Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра “Вычислительная техника”

Отчет

по лабораторной работе №2

по курсу “**Логика и основы алгоритмизации инженерных задач**”

на тему “ Простые структуры данных”

Выполнили студ. группы 24ВВВ4:

Бывшева А.Ю

Конкина С.В

Приняли:

к.т.н. доцент Юрова О.В.

к.э.н. доцент Акифьев И.В.

Пенза 2025

Цель работы: реализация списков посредством массивов или динамическая реализация.

Методические материалы: 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.

**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*.

1 задание

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

// Структура элемента очереди с приоритетом

struct node {

char inf[256];

int priority; // Приоритет: чем выше число, тем выше приоритет

struct node\* next;

};

// Глобальные указатели для очереди

struct node\* head = NULL;

struct node\* last = NULL;

// Прототипы функций

struct node\* get\_struct(void);

void enqueue(void);

void enqueue\_with\_priority(void);

void dequeue(void);

void review(void);

struct node\* find(char\* name);

void del(char\* name);

void queue\_front(void);

int is\_empty(void);

void clear\_queue(void);

void change\_priority(void);

// Функция создания нового элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL)

{

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

exit(1);

}

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

if (scanf("%255s", s) != 1)

{

printf("Ошибка ввода\n");

free(p);

return NULL;

}

if (\*s == 0)

{

printf("Запись не была произведена\n");

free(p);

return NULL;

}

strcpy(p->inf, s);

p->priority = 0; // Приоритет по умолчанию

p->next = NULL;

return p;

}

// Функция создания нового элемента с приоритетом

struct node\* get\_struct\_with\_priority(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);

}

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

if (scanf("%255s", s) != 1)

{

printf("Ошибка ввода\n");

free(p);

return NULL;

}

if (\*s == 0)

{

printf("Запись не была произведена\n");

free(p);

return NULL;

}

printf("Введите приоритет (целое число, чем выше - тем важнее): \n");

if (scanf("%d", &priority) != 1)

{

printf("Ошибка ввода приоритета\n");

free(p);

return NULL;

}

strcpy(p->inf, s);

p->priority = priority;

p->next = NULL;

return p;

}

// Добавление элемента в конец очереди (без приоритета)

void enqueue(void)

{

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

if (p == NULL) {

return;

}

if (head == NULL)

{

head = p;

last = p;

}

else

{

last->next = p;

last = p;

}

printf("Элемент '%s' добавлен в очередь\n", p->inf);

}

// Добавление элемента с приоритетом (в правильную позицию)

void enqueue\_with\_priority(void)

{

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

if (p == NULL) {

return;

}

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

if (head == NULL || p->priority > head->priority)

{

p->next = head;

head = p;

if (last == NULL) {

last = p;

}

printf("Элемент '%s' с приоритетом %d добавлен в начало очереди\n", p->inf, p->priority);

return;

}

// Поиск правильной позиции для вставки

struct node\* current = head;

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

{

current = current->next;

}

p->next = current->next;

current->next = p;

// Если вставляем в конец, обновляем last

if (p->next == NULL) {

last = p;

}

printf("Элемент '%s' с приоритетом %d добавлен в очередь\n", p->inf, p->priority);

}

// Изменение приоритета элемента

void change\_priority(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

char name[256];

int new\_priority;

printf("Введите название элемента для изменения приоритета: ");

scanf("%255s", name);

printf("Введите новый приоритет: ");

if (scanf("%d", &new\_priority) != 1)

{

printf("Ошибка ввода приоритета\n");

return;

}

// Находим элемент

struct node\* current = head;

struct node\* prev = NULL;

struct node\* target = NULL;

while (current != NULL)

{

if (strcmp(current->inf, name) == 0)

{

target = current;

break;

}

prev = current;

current = current->next;

}

if (target == NULL)

{

printf("Элемент '%s' не найден\n", name);

return;

}

// Удаляем элемент из текущей позиции

if (prev == NULL)

{

head = target->next;

if (target == last) {

last = NULL;

}

}

else

{

prev->next = target->next;

if (target == last) {

last = prev;

