|  |
| --- |
| 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: Елисеев И.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](#_3znysh7)

[2.2.1 Определение структуры узла однонаправленного списка 5](#_2et92p0)

[2.2.2 Процесс выполнения операций 5](#_tyjcwt)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 8](#_3dy6vkm)

[2.3.1 Реализация на языке программирования C++ 8](#_1t3h5sf)

[2.3.2 Тестирование 12](#_4d34og8)

[3 КОНТРОЛЬНЫЕ ВОПРОСЫ 17](#_3rdcrjn)

[4 ВЫВОДЫ 25](#_lnxbz9)

[5 ЛИТЕРАТУРА 26](#_35nkun2)

# **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**

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

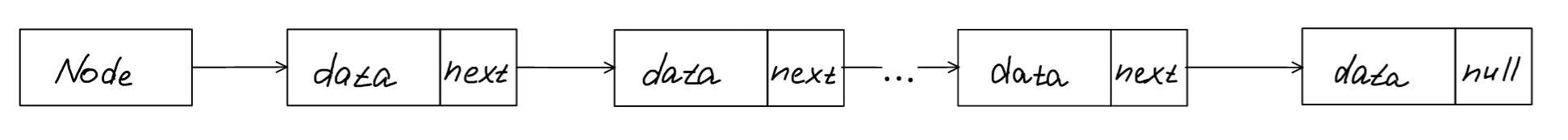


Рисунок 1 - Изображение структуры данных

### **2.2.2 Процесс выполнения операций**

1. Добавление нового узла в начало списка.

Отобразим выполнение данной операции на рисунке 2.

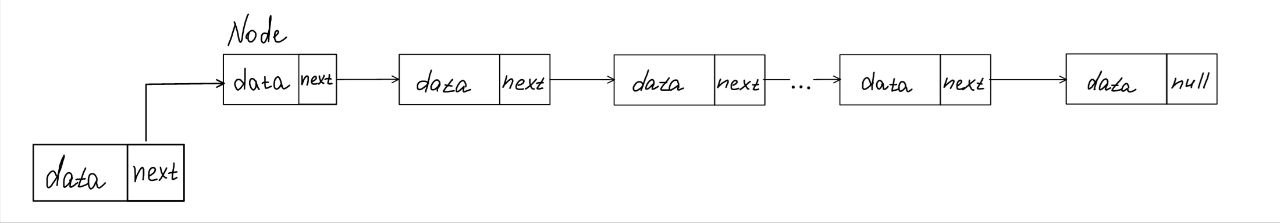


Рисунок 2 - Изображение добавление нового узла

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.

Таблица 1 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 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 |
| 2 |  | 5 | 5 |

1. Удаление узла по ключу.

Отобразим выполнение данной операции на рисунке 3.

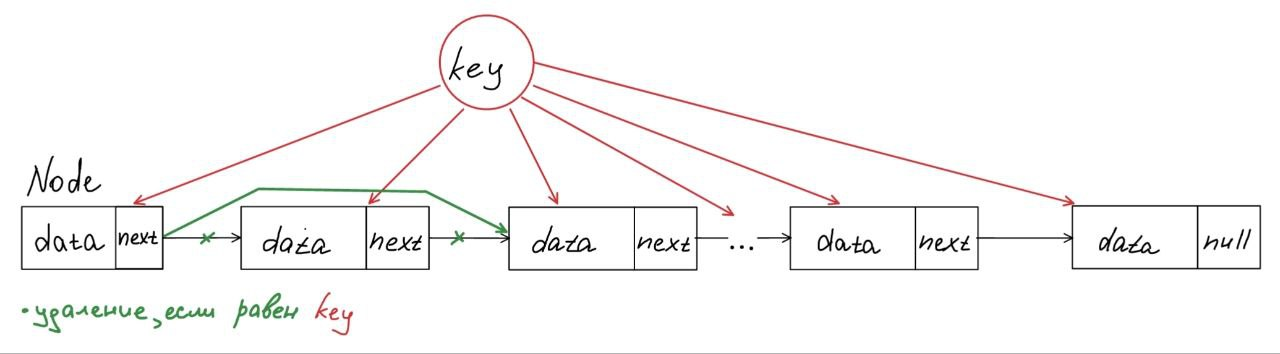


Рисунок 3 - Изображение удаления узла по ключу

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 2. Данные для тестирования будут приведены в таблице 2.

