|  |
| --- |
| 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: Харченко А.А. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ 4](#_30j0zll)

[2.1 Формулировка задачи 4](#_1fob9te)

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

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

[2.2.2 Процесс выполнения операций 6](#_3dy6vkm)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 11](#_bubrpb7ydlxv)

[2.3.1 Реализация на языке программирования C++ 11](#_4d34og8)

[2.3.2 Тестирование 18](#_2s8eyo1)

[2.4 Вывод по заданию 22](#_lnxbz9)

[3 КОНТРОЛЬНЫЕ ВОПРОСЫ 24](#_49x2ik5)

[4 ВЫВОДЫ 30](#_n59xv9kokqs4)

[5 ЛИТЕРАТУРА 31](#_2p2csry)

# **1 ЦЕЛЬ**

Получить знания и практические навыки управления динамическим однонаправленным списком.

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи**

В списке 27, 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) представляет собой динамическую структуру данных, где каждый элемент, называемый узлом, содержит какое-то значение и указатель на следующий элемент в списке. Последний узел списка указывает на NULL, обозначая конец списка. Однонаправленные списки обладают рядом преимуществ, таких как эффективные операции вставки и удаления элементов, но требуют больше памяти для хранения указателей.

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.

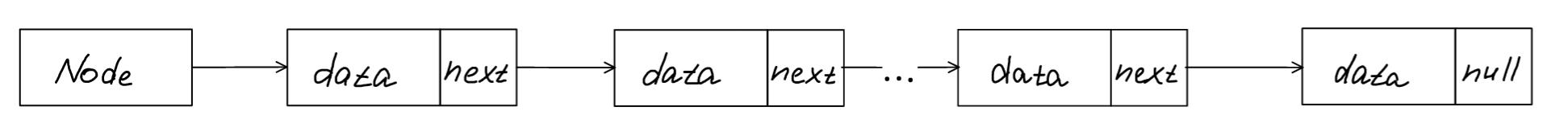


Рисунок 1 - Изображение структуры данных

### **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.

1. Добавление узла в начале списка

Рассмотрим выполнение операции добавления нового узла в начало списка, алгоритм которой представлен на рисунке 2.

Создается новый узел newNode с данными newData. Устанавливается указатель next нового узла на текущее начало списка. Обновляется указатель head, чтобы он указывал на новый узел, тем самым делая его новым началом списка.

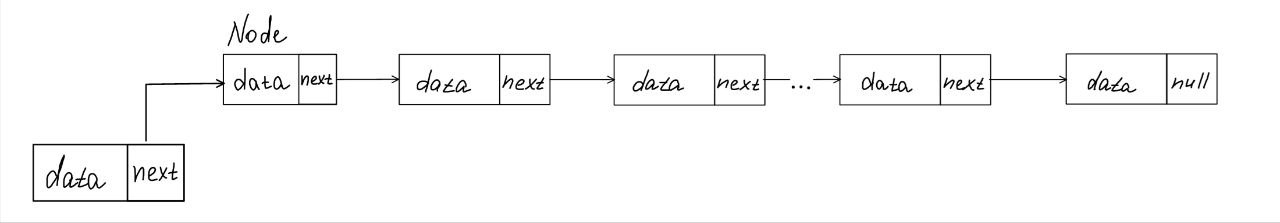


Рисунок 2 - Изображение добавление нового узла

Реализация кода предаставлено в блок коде 2. Для проверки работы кода приведены данные в таблице 1.

Таблица 1 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 5essssq7qlq5 | k | k5essssq7qlq5 |
| 2 |  | k | k |
| 3 | 6 7 8 9 a b c | k | k 6 7 8 9 a b c |

1. Удаление узла по ключу.

Начинается обход списка с начала. Проверяется наличие узла и его значение. Если значение узла совпадает с ключом, то удаляется этот узел и обновляется указатель head, если удаляемый узел находится в начале списка. Если значение не совпадает с ключом, то продолжается обход списка до тех пор, пока не будет найден узел с нужным значением или не будет достигнут конец списка. Если узел с заданным значением не найден, то метод завершается. Если узел найден, то переустанавливается указатель предыдущего узла prev, чтобы он указывал на узел, следующий за удаляемым узлом. Удаляется найденный узел.

