Министерство образования и науки Российской Федерации

Федеральное государственное образовательное учреждение высшего профессионального образования

«Московский авиационный институт»

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

2 семестр

# Курсовой проект

## По курсу «Алгоритмы и структуры данных»

Задание VIII

Работ сдал: студент группы М8О-204Б-22

Филиппов Фёдор Иванович

Работу принял: преподаватель информатики

Потенко Максим Алексеевич

Москва, 2023

# Оглавление

# Цель работы

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением списка на динамические структуры. Навигацию по списку следует реализовать с применением итераторов. Предусмотреть выполнение одного нестандартного и четырех стандартных действий:

1. Печать списка
2. Вставка нового элемента в список
3. Удаление элемента из списка
4. Подсчет длины списка

# Задание

*Тип элемента списка:* Вещественный.

*Вид списка:* Линейный однонаправленный.

*Нестандартное действие:* Переставить элементы списка в обратном порядке.

# Теоретическая справка

## Линейный список

Линейный список в программировании представляет собой структуру данных, которая позволяет хранить и организовывать элементы последовательно. Каждый элемент списка, называемый узлом, содержит данные и указатель на следующий узел. Это позволяет обеспечить гибкую и динамическую структуру данных, где элементы могут быть добавлены и удалены по мере необходимости.

Основные операции, которые можно выполнять со списком, включают добавление элемента, удаление элемента, поиск элемента, получение размера списка и печать списка. Вот некоторые общие операции для работы с линейным списком:

1. Добавление элемента: Для добавления элемента в список необходимо создать новый узел, заполнить его данными и установить указатель предыдущего узла на новый узел, а указатель нового узла на следующий узел. Если список пуст, то новый узел становится первым элементом списка.

2. Удаление элемента: Для удаления элемента из списка необходимо найти узел, предшествующий удаляемому узлу, и изменить его указатель на следующий узел. Затем можно освободить память, занятую удаляемым узлом.

3. Поиск элемента: Для поиска элемента в списке нужно последовательно просматривать узлы, сравнивая данные каждого узла с заданным значением. Если совпадение найдено, возвращается указатель на этот узел. Если элемент не найден, возвращается специальное значение, обозначающее отсутствие элемента.

4. Получение размера списка: Для определения размера списка необходимо пройти по всем узлам, подсчитывая их количество, пока не будет достигнут конец списка.

5. Печать списка: Для вывода содержимого списка на экран нужно последовательно просмотреть все узлы и вывести их данные.

Линейные списки полезны, когда требуется динамически изменять размер структуры данных, добавлять или удалять элементы в середине списка. Они широко используются для реализации других структур данных, таких как стеки, очереди и связанные списки. Понимание работы с линейными списками позволяет разрабатывать эффективные алгоритмы и решать разнообразные задачи в программировании.

# Используемое оборудование

ЭВМ iMac 2012 Late 2012 21.5’

Процессор Intel Core i5 4 ядра

ОП 16324 Мб

НМД 524288 Мб

# Реализация программы

## Описание функций spisok.c

**create_list():**

Инициализирует и возвращает новый пустой список.

Выделяет память под новый список.

Если происходит ошибка выделения памяти, функция выводит сообщение об ошибке и завершает программу.

Инициализирует размер списка как 0.

Устанавливает корневой узел списка как NULL.

***append(list l, double value)**:**

Добавляет новый элемент со значением 'value' в конец списка 'l'.

Выделяет память под новый узел.

Если происходит ошибка выделения памяти, функция выводит сообщение об ошибке и завершает программу.

Если список пуст, новый элемент становится корневым узлом.

Если список не пуст, новый элемент добавляется в конец списка.

***remove_node(list l, int index)**:**

Удаляет узел с заданным индексом 'index' из списка 'l'.

Проверяет, находится ли индекс в пределах допустимого диапазона.

Если индекс равен 0, удаляется первый элемент списка.

Если индекс больше 0, удаляется элемент, не являющийся первым.

Освобождает память, выделенную под удаленный узел.

***length(const list l)**:**

Возвращает текущий размер списка 'l'.

***print_list(list l)**:**

Выводит значения всех элементов списка 'l' на экран.

Использует формат с плавающей точкой с двумя знаками после запятой.

***free_list(list l)**:**

Освобождает память, выделенную под все элементы списка 'l' и сам список.

***reverse_list(list l)**:**

Изменяет порядок элементов списка 'l' на обратный.

