|  |
| --- |
|  |
| 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: Зернов Н.А. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ 4](#_30j0zll)

[2.1 Формулировка задачи (В списке №13, Вариант 13) 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 13](#_bubrpb7ydlxv)

[2.3.1 Реализация на языке программирования C++ 13](#_4d34og8)

[2.3.2 Тестирование 17](#_2s8eyo1)

[2.4 Вывод по заданию 19](#_lnxbz9)

[3 КОНТРОЛЬНЫЕ ВОПРОСЫ 22](#_49x2ik5)

[4 ВЫВОДЫ 29](#_n59xv9kokqs4)

[5 ЛИТЕРАТУРА 30](#_2p2csry)

# **1 ЦЕЛЬ**

Получить знания и практические навыки управления динамическим однонаправленным списком.

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи (В списке №13, Вариант 13)**

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: int.

Дополнительные операции: Дан массив из n указателей на вершины списков. Структура узла списка содержит ключ (информационная часть узла) и ссылку на следующий узел.

1. Разработать функцию, которая вставляет переданный в качестве параметра ключ в i-ый список массива. Индекс i определяется по правилу: i=key%n. Некоторые элементы массива могут остаться nullptr.

2. Разработать функцию для удаления значение ключа из списка.

3. Разработать функцию, которая находит узел со значением ключа и возвращает указатель на найденный узел.

## **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 - элемент односвязного списка. В данной структуре содержится переменная data типа int, которая хранит значение элемента, и указатель на следующий элемент списка next. Конструктор Node принимает значение типа int 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**(**int 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.

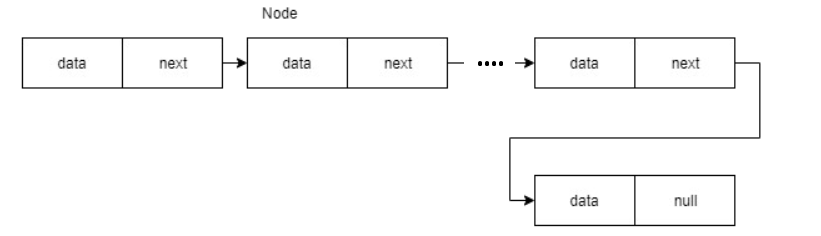


Рисунок 2 - Изображение добавление нового узла

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 OutputList**()**

**{**

Node**\*** temp **=** head**;**

**while** **(**temp **!=** **nullptr)** **{**

cout **<<** temp**->**data **<<** " "**;**

temp **=** temp**->**next**;**

**}**

cout **<<** endl**;**

**}**

Блок кода 2 - Добавление нового узла в начале списка

Таблица 1 - Данные для тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Добавить | Ожидаемый результат |
| 1 | 1 2 3 4 5 6 7 | 8 | 8 1 2 3 4 5 6 7 |
| 2 | 11 12 | 7 | 7 11 12 |
| 3 |  | 6 | 6 |
| 4 | 2 6 9 4 9 90 2 6 80 1 | 5 | 5 2 6 9 4 9 90 2 6 80 1 |
| 5 | 1 1 1 1 1 | 4 | 4 1 1 1 1 1 |

1. Удаление узла по ключу.

Отобразим выполнение данной операции на рисунке 3.

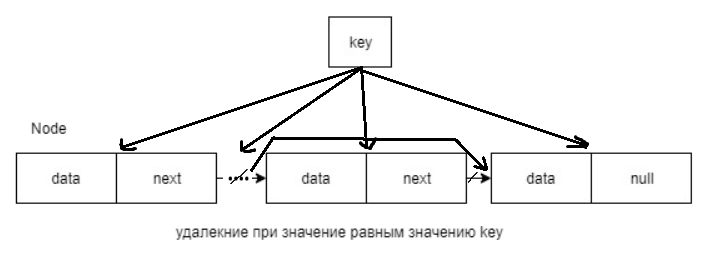


Рисунок 3 - Изображение удаления узла по ключу

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**(**int key**)**

**{** // Метод удаления ноды по ключу

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 | 8 1 2 3 4 5 6 7 | 8 | 1 2 3 4 5 6 7 |
| 2 | 7 11 12 | 7 | 11 12 |
| 3 | 6 | 6 |  |
| 4 | 5 2 6 9 4 9 90 2 6 80 1 | 5 | 2 6 9 4 9 90 2 6 80 1 |
| 5 | 4 1 1 1 1 1 | 4 | 1 1 1 1 1 |

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 OutputList()

{

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 | 1 2 3 4 5 6 7 | 1 2 3 4 5 6 7 |
| 2 | 11 12 | 11 12 |
| 3 |  |  |
| 4 | 2 6 9 4 9 90 2 6 80 1 | 2 6 9 4 9 90 2 6 80 1 |
| 5 | 1 1 1 1 1 | 1 1 1 1 1 |

4. Разработать функцию, которая вставляет переданный в качестве параметра ключ в i-ый список массива. Индекс i определяется по правилу: i=key%n. Некоторые элементы массива могут остаться nullptr.