Таблица 2 - Данные для тестирования

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 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 |
| 2 | 5 | 5 |  |

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.

1. Поиск самых длинных последовательностей одинаковых элементов

Пройдем по списку, найдем самую длинную последовательность одинаковых символов и выведем информацию о ней.

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

Таблица 3 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | a 3 3 2 1 c c c f b b 3 | Длина самой длинной последовательности: 3 состоящей из c |
| 2 |  | Длина самой длинной последовательности: 0 состоящей из |

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

Начинается обход списка с начала с помощью указателя temp. Пока temp не равен nullptr. Инициализируется временный указатель curr как указатель на текущий узел temp. Внутренний цикл while продолжается, пока curr->next не равен nullptr и данные curr->next совпадают с данными текущего узла temp. По завершении внутреннего цикла curr указывает на последний узел в группе одинаковых элементов. Устанавливается указатель next текущего узла temp на узел после группы (то есть на узел после curr). temp перемещается на следующий узел после группы одинаковых элементов.

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

Таблица 4 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | a 3 3 2 1 c c c f b b 3 | a 3 2 1 c f b 3 |
| 2 |  |  |

1. Создание нового списка с уникальными цифровыми значениями.

Начинается обход исходного списка с его начала с помощью указателя temp. Создается новый пустой список newList для уникальных цифр. Пока temp не достигнет конца списка. Получается значение текущего узла temp и сохраняется в digit. Проверяется, есть ли digit в newList с помощью временного указателя checkTemp. Если digit уникален и является цифрой, он добавляется в newList методом AddendumFront. temp перемещается на следующий узел. После завершения обхода исходного списка выводится новый список newList с помощью метода outputList.

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

Таблица 5 - Данные для тестирования

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | a 3 2 1 c f b 3 | 1 2 |
| 2 |  |  |

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

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

Реализуем данный алгоритм на языке C++(блок кода 2). Для реализации понадобятся такие библиотеки, как iostream.

| #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('3');  list.AddendumFront('b');  list.AddendumFront('b');  list.AddendumFront('f');  list.AddendumFront('c');  list.AddendumFront('c');  list.AddendumFront('c');  list.AddendumFront('1');  list.AddendumFront('2');  list.AddendumFront('3');  list.AddendumFront('3');  list.AddendumFront('a');  } else {  cout << "Ошибка";  return 1;  }  do {  cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;  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 << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;  cout << "Выберете дейсвие: ";    cin >> c;    switch(c) {  case 1:  cout << "Список: ";  list.outputList();  break;  case 2:  list.AddendumFront('5');  list.outputList();  break;  case 3:  list.delNode('5');  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 1-5. Результаты тестирования №1 строки представлены на рисунках 3-4.

