|  |
| --- |
| 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: |
|  | 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 6](#_tyjcwt)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 11](#_3dy6vkm)

[2.3.1 Реализация на языке программирования C++ 11](#_1t3h5sf)

[2.3.2 Тестирование 15](#_4d34og8)

[2.4 Вывод по заданию 16](#_2s8eyo1)

[3 КОНТРОЛЬНЫЕ ВОПРОСЫ 17](#_3rdcrjn)

[ЗАКЛЮЧЕНИЕ 24](#_lnxbz9)

[ЛИТЕРАТУРА 25](#_35nkun2)

# **1 ЦЕЛЬ**

Получить знания и практические навыки управления динамическим однонаправленным списком.

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи**

Вариант 2

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: float

Дополнительные операции:

Даны два линейных однонаправленных списка L1 и L2.

1. Разработать функцию, которая формирует список L, включив в него по одному разу элементы, значения которых входят одновременно в оба списка L1 и L2.

2. Разработать функцию, которая удаляет узел списка L2, расположенный перед узлом, содержащим отрицательное значение. И так для всех узлов, содержащих отрицательное значение.

3. Разработать функцию, которая вставляет новый узел с заданным значением перед каждым узлом списка L1, содержащим нечетное значение.

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

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

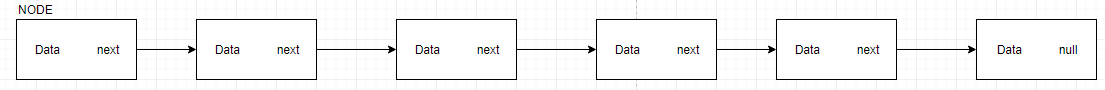
Однонаправленный список (или односвязный список) - это динамическая структура данных, состоящая из узлов. Каждый узел будет иметь какое-то значение и указатель на следующий узел. Последний узел списка ссылается на NULL, что означает конец списка. Однонаправленные списки обладают преимуществами в сравнении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

Определим структуру узла однонаправленного списка согласно варианту.

Структура Node представляет собой элемент односвязного списка. В данной структуре содержится переменная data типа char, которая хранит значение элемента, и указатель на следующий элемент списка next. Конструктор Node принимает значение типа char и инициализирует переменную data этим значением, а указатель next присваивается значение nullptr. Данное описание структуры представлено в виде кода на C++ в листинге 1, а его изображение на рисунке 2.1.

*Листинг 1 - Узел*

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



**Рисунок 2.1 - 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, формирование списка L, включив в него по одному разу элементы, значения которых входят одновременно в оба списка L1 и L2, удаление узла списка L2, расположенный перед узлом, содержащим отрицательное значение,

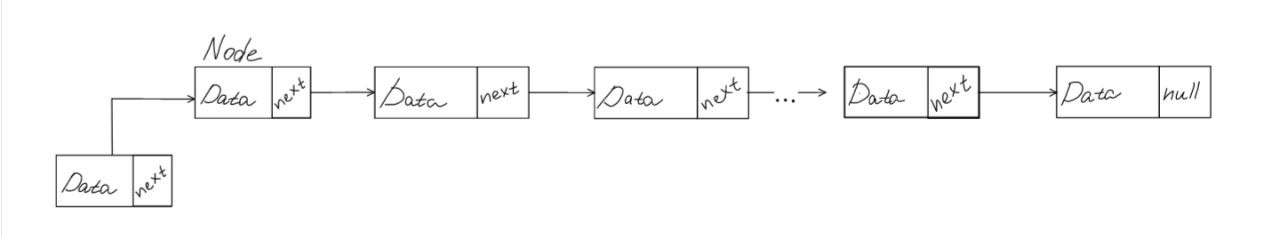
Изобразим, рассмотрим алгоритм, реализуем и предоставим таблицу с данными для тестирования данных операций.

1. Добавление нового узла в начало списка.

Каждый узел содержит два поля данных: data, которое является переменной типа float и хранит значение узла, и next, который является указателем на следующий узел в списке (Node\*).

