| void insertBeforeFirst(Node\*& head, char key) {  Node\* newNode = createNode(key);  newNode->next = head;  head = newNode; } |
| --- |

Блок кода 2 - Добавление нового узла в начало списка

Таблица 1 - Данные для тестирования

| № | Входные данные | | Добавить | Ожидаемый результат | |
| --- | --- | --- | --- | --- | --- |
| L1 | L2 | L1 | L2 |
| 1 | z z z z z | h a p p y | s | s z z z z z | s h a p p y |
| 2 | s a d | i 4 3 | 5 | 5 s a d | 5 i 4 3 |
| 3 | y y y y | y y y y | r | r y y y y | r y y y y |

#### 2.2.2.2 Удаление узла по ключу.

Создаются временные указатели temp и prev, которые используются для перемещения по списку и отслеживания предыдущего узла. Цикл выполняется до тех пор, пока указатель temp не достигнет конца списка (т.е., не станет равным nullptr) или пока не будет найден узел с ключом, равным key. Это позволяет найти нужный узел для удаления. Если узел с ключом key не найден, выводится сообщение об этом, и функция завершает свою работу. Если найденный узел не первый в списке, то его предыдущий узел prev перенаправляется на следующий узел после удаляемого. Если удаляемый узел является первым в списке, то указатель head перенаправляется на следующий узел после удаляемого, чтобы обновить начало списка. Удаляется узел, который содержит указанный ключ key, освобождая выделенную память.

Отобразим выполнение данной операции на рисунке 3.

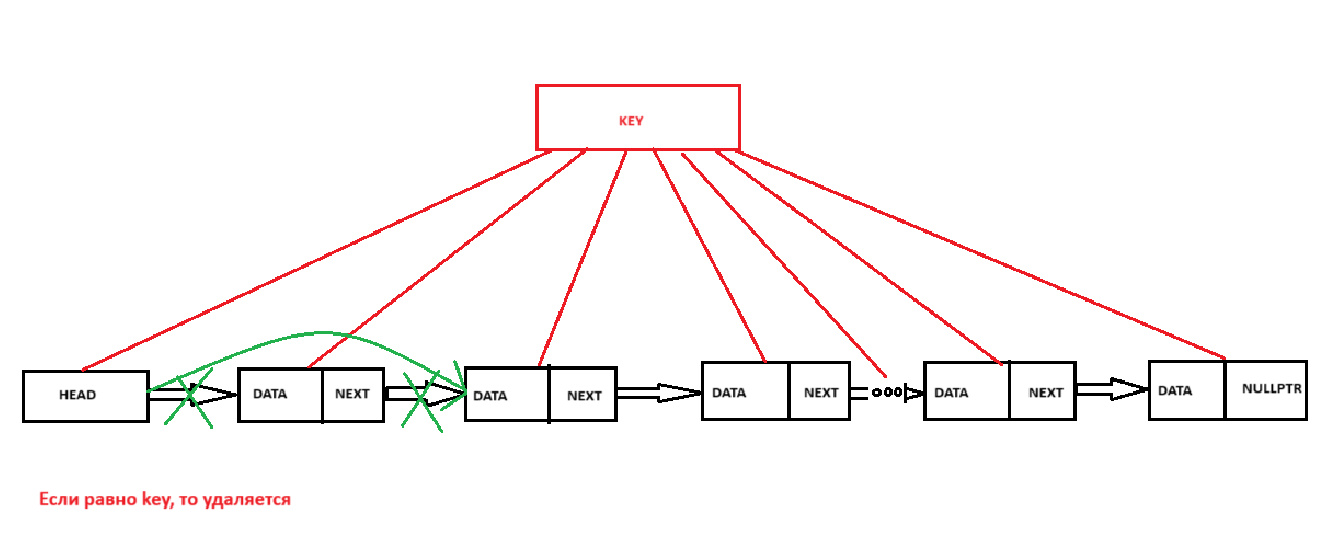


Рисунок 3 - Изображение удаления узла по ключу

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

| void deleteNodeByKey(Node\*& head, char key) {  Node\* temp = head;  Node\* prev = nullptr;   while (temp != nullptr && temp->data != key) {  prev = temp;  temp = temp->next;  }   if (temp == nullptr) {  cout << "Узел с ключом " << key << " не найден." << endl;  return;  }   if (prev != nullptr) {  prev->next = temp->next;  } else {  head = temp->next;  }   delete temp; } |
| --- |

Блок кода 3 - Удаление узла по ключу

Таблица 2 - Данные для тестирования

| № | Входные данные | | Удалить | Ожидаемый результат | |
| --- | --- | --- | --- | --- | --- |
| L1 | L2 | L1 | L2 |
| 1 | s z z z z z | s h a p p y | s | z z z z z | h a p p y |
| 2 | 5 s a d | 5 i 4 3 | 5 | s a d | i 4 3 |
| 3 | r y y y y | r y y y y | r | y y y y | y y y y |

#### 2.2.2.3 Вывод элементов списка на экран.

Создается временный указатель temp, который устанавливается в начало списка (head). Это позволяет сохранить начальную ссылку на список и избежать изменения переменной head, что может быть важно для последующего использования списка. Цикл выполняется до тех пор, пока указатель temp не достигнет конца списка (т.е., не станет равным nullptr). Это обеспечивает итерацию по всему списку. На каждой итерации цикла выводится значение поля data текущего узла (temp) на консоль, с последующим пробелом. Таким образом, выводится содержимое каждого узла списка. Указатель temp перемещается на следующий узел списка, указывая на него через поле next. Это обеспечивает перемещение по списку и переход к следующему элементу. После завершения цикла выводится символ новой строки для отделения вывода списка от других сообщений или данных в консоли. Это обеспечивает более читаемый вывод списка.

