|  |
| --- |
| 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: Жаворонкова А.А. |
|  | 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](#_2et92p0)

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

[2.2.2 Процесс выполнения операций 6](#_3dy6vkm)

[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 19](#_2s8eyo1)

[2.4 Вывод по заданию 25](#_lnxbz9)

[3 КОНТРОЛЬНЫЕ ВОПРОСЫ 27](#_49x2ik5)

[4 ВЫВОДЫ 35](#_n59xv9kokqs4)

[5 ЛИТЕРАТУРА 36](#_2p2csry)

# **1 ЦЕЛЬ**

Получить знания и практические навыки управления динамическим однонаправленным списком.

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи (В списке №10, Вариант 10)**

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:

Дан линейный однонаправленный список L

1. Разработать функцию, определяющую в списке L самую длинную последовательность одинаковых символов.

2. Разработать функцию, которая в каждой последовательности одинаковых символов оставляет только один.

3. Разработать функцию, которая создает новый список из цифр исходного, выполняя вставку элемента в новый список в порядке возрастания цифр. В новом списке не может быть повторяющихся цифр.

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

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

Однонаправленный список (или односвязный список) - это динамическая структура данных, состоящая из узлов. Каждый узел будет иметь какое-то значение и указатель на следующий узел. Последний узел списка ссылается на NULL, что означает конец списка. Однонаправленные списки обладают преимуществами в сравнении с массивами, так как позволяют эффективно вставлять и удалять элементы, но требуют больше памяти для хранения указателей.

Определим структуру узла однонаправленного списка согласно варианту.

Структура Node представляет собой элемент односвязного списка. В данной структуре содержится переменная 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 присваивается значение nullptr. Данное описание структуры представлено в виде кода на C++ в блоке кода 1, а его изображение на рисунке 1.

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

Блок кода 1 - Структура

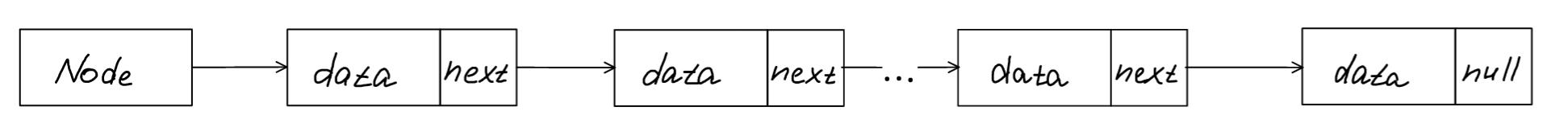


Рисунок 1 - Изображение структуры данных

### **2.2.2 Процесс выполнения операций**

На основании индивидуального варианта, можно сказать, что используются операции: добавления нового узла в начало списка, удаления узла по ключу, вывод элементов списка на экран, поиск самых длинных последовательностей одинаковых узлов, оставление по одному элементу из каждой группы одинаковых элементов, создание нового списка с уникальными цифровыми значениями.

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

1. Добавление нового узла в начало списка.

Отобразим выполнение данной операции на рисунке 2.

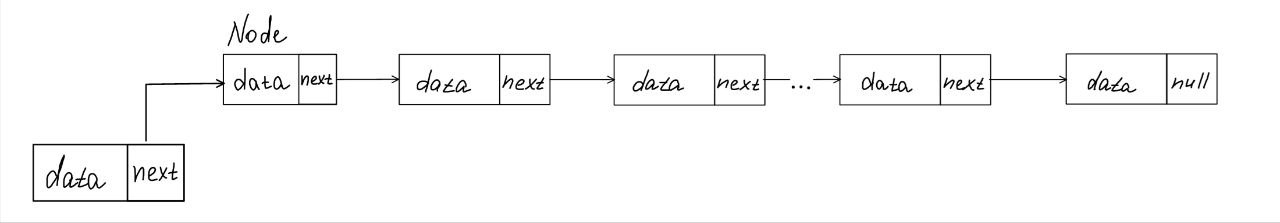


Рисунок 2 - Изображение добавление нового узла

Необходимо создать сам узел с заданным (переданным в функцию) значением. Создается новый узел с переданным значением, затем значение указателя на следующий узел этого нового узла устанавливается на текущий первый узел списка, и обновляется указатель head на новый узел.

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

| void insertFront(char newData) {  Node\* newNode = new Node(newData);  newNode->next = head;  head = newNode;  } |
| --- |

Блок кода 2 - Добавление нового узла в начале списка

Таблица 1 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 1 2 2 2 3 3 | 5 | 5 3 3 2 2 2 1 |
| 2 | a b c d e | 5 | 5 e d c b a |
| 3 |  | 5 | 5 |
| 4 | x x x x x x x x x x x x | 5 | 5 x x x x x x x x x x x x |
| 5 | 3 b b f c c c 1 2 3 3 a | 5 | 5 a 3 3 2 1 c c c f b b 3 |

1. Удаление узла по ключу.

Отобразим выполнение данной операции на рисунке 3.

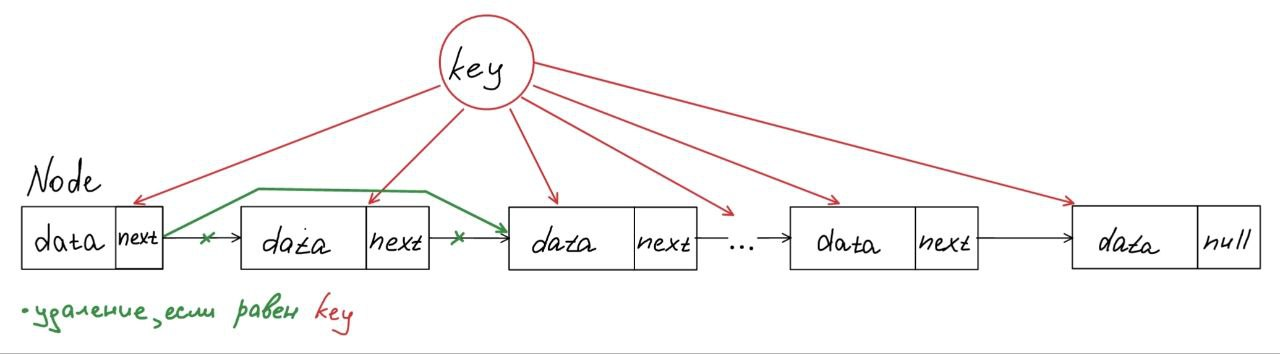


Рисунок 3 - Изображение удаления узла по ключу

Создаются указатели temp на первый узел, и prev – на следующий после первого. Затем, пока temp не пустой и пока значение текущего узла temp не равно ключу, при каждой итерации перенаправляем temp и prev на следующий после них узел. Если указатель prev пустой, то сообщаем об ошибке, иначе просто удаляем узел prev.