Метод InsertKey принимает два параметра: key - значение ключа, которое нужно вставить в список, и n - количество списков в массиве. Он вычисляет индекс i для вставки нового ключа в список по формуле i=key%n.

Если индекс равен 0, значит, новый узел должен быть вставлен в начало списка. В этом случае метод создает новый узел с переданным ключом, устанавливает его указатель на следующий узел равным текущему началу списка, а затем делает этот новый узел новым началом списка.

Если индекс не равен 0, метод начинает обход списка, чтобы найти узел, перед которым нужно вставить новый ключ. Когда он находит нужный узел (или достигает конца списка, если индекс находится за его пределами), он создает новый узел с переданным ключом, устанавливает его указатель на следующий узел равным указателю на следующий узел временного узла, а затем устанавливает указатель временного узла на новый узел. Таким образом, новый узел вставляется в список перед указанным индексом i.

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

void InsertKey**(**int key**,** int n**)** **{**

int index **=** key **%** n**;**

Node**\*** newNode **=** **new** Node**(**key**);**

newNode**->**next **=** **nullptr;**

**if** **(**index **==** 0**)** **{**

newNode**->**next **=** head**;**

head **=** newNode**;**

**return;**

**}**

Node**\*** temp **=** head**;**

**for** **(**int i **=** 0**;** i **<** index **-** 1 **&&** temp **!=** **nullptr;** i**++)** **{**

temp **=** temp**->**next**;**

**}**

**if** **(**temp **==** **nullptr)** **{**

cout **<<** "Индекс вне диапазона" **<<** endl**;**

**return;**

**}**

newNode**->**next **=** temp**->**next**;**

temp**->**next **=** newNode**;**

**}**

Блок кода 5 – вставка по ключу

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Вставка,  ключ | Ожидаемый результат |
| 1 | 8 1 2 3 4 5 6 7 | 7 5 | 7 6 7 5 4 3 2 1 8 |
| 2 | 7 11 12 | 7 5 | 12 11 7 7 |
| 3 |  | 7 5 |  |
| 4 | 5 2 6 9 4 9 90 2 6 80 1 | 7 5 | 1 80 7 6 2 90 9 4 9 6 2 5 |
| 5 | 4 1 1 1 1 1 | 7 5 | 1 1 7 1 1 1 4 |

Таблица 4 - Данные для тестирования

1. Метод для удаления значение ключа из списка

Метод DeleteNode принимает параметр key, который представляет значение ключа, которое нужно удалить из списка.

Сначала метод устанавливает временный указатель temp на начало списка и указатель prev на nullptr, что означает отсутствие предыдущего узла. Затем метод проверяет, содержит ли первый узел списка ключ, который нужно удалить. Если да, то указатель на начало списка перенаправляется на следующий узел, а память, занимаемая первым узлом, освобождается.

Если ключ не найден в первом узле, метод начинает обход списка, проверяя каждый узел на соответствие ключу. При этом он сохраняет указатель на предыдущий узел в переменной prev. Когда нужный узел найден или достигнут конец списка, метод прекращает выполнение.

Если узел со значением ключа был найден, то указатель на следующий узел предыдущего узла устанавливается равным указателю на следующий узел текущего узла, тем самым "пропуская" текущий узел в списке. Затем память, занимаемая текущим узлом, освобождается.

Это позволяет удалить узел с заданным значением ключа из списка.

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

void DeleteNode**(**int 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**;**

**}**

Блок кода 6 – реализация метода DeleteNode

Таблица 5 - Данные для тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Удалить | Ожидаемый результат |
| 1 | 8 1 2 3 4 5 6 7 | 8 | 1 2 3 4 5 6 7 |
| 2 | 7 11 12 | 7 | 11 12 |
| 3 | 6 | 6 |  |
| 4 | 5 2 6 9 4 9 90 2 6 80 1 | 5 | 2 6 9 4 9 90 2 6 80 1 |
| 5 | 4 1 1 1 1 1 | 4 | 1. 1 1 1 1 |

1. Метод, который находит узел со значением ключа и возвращает указатель на найденный узел..

Метод FindNode принимает параметр key, который представляет значение ключа, который нужно найти в списке.

Сначала метод устанавливает временный указатель temp на начало списка. Затем метод начинает обход списка, проверяя каждый узел. Если значение текущего узла равно ключу, то метод возвращает указатель на этот узел. Если узел со значением ключа не найден в списке (т.е. если temp становится равным nullptr), метод завершает свое выполнение и возвращает nullptr.