Отобразим выполнение данной операции на рисунке 3.

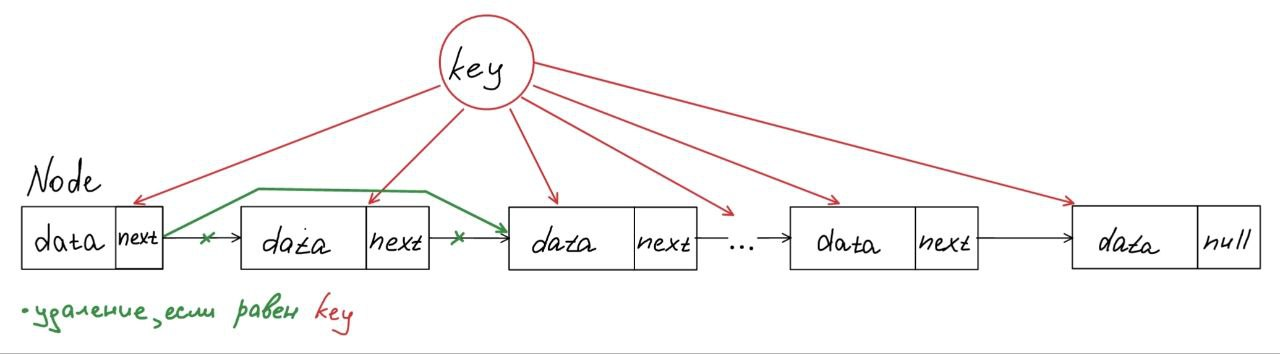


Рисунок 3 - Изображение удаления узла по ключу

Реализация данного алгоритма представлена в блок коде 2 под именем delList. Данные для тестирования будут приведены в таблице 2.

Таблица 2 - Данные для тестирования

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | k5essssq7lq5 | k | 5essssq7qlq5 |
| 2 |  | k |  |
| 3 | k 6 7 8 9 a b c | k | 6 7 8 9 a b c |

1. Вывод элементов списка на экран.

Начинается обход списка с его начала, с помощью указателя temp.

Пока указатель temp не станет равным nullptr (то есть не достигнет конца списка): Выводится значение текущего узла temp->data. Указатель temp перемещается на следующий элемент списка. После завершения обхода списка выводится символ новой строки для читаемости.

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

Таблица 3 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 5essssq7qlq5 | 5essssq7qlq5 |
| 2 |  |  |
| 3 | 6 7 8 9 a b c | 6 7 8 9 a b c |

1. Поиск самых длинных последовательностей одинаковых элементов

Начинается обход списка с его начала, с помощью указателя temp.

Инициализируются переменные maxCount (максимальная длина последовательности) и maxChar (символ с максимальной длиной последовательности) как 0 и пустой символ соответственно. Пока указатель temp не станет равным nullptr (то есть не достигнет конца списка):

* Инициализируется переменная count как 1, представляющая длину текущей последовательности символов.
* Запоминается текущий символ в переменной currChar.
* Во внутреннем цикле происходит сравнение текущего символа с символами следующих узлов. Если символы равны, увеличивается счетчик count и указатель temp перемещается на следующий элемент списка.
* Если длина текущей последовательности (count) превышает максимальную длину (maxCount), то значения maxCount и maxChar обновляются текущими значениями count и currChar.
* Указатель temp перемещается на следующий элемент списка.

После завершения обхода списка выводится сообщение о длине самой длинной последовательности и символе, из которого она состоит.

Отобразим выполнение данной операции на рисунке 4.

