**Отчёт по курсовой работе 8**

по курсу Алгоритмы и структуры данных

Студент группы М8О-103Б-23, Махмутов Дэниз Ирикович, № по списку 17

Контакты www, e-mail, icq, skype e-mail: lustmodesta@gmai.com

Работа выполнена «20» мая 2024 г.

Преподаватель: Никулин Сергей Петрович каф. 806

Входной контроль знаний с оценкой \_\_\_\_\_\_

Отчет сдан «22» мая 2024 г., итоговая оценка \_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_

1. **Тема: Линейные списки**
2. **Цель работы:**

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением

списка на динамические структуры

1. **Задание (***вариант №17***):** ТИП ЭЛЕМЕНТА СПИСКА: Вещественный, ВИД СПИСКА линейный однонаправленный с барьерным элементом;

НЕСТАНДАРТНОЕ ДЕЙСТВИЕ удалить из списка все элементы, предшествующие и последующие заданному значению;

#### Дек - Поиск и удаление максимального элемента - Сортировка линейным выбором

**Оборудование (лабораторное):**

ЭВМ -, процессор -, имя узла сети - с ОП - ГБ,

НМД - ГБ, терминал- адрес -, принтер -

Другие устройства –

*Оборудование ПЭВМ студента, если использовалось:*

Процессор: AMD Ryzen 5 5600x c ОП 16 Гб, НМД 1024 Гб. Монитор 1920×1080 пикс.

Другие устройства –

1. **Программное обеспечение (лабораторное):**

Операционная система семейства -, наименование - версия -

интерпретатор команд - версия

Система программирования - версия -

Редактор текстов - версия -

Утилиты операционной системы -

Прикладные системы и программы -

Местонахождение и имена файлов программ и данных –

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства GNU/Linux наименование Ubuntu версия 20.04.3 LTS

Интерпретатор команд GNOME Terminal версия 3.36.2

Система программирования – версия –

Редактор текстов emacs версия 3.24.14

Прикладные системы и программы –

Местонахождение и имена файлов программ и данных на домашнем компьютере –

1. **Идея, метод, алгоритм**  решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)
2. **Инициализация списка**: initializeList создает барьерный элемент.
3. **Печать списка**: printList проходит по списку и печатает каждый элемент.
4. **Вставка нового элемента**: insertNode добавляет новый элемент в конец списка.
5. **Удаление элемента**: deleteNode находит и удаляет элемент со значением value.
6. **Подсчет длины списка**: getListLength возвращает количество элементов в списке.
7. **Удаление элементов вокруг заданного значения**: deleteSurroundingNodes удаляет все элементы перед и после элемента со значением value, включая сам элемент.
8. **Итератор**: Структура Iterator и функции для инициализации и работы с ним (getIterator, hasNext, next) позволяют обходить список.
9. **Меню**: showMenu отображает доступные действия. Основной цикл в main позволяет пользователю выбирать действия и многократно их выполнять.
10. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

**main.c**

**#include <stdio.h>**

#include <stdlib.h>

#include <string.h>

// Структура узла списка

typedef struct Node {

    char data[100];

    struct Node\* next;

} Node;

// Инициализация списка с барьерным элементом

Node\* initializeList() {

    Node\* barrier = (Node\*)malloc(sizeof(Node));

    strcpy(barrier->data, "BARRIER");

    barrier->next = NULL;

    return barrier;

}

// Печать списка

void printList(Node\* head) {

    Node\* current = head->next;

    while (current != NULL) {

        printf("%s -> ", current->data);

        current = current->next;

    }

    printf("NULL\n");

}

// Вставка нового элемента в конец списка

void insertNode(Node\* head, const char\* value) {

    Node\* current = head;

    while (current->next != NULL) {

        current = current->next;

    }

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    strcpy(newNode->data, value);

    newNode->next = NULL;

    current->next = newNode;

}

// Удаление элемента из списка

void deleteNode(Node\* head, const char\* value) {

    Node\* current = head;

    while (current->next != NULL && strcmp(current->next->data, value) != 0) {

        current = current->next;

    }

    if (current->next != NULL) {

        Node\* temp = current->next;

        current->next = temp->next;

        free(temp);

    }

}

// Подсчет длины списка

int getListLength(Node\* head) {

    int length = 0;