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

| void deleteNode(char key) {  Node\* temp = head;  Node\* prev = nullptr;  if (temp != nullptr && temp->data == key) {  head = temp->next;  delete temp;  return;  }  while (temp != nullptr && temp->data != key) {  prev = temp;  temp = temp->next;  }  if (temp == nullptr) {  return;  }  prev->next = temp->next;  delete temp;  } |
| --- |

Блок кода 3 - Удаление узла по ключу

Таблица 2 - Данные для тестирования

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 5 3 3 2 2 2 1 | 5 | 3 3 2 2 2 1 |
| 2 | 5 e d c b a | 5 | e d c b a |
| 3 | 5 | 5 |  |
| 4 | 5 x x x x x x x x x x x x | 5 | x x x x x x x x x x x x |
| 5 | 5 a 3 3 2 1 c c c f b b 3 | 5 | a 3 3 2 1 c c c f b b 3 |

1. Вывод элементов списка на экран.

В структуре Node в функции вывода всего списка (если список не пуст) направляем указатель temp на первый узел списка и выводим значения узлов, пока указатель temp не пустой. При каждой итерации перенаправляем temp на следующий узел.

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

| void printList() {  *// Начинаем обходить список, выводим значения узлов, переходим к следующему элементу, выводим символ новой строки*  Node\* temp = head;  while (temp != nullptr) {  cout << temp->data << " ";  temp = temp->next;  }  cout << endl;  } |
| --- |

Блок кода 4 - Вывод элементов на экран

Таблица 3 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 3 3 2 2 2 1 | 3 3 2 2 2 1 |
| 2 | e d c b a | e d c b a |
| 3 |  |  |
| 4 | x x x x x x x x x x x x | x x x x x x x x x x x x |
| 5 | a 3 3 2 1 c c c f b b 3 | a 3 3 2 1 c c c f b b 3 |

1. Поиск самых длинных последовательностей одинаковых элементов

Отобразим выполнение данной операции на рисунке 4.

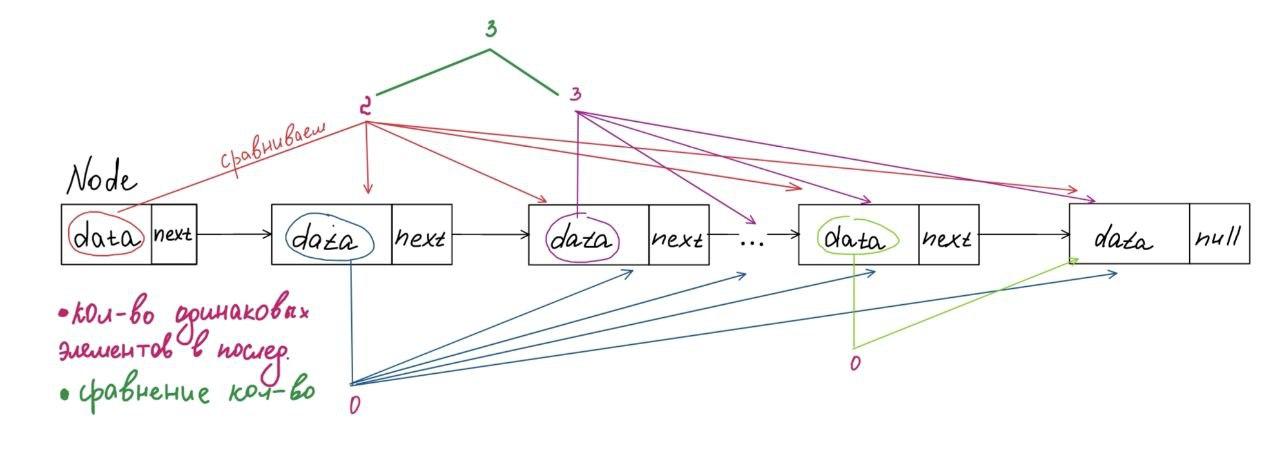


Рисунок 4 - Изображение поиска самых длинных последовательности одинаковых элементов

Установим указатель temp на начало списка head.Инициализируем переменные maxCount (макс. длина последовательности) и maxChar (символ самой длинной последовательности) как 0 и нулевой символ соответственно. Пока temp не пустой происходит инициализация count как 1 и currChar как символ из temp->data. Пока следующий узел не пустой и его данные равны currChar, увеличивать count и перемещать temp. Если count оказывается больше maxCount, то обновляется maxCount и maxChar. Переместить temp к следующему узлу. Вывести длину и символ самой длинной последовательности.

Этот алгоритм проходит по списку, находит самую длинную последовательность одинаковых символов и выводит информацию о ней.

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

| void longestSequence() {  Node\* temp = head;  int maxCount = 0;  char maxChar = '\0';  while (temp != nullptr) {  int count = 1;  char currChar = temp->data;  while (temp->next != nullptr && temp->next->data == currChar) {  count++;  temp = temp->next;  }  if (count > maxCount) {  maxCount = count;  maxChar = currChar;  }  temp = temp->next;  }  cout << "Длина самой длинной последовательности: " << maxCount << " состоящей из " << maxChar << endl;  } |
| --- |

Блок кода 5 - Поиск самых длинных последовательностей одинаковых узлов

Таблица 4 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 3 3 2 2 2 1 | Длина самой длинной последовательности: 3 состоящей из 2 |
| 2 | e d c b a | Длина самой длинной последовательности: e состоящей из 1 |
| 3 |  | Длина самой длинной последовательности: 0 состоящей из \0 |
| 4 | x x x x x x x x x x x x | Длина самой длинной последовательности: 12 состоящей из х |
| 5 | a 3 3 2 1 c c c f b b 3 | Длина самой длинной последовательности: 3 состоящей из с |

1. Оставить по одному элементу из каждой группы одинаковых элементов.