Конструктор Node принимает аргумент float val, который инициализирует поле data узла. Поле next инициализируется значением nullptr, указывая на то, что изначально узел не имеет следующего узла в списке.

Рассмотрим алгоритм добавления нового узла в начало списка, который представлен на рисунке 2.2. Код для реализации этого алгоритма приведен в листинге 2.



**Рисунок 2.2 - Изображение добавление нового узла**

Данные для тестирования будут приведены в таблице 1.

**Таблица 1 - Данные для тестирования**

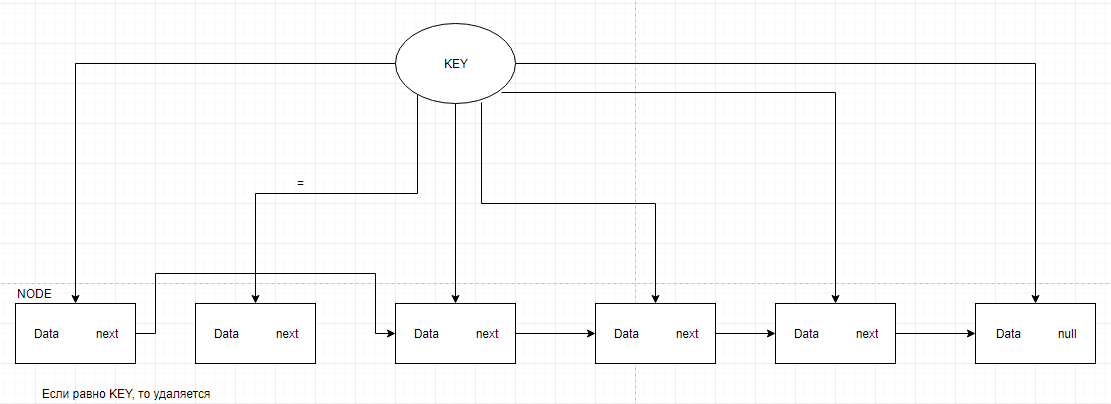
| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 67.8 46.3 53.2 2.53 1.6 | 5.5 | 5.5 67.8 46.3 53.2 2.53 1.6 |
| 2 | 5.2 -4.1 3.7 -2.3 1.5 | 3.4 | 3.4 5.2 -4.1 3.7 -2.3 1.5 |

1. Удаление узла по ключу.

Функция сначала проверяет, пуст ли список. Если список пуст, функция завершает свою работу. Затем она проверяет, является ли первый узел узлом, который нужно удалить. Если да, то первый узел переназначается на следующий узел, а старый первый узел освобождается из памяти. Если первый узел не является тем, который нужно удалить, происходит поиск узла с ключом key. Пока не достигнут конец списка и ключ текущего узла не равен key, функция переходит к следующему узлу. Если найден узел с ключом key, он удаляется из списка. Для этого переназначается указатель next предыдущего узла так, чтобы он указывал на узел после удаляемого узла, а затем освобождается память, занятая удаляемым узлом.

Реализация данного алгоритма представлена в листинге 2.

Отобразим выполнение данной операции на рисунке 3.



**Рисунок 3 - Изображение операции**

Данные для тестирования будут приведены в таблице 2.

**Таблица 2 - Данные для тестирования**

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 5.5 67.8 46.3 53.2 2.53 1.6 | 5.5 | 67.8 46.3 53.2 2.53 1.6 |
| 2 | 3.4 5.2 -4.1 3.7 -2.3 1.5 | 3.4 | 5.2 -4.1 3.7 -2.3 1.5 |

1. Вывод элементов списка на экран.

Функция начинает с проверки, является ли указатель head пустым. Если head равен nullptr, это означает, что список пуст, и функция завершает свою работу. Затем функция переходит в цикл while, который будет выполняться, пока указатель head не станет nullptr. Это означает, что мы еще не достигли конца списка. На каждой итерации цикла функция выводит значение поля data текущего узла в консоль с помощью cout. Затем указатель head переносится на следующий узел списка, чтобы перейти к следующей итерации цикла. Когда цикл завершается (когда head становится равным nullptr), функция выводит перевод строки, чтобы завершить вывод списка.