Для этого используется изменение указателей на следующий элемент.

# Распечатка программы

**Spisok.h**

```
#ifndef SPISOK
#define SPISOK

#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    double value;
    struct node* next;
} node;
```

```
typedef struct list {
    int size;
    struct node* root;
} list;

// Инициализация пустого списка
list* create_list();

// Добавление элемента в конец списка
void append(list* l, double value);

// Удаление элемента по индексу
void remove_node(list* l, int idx);

// Длина списка
int length(const list* l);

// Вывод содержимого списка
void print_list(list* l);

// Освобождение памяти, выделенной под список
void free_list(list* l);

// Перестановка списка в обратном порядке
void reverse_list(list* l);

#endif
```

## Spisok.c

```
#include "spisok.h"
```

```
#include <stdio.h>
#include <stdlib.h>

// Инициализация пустого списка
list* create_list() {
    list* new_list = (list*)malloc(sizeof(list));
    if (new_list == NULL) {
        perror("Ошибка выделения памяти");
        exit(1);
    }
    new_list->size = 0;
    new_list->root = NULL;
    return new_list;
}

// Добавление элемента в конец списка
void append(list* l, double value) {
    node* n = (node*)malloc(sizeof(node));
    if (n == NULL) {
        perror("Ошибка выделения памяти");
        exit(1);
    }
    n->value = value;
    n->next = NULL;

    if (l->root == NULL) {
        l->root = n;
    } else {
```

```
        node* current = l->root;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = n;
    }
    l->size++;
    return;
}

void remove_node(list* l, int index) {
    if (index < 0 || index >= l->size) {
        printf("Индекс за пределами допустимого диапазона\n");
        return;
    }

    if (index == 0) {
        // Удаляем первый элемент
        node* temp = l->root;
        l->root = l->root->next;
        free(temp);
    } else {
        // Удаляем элемент, не являющийся первым
        node* cur = l->root;
        for (int i = 0; i < index - 1; i++) {
            cur = cur->next;
        }
        node* temp = cur->next;
        cur->next = temp->next;
```

```
        free(temp);
    }

    l->size--;
    return;
}

// Узнать длину списка
int length(const list* l) {
    return l->size;
}

// Вывод содержимого списка
void print_list(list* l) {
    node* current = l->root;
    while (current != NULL) {
        printf("[%.2lf] ", current->value);
        current = current->next;
    }
    printf("\n");
    return;
}

// Освобождение памяти, выделенной под список
void free_list(list* l) {
    node* cur = l->root;
    while (cur != NULL) {
        node* next = cur->next;
        free(cur);
```

```
        cur = next;
    }
    free(l);
    return;
}

// Перестановка списка в обратном порядке
void reverse_list(list* l) {
    node* prev = NULL;
    node* cur = l->root;
    node* next = NULL;

    while (cur != NULL) {
        next = cur->next;
        cur->next = prev;
        prev = cur;
        cur = next;
    }

    l->root = prev;
    return;
}
```