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

| void printList(Node\* head) {  Node\* temp = head;  while (temp != nullptr) {  cout << temp->data << " ";  temp = temp->next;  }  cout << endl; } |
| --- |

Блок кода 4 - Вывод списка на экран

Таблица 3 - Данные для тестирования

| № | Входные данные | | Ожидаемый результат | |
| --- | --- | --- | --- | --- |
| L1 | L2 | L1 | L2 |
| 1 | z z z z z | h a p p y | z z z z z | h a p p y |
| 2 | s a d | i 4 3 | s a d | i 4 3 |
| 3 | y y y y | y y y y | y y y y | y y y y |

#### 2.2.2.4 Проверка на равенство списков L1 и L2

Цикл выполняется до тех пор, пока указатели l1 и l2 не достигнут конца своих соответствующих списков или пока не будет найдено несоответствие между узлами списков. Это позволяет сравнить каждый узел из двух списков. Если данные текущих узлов l1 и l2 не совпадают, то списки не равны, и функция возвращает false. Указатели l1 и l2 сдвигаются на следующие узлы в своих списках для продолжения сравнения.Если оба указателя достигли конца своих списков одновременно (т.е., оба равны nullptr), это означает, что все узлы списков были совпадающими, и функция возвращает true. Если один из указателей не равен nullptr, это означает, что списки имеют разную длину, и функция также вернет false.

Отобразим выполнение данной операции на рисунке 4.

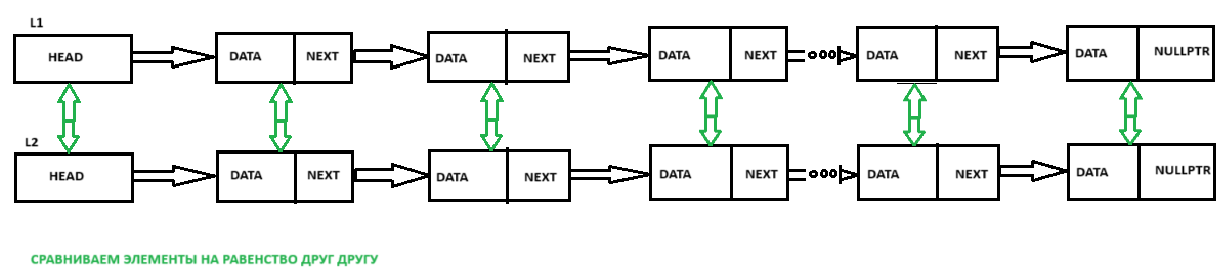


Рисунок 4 - Проверка на равенство списков

Реализация данного алгоритма представлена в блок коде 5 под именем Same. Данные для тестирования будут приведены в таблице 4.

| bool isEqual(Node\* l1, Node\* l2) {  while (l1 != nullptr && l2 != nullptr) {  if (l1->data != l2->data) {  return false;  }  l1 = l1->next;  l2 = l2->next;  }  return l1 == nullptr && l2 == nullptr; } |
| --- |

Блок кода 5 - Проверка на равенство списков

Таблица 4 - Данные для тестирования

| № | Входные данные | | Ожидаемый результат | |
| --- | --- | --- | --- | --- |
| L1 | L2 |
| 1 | z z z z z | h a p p y | Списки L1 и L2 не равны. | |
| 2 | s a d | i 4 3 | Списки L1 и L2 не равны. | |
| 3 | y y y y | y y y y | Списки L1 и L2 равны. | |

#### 2.2.2.5 Вставление в список L1 последний элемент списка L2

Если список l1 пустой, то мы просто присваиваем ему список l2. Это позволяет обрабатывать случай, когда список l1 изначально пустой. Проходимся по списку l1 до его последнего элемента. Это нужно для того, чтобы найти текущий последний элемент и его указатель next установить на первый элемент списка l2. Устанавливаем указатель next последнего элемента списка l1 на первый элемент списка l2, вставляя таким образом весь список l2 в конец списка l1.

Отобразим выполнение данной операции на рисунке 5.

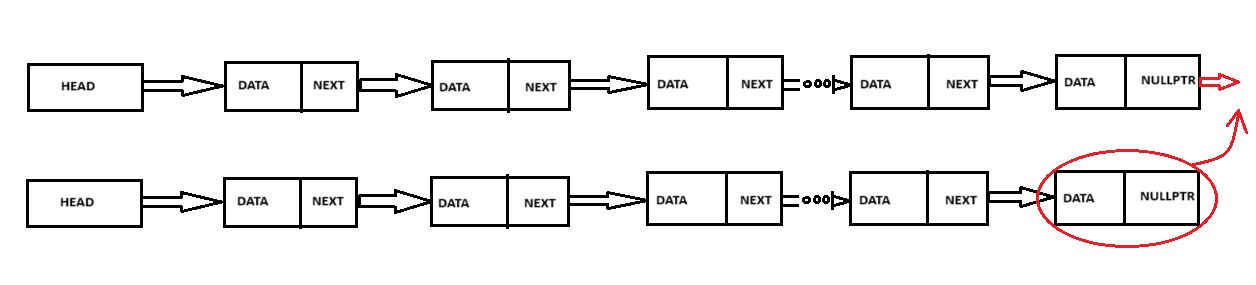


Рисунок 5 - Изобразим оставление одного элемента из каждой группы одинаковых элементов

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

| void insertLastOfL2IntoL1(Node\*& L1, Node\*& L2) {  if (L2 == nullptr) {  return;  }   Node\* lastElem = L2;  while (lastElem->next != nullptr) {  lastElem = lastElem->next;  }   Node\* current = L1;  while (current->next != nullptr) {  current = current->next;  }   current->next = new Node(lastElem->data); } |
| --- |

