Министерство науки и высшего 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

Куничкина В.А.

Суркова Д.А.

**Приняли:**

к.т.н. Деев М.В.

к.т.н. доцент Юрова О.В.

Пенза 2025

Цель работы: Реализовать приоритетную очередь.

**Лабораторное задание:**

**Задание 1:**

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*.

**Теоретическая часть:**

Для оценки времени выполнения программ языка Си или их частей могут использоваться средства, предоставляемые библиотекой **time.h**. Данная библиотека содержит описания типов и прототипы функций для работы с датой и временем.

Типы данных:

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.

**Практическая часть:**

**Листинг:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <Windows.h>

struct node {

char inf[256]; // полезная информация

unsigned long long int priority; // приоритет элемента

struct node\* next; // ссылка на следующий элемент

};

struct node\* head = NULL, \* last = NULL; // указатели на первый и последний элементы списка

struct node\* get\_struct(void) {

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

if (p == NULL) {

printf("Ошибка при распределении памяти\n");

exit(1);

}

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

scanf\_s("%s", p->inf, (unsigned)\_countof(p->inf));

p->next = NULL;

return p;

}

void spstore(int priority) {

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

p->priority = priority;

if (head == NULL || head->priority < priority) {

p->next = head;

head = p;

if (last == NULL) last = p;

}

else {

struct node\* current = head;

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

current = current->next;

}

p->next = current->next;

current->next = p;

if (p->next == NULL) last = p;

}

}

void review(void) {

struct node\* struc = head;

if (head == NULL) {

printf("Список пуст\n");

return;

}

while (struc) {

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

struc = struc->next;

}

}

void del(char\* name) {

struct node\* struc = head;

struct node\* prev = NULL;

int flag = 0;

char new\_info[100];

if (head == NULL) {

printf("Список пуст\n");

return;

}

printf("Введите новое значение для изменения: ");

scanf\_s("%s", new\_info);

if (strcmp(name, struc->inf) == 0) {

flag = 1;

strcpy\_s(struc->inf, new\_info);

struc = struc->next;

}

else {

prev = struc;

struc = struc->next;

}

while (struc) {

if (strcmp(name, struc->inf) == 0) {

flag = 1;

strcpy\_s(struc->inf, new\_info);

struc = struc->next;

}

else {

prev = struc;

struc = struc->next;

}

}

if (flag == 0) {

printf("Элемент не найден\n");

}

}

//очередь

void enqueue(void) {

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

p->priority = 0;

if (head == NULL) {

head = p;

last = p;

}

else {

last->next = p;

last = p;

}

}

struct node\* dequeue(void) {

if (head == NULL) {

printf("Очередь пуста\n");

return NULL;

}

struct node\* temp = head;

head = head->next;

if (head == NULL) last = NULL;

return temp;

}

//стек

void push(void) {

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

p->priority = 0;

p->next = head;

head = p;

if (last == NULL) last = p;

}

struct node\* pop(void) {

if (head == NULL) {

printf("Стек пуст\n");

return NULL;

}

struct node\* temp = head;

head = head->next;

if (head == NULL) last = NULL;

return temp;

}

int main() {

int negr;

unsigned long long priority;

char name[256];

struct node\* removed;

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

while (1) {

printf("\nВыберите действие:\n");

printf("1. Добавить элемент в приоритетную очередь\n");

printf("2. Просмотреть приоритетную очередь\n");

printf("3. Изменить элемент из приоритетной очереди\n");

printf("4. Добавить элемент в очередь\n");

printf("5. Удалить элемент из очереди\n");

printf("6. Добавить элемент в стек\n");

printf("7. Удалить элемент из стека\n");

printf("0. Выход из программы\n");

scanf\_s("%d", &negr);

switch (negr) {

case 1:

printf("Введите приоритет элемента: ");

scanf\_s("%llu", &priority);

spstore(priority);

break;

case 2:

review();

break;

case 3:

printf("Введите имя элемента для удаления: ");

scanf\_s("%s", name, (unsigned)\_countof(name));

del(name);

break;

case 4:

enqueue();

break;

case 5:

removed = dequeue();

if (removed) {

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

free(removed);

}

break;

case 6:

push();

break;

case 7:

removed = pop();

if (removed) {

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

free(removed);

}

break;

case 0:

\_getch();

return 0;

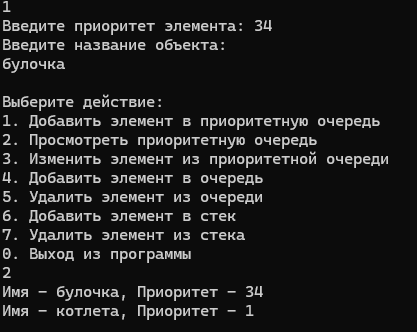
}

}

return 0;

}

**Результат работы программы:**

****

**Вывод**

В ходе выполнения лабораторной работы были изучены и практически применены основные принципы работы с динамическими списками, очередью и стеком в языке Си