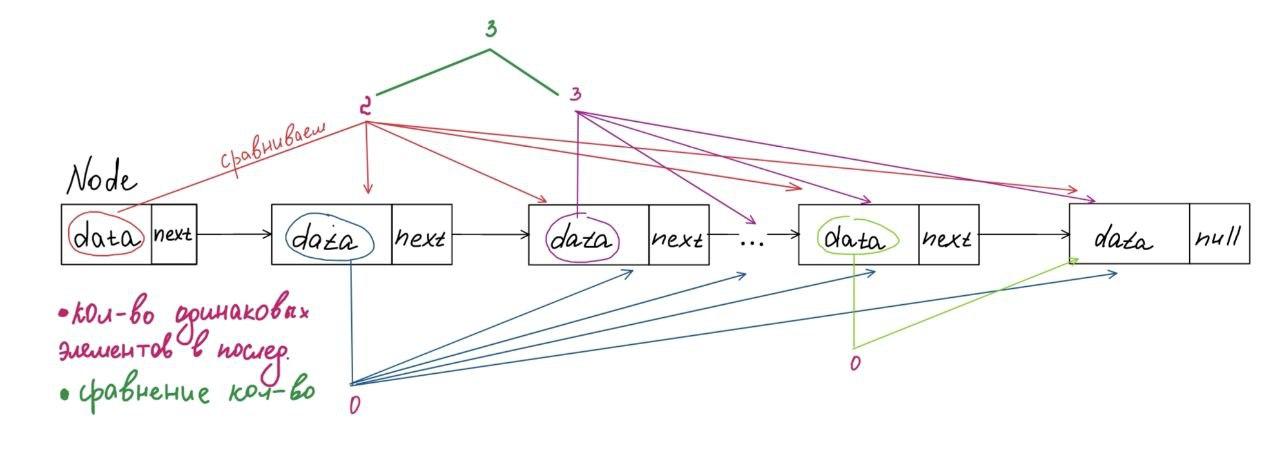


Рисунок 4 - Изображение поиска самых длинных последовательности одинаковых элементов

Реализация данного алгоритма представлена в блок коде 2 под именем Same. Данные для тестирования будут приведены в таблице 4.

Таблица 4 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 5essssq7qlq5 | Длина самой длинной последовательности: 4 состоящей из s |
| 2 |  | Длина самой длинной последовательности: 0 состоящей из \0 |
| 3 | 6 7 8 9 a b c | Длина самой длинной последовательности: 1 состоящей из 6 |

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

Отобразим выполнение данной операции на рисунке 5.

Начинается обход списка с его начала, с помощью указателя temp. Пока указатель temp не станет равным nullptr (то есть не достигнет конца списка):

* Инициализируется временный указатель curr как указатель на текущий узел temp.
* Во внутреннем цикле while продолжается обход списка от текущего узла curr, пока следующий узел (curr->next) не станет равным nullptr и значение его данных (data) совпадает с данными текущего узла temp.
* После завершения внутреннего цикла curr указывает на последний узел в группе одинаковых элементов.
* Устанавливается указатель next текущего узла temp на узел, следующий за последним узлом в группе (то есть на узел после curr).
* Указатель temp перемещается на следующий узел после группы одинаковых элементов.

Реализация данного алгоритма представлена в блок коде 2 под именем leave. Данные для тестирования будут приведены в таблице 5.

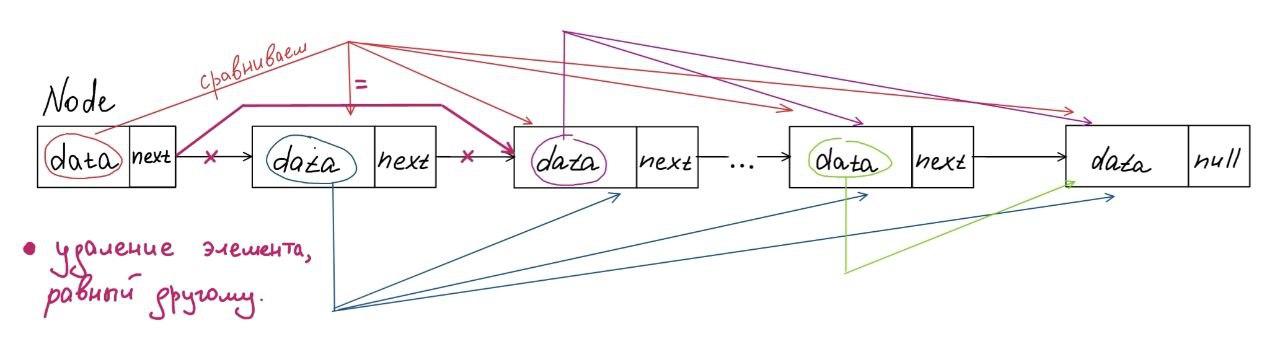


Рисунок 5 - Изобразим оставление одного элемента из каждой группы одинаковых элементов