Блок кода 6 - Вставка в L1 последнего элемента L2

Таблица 5 - Данные для тестирования

| № | Входные данные | | Ожидаемый результат | |
| --- | --- | --- | --- | --- |
| L1 | L2 |
| 1 | z z z z z | h a p p y | z z z z h | |
| 2 | s a d | i 4 3 | s a d 3 | |
| 3 | y y y y | y y y y | y y y y y | |

#### 2.2.2.6 Удаление из списка L2, узлы, содержащие цифровые значения

Начинаем с указателя temp, указывающего на голову списка head. Затем мы перебираем все узлы списка. Проверяем, содержит ли текущий узел цифру. Если да, то мы удаляем текущий узел из списка, перенаправляя указатель предыдущего узла на следующий узел после текущего. Если текущий узел - это головной узел списка, то мы обновляем голову списка на следующий узел после текущего. Затем мы освобождаем память, выделенную для удаляемого узла. Далее мы перемещаем указатель temp на следующий узел, чтобы продолжить перебор списка. Если текущий узел не содержит цифру, мы просто обновляем указатель prev на текущий узел и перемещаем указатель temp на следующий узел.

Отобразим выполнение данной операции на рисунке 6.

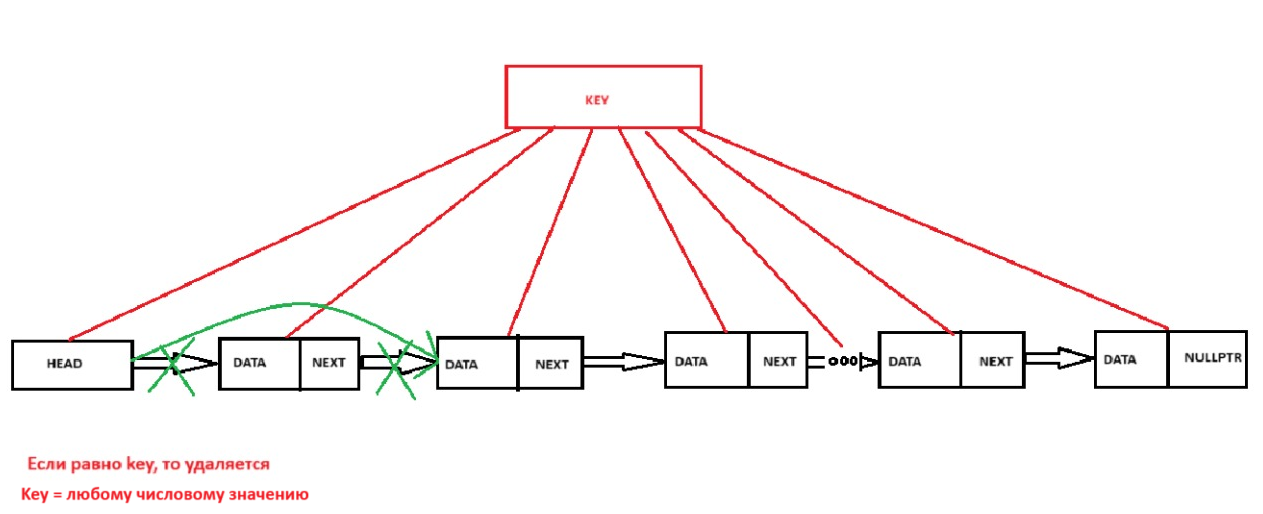


Рисунок 6 - Удаление узлов с числовыми значениями

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

| void deleteDigits(Node\*& head) {  Node\* temp = head;  Node\* prev = nullptr;   while (temp != nullptr) {  if (isdigit(temp->data)) {  if (prev != nullptr) {  prev->next = temp->next;  } else {  head = temp->next;  }  delete temp;  temp = prev != nullptr ? prev->next : head;  } else {  prev = temp;  temp = temp->next;  }  } } |
| --- |

Блок кода 7 - Удаление из списка L2, узлы, содержащие цифровые значения

Таблица 6 - Данные для тестирования

| № | Входные данные | | Ожидаемый результат | |
| --- | --- | --- | --- | --- |
| L1 | L2 |
| 1 | z z z z z | h a p p y | h a p p y | |
| 2 | s a d | i 4 3 | i | |
| 3 | y y y y | y y y y | y y y y | |

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

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

Для реализации данного алгоритма на языке C++ (блок кода 8) требуется использование стандартной библиотеки iostream. Эта библиотека предоставляет классы, функции и переменные для организации ввода и вывода данных через стандартные потоки: cout для вывода на консоль и cin для ввода с консоли.

В рамках программы также будут использованы классы, которые являются абстракциями, описывающими методы и свойства объектов, а также объекты, которые представляют конкретные экземпляры этих классов.

Кроме того, в программе будут использоваться структуры, которые представляют собой производные типы данных и описывают определенные сущности, аналогично классам.

