Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

Отчёт

по лабораторной работе №3

по курсу «Л и О А в ИЗ»

на тему «Динамические списки»

Выполнили студенты группы 24ВВВ4:

Кондратьев С.В.

Кошелев Р.Д.

Приняли к.т.н., доцент:

Юрова О.В.

к.э.н., доцент:

Акифьев И.В.

Пенза 2025

Цель: освоить практические навыки работы с динамическими структурами данных на языке С.

Общие сведения.

Список представляет собой последовательность элементов определенного типа. Простейший тип списка – линейный, когда для каждого из элементов, кроме последнего, имеется следующий, и для каждого, кроме первого имеется предыдущий элемент.

Возможна реализация списков посредством массивов или динамическая реализация.

Динамические списки относятся к динамическим структурам и используются, когда размер данных заранее неизвестен. Созданием динамических данных должна заниматься сама программа во время своего исполнения, этим достигается эффективное распределение памяти, но снижается эффективность доступа к элементам.

Динамические структуры данных отличаются от статических двумя основными свойствами:

1) в них нельзя обеспечить хранение в заголовке всей информации о структуре, поэтому каждый элемент должен содержать информацию, логически связывающую его с другими элементами структуры;
2) для них зачастую не удобно использовать единый массив смежных

элементов памяти, поэтому необходимо предусматривать ту или иную схему динамического управления памятью.

Для обращения к динамическим данным применяют указатели.

Набор операций над списком будет включать добавление и удаление элементов, поиск элементов списка.

Различают односвязные, двусвязные и циклические списки.

В простейшем случае каждый элемент содержит всего одну ссылку на следующий элемент, такой список называется односвязным.

В простейшем случае для создания элемента списка используется структура, в которой объединяются полезная информация и ссылка на следующий элемент списка

Задание

1) Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект с большим приоритетом становится перед объектом с меньшим приоритетом).

2) *На основе приведенного кода реализуйте структуру данных *Очередь.*

3) *На основе приведенного кода реализуйте структуру данных *Стек.*

Листинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include <string.h>
struct node
{
   char inf[256];
   int priority;
   struct node* next;
};
struct node* head = NULL, * last = NULL;
int dlinna = 0;
char find_el[256];
struct node* get_struct(void);
void spstore(void);
void review(void);
```

```
void del(char* name);
struct node* find(char* name);
struct node* get_struct(void)
{
   struct node* p = NULL;
   char s[256];
   int priority;
   if ((p = (struct node*)malloc(sizeof(struct node))) == NULL)
   {
      printf("Ошибка при распределении памяти\n");
      exit(1);
   }
int c;
while ((c = getchar()) != '\n' && c != EOF);
printf("Введите название объекта: ");
fgets(s, sizeof(s), stdin);
   printf("Введите название объекта: ");
   fgets(s, sizeof(s), stdin);
   s[strcspn(s, "\n")] = 0;
   printf("Введите приоритет объекта (целое число): ");
   scanf("%d", &priority);
   if (strlen(s) == 0)
   {
      printf("Запись не была произведена\n");
      free(p);
      return NULL;
   }
   strcpy(p->inf, s);
   p->priority = priority;
   p->next = NULL;
```

```
    return p;
}
void spstore(void)
{
    struct node* p = NULL;
    struct node* current = NULL;
    struct node* prev = NULL;

    p = get_struct();
    if (p == NULL)
    {
        return;
    }
    if (head == NULL)
    {
        head = p;
        last = p;
        return;
    }
    current = head;
    prev = NULL;
    while (current != NULL && current->priority >= p->priority)
    {
        prev = current;
        current = current->next;
    }
    if (prev == NULL)
    {
        p->next = head;
        head = p;
```

```
        }
        else
        {
            prev->next = p;
            p->next = current;

            if (current == NULL)
            {
                last = p;
            }
        }
    }
    void review(void)
    {
        struct node* struc = head;
        if (head == NULL)
        {
            printf("Список пуст\n");
            return;
        }
        printf("Содержимое списка (элементы упорядочены по убыванию приоритета):\n");
        while (struc)
        {
            printf("Имя - %s, Приоритет - %d\n", struc->inf, struc->priority);
            struc = struc->next;
        }
    }
    struct node* find(char* name)
    {
```