Таблица 5 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 5essssq7qlq5 | 5 q l q 7 q s e 5 |
| 2 |  |  |
| 3 | 6 7 8 9 a b c | 6 7 8 9 a b c |

1. Создание нового списка с уникальными цифровыми значениями.

Начинается обход исходного списка с его начала, с помощью указателя temp. Создается новый пустой список newList, в который будут добавлены уникальные цифры. Пока указатель temp не станет равным nullptr (то есть не достигнет конца списка):

* Получается значение текущего узла temp и сохраняется в переменной digit.
* Устанавливается флаг isUnique в значение true, предполагая, что текущая цифра уникальна.
* Создается временный указатель checkTemp, который начинает обход списка head для проверки наличия текущей цифры в списке newList.
* Во внутреннем цикле while проверяется каждый узел списка newList на совпадение его значения с текущей цифрой digit. Если такое совпадение найдено, флаг isUnique устанавливается в false.
* Если текущая цифра digit уникальна и является цифрой, она добавляется в список newList с помощью метода AddendumFront.
* Указатель temp перемещается на следующий узел исходного списка.

После завершения обхода исходного списка выводится сообщение о новом списке из цифр с помощью метода outputList списка newList.

Реализация данного алгоритма представлена в блок коде 2 под именем create. Данные для тестирования будут приведены в таблице 6.

Таблица 6 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 5essssq7qlq5 | 7 |
| 2 |  |  |
| 3 | 6 7 8 9 a b c | 6 7 8 9 |

## **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 2). Для реализации понадобятся такие библиотеки, как iostream.

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).

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>  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 L { private:  Node\* head; public:  L() : head(nullptr) {}// Конструктор класса LinkedList, инициализирует указатель на начало списка как nullptr  // Метод для добавления нового узла в начало списка  void AddendumFront(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 delNode(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 outputList() {  // 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 Same() {  // 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 leave() {  // 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 create() {  Node\* temp = head;  L newList;  while (temp != nullptr) {  char digit = temp->data;  bool isUnique = true;  Node\* checkTemp = head;  // Проверяем, есть ли текущая цифра в списке newList  while (checkTemp != nullptr) {  if (checkTemp->data == digit && checkTemp != temp) {  isUnique = false;  break;  }  checkTemp = checkTemp->next;  }  // Если цифра уникальна, добавляем ее в новый список  if (isUnique && isdigit(digit)) {  newList.AddendumFront(digit);  }  temp = temp->next;  }  cout << "Новый список из цифр: ";  newList.outputList();  } }; int main() {  L list;  int choice, c;  cout << "Выберите 1(ручной ввода) и 2 (использования существующих 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.AddendumFront(value);  }  } else if (choice == 2) {  // Готовый список для тестирования  list.AddendumFront('5');  list.AddendumFront('e');  list.AddendumFront('s');  list.AddendumFront('s');  list.AddendumFront('s');  list.AddendumFront('s');  list.AddendumFront('q');  list.AddendumFront('7');  list.AddendumFront('q');  list.AddendumFront('l');  list.AddendumFront('q');  list.AddendumFront('5');  } 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. 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. Выход из программы" << endl;  cout << "Выберете дейсвие: ";    cin >> c;    switch(c) {  case 1:  cout << "Список: ";  list.outputList();  break;  case 2:  list.AddendumFront('k');  list.outputList();  break;  case 3:  list.delNode('k');  list.outputList();  break;  case 4:  list.Same();  break;  case 5:  list.leave();  list.outputList();  break;  case 6:  list.create();  break;  case 7:  cout << "Выход из программы." << endl;  break;  default:  cout << "Ошибка" << endl;  }  } while (c!= 7);    return 0; } |
| --- |

Блок кода 2 – 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.