}

}

// Обновляем приоритет и вставляем обратно с новым приоритетом

target->priority = new\_priority;

target->next = NULL;

// Вставляем элемент в правильную позицию по новому приоритету

if (head == NULL || target->priority > head->priority)

{

target->next = head;

head = target;

if (last == NULL) {

last = target;

}

}

else

{

struct node\* curr = head;

while (curr->next != NULL && curr->next->priority >= target->priority)

{

curr = curr->next;

}

target->next = curr->next;

curr->next = target;

if (target->next == NULL) {

last = target;

}

}

printf("Приоритет элемента '%s' изменен на %d\n", name, new\_priority);

}

void dequeue(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

struct node\* temp = head;

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

head = head->next;

free(temp);

if (head == NULL) {

last = NULL;

}

}

void queue\_front(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

}

else

{

printf("Первый элемент очереди: %s (приоритет: %d)\n", head->inf, head->priority);

}

}

int is\_empty(void)

{

return head == NULL;

}

// Просмотр содержимого очереди

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

printf("Содержимое очереди (от начала к концу):\n");

int position = 1;

while (struc)

{

printf("%d. %s (приоритет: %d)\n", position++, struc->inf, struc->priority);

struc = struc->next;

}

}

// Поиск элемента по содержимому

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста\n");

return NULL;

}

while (struc)

{

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

{

printf("Элемент '%s' найден (приоритет: %d)\n", name, struc->priority);

return struc;

}

struc = struc->next;

}

printf("Элемент '%s' не найден\n", name);

return NULL;

}

// Удаление всех элементов с указанным значением

void del(char\* name)

{

struct node\* current = head;

struct node\* prev = NULL;

int count = 0;

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

// Проходим по всей очереди и удаляем все вхождения

while (current != NULL)

{

if (strcmp(name, current->inf) == 0)

{

// Найден элемент для удаления

struct node\* temp = current;

if (prev == NULL)

{

// Удаляем первый элемент

head = current->next;

current = head;

}

else

{

// Удаляем элемент из середины или конца

prev->next = current->next;

current = current->next;

}

// Обновляем last если нужно

if (temp == last)

{

last = prev;

}

free(temp);

count++;

}

else

{

// Переходим к следующему элементу

prev = current;

current = current->next;

}

}

if (count > 0)

{

printf("Удалено %d элементов с именем '%s'\n", count, name);

}

else

{

printf("Элементы с именем '%s' не найдены\n", name);

}

}

// Функция очистки всей очереди

void clear\_queue(void)

{

struct node\* current = head;

struct node\* next;

while (current != NULL)

{

next = current->next;

free(current);

current = next;

}

head = NULL;

last = NULL;

printf("Очередь очищена\n");

}

// Меню для тестирования очереди

void menu(void)

{

int choice;

char name[256];

while (1)

{

printf("\n=== Меню очереди с приоритетами ===\n");

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

printf("2. Добавить элемент с приоритетом\n");

printf("3. Удалить элемент по значению\n");

printf("4. Поиск элемента\n");

printf("5. Изменить приоритет элемента\n");

printf("6. Просмотреть всю очередь\n");

printf("0. Выход\n");

printf("Выберите действие: ");

if (scanf("%d", &choice) != 1) {

printf("Ошибка ввода\n");

while (getchar() != '\n'); // очистка буфера

continue;

}

switch (choice)

{

case 1:

enqueue();

break;

case 2:

enqueue\_with\_priority();

break;

case 3:

printf("Введите элемент для удаления: ");

scanf("%255s", name);

del(name);

break;

case 4:

printf("Введите элемент для поиска: ");

scanf("%255s", name);

find(name);

break;

case 5:

change\_priority();

break;

case 6:

review();

break;

case 0:

clear\_queue();

printf("Выход\n");

return;

default:

printf("Неверный выбор\n");