```
    struct node* struc = head;
    if (head == NULL)
    {
        printf("Список пуст\n");
        return NULL;
    }

    while (struc)
    {
        if (strcmp(name, struc->inf) == 0)
        {
            return struc;
        }
        struc = struc->next;
    }
    printf("Элемент не найден\n");
    return NULL;
}
void del(char* name)
{
    struct node* struc = head;
    struct node* prev = NULL;
    int flag = 0;
    if (head == NULL)
    {
        printf("Список пуст\n");
        return;
    }
    if (strcmp(name, struc->inf) == 0)
    {
```

```
    flag = 1;
    head = struc->next;
    free(struc);
    struc = head;
}
else
{
    prev = struc;
    struc = struc->next;
}
while (struc)
{
    if (strcmp(name, struc->inf) == 0)
    {
        flag = 1;
        if (struc->next)
        {
            prev->next = struc->next;
            free(struc);
            struc = prev->next;
        }
        else
        {
            prev->next = NULL;
            last = prev;
            free(struc);
            return;
        }
    }
    else
```

```
        {
            prev = struc;
            struc = struc->next;
        }
    }
    if (flag == 0)
    {
        printf("Элемент не найден\n");
    }
}
struct node* peek(void)
{
    return head;
}
struct node* dequeue(void)
{
    struct node* temp = head;
    if (head == NULL)
    {
        printf("Очередь пуста\n");
        return NULL;
    }
    head = head->next;
    if (head == NULL)
    {
        last = NULL;
    }
    temp->next = NULL;
    return temp;
}
```

```
int main() {
        setlocale(LC_ALL, "rus");
    int choice;
    char name[256];
    struct node* temp;
    while (1)
    {
        printf("\n1. Добавить элемент в очередь\n");
        printf("2. Просмотреть очередь\n");
        printf("3. Найти элемент\n");
        printf("4. Удалить элемент\n");
        printf("5. Показать элемент с наивысшим приоритетом\n");
        printf("6. Извлечь элемент с наивысшим приоритетом\n");
        printf("7. Выход\n");
        printf("Выберите действие: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            spstore();
            break;
        case 2:
            review();
            break;
        case 3:
            printf("Введите имя для поиска: ");
            scanf("%s", name);
            temp = find(name);
            if (temp != NULL)
            {
```

```
                printf("Найден: %s с приоритетом %d\n", temp->inf, temp->priority);
            }
            break;
        case 4:
            printf("Введите имя для удаления: ");
            scanf("%s", name);
            del(name);
            break;
        case 5:
            temp = peek();
            if (temp != NULL)
            {
                printf("Элемент с наивысшим приоритетом: %s (приоритет: %d)\n",
                    temp->inf, temp->priority);
            }
            break;
        case 6:
            temp = dequeue();
            if (temp != NULL)
            {
                printf("Извлечен элемент: %s (приоритет: %d)\n",
                    temp->inf, temp->priority);
                free(temp);
            }
            break;
        case 7:
            // Очистка памяти перед выходом
            while (head != NULL)
```