| #include <iostream>  using namespace std;  // Структура узла списка struct Node {  char data;  Node\* next;  Node(char d) : data(d), next(nullptr) {} };  // Функция создания нового узла Node\* createNode(char key) {  return new Node(key); }  // Функция вставки нового узла перед первым узлом void insertBeforeFirst(Node\*& head, char key) {  Node\* newNode = createNode(key);  newNode->next = head;  head = newNode; }  // Функция удаления узла по ключу void deleteNodeByKey(Node\*& head, char key) {  Node\* temp = head;  Node\* prev = nullptr;   while (temp != nullptr && temp->data != key) {  prev = temp;  temp = temp->next;  }   if (temp == nullptr) {  cout << "Узел с ключом " << key << " не найден." << endl;  return;  }   if (prev != nullptr) {  prev->next = temp->next;  }  else {  head = temp->next;  }   delete temp; }  void printList(Node\* head) {  Node\* current = head;  while (current != nullptr) {  cout << current->data << " ";  current = current->next;  }  cout << endl; }  // Функция для создания нового списка вручную Node\* createManualList() {  Node\* head = nullptr;  char key;  cout << "Введите элементы списка (введите '.' для завершения ввода): ";  while (cin >> key && key != '.') {  insertBeforeFirst(head, key);  }  return head; }  // Функция для использования уже готового списка Node\* useReadyList(char\* keys, int n) {  Node\* head = nullptr;  for (int i = 0; i < n; ++i) {  insertBeforeFirst(head, keys[i]);  }  return head; }  // Функция для проверки на равенство двух списков bool isEqual(Node\* l1, Node\* l2) {  while (l1 != nullptr && l2 != nullptr) {  if (l1->data != l2->data) {  return false;  }  l1 = l1->next;  l2 = l2->next;  }  return l1 == nullptr && l2 == nullptr; }  // Функция для вставки в список L1 последнего элемента списка L2 void insertLastOfL2IntoL1(Node\*& L1, Node\*& L2) {  if (L2 == nullptr) {  return;  }   Node\* lastElem = L2;  while (lastElem->next != nullptr) {  lastElem = lastElem->next;  }   Node\* current = L1;  while (current->next != nullptr) {  current = current->next;  }   current->next = new Node(lastElem->data); }  // Функция для удаления из списка L2 узлов, содержащих цифровые значения void deleteDigits(Node\*& head) {  Node\* temp = head;  Node\* prev = nullptr;   while (temp != nullptr) {  if (isdigit(temp->data)) {  if (prev != nullptr) {  prev->next = temp->next;  }  else {  head = temp->next;  }  delete temp;  temp = prev != nullptr ? prev->next : head;  }  else {  prev = temp;  temp = temp->next;  }  } }  int main() {  setlocale(LC\_ALL, "RUS");  int choice, c;  Node\* L1 = nullptr;  Node\* L2 = nullptr;   cout << "Выберите 1(ручной ввод) и 2 (использования существующих данных): ";  cin >> choice;   if (choice == 1) {  cout << "Создание списка L1 вручную:" << endl;  L1 = createManualList();  cout << "Список L1: ";  printList(L1);   cout << "Создание списка L2 вручную:" << endl;  L2 = createManualList();  cout << "Список L2: ";  printList(L2);  }   else if (choice == 2) {  L1 = new Node('z');  L1->next = new Node('z');  L1->next->next = new Node('z');  L1->next->next->next = new Node('z');  L1->next->next->next->next = new Node('z');   L2 = new Node('h');  L2->next = new Node('a');  L2->next->next = new Node('p');  L2->next->next->next = new Node('p');  L2->next->next->next->next = new Node('y');  cout << "Список L1: ";  printList(L1);  cout << "Список L2: ";  printList(L2);  }   else {  cout << "Ошибка";  return 1;  }   do {  cout << "Меню:" << endl;  cout << "1. Вывод списка" << endl;  cout << "2. Добавление нового узла в начале списка" << endl;  cout << "3. Удаление узла с заданным значением" << endl;  cout << "4. Проверка на равенство списков" << endl;  cout << "5. Вставка в список L1 последнего элемента из списка L2" << endl;  cout << "6. Удаление из списка L2 узлов, содержащих цифровые значения" << endl;  cout << "7. Выход из программы" << endl;  cout << "Выберете дейсвие: ";   cin >> c;   switch (c) {   case 1:  cout << "Список L1: ";  printList(L1);  cout << "Список L2: ";  printList(L2);  break;   case 2:  char key;  cout << "Введите значение для нового узла: ";  cin >> key;  insertBeforeFirst(L1, key);  insertBeforeFirst(L2, key);  cout << "Список L1 после добавления нового узла: ";  printList(L1);  cout << "Список L2 после добавления нового узла: ";  printList(L2);  break;   case 3:  char delKey;  cout << "Введите значение узла для удаления: ";  cin >> delKey;  deleteNodeByKey(L1, delKey);  deleteNodeByKey(L2, delKey);  cout << "Список L1 после удаления узла: ";  printList(L1);  cout << "Список L2 после удаления узла: ";  printList(L2);  break;   case 4:  cout << "Списки L1 и L2 " << (isEqual(L1, L2) ? "равны." : "не равны.") << endl;  break;   case 5:  insertLastOfL2IntoL1(L1, L2);  cout << "Список L1 после вставки последнего элемента из L2: ";  printList(L1);  break;   case 6:  deleteDigits(L2);  cout << "Список L2 после удаления цифровых значений: ";  printList(L2);  break;   case 7:  cout << "Выход из программы." << endl;  break;   default:  cout << "Ошибка" << endl;  }  } while (c != 7);   return 0; } |
| --- |

Блок кода 8 – Программа однонаправленного списка с выбором операций

### **2.3.2 Тестирование**

Проведем тестирование основываясь на данные из таблиц 1-6. Элементы готового списка являются №1 строкой таблиц 1-6. Следовательно для тестирования строки №1 данные вручную не вводятся. Для строк №2-3 данные вводятся вручную. Результаты тестирования №1 строки представлены на рисунках 6-7.