Отобразим выполнение данной операции на рисунке 5.

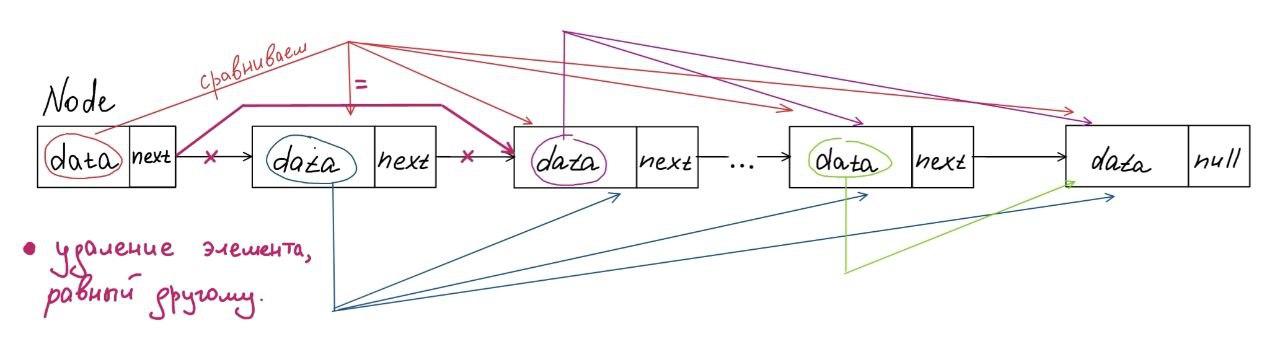


Рисунок 5 - Изобразим оставление одного элемента из каждой группы одинаковых элементов

Установим указатель temp на начало списка. Пока temp не достигнет конца списка (не станет равным nullptr), создаётся новый указатель curr и инициализируем его значением temp. Пока следующий узел после curr не является nullptr и данные этого узла равны данным узла, на который указывает temp, указатель curr перемещается к следующему узлу. Указываем, что следующий узел после temp равен следующему узлу после curr (удаляем все повторы текущего символа). Перемещаем указатель temp к следующему узлу.

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

| void leaveOneOfEach() {  Node\* temp = head;  while (temp != nullptr) {  Node\* curr = temp;  while (curr->next != nullptr && curr->next->data == temp->data) {  curr = curr->next;  }  temp->next = curr->next;  temp = temp->next;  }  } |
| --- |

Блок кода 6 - Оставление по одному элементу из каждой группы одинаковых элементов

Таблица 5 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 3 3 2 2 2 1 | 3 2 1 |
| 2 | e d c b a | e d c b a |
| 3 |  |  |
| 4 | x x x x x x x x x x x x | х |
| 5 | a 3 3 2 1 c c c f b b 3 | a 3 2 1 c f b 3 |

1. Создание нового списка с уникальными цифровыми значениями.

Создаем пустой вектор uniqueDigits для хранения уникальных цифр. Установим указатель temp на начало списка. Пока temp не достигнет конца списка (не станет равным nullptr), получаем цифру из данных узла, на который указывает temp. Если цифра является цифровым символом и отсутствует в векторе uniqueDigits, то добавляем цифру в конец вектора uniqueDigits. Указатель temp перемещается к следующему узлу. Создаем новый пустой связанный список newList. Все цифры в векторе uniqueDigits вставляем цифру в начало списка newList.

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

| void createNewListFromDigits() {  vector<char> uniqueDigits;  Node\* temp = head;  while (temp != nullptr) {  char digit = temp->data;  if (isdigit(digit) && std::find(uniqueDigits.begin(), uniqueDigits.end(), digit) == uniqueDigits.end()) {  uniqueDigits.push\_back(digit);  }  temp = temp->next;  }  LinkedList newList;  for (char digit : uniqueDigits) {  newList.insertFront(digit);  }  cout << "Новый список из цифр: ";  newList.printList();  } |
| --- |

Блок кода 7 - Создание нового списка с уникальными цифровыми значениями

Таблица 6 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 3 2 1 | 1 2 3 |
| 2 | e d c b a |  |
| 3 |  |  |
| 4 | х |  |
| 5 | a 3 2 1 c f b 3 | 1 2 3 |

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

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

Реализуем данный алгоритм на языке C++(блок кода 8). Для реализации понадобятся такие библиотеки, как iostream, random, chrono, fstream, algorithm, vector, string.

iostream: библиотека в C++, являющаяся заголовочным файлом с классами, функциями и переменными для предоставления ввода и вывода данных через стандартные потоки cout (для вывода на консоль) и cin (для ввода с консоли).

algorithm: библиотека в C++, содержащая широкий набор стандартных алгоритмов, таких как сортировка, поиск, преобразование контейнеров и другие.

vector: библиотека в C++, представляющая собой динамический массив, позволяющий хранить и управлять элементами одного типа данных.

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

В программе также будут использоваться структура (представляет собой производный тип данных, который представляет какую-то определенную сущность, также как и класс.)