Таким образом, метод FindNode позволяет найти узел в списке по заданному значению ключа и возвращает указатель на этот узел. Если узел не найден, возвращается nullptr.

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

Node**\*** FindNode**(**int key**)** **{**

Node**\*** temp **=** head**;**

**while** **(**temp **!=** **nullptr)** **{**

**if** **(**temp**->**data **==** key**)** **{**

**return** temp**;**

**}**

temp **=** temp**->**next**;**

**}**

**return** **nullptr;**

**}**

Блок кода 7 - Создание нового списка с уникальными цифровыми значениями

Таблица 6 - Данные для тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Ключ | Ожидаемый результат |
| 1 | 8 1 2 3 4 5 6 7 | 4 | 4 |
| 2 | 7 11 12 | 11 | 11 |
| 3 |  | 6 | не найден |
| 4 | 5 2 6 9 4 9 90 2 6 80 1 | 9 | 9 |
| 5 | 4 1 1 1 1 1 | 1 | 1 |

## **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.

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**;**

struct Node **{**

int data**;**

Node**\*** next**;**

Node**(**int d**)** **:** data**(**d**),** next**(nullptr)** **{}**

**};**

// Определяем класс односвязного списка с приватным указателем на начало списка

class List **{**

private**:**

Node**\*** head**;**

public**:**

List**():**head**(nullptr)** **{}** // Конструктор класса List, инициализирует указатель на начало списка как nullptr

// Метод для добавления нового узла в начало списка

void PushFront**(**int 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 удаления ноды по ключу

void DeleteNode**(**int 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**;**

**}**

// Метод для вставки ключа в i-ый список массива

void InsertKey**(**int key**,** int n**)** **{**

int index **=** key **%** n**;**

Node**\*** newNode **=** **new** Node**(**key**);**

newNode**->**next **=** **nullptr;**

**if** **(**index **==** 0**)** **{**

newNode**->**next **=** head**;**

head **=** newNode**;**

**return;**

**}**

Node**\*** temp **=** head**;**

**for** **(**int i **=** 0**;** i **<** index **-** 1 **&&** temp **!=** **nullptr;** i**++)** **{**

temp **=** temp**->**next**;**

**}**

**if** **(**temp **==** **nullptr)** **{**

cout **<<** "Индекс вне диапазона" **<<** endl**;**

**return;**

**}**

newNode**->**next **=** temp**->**next**;**

temp**->**next **=** newNode**;**

**}**

// Метод для поиска узла с заданным ключом и возврата указателя на него

Node**\*** FindNode**(**int key**)** **{**

Node**\*** temp **=** head**;**

**while** **(**temp **!=** **nullptr)** **{**

**if** **(**temp**->**data **==** key**)** **{**

**return** temp**;**

**}**

temp **=** temp**->**next**;**

**}**

**return** **nullptr;**

**}**

**};**

int main**()** **{**

setlocale**(**LC\_ALL**,** "Rus"**);**

List list**;** // Создаем объект класса List(по факту собственная реализация контейнера List

int type\_input**;**

cout **<<** "Выберите 0 для ручного ввода данных и 1 для использования уже имеющихся данных: "**;**

cin **>>** type\_input**;**

**if** **(**type\_input **==** 0**)** **{**

int size**;**

int value**;**

cout **<<** "Введите размер списка: "**;**

cin **>>** size**;**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)** **{**

cout **<<** "Введите значение: "**;**

cin **>>** value**;**

list**.**PushFront**(**value**);**

**}**

**}**

**else** **if** **(**type\_input **==** 1**)** **{**

// Готовый вводные данные

// 2 6 9 4 9 90 2 6 80 1

list**.**PushFront**(**2**);**

list**.**PushFront**(**6**);**

list**.**PushFront**(**9**);**

list**.**PushFront**(**4**);**

list**.**PushFront**(**9**);**

list**.**PushFront**(**90**);**

list**.**PushFront**(**2**);**

list**.**PushFront**(**6**);**

list**.**PushFront**(**80**);**

list**.**PushFront**(**1**);**

**}**

**else** **{**

cout **<<** "Ошибка" **<<** endl**;**

**return** **-**1**;**

**}**

int choice\_method**;** // Переменная для выбора метода

**do** **{** // Реализация меню выбора методов для класса List объекта list

cout **<<** "Меню:" **<<** endl**;**

cout **<<** "0. Вывод списка на экран" **<<** endl**;**

cout **<<** "1. Добавление нового узла в начале списка" **<<** endl**;**

cout **<<** "2. Удаление узла с заданным значением" **<<** endl**;**

cout **<<** "3. Метод, который вставляет переданный в качестве параметра ключ в i-ый список массива. Индекс i определяется по правилу: i=key%n. Некоторые элементы массива могут остаться nullptr." **<<** endl**;**