Реализация в листинге 2.

1. Формирование списка, включив в него по одному разу элементы, значения которых входят одновременно в оба списка

Создается новый пустой список intersectionList, который будет содержать пересечение элементов из списков L1 и L2. Устанавливается указатель currentL1 на начало списка L1. Запускается цикл, который проходит по всем узлам списка L1, пока указатель currentL1 не станет равным nullptr. На каждом шаге цикла проверяется, содержится ли значение текущего узла списка L1 в списке L2. Для этого используется функция search, которая выполняет поиск значения в списке L2. Если значение текущего узла списка L1 присутствует в списке L2, то это значение добавляется в список intersectionList перед первым узлом. Указатель current L1 перемещается на следующий узел списка L1. Цикл завершается, когда все узлы списка L1 пройдены. Функция возвращает список intersectionList, содержащий пересечение элементов из списков L1 и L2.

Реализация данного алгоритма представлена в блок коде 5. Данные для тестирования будут приведены в таблице 3, в которой представлены все тестовые данные по дополнительным операциям. Данные по данной операции будут представлены в 1 строке.

1. Удаление узла списка L2, расположенный перед узлом, содержащим отрицательное значение

Устанавливается указатель current на начало списка, то есть на головной узел. Запускается цикл, который будет продолжаться, пока указатель current не достигнет конца списка (то есть не станет равным nullptr). На каждом шаге цикла проверяется значение данных текущего узла списка (current->data) на равенство заданному ключу (key). Если значение данных текущего узла равно заданному ключу, то метод возвращает true, что указывает на то, что элемент найден. Если значение данных текущего узла не равно заданному ключу, указатель current перемещается на следующий узел списка. Цикл продолжается до тех пор, пока не будет пройден весь список или пока не будет найден элемент с заданным ключом. Если элемент с заданным ключом не был найден после прохода по всему списку, метод возвращает false, что указывает на то, что элемент отсутствует в списке. Устанавливается указатель current на начало списка, то есть на головной узел. Устанавливается указатель prev на nullptr, так как изначально перед головным узлом нет других узлов. Запускается цикл, который будет продолжаться, пока указатель current не достигнет конца списка (то есть не станет равным nullptr). На каждом шаге цикла проверяется значение данных текущего узла списка (current->data) на отрицательность. Если значение данных текущего узла отрицательно, цикл прерывается с помощью оператора break, так как далее необходимо сохранить все узлы. В противном случае (если значение данных текущего узла не отрицательно), указатель prev устанавливается на текущий узел, а указатель current перемещается на следующий узел списка. После завершения цикла проверяется условие, что указатель current не равен nullptr (то есть список не закончился) и указатель prev не равен nullptr (то есть текущий узел не первый в списке). Если оба условия выполняются, то устанавливается ссылка next предыдущего узла prev на nullptr, чтобы обрубить связь с последующими узлами. Затем освобождается память, занятая текущим узлом current, с помощью оператора delete. Выполнение метода завершается.

Реализация данного алгоритма представлена в листинге 2. Данные для тестирования будут приведены в таблице 3, в которой представлены все тестовые данные по дополнительным операциям. Данные по данной операции будут представлены во 2 строке.