| #include <iostream> #include <vector> #include <algorithm> using namespace std; *// Определяем структуру узла односвязного списка* struct Node {  char data;  Node\* next;  Node(char d) : data(d), next(nullptr) {} }; *// Определяем класс односвязного списка с приватным указателем на начало списка* class LinkedList { private:  Node\* head; public:  LinkedList() : head(nullptr) {}*// Конструктор класса LinkedList, инициализирует указатель на начало списка как nullptr*  *// Метод для добавления нового узла в начало списка*  void insertFront(char newData) {  *// Создаем новый узел, присваиваем ему значение, указываем, что следующий элемент - текущее начало, обновляем начало списка*  Node\* newNode = new Node(newData);  newNode->next = head;  head = newNode;  }  *// Метод для удаления узла с заданным значением*  void deleteNode(char key) {  *// Начинаем обходить список, проверяем наличие узла и его значение, удаляем его и обновляем указатели*  Node\* temp = head;  Node\* prev = nullptr;  if (temp != nullptr && temp->data == key) {  head = temp->next;  delete temp;  return;  }  while (temp != nullptr && temp->data != key) {  prev = temp;  temp = temp->next;  }  if (temp == nullptr) {  return;  }  prev->next = temp->next;  delete temp;  }  *// Метод для вывода элементов списка на экран*  void printList() {  *// Начинаем обходить список, выводим значения узлов, переходим к следующему элементу, выводим символ новой строки*  Node\* temp = head;  while (temp != nullptr) {  cout << temp->data << " ";  temp = temp->next;  }  cout << endl;  }  *// Подзадача 1. Нахождение самой длинной последовательности одинаковых символов*  void longestSequence() {  *// Начинаем обходить список, считаем последовательные одинаковые символы, выбираем максимальную длину и символ*  Node\* temp = head;  int maxCount = 0;  char maxChar = '\0';  while (temp != nullptr) {  int count = 1;  char currChar = temp->data;  while (temp->next != nullptr && temp->next->data == currChar) {  count++;  temp = temp->next;  }  if (count > maxCount) {  maxCount = count;  maxChar = currChar;  }  temp = temp->next;  }  cout << "Длинна самой длинной последовательности: " << maxCount << " состоящей из " << maxChar << endl;  }  *// Подзадача 2. Оставить по одному элементу из каждой группы одинаковых элементов*  void leaveOneOfEach() {  *// Начинаем обходить список, оставляем только один узел из каждой группы одинаковых элементов*  Node\* temp = head;  while (temp != nullptr) {  Node\* curr = temp;  while (curr->next != nullptr && curr->next->data == temp->data) {  curr = curr->next;  }  temp->next = curr->next;  temp = temp->next;  }  }  *// Подзадача 3. Создать новый список из уникальных цифр*  void createNewListFromDigits() {  *// Создаем новый вектор и начинаем обходить список, добавляем уникальные цифры в вектор, создаем новый список на основе вектора и выводим его*  vector<char> uniqueDigits;  Node\* temp = head;  while (temp != nullptr) {  char digit = temp->data;  if (isdigit(digit) && std::find(uniqueDigits.begin(), uniqueDigits.end(), digit) == uniqueDigits.end()) {  uniqueDigits.push\_back(digit);  }  temp = temp->next;  }  LinkedList newList;  for (char digit : uniqueDigits) {  newList.insertFront(digit);  }  cout << "Новый список из цифр: ";  newList.printList();  } }; int main() {  LinkedList list;*// Создаем объект класса LinkedList*  int choice, choice\_method;  cout << "Выберите 1 для ручного ввода данных и 2 для использования уже имеющихся данных: ";  cin >> choice;  if (choice == 1) {  int size;  char value;  cout << "Введите размер списка: ";  cin >> size;  for (int i = 0; i < size; i++) {  cout << "Введите значение: ";  cin >> value;  list.insertFront(value);  }  } else if (choice == 2) {  *// Готовый список для тестирования*  list.insertFront('3');  list.insertFront('b');  list.insertFront('b');  list.insertFront('f');  list.insertFront('c');  list.insertFront('c');  list.insertFront('c');  list.insertFront('1');  list.insertFront('2');  list.insertFront('3');  list.insertFront('3');  list.insertFront('a');  } else {  cout << "Ошибка";  return 1;  }  do {  cout << "Меню:" << endl;  cout << "1. Вывод списка на экран" << endl;  cout << "2. Добавление нового узла в начале списка" << endl;  cout << "3. Удаление узла с заданным значением" << endl;  cout << "4. Нахождение самой длинной последовательности одинаковых символов" << endl;  cout << "5. Оставить по одному элементу из каждой группы одинаковых элементов" << endl;  cout << "6. Создать новый список из уникальных цифр" << endl;  cout << "7. Выполнение всех операций по порядку" << endl;  cout << "8. Выход из программы" << endl;  cout << "Введите пункт от 1 до 8: ";    cin >> choice\_method;    switch(choice\_method) {  case 1:  cout << "Исходный список: ";  list.printList();  break;  case 2:  list.insertFront('5');  break;  case 3:  list.deleteNode('5');  break;  case 4:  list.longestSequence();  break;  case 5:  list.leaveOneOfEach();  break;  case 6:  list.createNewListFromDigits();  break;  case 7:  cout << "Исходный список: ";  list.printList();  *// Тестирование функций*  list.insertFront('5');  cout << "Список после добавления нового узла: ";  list.printList();  list.deleteNode('5');  cout << "Список после удаления нового узла: ";  list.printList();  list.longestSequence();  list.leaveOneOfEach();  cout << "Список после того, как остались по одному элементу из каждой группы одинаковых элементов: ";  list.printList();  list.createNewListFromDigits();  break;  case 8:  cout << "Выход из программы." << endl;  break;  default:  cout << "Неверный выбор. Попробуйте снова." << endl;  }  } while (choice\_method != 8);    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 №5 строкой таблиц 1-6. Следовательно для тестирования строки №5 данные вручную не вводятся. Для строк №1-4 данные вводятся вручную. №1 строка будет тестироваться с выбором каждой операции отдельно. Дальнейшие строки будут выполняться с выполнением всех операций по порядку. Результаты тестирования №1 строки представлены на рисунках 6-8.