}

}

}

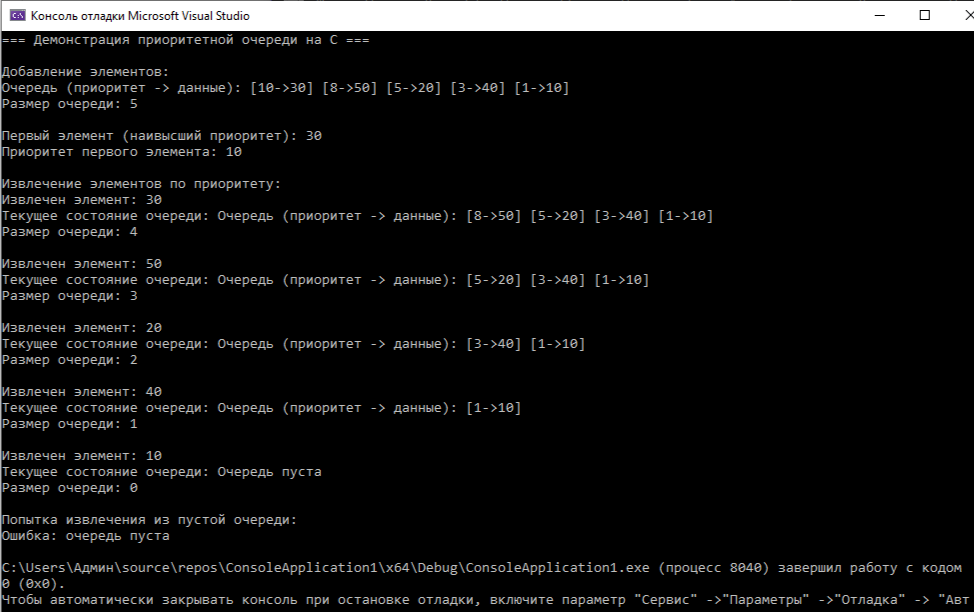
int main(void)

{

setlocale(LC\_ALL, "Russian");

menu();

return 0;

}

2 задание

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

// Структура элемента очереди

struct node {

char inf[256];

struct node\* next;

};

// Глобальные указатели для очереди

struct node\* head = NULL; // начало очереди (первый элемент)

struct node\* last = NULL; // конец очереди (последний элемент)

// Прототипы функций

struct node\* get\_struct(void);

void enqueue(void); // добавление в очередь (аналог spstore)

void dequeue(void); // удаление из начала очереди

void review(void);

struct node\* find(char\* name);

void del(char\* name);

void queue\_front(void); // просмотр первого элемента без удаления

int is\_empty(void); // проверка пустоты очереди

// Функция создания нового элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL)

{

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

exit(1);

}

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

if (scanf("%255s", s) != 1) // безопасный ввод с ограничением длины

{

printf("Ошибка ввода\n");

free(p);

return NULL;

}

if (\*s == 0)

{

printf("Запись не была произведена\n");

free(p);

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p;

}

// Добавление элемента в конец очереди (enqueue)

void enqueue(void)

{

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

if (p == NULL) {

return;

}

if (head == NULL) // если очередь пуста

{

head = p;

last = p;

}

else // добавление в конец

{

last->next = p;

last = p;

}

printf("Элемент '%s' добавлен в очередь\n", p->inf);

}

// Удаление элемента из начала очереди (dequeue)

void dequeue(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

struct node\* temp = head;

printf("Удаляем элемент: %s\n", head->inf);

head = head->next;

free(temp);

// Если очередь стала пустой, обнуляем last

if (head == NULL) {

last = NULL;

}

}

// Просмотр первого элемента без удаления

void queue\_front(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

}

else

{

printf("Первый элемент очереди: %s\n", head->inf);

}

}

// Проверка пустоты очереди

int is\_empty(void)

{

return head == NULL;

}

// Просмотр содержимого очереди

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

printf("Содержимое очереди (от начала к концу):\n");

int position = 1;

while (struc)

{

printf("%d. %s\n", position++, struc->inf);

struc = struc->next;

}

}

// Поиск элемента по содержимому

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста\n");

return NULL;

}

while (struc)

{

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

{

return struc;

}

struc = struc->next;

}

printf("Элемент '%s' не найден\n", name);

return NULL;