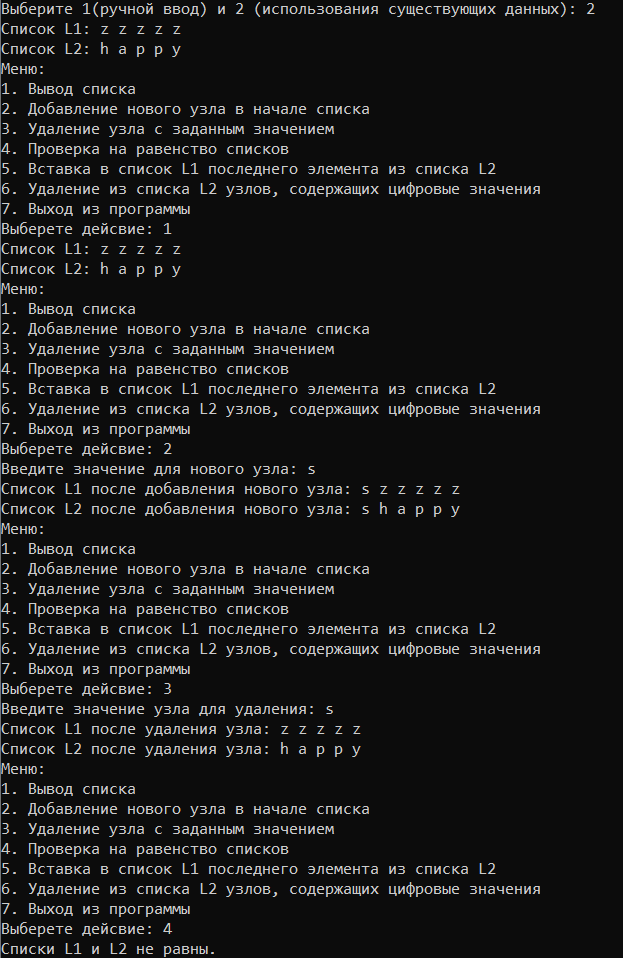


Рисунок 6 - Тестирование программы на данных №1 строки

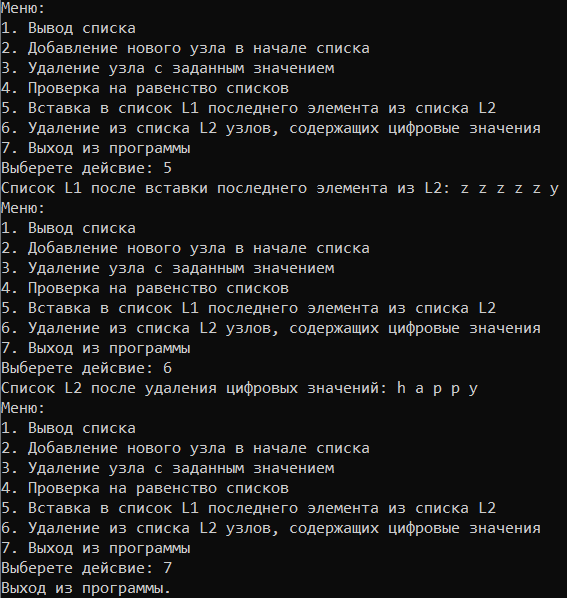


Рисунок 7 - Тестирование программы на данных №1 строки

Проведение тестирования на основе №1 строки таблиц 1-6, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

Результаты тестирования №2 строки представлены на рисунке 8-9.