1. Добавление нового узла с заданным значением перед каждым узлом списка L1, содержащим нечетное значение

Устанавливается указатель current на начало списка, то есть на головной узел. Устанавливается указатель prev на nullptr, так как изначально перед головным узлом нет других узлов. Запускается цикл, который будет продолжаться, пока указатель current не достигнет конца списка (то есть не станет равным nullptr). На каждом шаге цикла проверяется, является ли значение данных текущего узла списка (current->data) нечетным числом. Для этого вычисляется остаток от деления на 2 и проверяется, не равен ли он нулю. Если значение данных текущего узла нечетное, то создается новый узел newNode с переданным значением value. Проверяется, есть ли у предыдущего узла prev следующий узел. Если да, то ссылка next предыдущего узла устанавливается на новый узел newNode, чтобы он указывал на него. В противном случае (если предыдущего узла нет), головной указатель head списка устанавливается на новый узел newNode. Устанавливается ссылка next нового узла newNode на текущий узел current, чтобы он указывал на него. Указатель prev обновляется и устанавливается на новый узел newNode, чтобы он указывал на последний вставленный узел. Затем указатель current идет на следующий узел списка. Цикл повторяется до тех пор, пока не будет пройден весь список.

Реализация данного алгоритма представлена в листинге 2. Данные для тестирования будут приведены в таблице 3, в которой представлены все тестовые данные по дополнительным операциям. Данные по данной операции будут представлены в 3 строке.

**Таблица 3 - Данные для тестирования**

| № | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | L1: 3.7 -2.3 1.5  L2: 3.7 5.2 -1.5 | 3.7 |
| 2 | L1: 3.7 -2.3 1.5  L2: 3.7 5.2 -1.5 | 3.7 5.2 |
| 3 | L1: 3.7 -2.3 1.5  L2: 3.7 5.2 -1.5 | 8.9 5.2 -4.1 8.9 3.7 -2.3 8.9 1.5 |

## **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.

*Листинг 2 – Программа однонаправленного списка с выбором операций*

| #include <iostream>  using namespace std;  struct Node {  float data;  Node\* next;  Node(float value) : data(value), next(nullptr) {}  };  class SinglyLinkedList {  public:  SinglyLinkedList() : head(nullptr) {}  // Функция для вставки нового узла перед первым узлом  void insertBeforeFirst(float value) {  Node\* newNode = new Node(value);  newNode->next = head;  head = newNode;  }  // Функция для удаления узла по ключу (значению)  void deleteByKey(float key) {  Node\* current = head;  Node\* prev = nullptr;  // Поиск узла с заданным ключом  while (current != nullptr && current->data != key) {  prev = current;  current = current->next;  }  // Если узел с ключом найден, удаляем его  if (current != nullptr) {  if (prev != nullptr) {  prev->next = current->next;  }  else {  head = current->next;  }  delete current;  }  } |
| --- |

*Листинг 2 – Продолжение*

| // Функция для вывода списка в консоль  void printList() {  Node\* current = head;  while (current != nullptr) {  cout << current->data << " ";  current = current->next;  }  cout << endl;  }  // Функция для формирования списка, включив в него по одному разу элементы, значения которых входят одновременно в оба списка L1 и L2.  SinglyLinkedList intersection(const SinglyLinkedList& L1, const SinglyLinkedList& L2) {  SinglyLinkedList intersectionList;  Node\* currentL1 = L1.head;  while (currentL1 != nullptr) {  if (L2.search(currentL1->data)) {  intersectionList.insertBeforeFirst(currentL1->data);  }  currentL1 = currentL1->next;  }  return intersectionList;  }  // Функция для поиска элемента в списке  bool search(float key) const {  Node\* current = head;  while (current != nullptr) {  if (current->data == key) {  return true;  }  current = current->next;  }  return false;  }  // Функция для удаления узлов списка L2, расположенных перед узлом, содержащим отрицательное значение.  void deleteBeforeNegative() {  Node\* current = head;  Node\* prev = nullptr;  while (current != nullptr) {  if (current->data < 0) {  break;  }  prev = current;  current = current->next;  }  if (current != nullptr && prev != nullptr) {  prev->next = nullptr;  delete current;  }  } |
| --- |