}

// Удаление элемента по содержимому (нестандартная операция для очереди)

void del(char\* name)

{

struct node\* struc = head;

struct node\* prev = NULL;

int flag = 0;

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

// Поиск элемента для удаления

while (struc)

{

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

{

flag = 1;

if (prev == NULL) // удаление первого элемента

{

head = struc->next;

// Если удаляем последний элемент

if (struc == last) {

last = NULL;

}

free(struc);

struc = head;

}

else // удаление из середины или конца

{

prev->next = struc->next;

// Если удаляем последний элемент

if (struc ==

last) {

last = prev;

}

free(struc);

struc = prev->next;

}

printf("Элемент '%s' удален\n", name);

break; // удаляем только первое вхождение

}

else

{

prev = struc;

struc = struc->next;

}

}

if (!flag)

{

printf("Элемент '%s' не найден\n", name);

}

}

// Функция очистки всей очереди

void clear\_queue(void)

{

struct node\* current = head;

struct node\* next;

while (current != NULL)

{

next = current->next;

free(current);

current = next;

}

head = NULL;

last = NULL;

printf("Очередь очищена\n");

}

// Меню для тестирования очереди

void menu(void)

{

int choice;

char name[256];

while (1)

{

printf("\n=== Меню очереди ===\n");

printf("1. Добавить элемент (enqueue)\n");

printf("2. Удалить элемент из начала (dequeue)\n");

printf("3. Просмотреть первый элемент\n");

printf("4. Просмотреть всю очередь\n");

printf("5. Поиск элемента\n");

printf("6. Удалить элемент по значению\n");

printf("7. Проверить пустоту очереди\n");

printf("8. Очистить очередь\n");

printf("0. Выход\n");

printf("Выберите действие: ");

if (scanf("%d", &choice) != 1) {

printf("Ошибка ввода\n");

while (getchar() != '\n'); // очистка буфера

continue;

}

switch (choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

queue\_front();

break;

case 4:

review();

break;

case 5:

printf("Введите элемент для поиска: ");

scanf("%255s", name);

find(name);

break;

case 6:

printf("Введите элемент для удаления: ");

scanf("%255s", name);

del(name);

break;

case 7:

if (is\_empty()) {

printf("Очередь пуста\n");

}

else {

printf("Очередь не пуста\n");

}

break;

case 8:

clear\_queue();

break;

case 0:

clear\_queue();

printf("Выход\n");

return;

default:

printf("Неверный выбор\n");

}

}

}

int main(void)

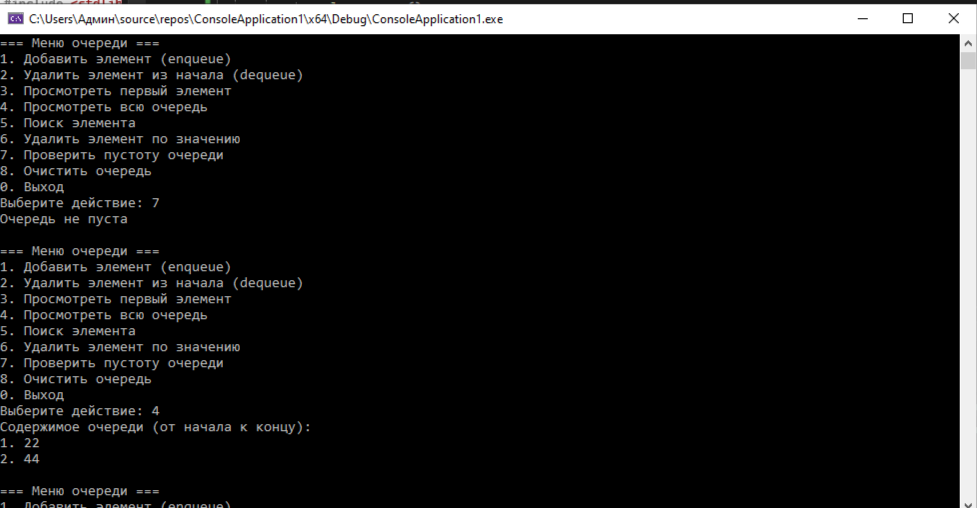
{

setlocale(LC\_ALL, "Russian");

menu();

return 0;