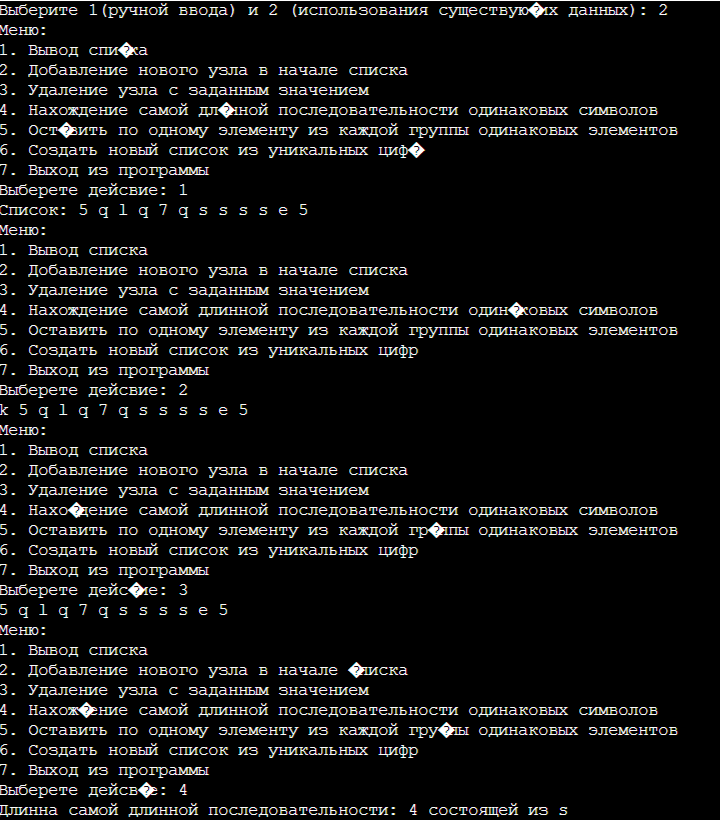


Рисунок 6 - Тестирование программы на данных №1 строки

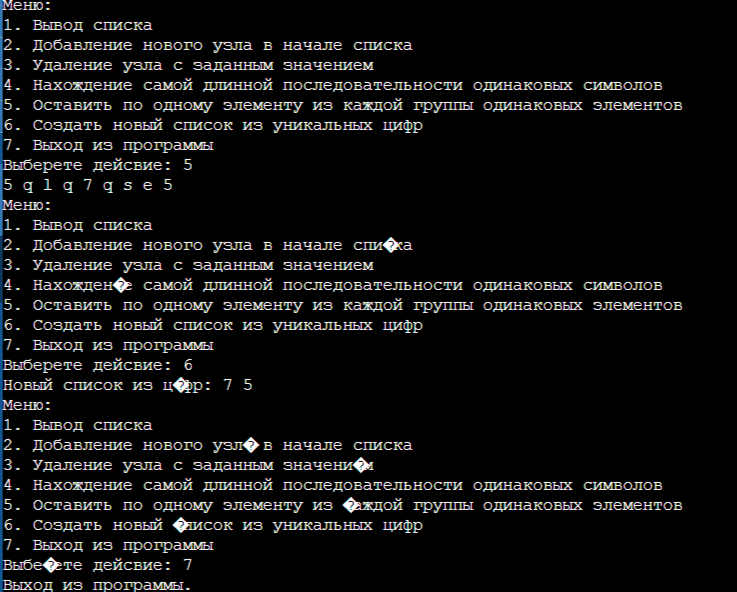


Рисунок 7 - Тестирование программы на данных №1 строки

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.