*Листинг 2 – Продолжение*

| // Функция для вставки нового узла с заданным значением перед каждым узлом списка L1, содержащим нечетное значение  void insertBeforeOdd(float value) {  Node\* current = head;  Node\* prev = nullptr;  while (current != nullptr) {  if ((int)current->data % 2 != 0) {  Node\* newNode = new Node(value);  if (prev != nullptr) {  prev->next = newNode;  }  else {  head = newNode;  }  newNode->next = current;  prev = newNode;  }  prev = current;  current = current->next;  }  }  private:  Node\* head;  };  int main() {  setlocale(LC\_ALL, "Russian");  SinglyLinkedList list;  int choice, choice\_method;;  cout << "Выберите 1 для ручного ввода данных и 2 для использования уже имеющихся данных: ";  cin >> choice;  if (choice == 1) {  int size;  float value;  cout << "Введите размер списка: ";  cin >> size;  cout << "Введите значения для списка: ";  for (int i = 0; i < size; ++i) {  cin >> value;  list.insertBeforeFirst(value);  }  }  else if (choice == 2) {  list.insertBeforeFirst(1.5);  list.insertBeforeFirst(-2.3);  list.insertBeforeFirst(3.7);  list.insertBeforeFirst(-4.1);  list.insertBeforeFirst(5.2);  }  else {  cout << "Ошибка: неверный выбор." << endl;  return 1;  } |
| --- |

*Листинг 2 – Продолжение*

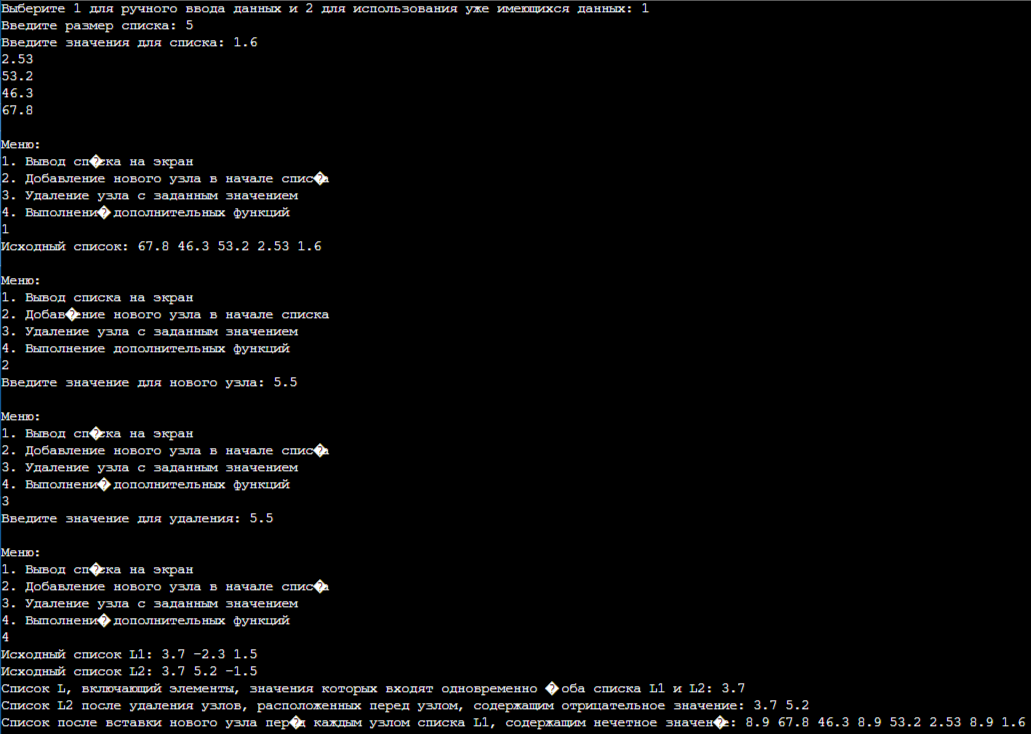
| // Меню для выполнения операций с списком  do {  cout << "\nМеню:" << endl;  cout << "1. Вывод списка на экран" << endl;  cout << "2. Добавление нового узла в начале списка" << endl;  cout << "3. Удаление узла с заданным значением" << endl;  cout << "4. Выполнение дополнительных функций" << endl;  cin >> choice\_method;  switch (choice\_method) {  case 1:  cout << "Исходный список: ";  list.printList();  break;  case 2:  float newValue;  cout << "Введите значение для нового узла: ";  cin >> newValue;  list.insertBeforeFirst(newValue);  break;  case 3:  float deleteValue;  cout << "Введите значение для удаления: ";  cin >> deleteValue;  list.deleteByKey(deleteValue);  break;  case 4:  // Создаем списки L1 и L2 для тестирования  SinglyLinkedList L1;  SinglyLinkedList L2;  // Заполняем списки значениями для тестирования  L1.insertBeforeFirst(1.5);  L1.insertBeforeFirst(-2.3);  L1.insertBeforeFirst(3.7);  L2.insertBeforeFirst(-4.1);  L2.insertBeforeFirst(5.2);  L2.insertBeforeFirst(6.6);  // Выводим исходные списки  cout << "Исходный список L1: ";  L1.printList();  cout << "Исходный список L2: ";  L2.printList();  // Формируем список, включающий значения из списков L1 и L2  SinglyLinkedList mergedList = list.intersection(L1, L2);  cout << "Список L, включающий элементы, значения которых входят одновременно в оба списка L1 и L2: ";  mergedList.printList();  // Удаляем из списка L2 все узлы, расположенные перед узлом, содержащим отрицательное значение  L2.deleteBeforeNegative();  cout << "Список L2 после удаления узлов, расположенных перед узлом, содержащим отрицательное значение: ";  L2.printList(); |
| --- |