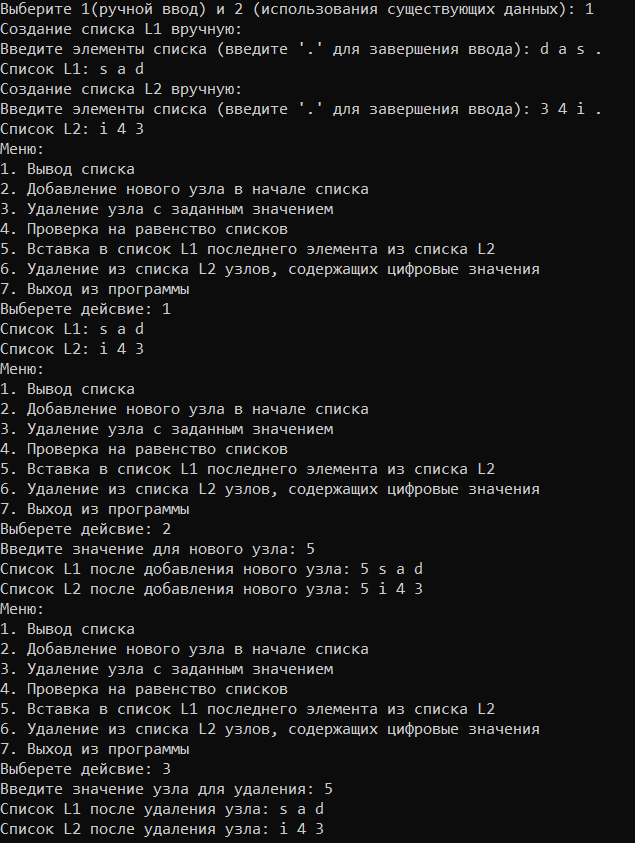


Рисунок 8 - Тестирование программы на данных №2 строки

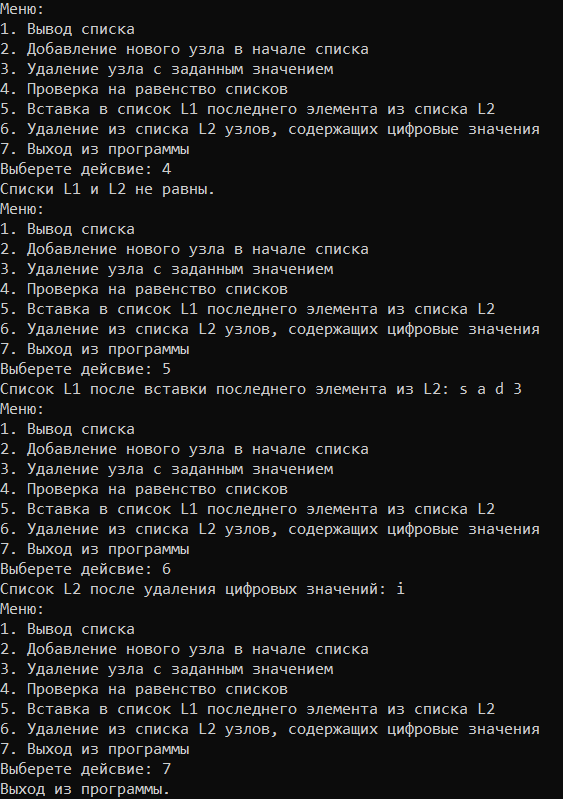


Рисунок 9 - Тестирование программы на данных №2 строки

Проведение тестирования на основе №2 строки таблиц 1-6, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

Результаты тестирования №3 строки представлены на рисунке 10-11.

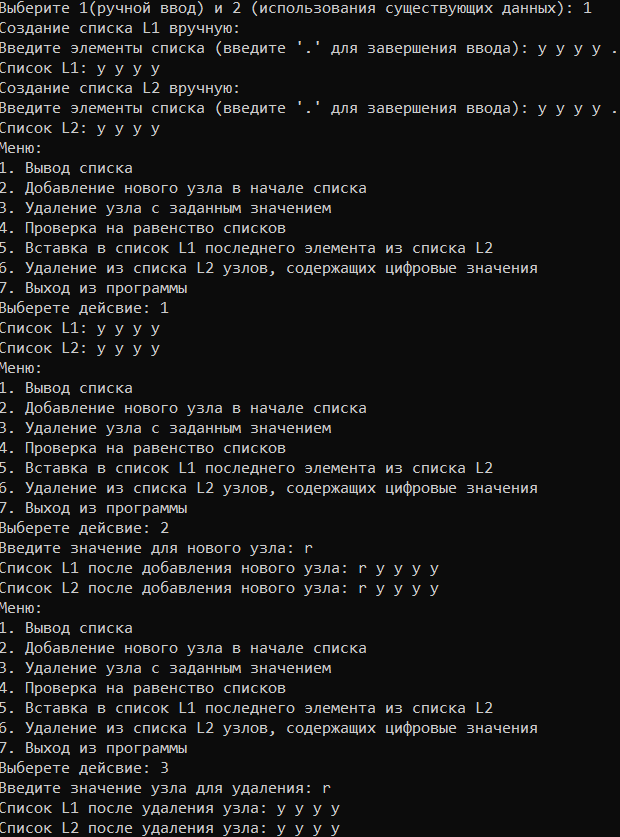


Рисунок 10 - Тестирование программы на данных №3 строки

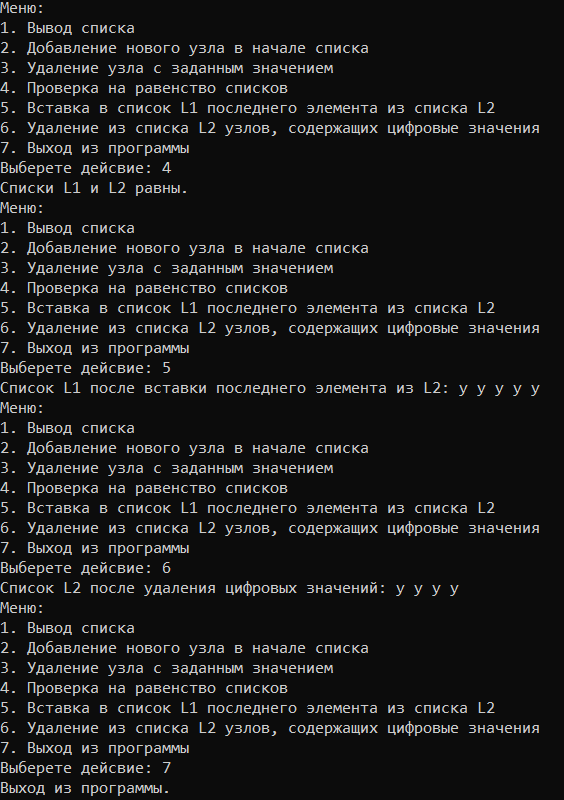


Рисунок 11 - Тестирование программы на данных №3 строки

Проведение тестирования на основе №3 строки таблиц 1-6, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

## **2.4 Вывод по заданию**

Однонаправленные списки - это структура данных, состоящая из узлов, каждый из которых содержит данные и ссылку на следующий узел. Они обеспечивают эффективное выполнение операций добавления и удаления элементов, так как для доступа к любому элементу необходимо последовательно перебирать все предыдущие элементы. Однонаправленные списки представляют собой удобную структуру данных для хранения и работы с последовательностями элементов и находят применение в различных задачах, требующих динамического управления данными.

Однонаправленный список поддерживает следующие операции:

* Вставка нового элемента: Добавление нового элемента в начало списка или в конец списка;
* Удаление элемента: Удаление элемента по его значению или позиции в списке;
* Поиск элемента: Поиск элемента по его значению в списке;
* Получение элемента: Получение значения элемента по его позиции в списке;
* Обход списка: Последовательный доступ к элементам списка для чтения или модификации значений;
* Проверка наличия элементов: Проверка, пуст ли список или содержит ли он определенный элемент;
* Получение размера списка: Получение количества элементов в списке;
* Объединение списков: Слияние двух списков в один;
* И многие другие.

Эти операции позволяют эффективно управлять данными в однонаправленном списке, делая его универсальной структурой данных для множества задач.

# **3 КОНТРОЛЬНЫЕ ВОПРОСЫ**

1) Расскажите о трех уровнях представления данных в программной системе.