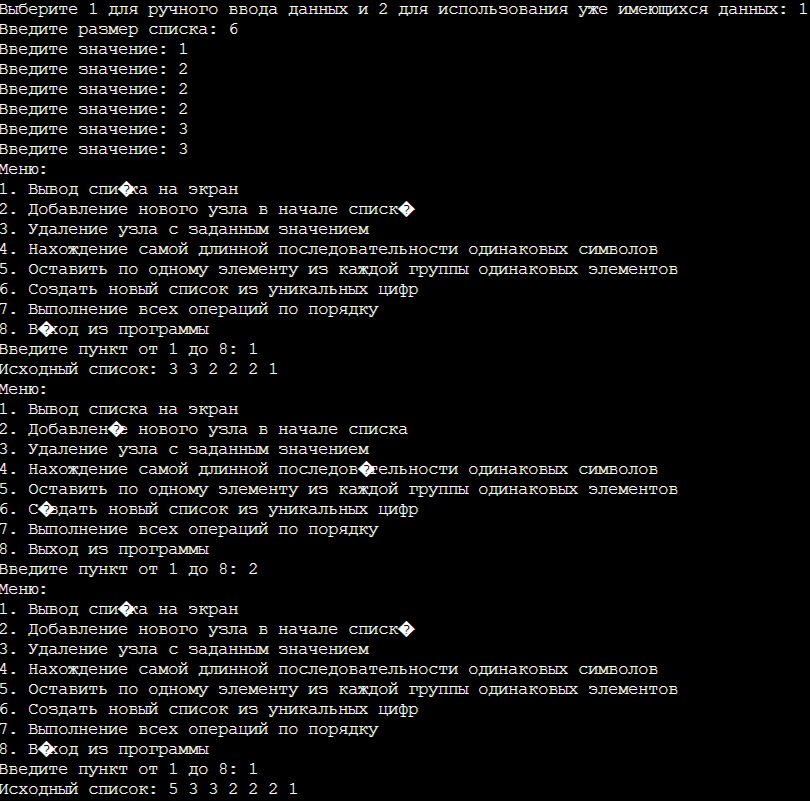


Рисунок 6 - Тестирование программы на данных №1 строки

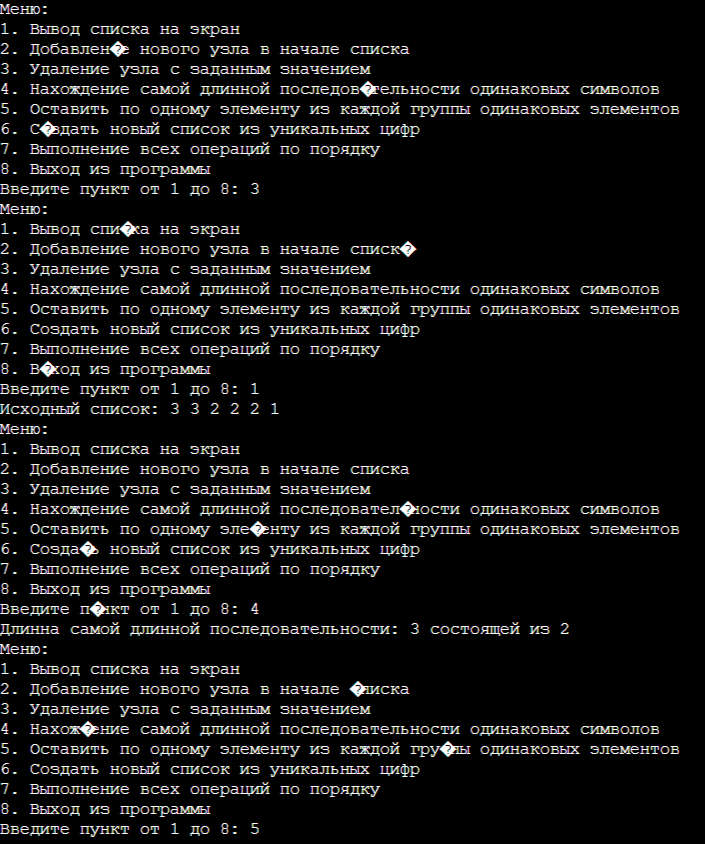


Рисунок 7 - Тестирование программы на данных №1 строки

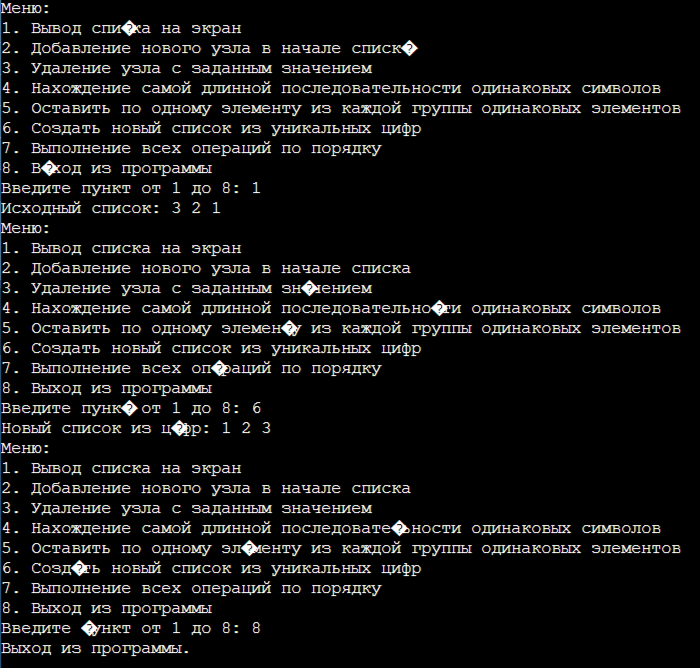


Рисунок 8 - Тестирование программы на данных №1 строки

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 9.