*Листинг 2 – Окончание*

| // Вставляем новый узел с заданным значением перед каждым узлом списка L1, содержащим нечетное значение  list.insertBeforeOdd(8.9);  cout << "Список после вставки нового узла перед каждым узлом списка L1, содержащим нечетное значение: ";  list.printList();  break;  }  } while (choice\_method != 4);  return 0;  } |
| --- |

### **2.3.2 Тестирование**

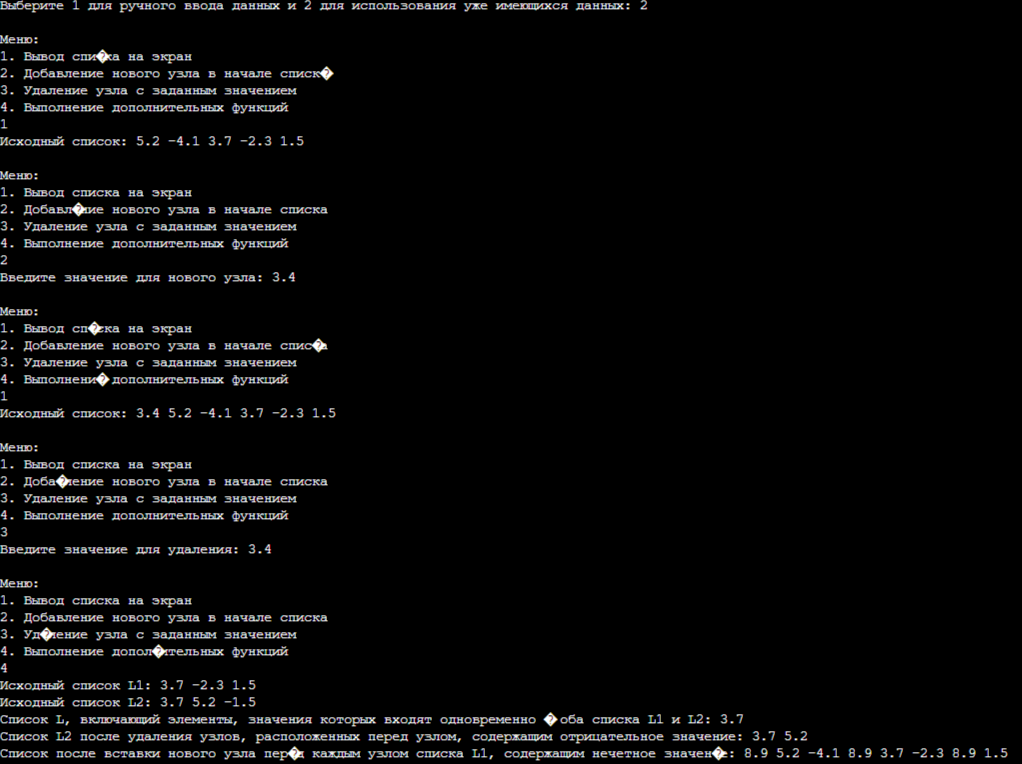
Проведем тестирование основываясь на данные из таблиц 1-3. Результаты тестирования №1 строки из таблиц 1-2 и всей таблицы 3 представлены на рисунке 2.4.



