Министерство образования Российской Федерации Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки»

**Выполнили:**

Студенты группы 24ВВВ1

Выдрин Д.А

Картушин Р.А

**Принял:**  к.т.н., Деев М.В

Пенза 2025

# Название

Динамические списки

**Задание**

1. Реализовать приоритетную очередь, путём добавления элемента в список всоответствии 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.

**Ход работы**

Написали программу, которая реализует приоритетную очередь, путём добавления элемента в список в соответсвии с приоритетом объекта, также реализовали структуры данных очередь и стек.

**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node {

char inf[256];

int priority;

struct node\* next;

};

struct node\* priority\_head = NULL;

struct node\* queue\_head = NULL;

struct node\* queue\_tail = NULL;

struct node\* stack\_top = NULL;

// Функция создания элемента с приоритетом

struct node\* get\_struct(void) {

struct node\* p = NULL;

int pr;

p = (struct node\*)malloc(sizeof(struct node));

printf("Введите название объекта: \n");

scanf("%255s", p->inf);

printf("Введите приоритет (целое число): \n");

scanf("%d", &pr);

p->priority = pr;

p->next = NULL;

return p;

}

// Функция создания элемента без приоритета

struct node\* get\_struct\_simple(void) {

struct node\* p = NULL;

p = (struct node\*)malloc(sizeof(struct node));

printf("Введите название объекта: \n");

scanf("%255s", p->inf);

p->priority = 0;

p->next = NULL;

return p;

}

// ПРИОРИТЕТНАЯ ОЧЕРЕДЬ

// Добавление элемента в приоритетную очередь

void priority\_enqueue(void) {

struct node\* p = get\_struct();

// Если очередь пуста или новый элемент имеет высший приоритет чем голова

if (priority\_head == NULL || p->priority > priority\_head->priority) {

p->next = priority\_head;

priority\_head = p;

}

else {

struct node\* current = priority\_head;

while (current->next != NULL && current->next->priority >= p->priority) {

current = current->next;

}

p->next = current->next;

current->next = p;

}

printf("Элемент добавлен в приоритетную очередь\n");

}

// Удаление элемента с наивысшим приоритетом

void priority\_dequeue(void) {

if (priority\_head == NULL) {

printf("Приоритетная очередь пуста!\n");

return;

}

struct node\* temp = priority\_head;

printf("Удален элемент: %s (приоритет: %d)\n", temp->inf, temp->priority);

priority\_head = priority\_head->next;

free(temp);

}

// Удаление элементов с выбранным приоритетом

void remove\_by\_priority(void) {

int tpriority;

printf("Введите приоритет для удаления: ");

scanf("%d", &tpriority);

struct node\* current = priority\_head;

struct node\* prev = NULL;

int count = 0;

while (current != NULL) {

if (current->priority == tpriority) {

struct node\* temp = current;

if (prev == NULL) {

priority\_head = current->next;

}

else {

prev->next = current->next;

}

current = current->next;

printf("Удален элемент: %s (приоритет: %d)\n", temp->inf, temp->priority);

free(temp);

count++;

}

else {

prev = current;

current = current->next;

}

}

if (count == 0) {

printf("Элементы с приоритетом %d не найдены\n", tpriority);

}

else {

printf("Удалено %d элементов с приоритетом %d\n", count, tpriority);

}

}

// Просмотр приоритетной очереди

void review\_priority(void) {

if (priority\_head == NULL) {

printf("Приоритетная очередь пуста!\n");

return;

}

struct node\* struc = priority\_head;

printf("Содержимое приоритетной очереди:\n");

while (struc) {

printf("Имя - %s, Приоритет - %d\n", struc->inf, struc->priority);

struc = struc->next;

}

}

// ОЧЕРЕДЬ

// Добавление элемента в очередь

void enqueue(void) {

struct node\* p = get\_struct\_simple();

if (queue\_head == NULL) {

queue\_head = p;

queue\_tail = p;

}

else {

queue\_tail->next = p;

queue\_tail = p;

}

printf("Элемент добавлен в очередь\n");

}

// Удаление элемента из очереди

void dequeue(void) {

if (queue\_head == NULL) {

printf("Очередь пуста!\n");

return;

}

struct node\* temp = queue\_head;

printf("Удален элемент: %s\n", temp->inf);

queue\_head = queue\_head->next;

if (queue\_head == NULL) {

queue\_tail = NULL;

}

free(temp);

}

// Просмотр очереди

void review\_queue(void) {

if (queue\_head == NULL) {

printf("Очередь пуста!\n");

return;

}

struct node\* struc = queue\_head;

printf("Содержимое очереди:\n");

while (struc) {

printf("Имя - %s\n", struc->inf);

struc = struc->next;

}

}

// CТЕК

// Добавление элемента в стек

void push(void) {

struct node\* p = get\_struct\_simple();

p->next = stack\_top;

stack\_top = p;

printf("Элемент добавлен в стек\n");

}

// Удаление элемента из стека

void pop(void) {

if (stack\_top == NULL) {

printf("Стек пуст!\n");

return;

}

struct node\* temp = stack\_top;

printf("Удален элемент: %s\n", temp->inf);

stack\_top = stack\_top->next;

free(temp);

}

// Смена первого и последнего элементов в стеке

void swap\_first\_last\_stack(void) {

if (stack\_top == NULL) {

printf("Стек пуст!\n");

return;

}

if (stack\_top->next == NULL) {

printf("В стеке только один элемент!\n");

return;

}

struct node\* last = stack\_top;

struct node\* prev = NULL;

while (last->next != NULL) {

prev = last;

last = last->next;