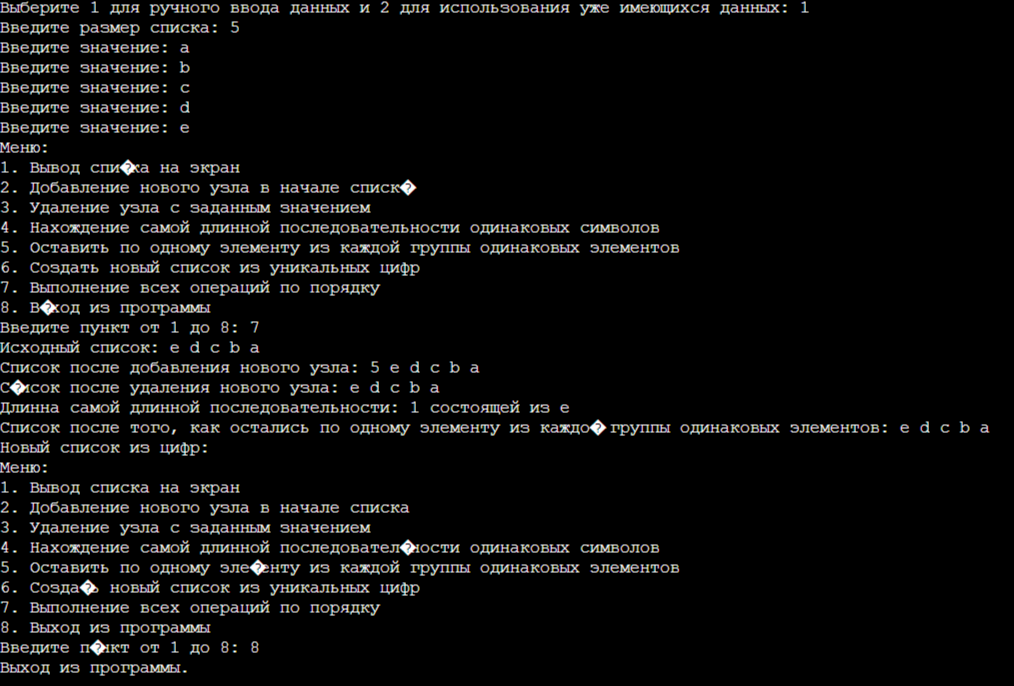


Рисунок 9 - Тестирование программы на данных №2 строки

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.

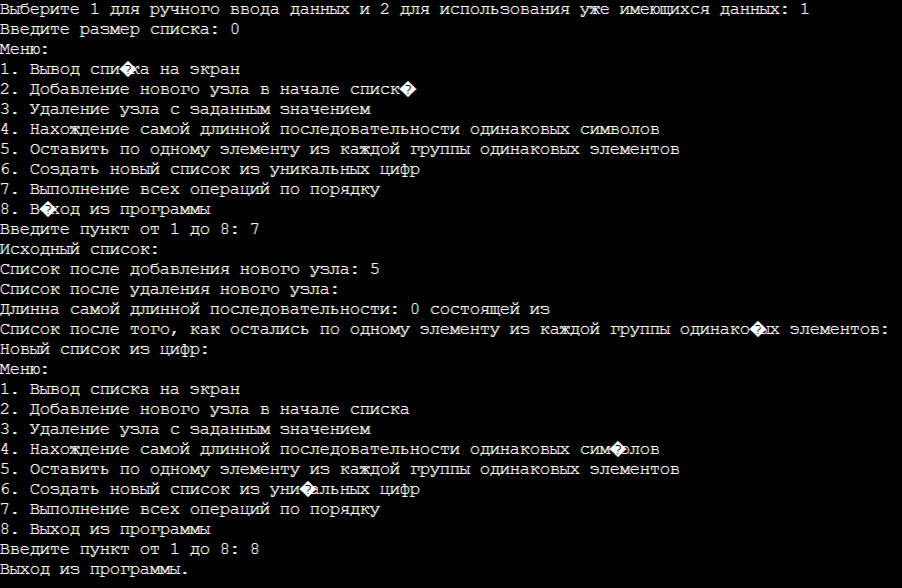


Рисунок 10 - Тестирование программы на данных №3 строки

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

Результаты тестирования №4 строки представлены на рисунке 11.

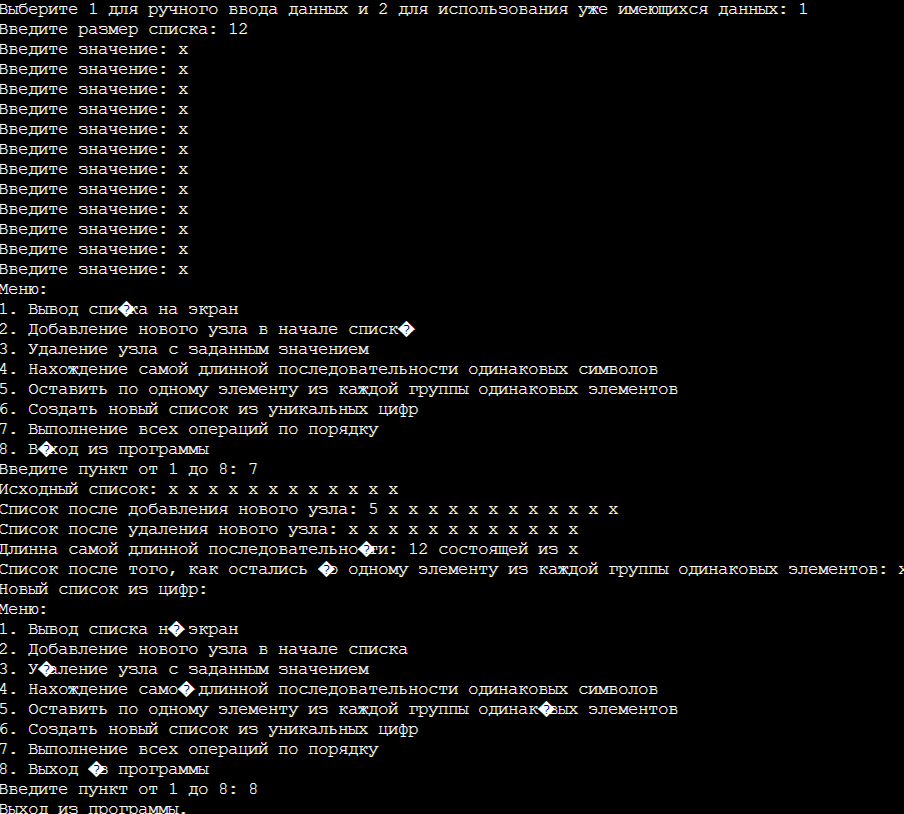


Рисунок 11 - Тестирование программы на данных №4 строки

Проведение тестирования на основе №4 строки таблиц 1-6, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

Результаты тестирования №5 строки представлены на рисунке 12.