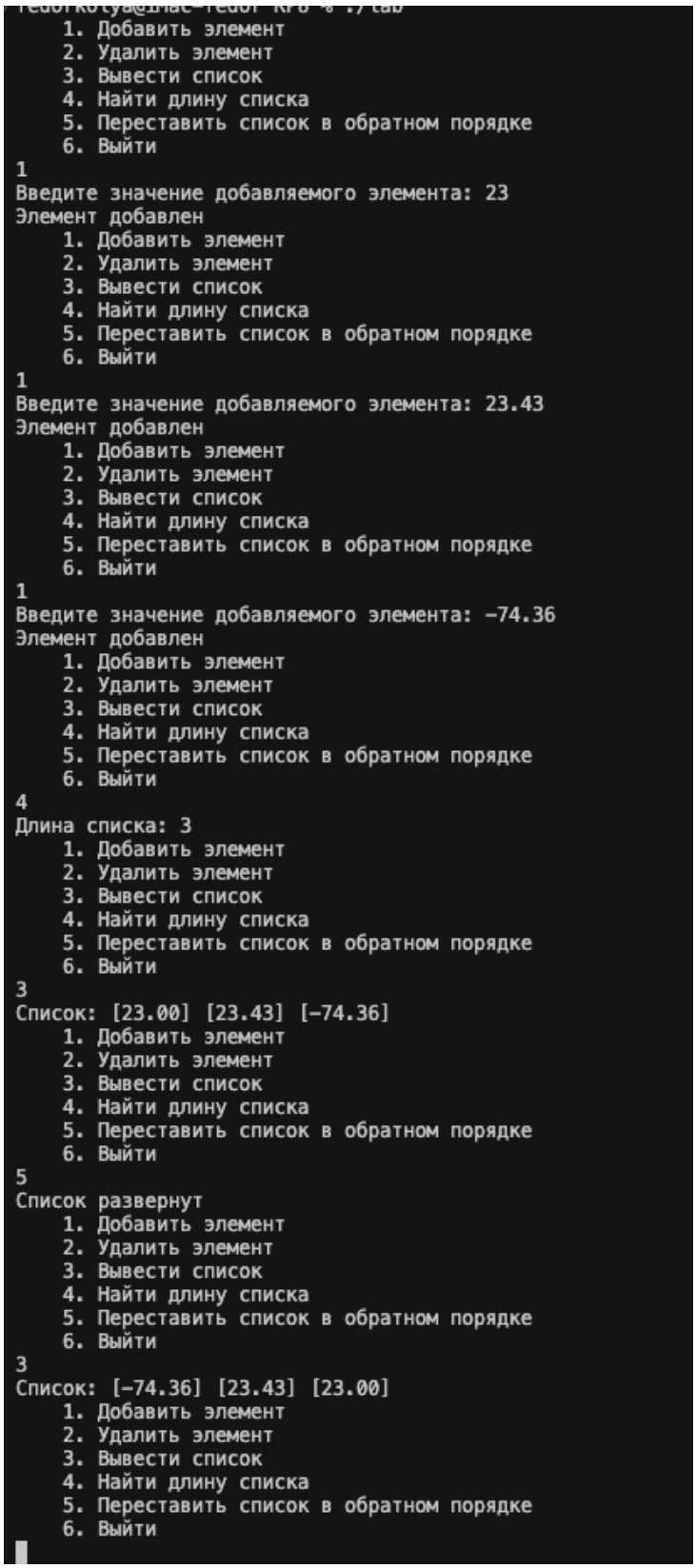
## Main.c

```
#include "spisok.h"
#include <stdio.h>

int main() {
```

```
list* l = create_list();

while (1) {
    double value;
    int choice = 0;
    printf("    1. Добавить элемент\n");
    printf("    2. Удалить элемент\n");
    printf("    3. Вывести список\n");
    printf("    4. Найти длину списка\n");
    printf("    5. Переставить список в обратном порядке\n");
    printf("    6. Выйти\n");

    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Введите значение добавляемого элемента: ");
            scanf("%lf", &value);
            append(l, value);
            printf("Элемент добавлен\n");
            break;
        case 2:
            printf("Введите номер элемента: ");
            int num;
            scanf("%d", &num);
            remove_node(l, num - 1);
            printf("Элемент удален\n");
            break;
        case 3:
```

```
            printf("Список: ");
            print_list(l);
            break;
        case 4:
            printf("Длина списка: %d\n", length(l));
            break;
        case 5:
            reverse_list(l);
            printf("Список развернут\n");
            break;
        case 6:
            free_list(l);
            return 0;
            break;
        default:
            printf("Некорректный ввод\n");
            break;
      }
   }
   return 0;
}
```

# Результат работы программы



# Вывод

В заключение хочется сказать, что проделанная работа укрепила мои знания о линейных списках и их обработке в языке C. В процессе выполнения задания я научился создавать и отлаживать программы, использующие линейные списки для организации данных.

Одной из ключевых задач было создание и управление динамической структурой данных. Я освоил работу с указателями и динамическим выделением памяти для создания и добавления новых элементов в список. Благодаря использованию итераторов я научился эффективно перемещаться по списку и выполнять операции вставки, удаления и обновления элементов.

Теперь операции, такие как добавление элемента в начало, конец или по определенной позиции, удаление элемента по индексу, получение размера списка, а также вывод списка на экран не представляют для меня трудности.

В результате я получил практические навыки работы с линейными списками и увидел их применение в реальных задачах программирования. Эти знания будут полезны для меня в будущем при разработке программ, требующих эффективной организации и обработки данных, особенно в случаях, когда количество элементов заранее неизвестно и может изменяться динамически.

# Использованные источники


1. https://ru.wikipedia.org/wiki/Связный_список
2. https://medium.com/nuances-of-programming/структуры-данных-которые-необходимо-знать-каждому-программисту-235d4315e5b9