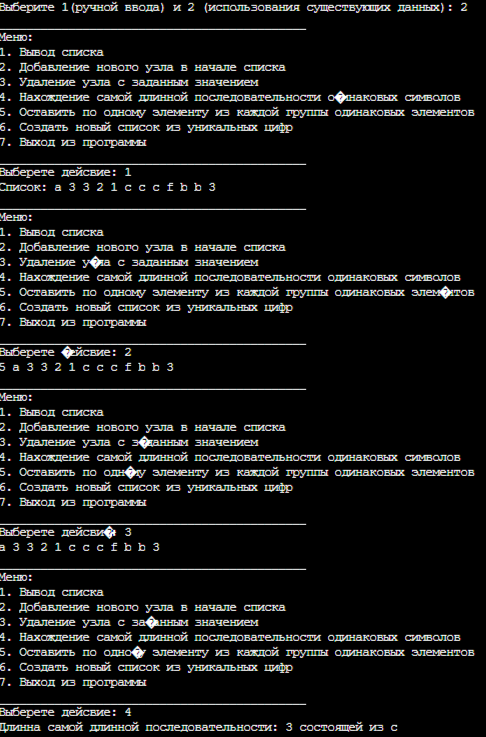


Рисунок 4 - Тестирование программы на данных №1 строки

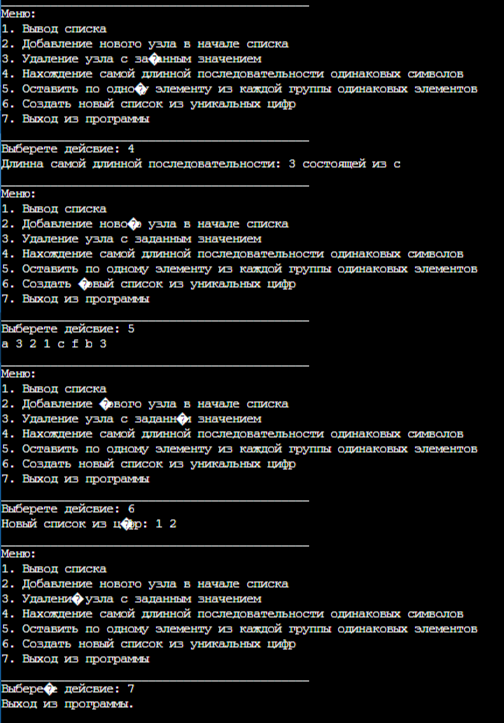


Рисунок 4 - Тестирование программы на данных №1 строки

Проведение тестирования на основе №1 строки таблиц 1-5, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

Результаты тестирования №2 строки представлены на рисунке 5-6.

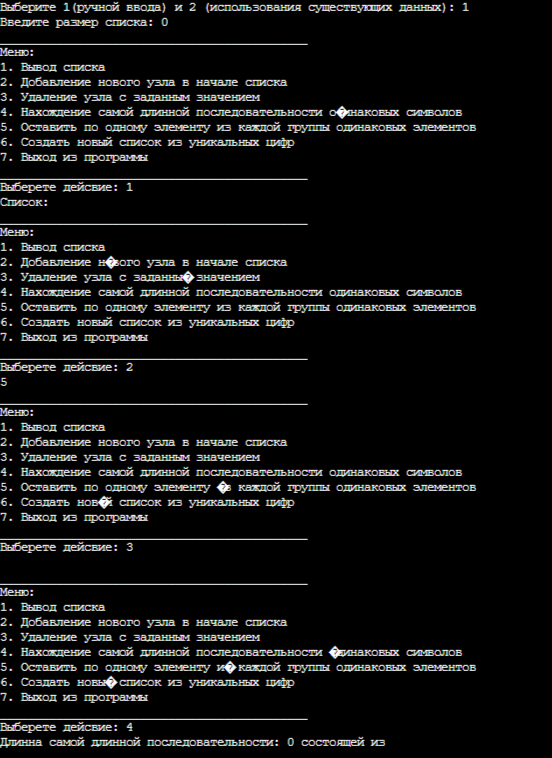


Рисунок 5 - Тестирование программы на данных №2 строки

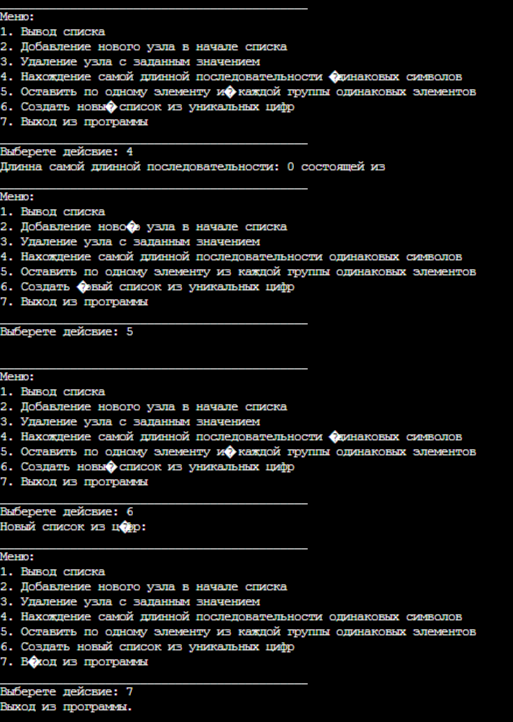


Рисунок 6 - Тестирование программы на данных №2 строки

Проведение тестирования на основе №2 строки таблиц 1-5, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

# **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.

| 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).