```
            {
                temp = head;
                head = head->next;
                free(temp);
            }
            exit(0);
        default:
            printf("Неверный выбор\n");
        }
    }
    return 0;
}
```

Результат работы программы

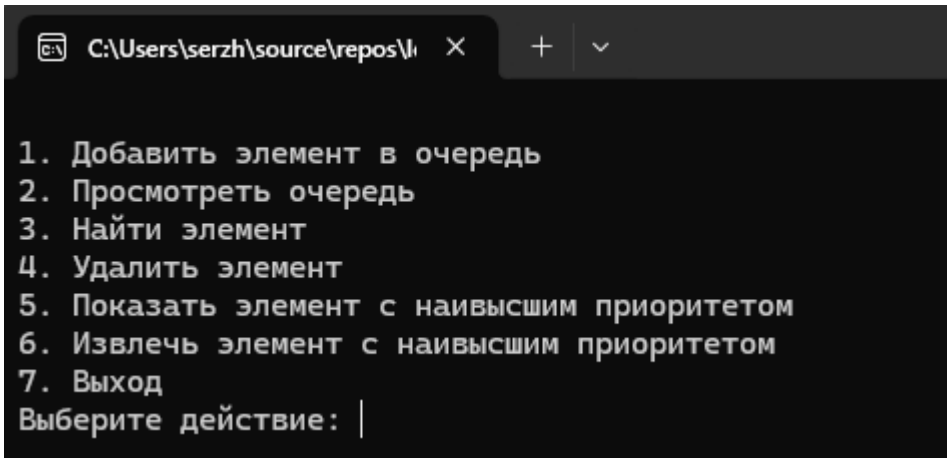

Рисунок 1 – Начальное меню

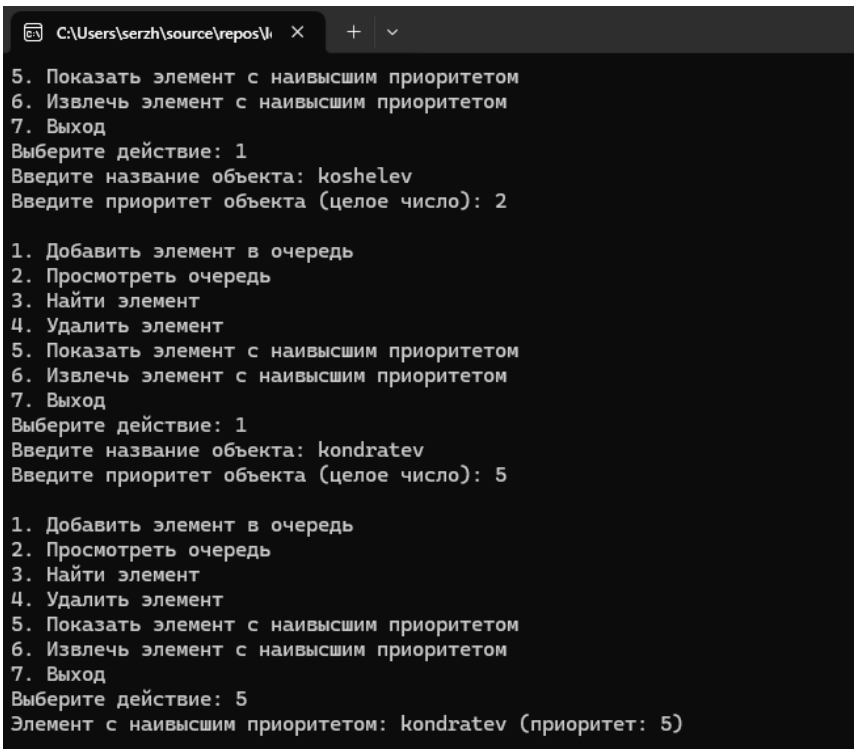

Рисунок 2 – Присваивание приоритета и показ элемента с наивысшим приоритетом

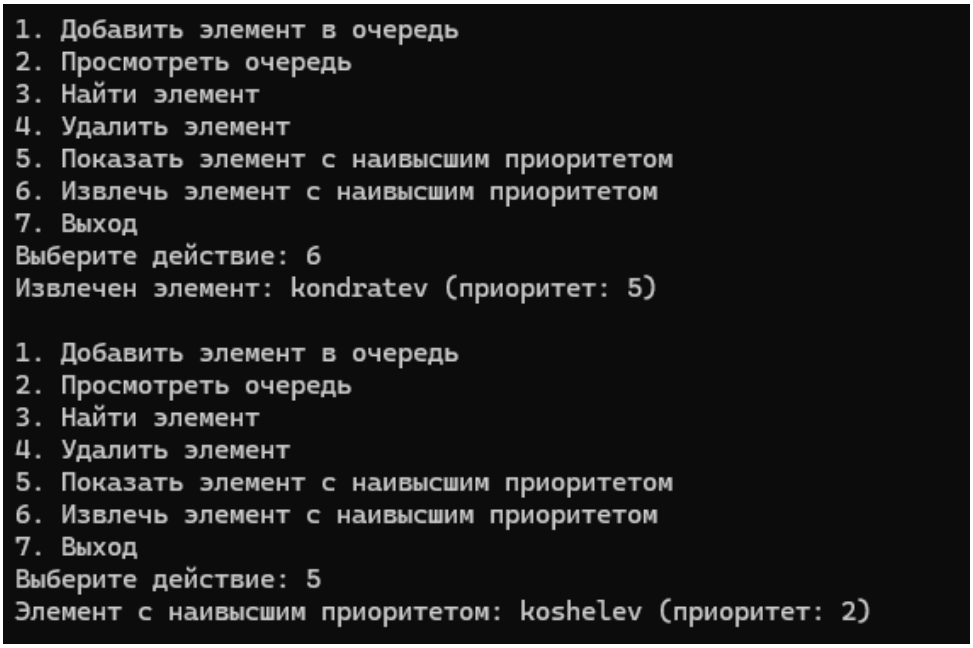

Рисунок 3 – Извлечение элемента с наивысшим приоритетом и показ элемента с наивысшим приоритетом

Вывод: в ходе выполнения данной лабораторной работы освоили практические навыки работы с динамическими структурами данных на языке С. Была реализована приоритетная очередь на основе односвязного списка.