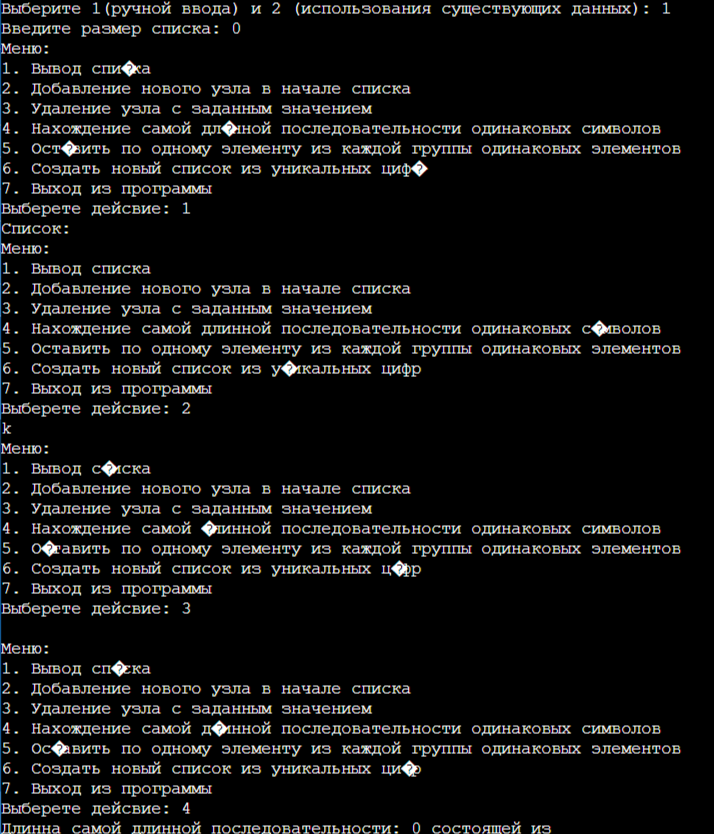


Рисунок 8 - Тестирование программы на данных №2 строки

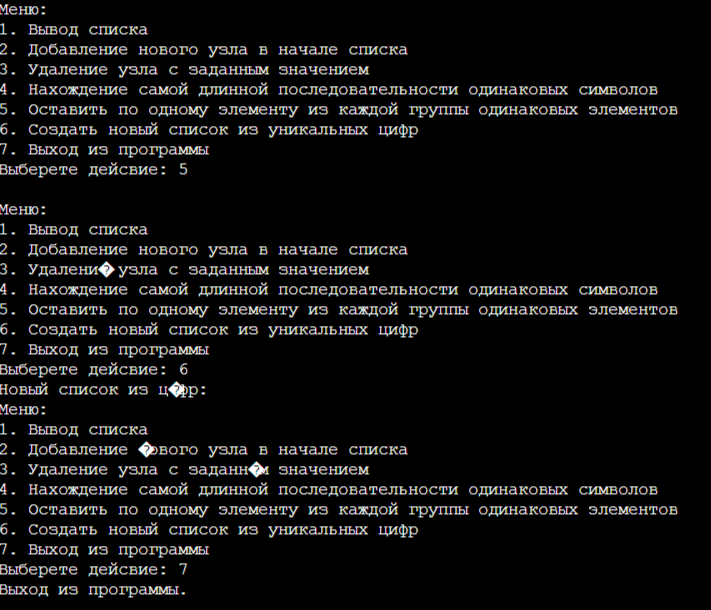


Рисунок 9 - Тестирование программы на данных №2 строки

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.