**Рисунок 4 - Тестирование программы на данных №1 строки**

Проведение тестирования, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

Результаты тестирования №2 строки таблиц 1-2 представлены на рисунке 2.5.



**Рисунок 2.5 - Тестирование программы на данных №2 строки**

Проведение тестирования, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

## **2.4 Вывод по заданию**

Однонаправленные списки состоят из узлов, каждый из которых содержит данные и ссылку на следующий узел. Эта структура данных обеспечивает эффективные операции добавления и удаления элементов, поскольку доступ к узлу возможен только через предшествующие элементы. Для управления такими списками важны структура узла, методы добавления и удаления элементов, а также операции поиска, изменения значений и сортировки. Однонаправленные списки представляют собой удобный инструмент для хранения и обработки последовательностей элементов в различных задача.

# **3 КОНТРОЛЬНЫЕ ВОПРОСЫ**

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

Три уровня представления данных в программной системе:

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

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

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

2) Что определяет тип данных?

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

3) Что определяет структура данных?

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

4) Расскажите о структуры хранения данных в компьютерных технологиях.

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

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

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

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

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

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

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

5) Дайте определение линейной структуре данных.

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

6) Дайте определение структуре данных линейный список.

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

7) Дайте определение структуре данных стек.

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

8) Дайте определение структуре данных очередь.

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

9) Чем стек отличается от структуры данных линейный список?

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

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

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

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

10) Какой из видов линейных списков лучше использовать, если

нужно введенную последовательность вывести наоборот?

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

11) Определите сложность алгоритма операции вставки элемента в

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

Сложность алгоритма операции вставки элемента в i-ую позицию:

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

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

12) Определите сложность алгоритма операции удаления элемента из

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

Сложность алгоритма операции удаления элемента из i-ой позиции:

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

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

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

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

14) Определите структур узла однонаправленного списка.

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

Структура узла содержит два поля:

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

15) Реализуйте алгоритм вывода линейного однонаправленного списка.

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

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

| #include <iostream>  using namespace std;  struct Node {  int data;  Node\* next;  };  void moveLastToFirst(Node\* &head) {  if (head == nullptr || head->next == nullptr) // если список пустой или состоит из одного элемента, ничего не делаем  return;    Node\* last = head;  Node\* secondLast = nullptr;  // Находим последний и предпоследний элементы списка  while (last->next != nullptr) {  secondLast = last;  last = last->next;  }  // Перемещаем последний элемент в начало списка  secondLast->next = nullptr;  last->next = head;  head = last;  }  // Пример использования:  int main() {  Node\* head = nullptr;  // Заполнение списка  for (int i = 1; i <= 5; ++i) {  Node\* newNode = new Node{i, nullptr};  newNode->next = head;  head = newNode;  }  // Перемещение последнего элемента в начало списка  moveLastToFirst(head);  // Вывод списка  Node\* current = head;  while (current != nullptr) {  cout << current->data << " ";  current = current->next;  }  return 0;  } |
| --- |

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

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

В указанном фрагменте кода лишнее действие - присваивание q->next=LL->next;, потому что оно выполняется перед условием if (LL==nullptr) и также после этого условия. Новый узел вставляется в начало списка.

# 

# **ЗАКЛЮЧЕНИЕ**

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.

# **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).