1. Физический уровень - хранение данных в битах и байтах.

2. Логический уровень - описание логической структуры данных.

3. Представление уровень - отображение данных в удобном для пользователя формате.

2) Что определяет тип данных?

Тип данных определяет формат, структуру и значения, которые может принимать определенная переменная или объект в программе. Тип данных определяет, какие операции можно выполнить с этими данными, как они будут храниться в памяти компьютера и какие правила применяются к их использованию. Тип данных также определяет размер памяти, выделяемый для переменной, и диапазон значений, которые могут быть ею сохранены.

3) Что определяет структура данных?

Структура данных определяет организацию и хранение данных в программе. Она определяет, как данные будут связаны и какие операции можно выполнять с ними.

4) Расскажите о структуры хранения данных в компьютерных технологиях.

В компьютерных технологиях данные хранятся в различных структурах данных:

1. Массивы: набор элементов, доступ к которым осуществляется по индексу. Используются для хранения коллекций однотипных данных.

2. Стеки: структура данных "последним пришел - первым ушел". Элементы добавляются и удаляются с вершины стека.

3. Очереди: структура данных "первым пришел - первым ушел". Элементы добавляются в конец очереди и удаляются с начала.

4. Связанные списки: элементы связаны ссылками. В двусвязном списке каждый элемент имеет ссылки на следующий и предыдущий элементы.

5. Деревья: структура данных, состоящая из узлов и ребер. Каждый узел может иметь нуль или более дочерних узлов.

6. Хеш-таблицы: используют хеш-функцию для преобразования ключей в индексы, обеспечивая быстрый доступ к данным.

5) Дайте определение линейной структуре данных.

Линейная структура данных - это структура данных, в которой элементы хранятся и обрабатываются последовательно, в одной линии. Каждый элемент имеет только одного предшественника и одного последователя, образуя цепочку элементов. Примеры линейных структур данных включают массивы, списки, стеки и очереди.

6) Дайте определение структуре данных линейный список.

Линейный список - это структура данных, в которой элементы хранятся последовательно, каждый из которых содержит информацию и ссылку на следующий элемент. Каждый элемент в линейном списке связан с предыдущим элементом и последующим элементом. Линейные списки могут быть однонаправленными (ссылка только на следующий элемент) или двунаправленными (ссылки на предыдущий и следующий элементы).

7) Дайте определение структуре данных стек.

Стек - это структура данных, которая работает по принципу Last-In-First-Out (LIFO), что означает, что последний добавленный элемент будет первым удаленным. Элементы добавляются и удаляются только с вершины стека. Операции добавления элемента в стек называются "push", а удаления - "pop".

8) Дайте определение структуре данных очередь.

Очередь - это структура данных, которая работает по принципу First-In-First-Out (FIFO), что означает, что первый добавленный элемент будет первым удаленным. Элементы добавляются в конец очереди (enqueue), а удаляются из начала (dequeue). Очередь используется в ситуациях, где требуется учет порядка поступления элементов, например, при организации задач для выполнения.

9) Чем стек отличается от структуры данных линейный список?

Основное различие между стеком и линейным списком заключается в порядке обработки элементов:

1. Стек (stack): Работает по принципу Last-In-First-Out (LIFO). Элементы добавляются и удаляются только с вершины стека с использованием операций push и pop.

2. Линейный список (linear list): Позволяет произвольный доступ к элементам для добавления, удаления и обновления. Элементы хранятся последовательно и могут быть доступны для обработки в любом месте списка.

Таким образом, стек обрабатывает элементы в порядке, обратном порядку их добавления, в то время как линейный список позволяет произвольный доступ к элементам.

10) Какой из видов линейных списков лучше использовать, если нужно введенную последовательность вывести наоборот?

Для вывода введенной последовательности наоборот лучше использовать двунаправленный линейный список (двусвязный список), который позволяет эффективно перемещаться как вперед, так и назад по элементам списка. Это делает обращение списка более эффективным и удобным для реализации задачи вывода элементов в обратном порядке.

11) Определите сложность алгоритма операции вставки элемента в i-ую позицию: а) массива; б) линейного списка.

а) Для массива операция вставки элемента в i-ую позицию имеет сложность O(n), так как может потребоваться сдвигать все элементы после i-ой позиции на одну позицию вправо, чтобы освободить место для нового элемента.

б) Для линейного списка операция вставки элемента в i-ую позицию имеет также сложность O(n), так как требуется пройти от начала списка до i-ой позиции, чтобы вставить новый элемент. Для этого нужно изменить ссылки на предыдущий и следующий элементы, связанные с i-ым элементом.

Таким образом, как в массиве, так и в линейном списке операция вставки элемента в i-ую позицию имеет сложность O(n).

12) Определите сложность алгоритма операции удаления элемента из i-ой позиции: а) массива; б) линейного списка.

а) Для массива операция удаления элемента из i-ой позиции также имеет сложность O(n), так как после удаления элемента из i-ой позиции может потребоваться сдвигать все элементы, находящиеся после i-ой позиции, на одну позицию влево, чтобы заполнить удаленное место.

б) Для линейного списка операция удаления элемента из i-ой позиции также имеет сложность O(n), поскольку перед удалением из i-ой позиции нужно связать предыдущий элемент с элементом, следующим за i-ым, а также изменить ссылку на i-ый элемент, чтобы не утратить целостность списка.

Итак, как в массиве, так и в линейном списке операция удаления элемента из i-ой позиции имеет сложность O(n).

13) В чем суть трюка Вирта при выполнении операции удаления элемента из списка?