}

if (stack\_top->next == last) {

last->next = stack\_top;

stack\_top->next = NULL;

stack\_top = last;

}

else {

struct node\* second = stack\_top->next;

last->next = second;

prev->next = stack\_top;

stack\_top->next = NULL;

stack\_top = last;

}

printf("Первый и последний элементы стека поменялись местами\n");

}

// Просмотр стека

void review\_stack(void) {

if (stack\_top == NULL) {

printf("Стек пуст!\n");

return;

}

struct node\* struc = stack\_top;

printf("Содержимое стека:\n");

while (struc) {

printf("Имя - %s\n", struc->inf);

struc = struc->next;

}

}

int main() {

setlocale(LC\_ALL, "Rus");

int choice;

int structure\_choice;

do {

printf("\n ГЛАВНОЕ МЕНЮ \n");

printf("1. Приоритетная очередь\n");

printf("2. Очередь \n");

printf("3. Стек \n");

printf("0. Выход\n");

printf("Выберите структуру данных: ");

scanf("%d", &structure\_choice);

if (structure\_choice == 0) {

break;

}

do {

printf("\n");

switch (structure\_choice) {

case 1:

printf("ПРИОРИТЕТНАЯ ОЧЕРЕДЬ\n");

printf("1. Добавить элемент\n");

printf("2. Удалить элемент \n");

printf("3. Удалить элементы с выбранным приоритетом\n");

printf("4. Просмотреть очередь\n");

printf("0. Назад\n");

break;

case 2:

printf("ОЧЕРЕДЬ\n");

printf("1. Добавить элемент \n");

printf("2. Удалить элемент \n");

printf("3. Просмотреть очередь\n");

printf("0. Назад\n");

break;

case 3:

printf("СТЕК\n");

printf("1. Добавить элемент \n");

printf("2. Удалить элемент \n");

printf("3. Поменять местами первый и последний элементы\n");

printf("4. Просмотреть стек\n");

printf("0. Назад\n");

break;

default:

printf("Неверный выбор!\n");

break;

}

printf("Выберите операцию: ");

scanf("%d", &choice);

switch (structure\_choice) {

case 1:

switch (choice) {

case 1:

priority\_enqueue();

break;

case 2:

priority\_dequeue();

break;

case 3:

remove\_by\_priority();

break;

case 4:

review\_priority();

break;

case 0:

break;

default:

printf("Неверный выбор!\n");

}

break;

case 2:

switch (choice) {

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

review\_queue();

break;

case 0:

break;

default:

printf("Неверный выбор!\n");

}

break;

case 3:

switch (choice) {

case 1:

push();

break;

case 2:

pop();

break;

case 3:

swap\_first\_last\_stack();

break;

case 4:

review\_stack();

break;

case 0:

break;

default:

printf("Неверный выбор!\n");

}

break;

}

} while (choice != 0);

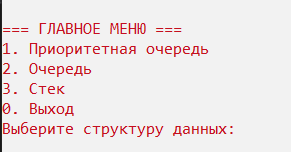
} while (structure\_choice != 0);

return 0;

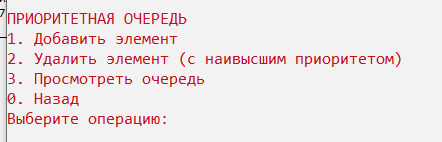
}

**Результат работы программы**

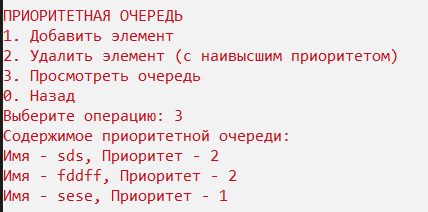
**1. Главное меню**



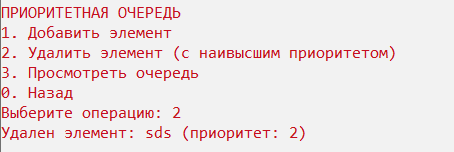
**2. Приорететная очередь**



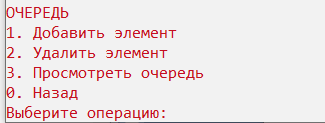
2.1 Просмотр очереди



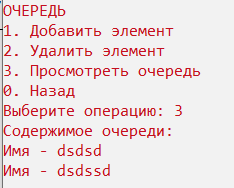
2.2 Удаление из очереди



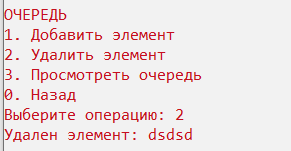
**3. Очередь**



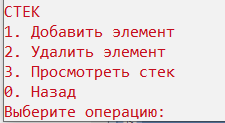
3.1 Просмотр очереди



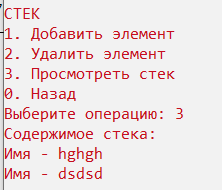
3.2 Удаление из очереди



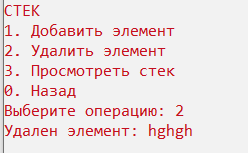
**4. Стек**

****

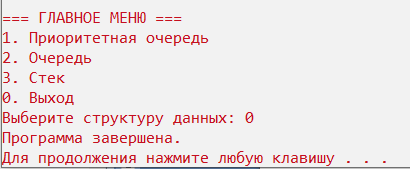
4.1 Просмотр очереди

****

4.2 Удаление из очереди



**5. Выход из программы**



**Вывод**

В ходе лабораторной работы мы научились добавлять приоритетную очередь, а так же создавать структуры данных очереди и стека.