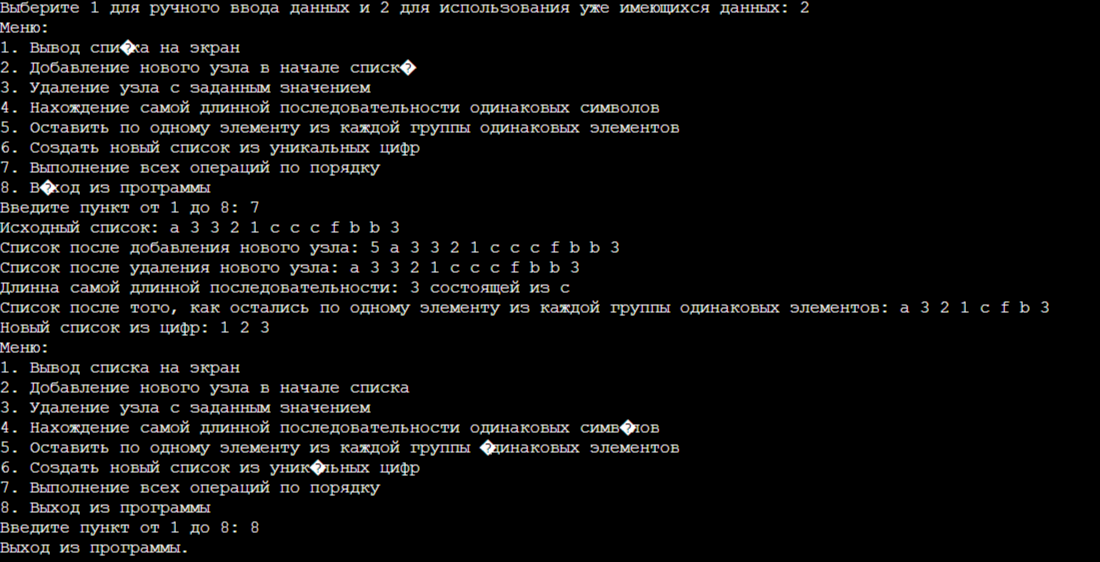


Рисунок 12 - Тестирование программы на данных №5 строки

Проведение тестирования на основе №5 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, которая состоит 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.

Для эффективного управления динамическим однонаправленным списком необходимо понимать следующие основные концепции:

1. Структура узла списка: узел включает в себя данные и указатель на следующий узел в списке.

2. Добавление элемента: необходимо уметь создавать новый узел и правильно настраивать ссылки между узлами, чтобы добавить элемент в список.

3. Удаление элемента: требуется умение корректно удалять узел из списка и перенаправлять ссылки для сохранения целостности списка.

4. Поиск элемента: нахождение определенного элемента в списке может потребовать прохода по всем узлам и сравнения значений.

5. Операции с элементами: изменение значений узлов, сортировка списка, нахождение минимального/максимального элемента и другие операции.

Однонаправленные списки являются удобной структурой данных для хранения и работы с последовательностями элементов и могут быть использованы в различных задачах.

# **3 КОНТРОЛЬНЫЕ ВОПРОСЫ**

1) Расскажите о трех уровнях представления данных в программной

системе.

Три уровня представления данных в программной системе:

a) Логический уровень: Этот уровень определяет, как данные организованы и представлены для конечного пользователя или приложения. На этом уровне определены сущности, их атрибуты и отношения между ними. Примерами могут служить таблицы в реляционных базах данных или объекты в объектно-ориентированных системах.

b) Физический уровень: Этот уровень описывает способы хранения данных на низком уровне. Здесь определяются структуры данных и форматы, используемые для эффективного хранения и доступа к данным на устройствах хранения, таких как жесткие диски или память компьютера.

c) Внешний уровень (Уровень представления): Этот уровень представляет собой специфическое представление данных для конкретного пользователя или приложения. Он скрывает сложность логического и физического уровней, предоставляя удобный интерфейс для работы с данными. Примером может быть пользовательский интерфейс программы или API для доступа к данным.

2) Что определяет тип данных?

Тип данных определяется набором значений, которые переменная может хранить, и операциями, которые можно выполнить над этими значениями. Тип данных также определяет, как память выделяется для переменной и как она интерпретируется при выполнении операций. Примеры типов данных включают целочисленные, вещественные, символьные, логические, строковые и составные типы данных.

3) Что определяет структура данных?

Структура данных определяет организацию и взаимосвязь элементов данных внутри программы или базы данных. Она определяет, как данные хранятся, как они могут быть доступны и какие операции можно выполнять над ними. Структура данных может быть простой, такой как массив или список, или составной, такой как дерево или граф. Хорошо выбранная структура данных может повысить эффективность выполнения операций с данными и упростить реализацию программы.

4) Расскажите о структуры хранения данных в компьютерных технологиях.

Структуры хранения данных в компьютерных технологиях определяют способы организации и хранения данных в памяти компьютера или на внешних устройствах хранения. Вот несколько примеров структур хранения данных:

a) Массивы: Это структуры данных, которые хранят элементы одного типа данных в последовательной памяти. Каждый элемент массива имеет свой индекс, который используется для доступа к нему.

b) Списки: Это структуры данных, которые хранят элементы в виде последовательности, где каждый элемент может быть связан с предыдущим и/или последующим элементом.

c) Деревья: Это иерархические структуры данных, где каждый элемент (узел) может иметь один или несколько дочерних элементов.

d) Графы: Это структуры данных, представляющие собой набор вершин и ребер, связывающих эти вершины.

e) Хеш-таблицы: Это структуры данных, которые используют хеш-функции для быстрого доступа к данным по ключу.

f) Файловые системы: Это структуры данных, используемые для организации и хранения файлов на внешних устройствах хранения, таких как жесткие диски или флеш-накопители.

5) Дайте определение линейной структуре данных.

Линейная структура данных - это структура данных, где элементы организованы в линейной последовательности, где каждый элемент имеет только один предшествующий элемент и один последующий элемент, за исключением первого и последнего элементов.

6) Дайте определение структуре данных линейный список.

Линейный список - это структура данных, представляющая собой последовательность элементов, где каждый элемент, называемый узлом, содержит данные и ссылку на следующий элемент в списке. Последний элемент списка имеет ссылку на NULL или пустое значение, указывающее на конец списка. Линейные списки могут быть однонаправленными (каждый узел содержит ссылку только на следующий узел) или двунаправленными (каждый узел содержит ссылки как на предыдущий, так и на следующий узел).

7) Дайте определение структуре данных стек.