Трюк Вирта (Wirth's trick) - это метод удаления элемента из односвязного связного списка, который позволяет выполнить операцию удаления за время \( O(1) \), то есть за постоянное время, независимо от размера списка. Этот метод основан на том, что элемент, который нужно удалить, заменяется следующим за ним элементом, а затем следующий элемент удаляется.

Основные шаги трюка Вирта:

1. Найдите элемент, который нужно удалить, и элемент, предшествующий ему.

2. Измените указатель предшествующего элемента так, чтобы он указывал на элемент, следующий за удаляемым элементом.

3. Удалите удаляемый элемент.

После выполнения этих шагов удаляемый элемент больше не будет доступен через список, и операция удаления будет выполнена за постоянное время.

Этот трюк полезен, когда операция удаления из списка должна быть выполнена быстро, без необходимости обходить весь список для поиска элемента, который нужно удалить. Однако он применим только в односвязном списке, поскольку для его работы требуется доступ к предыдущему элементу.

Трюк Вирта (Wirth's trick) - метод удаления элемента из односвязного связного списка за время \( O(1) \), заменяя элемент, который нужно удалить, на следующий за ним элемент, а затем удаляя следующий элемент.

14) Определите структур узла однонаправленного списка.

Структура узла однонаправленного списка обычно содержит два основных поля:

Значение (или данные): Это информация, которую хранит каждый узел списка. Значение может быть любого типа данных, например, целое число, строка, объект и т. д.

Указатель на следующий узел (или ссылка): Это указатель на следующий узел списка. Он позволяет связать узлы между собой, образуя цепочку. Если узел последний в списке, указатель обычно указывает на нулевой указатель или нулевое значение, чтобы обозначить конец списка.

Структура узла однонаправленного списка содержит два поля: значение узла и указатель на следующий узел.

15) Реализуйте алгоритм вывода линейного однонаправленного списка.

| #include <iostream>  template <typename T> struct Node {  T data;  Node\* next;    Node(const T& value) : data(value), next(nullptr) {} };  template <typename T> void printLinkedList(Node<T>\* head) {  Node<T>\* current = head;  while (current != nullptr) {  std::cout << current->data << " ";  current = current->next;  }  std::cout << std::endl; }  int main() {  // Пример списка: 1 -> 2 -> 3 -> nullptr  Node<int>\* head = new Node<int>(1);  head->next = new Node<int>(2);  head->next->next = new Node<int>(3);    // Вывод списка  cout << "Список: ";  printLinkedList(head);   // Освобождение памяти  Node<int>\* current = head;  while (current != nullptr) {  Node<int>\* next = current->next;  delete current;  current = next;  }   return 0; } |
| --- |

16) Приведите фрагмент кода программы на языке С++ выполнения операции перемещения последнего элемента в начало списка.

| #include <iostream>  template <typename T> struct Node {  T data;  Node\* next;    Node(const T& value) : data(value), next(nullptr) {} };  template <typename T> void moveLastToFront(Node<T>\*& head) {  if (head == nullptr || head->next == nullptr) {  // Если список пуст или содержит только один элемент, перемещать нечего  return;  }   Node<T>\* prev = nullptr;  Node<T>\* current = head;    // Доходим до последнего элемента и запоминаем его и предыдущий узел  while (current->next != nullptr) {  prev = current;  current = current->next;  }   // Перемещаем последний элемент в начало списка  prev->next = nullptr;  current->next = head;  head = current; }  int main() {  // Создаем список: 1 -> 2 -> 3 -> nullptr  Node<int>\* head = new Node<int>(1);  head->next = new Node<int>(2);  head->next->next = new Node<int>(3);    cout << "Исходный список: ";  printLinkedList(head);   // Перемещаем последний элемент в начало списка  moveLastToFront(head);   cout << "Список после перемещения последнего элемента в начало: ";  printLinkedList(head);   // Освобождение памяти  Node<int>\* current = head;  while (current != nullptr) {  Node<int>\* next = current->next;  delete current;  current = next;  }   return 0; } |
| --- |

17) Какое из действий лишнее в следующем фрагменте кода? Куда вставляется новый узел?

| struct Node{ int info;  Node\*next; }; typedef Node \*List; List L=new List; void insertToList(List LL, int x){  List q=new Node; q->info=x; q->next=0;  if (LL==nullptr) LL->next=q;  else  q->next=LL->next;  LL->next=q; } |
| --- |

В данном фрагменте кода лишним является создание нового указателя типа List и его инициализация с помощью new List. Поскольку List уже является указателем, необходимости в создании нового указателя на указатель нет.

Корректная версия фрагмента кода без лишнего действия выглядит так:

| struct Node {  int info;  Node\* next; };  typedef Node\* List;  void insertToList(List LL, int x) {  List q = new Node;  q->info = x;  q->next = nullptr;  if (LL == nullptr)  LL = q; // Вставляем новый узел в начало списка  else {  q->next = LL->next;  LL->next = q;  } } |
| --- |

Здесь новый узел вставляется либо в начало списка, если LL пуст, либо после текущего узла.

# 

# **4 ВЫВОДЫ**

В ходе выполнения практической работы были достигнуты следующие результаты:

* Освоены основные принципы работы с динамическим однонаправленным списком.
* Получены практические навыки работы с этой структурой данных.
* Проведен анализ структуры и операций, соответствующих индивидуальному варианту.
* Разработана программа для работы с выбранным типом данных.
* Реализованы функции для работы с этой структурой данных.
* Реализована возможность как ручного ввода данных, так и использования уже имеющихся списков для тестирования.
* Создано удобное меню выбора способа ввода данных и функций программы.
* Проведено тщательное тестирование программы с различными входными данными и методами ввода.

Таким образом, цель практической работы - овладение управлением динамическим однонаправленным списком - успешно достигнута.

# **5 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).