cout **<<** "4. Метод для удаления значение ключа из списка" **<<** endl**;**

cout **<<** "5. Метод, который находит узел со значением ключа и возвращает указатель на найденный узел" **<<** endl**;**

cout **<<** "6. Выполнение всех операций по порядку" **<<** endl**;**

cout **<<** "7. Завершение работы программы" **<<** endl**;**

cout **<<** "Введите номер от 0 до 7 включительно: "**;**

cin **>>** choice\_method**;**

**if** **(**choice\_method **==** 0**)** **{**

cout **<<** "Исходный список: "**;**

list**.**OutputList**();**

**}**

**else** **if** **(**choice\_method **==** 1**)**

list**.**PushFront**(**5**);**

**else** **if** **(**choice\_method **==** 2**)**

list**.**DeleteNode**(**5**);**

**else** **if** **(**choice\_method **==** 3**)** **{**

list**.**InsertKey**(**7**,** 5**);** // Вставляем ключ в i-ый список массива

cout **<<** "Список после вставки ключа: "**;**

list**.**OutputList**();**

**}**

**else** **if** **(**choice\_method **==** 4**)** **{**

// Удаляем значение ключа из списка

list**.**DeleteNode**(**7**);**

cout **<<** "Список после удаления ключа: "**;**

list**.**OutputList**();**

**}**

**else** **if** **(**choice\_method **==** 5**)** **{**

// Находим узел со значением ключа и выводим его

Node**\*** node **=** list**.**FindNode**(**80**);**

**if** **(**node **!=** **nullptr)**

cout **<<** "Узел со значением ключа найден: " **<<** node**->**data **<<** endl**;**

**else**

cout **<<** "Узел со значением ключа не найден" **<<** endl**;**

**}**

**else** **if** **(**choice\_method **==** 6**)** **{** //что бы запустить все методы и проверить сразу

cout **<<** "Исходный список: "**;**

list**.**OutputList**();**

list**.**PushFront**(**5**);**

cout **<<** "Список после добавления нового узла: "**;**

list**.**OutputList**();**

list**.**DeleteNode**(**5**);**

cout **<<** "Список после удаления нового узла: "**;**

list**.**OutputList**();**

list**.**InsertKey**(**7**,** 5**);** // Вставляем ключ в i-ый список массива

cout **<<** "Список после вставки ключа: "**;**

list**.**OutputList**();**

list**.**DeleteNode**(**7**);**

cout **<<** "Список после удаления ключа: "**;**

list**.**OutputList**();**

Node**\*** node **=** list**.**FindNode**(**80**);**

**if** **(**node **!=** **nullptr)**

cout **<<** "Узел со значением ключа найден: " **<<** node**->**data **<<** endl**;**

**else**

cout **<<** "Узел со значением ключа не найден" **<<** endl**;**

**}**

**else** **if** **(**choice\_method **==** 7**)** **{**

cout **<<** "Завершение работы программы" **<<** endl**;**

**return** 0**;**

**}**

**else**

cout **<<** "Ошибка, поробуйте еще раз." **<<** endl**;**

**}** **while** **(true);** // Будет выполнется пока пользователь не завршит программу

**}**

Блок кода 8 – Программа однонаправленного списка с выбором операций

### **2.3.2 Тестирование**

Проведем тестирование основываясь на данные из таблиц 1-6. Сперва проведя ввод данных ручной, потом готовый.