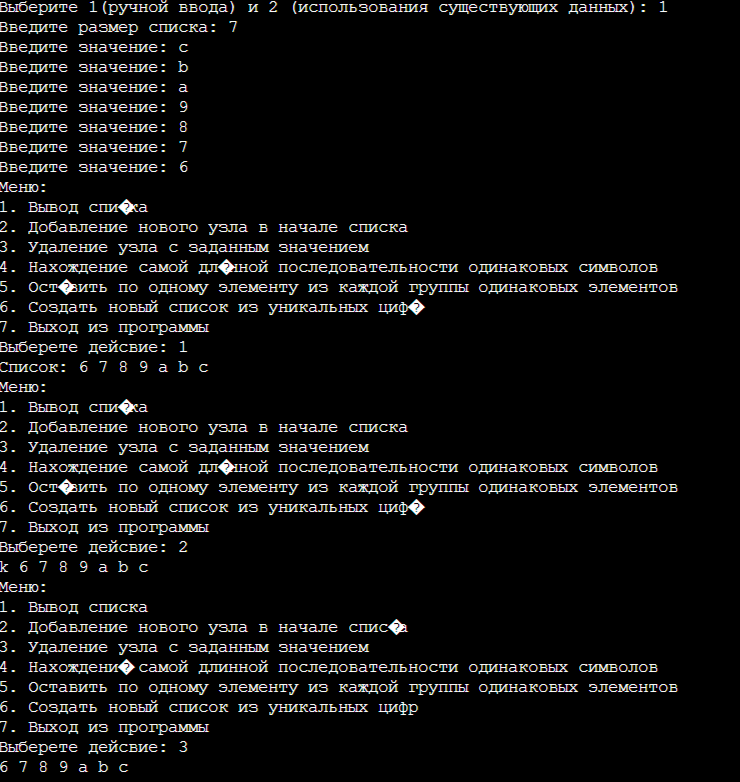


Рисунок 10 - Тестирование программы на данных №3 строки

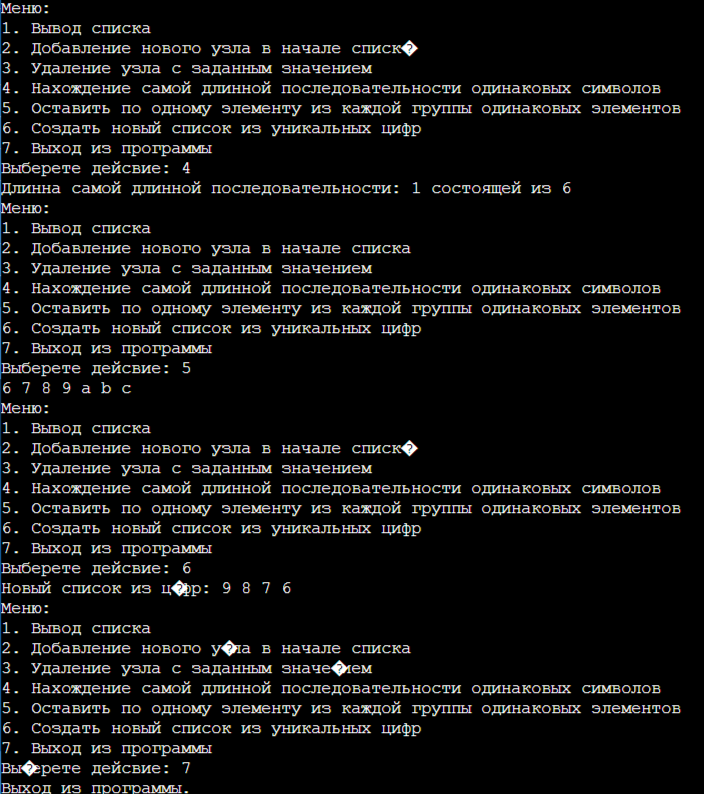


Рисунок 11 - Тестирование программы на данных №3 строки

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 представляют собой структуру данных, где каждый элемент, называемый узлом, содержит какую-то информацию и ссылку на следующий элемент в списке. Они обеспечивают эффективность операций добавления и удаления, поскольку для доступа к элементу достаточно просматривать только предшествующие узлы.

Чтобы эффективно управлять динамическим однонаправленным списком, важно понимать следующие основные концепции:

Структура узла списка: каждый узел содержит данные и указатель на следующий узел в списке.

Добавление элемента: нужно уметь создавать новый узел и корректно устанавливать ссылки между узлами, чтобы добавить элемент в список.

Удаление элемента: необходимо умение удалять узел из списка и перенаправлять ссылки для сохранения целостности списка.

Поиск элемента: нахождение конкретного элемента может потребовать просмотра всех узлов и сравнения их значений.