    Node\* current = head->next; // Пропускаем барьерный элемент

    while (current != NULL) {

        length++;

        current = current->next;

    }

    return length;

}

// Нестандартная операция: удалить из списка все элементы, предшествующие и последующие заданному значению

void deleteSurroundingNodes(Node\* head, const char\* value) {

    Node\* current = head->next; // Пропускаем барьерный элемент

    Node\* prev = head;

    Node\* temp;

    while (current != NULL && strcmp(current->data, value) != 0) {

        temp = current;

        current = current->next;

        free(temp); // Удаляем элементы до значения value

    }

    if (current != NULL) {

        // Нашли элемент со значением value

        prev->next = current;

        prev = current;

        while (prev->next != NULL) {

            temp = prev->next;

            prev->next = temp->next;

            free(temp);

        }

    } else {

        printf("Значение не найдено в списке.\n");

    }

}

// Итератор для списка

typedef struct Iterator {

    Node\* current;

} Iterator;

// Инициализация итератора

Iterator getIterator(Node\* head) {

    Iterator iter;

    iter.current = head->next; // Пропускаем барьерный элемент

    return iter;

}

// Проверка, есть ли следующий элемент

int hasNext(Iterator\* iter) {

    return iter->current != NULL;

}

// Переход к следующему элементу

void next(Iterator\* iter) {

    if (iter->current != NULL) {

        iter->current = iter->current->next;

    }

}

void showMenu() {

    printf("Выберите действие:\n");

    printf("1. Печать списка\n");

    printf("2. Вставка элемента\n");

    printf("3. Удаление элемента\n");

    printf("4. Подсчет длины списка\n");

    printf("5. Удалить элементы вокруг заданного значения\n");

    printf("6. Выход\n");

}

int main() {

    Node\* list = initializeList();

    int choice;

    char value[100];

    while (1) {

        showMenu();

        printf("Введите ваш выбор: ");

        scanf("%d", &choice);

        getchar(); // для удаления символа новой строки после ввода

        switch (choice) {

            case 1:

                printf("Список: ");

                printList(list);

                break;

            case 2:

                printf("Введите значение для вставки: ");

                fgets(value, sizeof(value), stdin);

                value[strcspn(value, "\n")] = '\0'; // удалить символ новой строки

                insertNode(list, value);

                break;

            case 3:

                printf("Введите значение для удаления: ");

                fgets(value, sizeof(value), stdin);

                value[strcspn(value, "\n")] = '\0'; // удалить символ новой строки

                deleteNode(list, value);

                break;

            case 4:

                printf("Длина списка: %d\n", getListLength(list));

                break;

            case 5:

                printf("Введите значение, вокруг которого удалить узлы: ");

                fgets(value, sizeof(value), stdin);

                value[strcspn(value, "\n")] = '\0'; // удалить символ новой строки

                deleteSurroundingNodes(list, value);

                break;

            case 6:

                printf("Выход...\n");

                // Освобождаем память перед выходом

                while (list != NULL) {

                    Node\* temp = list;

                    list = list->next;

                    free(temp);

                }

                return 0;

            default:

                printf("Неверный выбор. Пожалуйста, попробуйте снова.\n");

        }

    }

    return 0;

}

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*

*Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Распечатка протокола**  (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

unix@DESKTOP-MPQDBS2:~/labs/кп 7$ gcc main.c

unix@DESKTOP-MPQDBS2:~/labs/кп 7$ ./a.out

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 1

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 2

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 3

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 4

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 5

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 6

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 7

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 8

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 1

Список: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 5

Введите значение, вокруг которого удалить узлы: 5

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 1

Список: 5 -> NULL

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 1

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 1

Список: 5 -> 1 -> NULL

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 4

Длина списка: 2

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 2

Введите значение для вставки: 3

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 3

Введите значение для удаления: 1

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 1

Список: 5 -> 3 -> NULL

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 4

Длина списка: 2

Выберите действие:

1. Печать списка

2. Вставка элемента

3. Удаление элемента

4. Подсчет длины списка

5. Удалить элементы вокруг заданного значения

6. Выход

Введите ваш выбор: 6

Выход...

1. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора**
2. **Выводы**

За время выполнения ЛР я научился работать с абстрактными типами данных, реализовал стек и метод её сортировки.

Недочёты при выполнении задания могут быть устранены следующим образом: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_