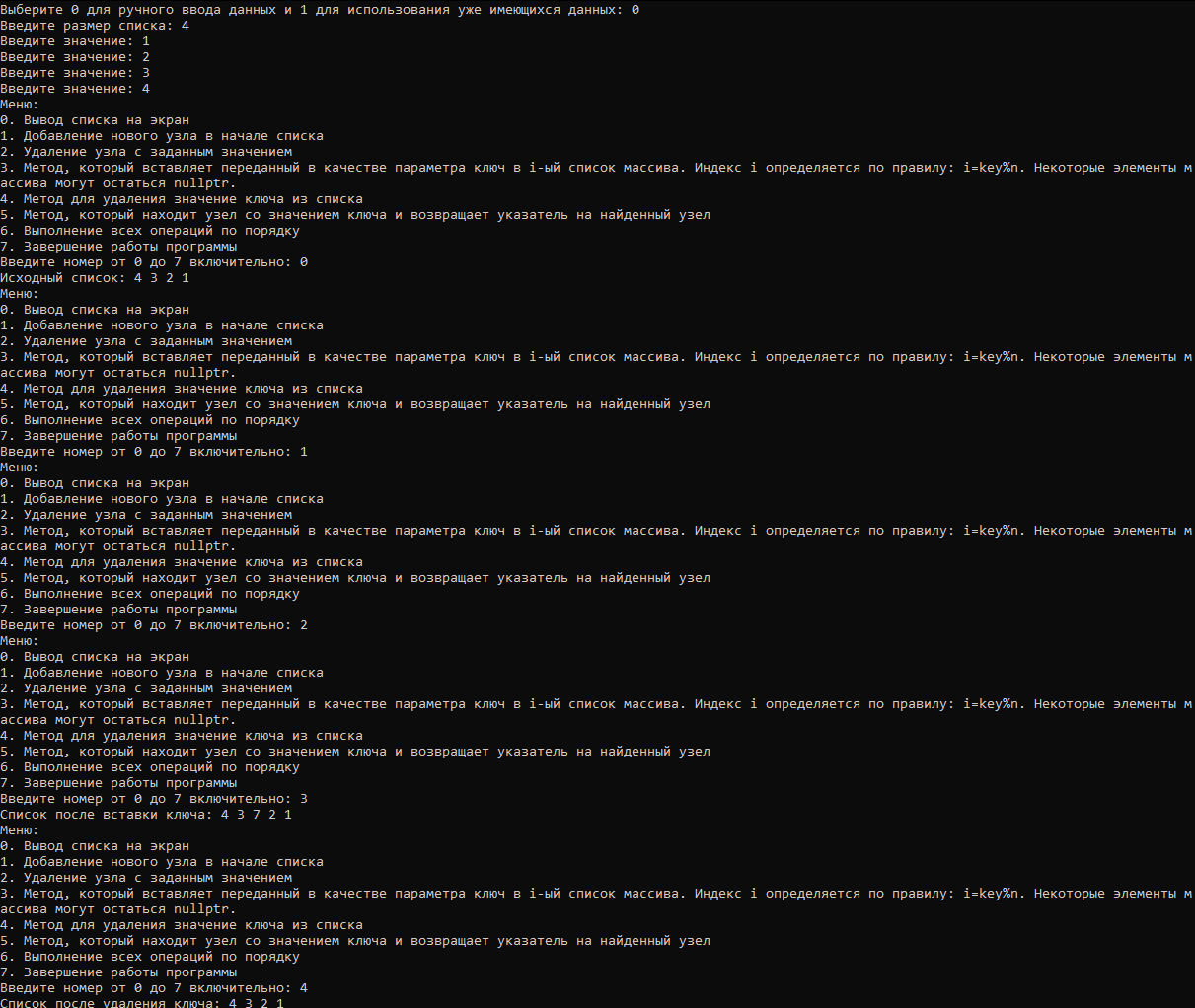


Рисунок 4 - Тестирование программы

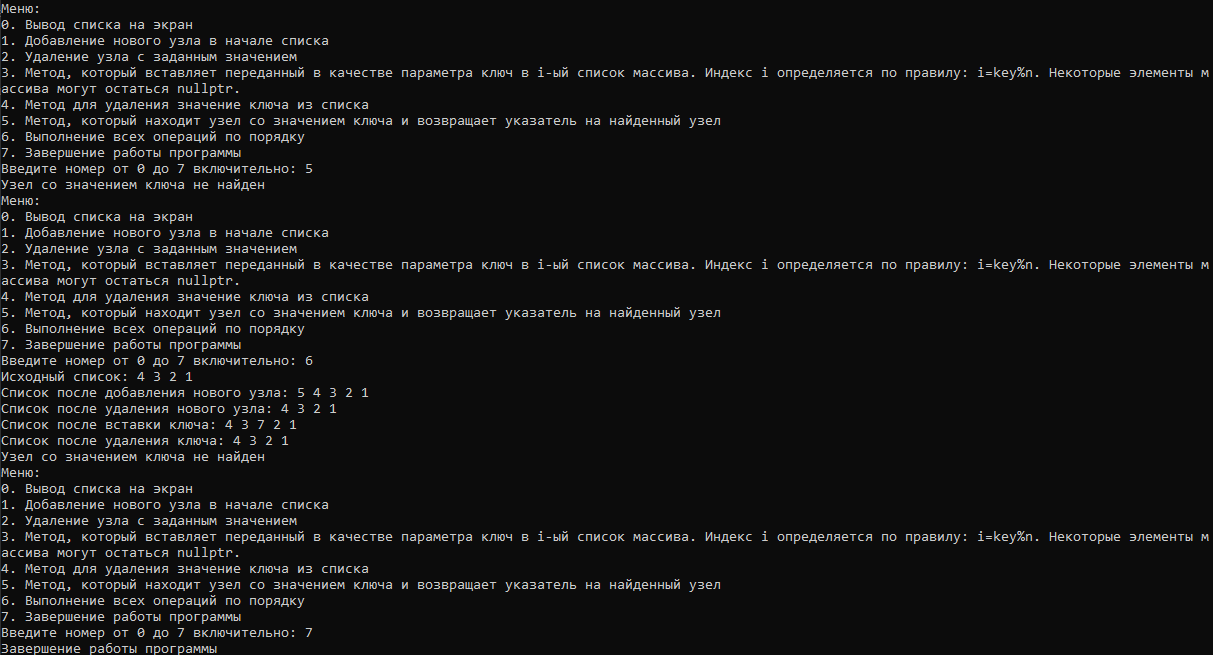


Рисунок 5 - Тестирование программы

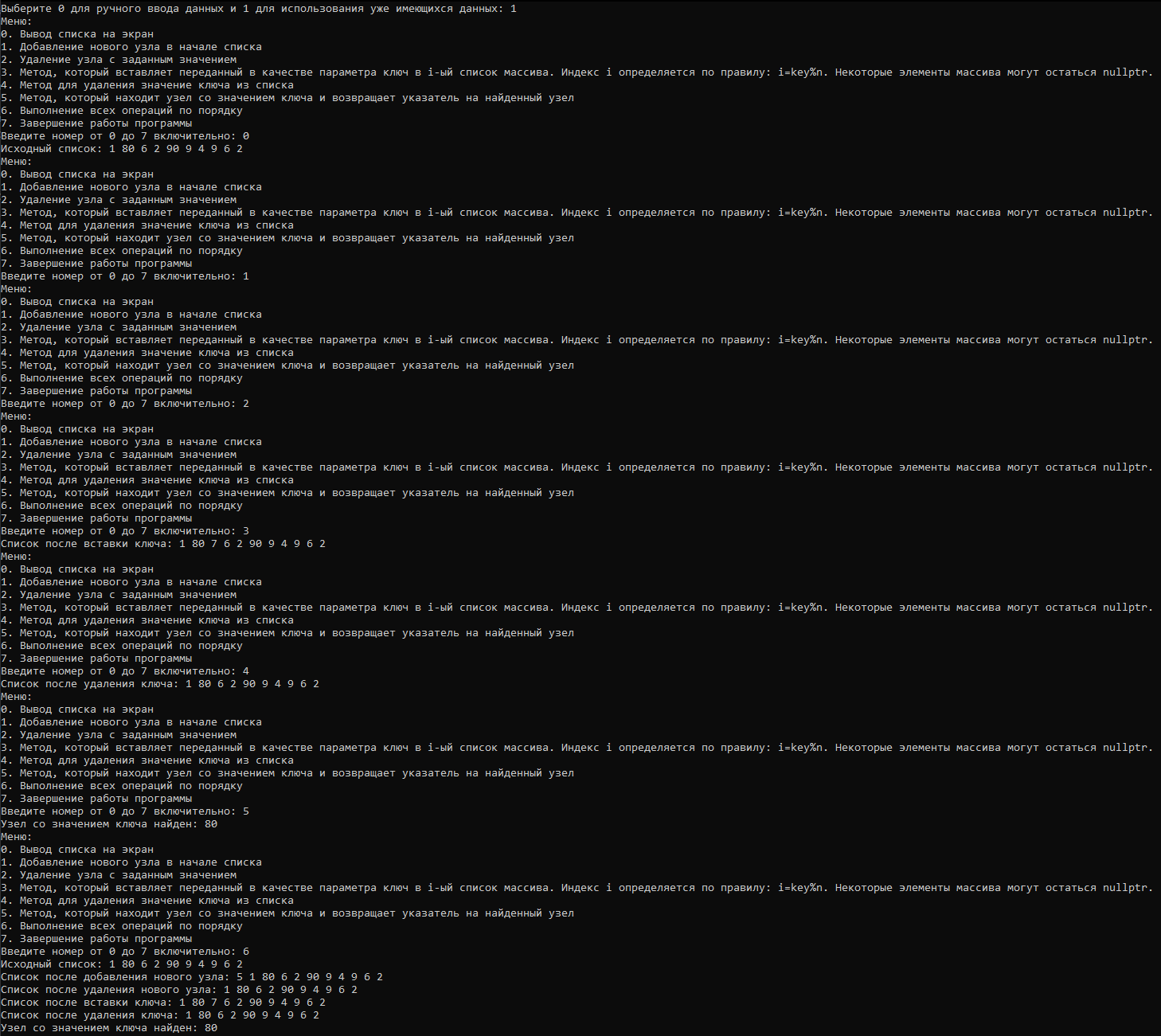


Рисунок 6 - Тестирование программы

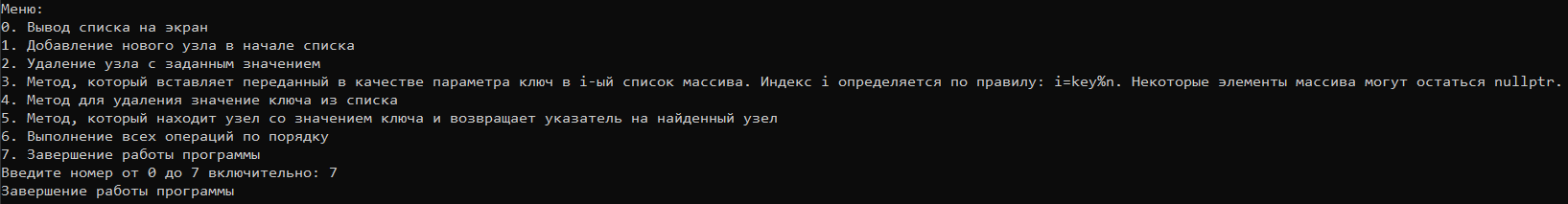
с

Рисунок 7 - Тестирование программы

## **2.4 Вывод по заданию**

Однонаправленные списки - структура данных, которая состоит из узлов, где каждый узел содержит данные и ссылку на следующий узел в списке. Эта структура обеспечивает эффективное выполнение операций добавления и удаления элементов. Однако для доступа к конкретному элементу необходимо последовательно перебирать все предыдущие элементы.

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

1. Структура узла списка: каждый узел содержит данные и указатель на следующий узел в списке. Это позволяет организовать линейную последовательность элементов.

2. Добавление элемента: для добавления элемента нужно создать новый узел и корректно настроить ссылки между узлами, чтобы вставить его в нужное место списка.

3. Удаление элемента: удаление элемента требует правильного перенаправления ссылок, чтобы исключить узел из списка и сохранить его целостность.

4. Поиск элемента: нахождение определенного элемента может потребовать прохода по всем узлам и сравнения значений с целевым элементом.