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

Однонаправленные списки представляют собой удобную структуру данных для хранения и манипулирования последовательностями элементов, и они могут быть применены в разнообразных задачах.

# **3 КОНТРОЛЬНЫЕ ВОПРОСЫ**

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

Три уровня представления данных в программной системе:

Физический уровень: определяет, как данные физически хранятся на устройствах хранения, например, на жестком диске или в памяти.

Логический уровень: определяет структуру и связи между данными, независимо от конкретной реализации физического уровня. Включает в себя схемы баз данных и структуры данных.

Уровень представления: определяет, как данные представлены для пользователя или приложения, например, с помощью интерфейса пользователя или API.

2) Что определяет тип данных?

Тип данных определяется множеством значений, которые могут принимать переменные данного типа, и операциями, которые могут быть выполнены над этими значениями. Тип данных также определяет размер и формат памяти, выделенной для хранения каждой переменной этого типа.

3) Что определяет структура данных?

Структура данных определяет способ организации и хранения данных, а также доступ к этим данным. Она включает в себя конкретные методы доступа, операции добавления и удаления данных, и способы организации элементов в памяти.

4) Расскажите о структуры хранения данных в компьютерных технологиях.

В компьютерных технологиях данные хранятся с использованием различных структур данных, таких как массивы, списки, деревья, хеш-таблицы и т. д. Эти структуры данных определяют способы организации, доступа и обработки данных в компьютерной программе.

5) Дайте определение линейной структуре данных.

Линейная структура данных - это структура, в которой элементы данных организованы последовательно, таким образом, что каждый элемент имеет ровно одного предшествующего и одного последующего соседа.

6) Дайте определение структуре данных линейный список.

Линейный список - это структура данных, состоящая из последовательности узлов, где каждый узел содержит данные и ссылку (или указатель) на следующий узел в списке.

7) Дайте определение структуре данных стек.

Стек - это абстрактная структура данных, основанная на принципе "последним пришел - первым вышел" (Last In, First Out, LIFO). Это означает, что последний элемент, добавленный в стек, будет первым, который будет удален из стека.

8) Дайте определение структуре данных очередь.

Очередь - это абстрактная структура данных, основанная на принципе "первым пришел - первым вышел" (First In, First Out, FIFO). Это означает, что первый элемент, добавленный в очередь, будет первым, который будет удален из очереди.

9) Чем стек отличается от структуры данных линейный список?

Стек отличается от линейного списка тем, что стек поддерживает только операции добавления и удаления элементов с одного конца (вершины стека), в то время как линейный список поддерживает операции добавления и удаления элементов с любого места в списке.

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

Если нужно введенную последовательность вывести наоборот, то лучше всего использовать двусвязный список. В двусвязном списке каждый узел содержит указатель как на следующий, так и на предыдущий элемент, что облегчает обход списка как в прямом, так и в обратном направлении.

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 требуется сдвинуть все элементы после вставляемого элемента на одну позицию вправо.

б) Для линейного списка: O(1), если известен узел, предшествующий вставляемому узлу. Иначе, если требуется поиск i

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

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

Трюк Вирта - это метод удаления узла из однонаправленного списка без использования указателя на предыдущий узел. Вместо этого узел, который следует удалить, копируется содержимое следующего узла, а затем следующий узел удаляется. Этот подход позволяет обойти ограничение однонаправленных списков, которые не имеют прямого доступа к предыдущему узлу.

14) Определите структур узла однонаправленного списка.

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

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.

| 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 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 выполнения практической работы были достигнуты следующие результаты:

* Освоены основы управления динамическим однонаправленным списком;
* Получены практические навыки работы с динамическим однонаправленным списком;
* Проведен анализ структуры и операций в соответствии с индивидуальным вариантом;
* Разработана программа для работы с выбранной структурой данных;
* Реализованы функции операций с этой структурой;
* Реализована возможность как ручного ввода данных, так и использования уже имеющихся списков для тестирования;
* Создано меню выбора способа ввода и функций программы;
* Проведено тестирование программы с различными входными данными и методами ввода.

Таким образом, цель практической работы, а именно 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).