Стек - это абстрактная структура данных, которая представляет собой коллекцию элементов, где операции добавления и удаления элементов происходят только с одного конца, называемого вершиной стека. Этот принцип называется "последним пришел - первым ушел" (Last-In-First-Out, LIFO). Стек поддерживает две основные операции: добавление элемента на вершину стека (push) и удаление элемента с вершины стека (pop).

8) Дайте определение структуре данных очередь.

Очередь - это абстрактная структура данных, которая представляет собой коллекцию элементов, где операции добавления происходят с одного конца, называемого хвостом очереди, а операции удаления происходят с другого конца, называемого головой очереди. Этот принцип называется "первым пришел - первым ушел" (First-In-First-Out, FIFO). Очередь поддерживает две основные операции: добавление элемента в конец очереди (enqueue) и удаление элемента из начала очереди (dequeue).

9) Чем стек отличается от структуры данных линейный список?

Стек и линейный список - это две разные структуры данных с разными принципами организации и доступа к элементам:

В стеке элементы добавляются и удаляются только с одного конца, вершины стека, в то время как в линейном списке элементы могут добавляться и удаляться с обоих концов списка.

В стеке применяется принцип LIFO (Last-In-First-Out), что означает, что последний добавленный элемент будет удален первым. В линейном списке порядок элементов сохраняется и операции доступа к элементам могут быть произвольными.

Стек обычно реализуется с использованием массивов или связанных списков, в то время как линейный список чаще всего реализуется с использованием связанных списков.

10) Какой из видов линейных списков лучше использовать, если

нужно введенную последовательность вывести наоборот?

Лучше использовать односвязный список, так как для вывода последовательности наоборот необходимо пройти от начала списка до конца, сохраняя элементы на пути в обратном порядке. Односвязный список позволяет легко обращаться к предыдущему элементу, что упрощает решение данной задачи.

11) Определите сложность алгоритма операции вставки элемента в

i-ую позицию: а) массива; б) линейного списка.

Сложность алгоритма операции вставки элемента в i-ую позицию:

а) Для массива: O(n), так как при вставке элемента в середину массива требуется сдвиг всех элементов справа от позиции i на одну позицию вправо.

б) Для линейного списка: O(1), так как для вставки элемента в i-ую позицию достаточно изменить ссылки узлов на предыдущий и следующий элементы.

12) Определите сложность алгоритма операции удаления элемента из

i-ой позиции: а) массива; б) линейного списка.

Сложность алгоритма операции удаления элемента из i-ой позиции:

а) Для массива: O(n), так как при удалении элемента из середины массива требуется сдвиг всех элементов справа от позиции i на одну позицию влево.

б) Для линейного списка: O(1), так как для удаления элемента из i-ой позиции достаточно перенаправить ссылки предыдущего и следующего узлов так, чтобы они обходили удаленный узел.

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

Трюк Вирта (или алгоритм Вирта) - это метод удаления элемента из однонаправленного связного списка без явного указания на предыдущий узел. Этот метод состоит в том, что вместо удаления выбранного узла, его данные заменяются данными следующего узла, а затем следующий узел удаляется. Таким образом, пропадает необходимость изменять ссылку предыдущего узла на новый следующий узел. Трюк Вирта позволяет сделать операцию удаления более эффективной и избежать необходимости перебирать список в поисках предыдущего узла.

14) Определите структур узла однонаправленного списка.

| struct Node {  int data; *// данные, которые хранятся в узле*  Node\* next; *// указатель на следующий узел* }; |
| --- |

Структура узла содержит два поля:

* data для хранения данных узла.
* next для указания на следующий узел в списке.

15) Реализуйте алгоритм вывода линейного однонаправленного

списка.

| #include <iostream> using namespace std; *// Определение структуры узла списка* struct Node {  int data; *// данные узла*  Node\* next; *// указатель на следующий узел* };  *// Функция для вывода списка* void printLinkedList(Node\* head) {  Node\* current = head; *// начинаем с головного узла*  while (current != nullptr) {  cout << current->data << " "; *// выводим данные текущего узла*  current = current->next; *// переходим к следующему узлу*  }  cout << endl; *// переход на новую строку после вывода списка* }  *// Пример использования* int main() {  *// Создаем список: 1 -> 2 -> 3 -> nullptr*  Node\* head = new Node{1, nullptr};  head->next = new Node{2, nullptr};  head->next->next = new Node{3, nullptr};  *// Выводим список*  cout << "Список: ";  printLinkedList(head);  return 0; } |
| --- |

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

| #include <iostream> using namespace std;  struct Node {  int data;  Node\* next; };  void moveLastToFirst(Node\* &head) {  if (head == nullptr || head->next == nullptr) *// если список пустой или состоит из одного элемента, ничего не делаем*  return;    Node\* last = head;  Node\* secondLast = nullptr;   *// Находим последний и предпоследний элементы списка*  while (last->next != nullptr) {  secondLast = last;  last = last->next;  }   *// Перемещаем последний элемент в начало списка*  secondLast->next = nullptr;  last->next = head;  head = last; }  *// Пример использования:* int main() {  Node\* head = nullptr;   *// Заполнение списка*  for (int i = 1; i <= 5; ++i) {  Node\* newNode = new Node{i, nullptr};  newNode->next = head;  head = newNode;  }   *// Перемещение последнего элемента в начало списка*  moveLastToFirst(head);   *// Вывод списка*  Node\* current = head;  while (current != nullptr) {  cout << current->data << " ";  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 указанном фрагменте кода лишнее действие - присваивание q->next=LL->next;, потому что оно выполняется перед условием if (LL==nullptr) и также после этого условия. Новый узел вставляется в начало списка.

# 

# **4 ВЫВОДЫ**

В ходе практической работы были выполнены следующие задачи:

- Получены знания по управлению динамическим однонаправленным списком;

- Получены практические навыки управления динамическим однонаправленным списком;

- 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;

- Была реализована возможность а) создания нового списка вручную, а также б) использования уже готового списка для тестирования заданий индивидуального варианта;

- Было реализовано меню выбора способа ввода и реализации функций;

- Проведено тестирование программы с различными данными и способами ввода;

Таким образом, главную цель практической работы, а именно получение знаний и практических навыков управления динамическим однонаправленным списком, можно считать выполненной.

# **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).

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