5. Операции с элементами: кроме добавления и удаления, список может подвергаться различным операциям, таким как изменение значений узлов, сортировка списка, поиск минимального/максимального элемента и другие.

Однонаправленные списки представляют собой удобную структуру данных для хранения и обработки последовательностей элементов и находят применение в различных задачах.

Мы говорим о структуре данных, где каждый элемент, или узел, состоит из данных и ссылки на следующий элемент в списке. Это называется однонаправленным списком. Этот тип структуры данных обеспечивает эффективность операций добавления и удаления элементов. Однако для доступа к определенному элементу необходимо последовательно пройти через все предыдущие элементы.

Для управления таким списком необходимо усвоить следующие основные концепции:

1. Структура узла списка: каждый узел содержит данные и указатель на следующий узел в списке, образуя цепочку.

2. Добавление элемента: чтобы добавить новый элемент, нужно создать узел с данными и настроить ссылки таким образом, чтобы новый узел стал частью списка.

3. Удаление элемента: при удалении элемента необходимо перенаправить ссылки, чтобы исключить узел из списка, сохраняя целостность.

4. Поиск элемента: для поиска определенного элемента может потребоваться последовательное сравнение значений с каждым элементом списка.

5. Операции с элементами: кроме добавления и удаления, можно проводить операции над элементами, такие как изменение их значений, сортировка, поиск минимального или максимального элемента и другие.

Такие списки представляют удобную структуру данных для работы с последовательностями элементов и находят широкое применение в различных областях программирования.

# **3 КОНТРОЛЬНЫЕ ВОПРОСЫ**

1) Три уровня представления данных в программной системе: Логический уровень: Этот уровень определяет структуру данных для конечного пользователя или приложения. Здесь устанавливаются сущности, их атрибуты и взаимосвязи. Примерами являются таблицы в реляционных базах данных или объекты в объектно-ориентированных системах. Физический уровень: Этот уровень описывает методы хранения данных на низком уровне. Здесь определяются структуры данных и форматы для эффективного доступа к данным на устройствах хранения, таких как жесткие диски или память компьютера. Внешний уровень (Уровень представления): Этот уровень обеспечивает специфическое представление данных для конкретного пользователя или приложения, скрывая сложности логического и физического уровней. Он предоставляет удобный интерфейс для работы с данными, например, пользовательский интерфейс программы или API для доступа к данным.

2) Что определяет тип данных?

Тип данных определяется набором возможных значений, которые переменная может принимать, а также операциями, которые можно выполнять над этими значениями. Он также указывает, как память выделяется для переменной и как она обрабатывается 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 между элементами данных в рамках программы или базы данных. Она устанавливает, как данные хранятся, как они могут быть доступны, и какие операции можно выполнять над ними. Структура данных может быть простой, например, массив или список, или составной, такой как дерево или граф. Выбор подходящей структуры данных способствует повышению эффективности операций с данными и упрощает реализацию программы.

4) Расскажите о структуры хранения данных в компьютерных технологиях.

Структуры хранения данных в информационных технологиях определяют методы организации и сохранения данных в памяти компьютера или на внешних устройствах хранения. Вот несколько примеров таких структур:

1. Массивы: это упорядоченные коллекции элементов одного типа данных, хранящиеся последовательно в памяти компьютера. Каждый элемент имеет свой индекс для доступа к нему.
2. Списки: это структуры данных, где элементы хранятся последовательно, и каждый элемент может быть связан с предыдущим и/или последующим.
3. Деревья: это иерархические структуры данных, где каждый элемент (узел) может иметь один или несколько дочерних элементов.
4. Графы: это структуры данных, представляющие собой совокупность вершин и ребер, определяющих их взаимосвязи.
5. Хеш-таблицы: это структуры данных, использующие хеш-функции для быстрого доступа к данным по ключу.
6. Файловые системы: это организационные структуры данных, применяемые для хранения и управления файлами на внешних носителях, таких как жесткие диски или флеш-накопители.

5) Дайте определение линейной структуре данных.

Линейная структура данных представляет собой организацию элементов в последовательной цепочке, где каждый элемент связан только с одним предшествующим и одним последующим элементом, за исключением первого и последнего элементов.

6) Дайте определение структуре данных линейный список.

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

7) Дайте определение структуре данных стек.

Стек представляет собой абстрактную структуру данных, в которой элементы хранятся в порядке "последним пришел - первым ушел" (Last-In-First-Out, LIFO). Он состоит из коллекции элементов, где операции добавления и удаления происходят только с одного конца, который называется вершиной стека. Основные операции в стеке включают добавление элемента на вершину стека (push) и удаление элемента с вершины стека (pop).

8) Дайте определение структуре данных очередь.

Очередь - абстрактная структура данных, где элементы хранятся в порядке "первым пришел - первым ушел" (First-In-First-Out, FIFO). Это означает, что операции добавления элементов происходят с одного конца, называемого хвостом очереди, а операции удаления элементов - с другого конца, называемого головой очереди. Основные операции в очереди включают 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 различные структуры данных с уникальными 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), где 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, тогда как линейный список чаще всего реализуется с использованием связанных списков.

10) Какой из видов линейных списков лучше использовать, если

нужно введенную последовательность вывести наоборот?

Рекомендуется применять односвязный список, поскольку для вывода последовательности в обратном порядке требуется пройти от начала списка до конца, сохраняя элементы на пути. Односвязный список облегчает доступ к предыдущему элементу, что упрощает выполнение 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.

а) Для массива: O(n), поскольку при удалении элемента из середины массива необходимо сдвинуть все элементы справа от позиции i на одну позицию влево.

б) Для линейного списка: O(1), потому что для удаления элемента из i-ой позиции достаточно перенаправить ссылки предыдущего и следующего узлов так, чтобы они обходили удаленный узел. Это выполняется за постоянное время и не зависит от размера списка.

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

Метод удаления элемента из однонаправленного связного списка без явного указания на предыдущий узел, известный как "трюк Вирта" или алгоритм Вирта, основан на замене данных удаляемого узла данными следующего узла, а затем удалении следующего узла. Это позволяет избежать необходимости изменения ссылки предыдущего узла на новый следующий узел. Таким образом, операция удаления становится более эффективной, и нет 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:

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

15) Реализуйте алгоритм вывода линейного однонаправленного

списка.

#include <iostream>

**using** **namespace** std**;**

struct Node **{** // Определение структуры узла списка

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 выполнения практической работы были достигнуты следующие цели:

1. Освоены принципы управления динамическим однонаправленным списком, как в теории, так и на практике.

2. Проведен анализ структуры и операций в соответствии с индивидуальным вариантом задания.

3. Реализована программа для работы с этой структурой и выполнения операций, предусмотренных заданием.

4. Созданы возможности как для вручную создания нового списка, так и для использования уже готового списка в целях тестирования.

5. Разработано меню для выбора способа ввода данных и выполнения функций.

6. Произведено тестирование программы с разнообразными данными и способами ввода.

Таким образом, главная цель практической работы — приобретение знаний и практических навыков управления динамическим однонаправленным списком — была 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).

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).