}



3 задание

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

// Структура элемента стека

struct node {

char inf[256];

struct node\* next;

};

// Глобальные указатели для стека

struct node\* top = NULL; // Вершина стека (вместо head)

// Прототипы функций

void push(void); // Добавление элемента в стек (вместо spstore)

void pop(void); // Удаление элемента из стека

void peek(void); // Просмотр верхнего элемента

void display(void); // Просмотр всего стека (вместо review)

int isEmpty(void); // Проверка на пустоту

struct node\* get\_node(void); // Создание нового элемента

// Создание нового элемента стека

struct node\* get\_node(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL)

{

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

exit(1);

}

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

// Безопасный ввод с проверкой

if (scanf("%255s", s) != 1) {

printf("Ошибка ввода\n");

free(p);

return NULL;

}

if (strlen(s) == 0)

{

printf("Запись не была произведена\n");

free(p);

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p;

}

// Добавление элемента в стек (Push)

void push(void)

{

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

if (p == NULL) {

return; // Не удалось создать элемент

}

if (top == NULL) {

// Стек пуст

top = p;

}

else {

// Добавляем на вершину стека

p->next = top;

top = p;

}

printf("Элемент '%s' добавлен в стек\n", p->inf);

}

// Удаление элемента из стека (Pop)

void pop(void)

{

if (isEmpty()) {

printf("Стек пуст - невозможно удалить элемент\n");

return;

}

struct node\* temp = top;

top = top->next;

printf("Элемент '%s' удален из стека\n", temp->inf);

free(temp);

}

// Просмотр верхнего элемента (Peek)

void peek(void)

{

if (isEmpty()) {

printf("Стек пуст\n");

return;

}

printf("Верхний элемент: %s\n", top->inf);

}

// Просмотр всего стека

void display(void)

{

if (isEmpty()) {

printf("Стек пуст\n");

return;

}

struct node\* current = top;

printf("Содержимое стека (сверху вниз):\n");

while (current != NULL) {

printf("| %s |\n", current->inf);

current = current->next;

}

printf("+----+\n");

}

// Проверка, пуст ли стек

int isEmpty(void)

{

return top == NULL;

}

// Очистка всего стека

void clear\_stack(void)

{

while (!isEmpty()) {

pop();

}

}

// Поиск элемента в стеке

struct node\* find(char\* name)

{

if (isEmpty()) {

printf("Стек пуст\n");

return NULL;

}

struct node\* current = top;

while (current != NULL) {

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

return current;

}

current = current->next;

}

printf("Элемент '%s' не найден в стеке\n", name);

return NULL;

}

// Основное меню программы

void menu(void)

{

int choice;

char search\_name[256];

do {

printf("\n=== Реализация Стека ===\n");

printf("1. Добавить элемент (Push)\n");

printf("2. Удалить элемент (Pop)\n");

printf("3. Просмотреть верхний элемент (Peek)\n");

printf("4. Показать весь стек\n");

printf("5. Поиск элемента\n");

printf("6. Очистить стек\n");

printf("7. Проверить пустоту стека\n");

printf("0. Выход\n");

printf("Выберите действие: ");

if (scanf("%d", &choice) != 1) {

printf("Ошибка ввода! Пожалуйста, введите число.\n");

// Очистка буфера ввода

while (getchar() != '\n');

continue;

}

switch (choice) {

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

display();

break;

case 5:

printf("Введите имя для поиска: ");

scanf("%255s", search\_name);

find(search\_name);

break;

case 6:

clear\_stack();

printf("Стек очищен\n");

break;

case 7:

if (isEmpty()) {

printf("Стек пуст\n");

}

else {

printf("Стек не пуст\n");

}

break;

case 0:

printf("Выход из программы\n");

break;

default:

printf("Неверный выбор! Попробуйте снова.\n");

}

} while (choice != 0);

}

// Главная функция

int main(void)

{

setlocale(LC\_ALL, "Russian");

printf("Программа реализует структуру данных Стек\n");

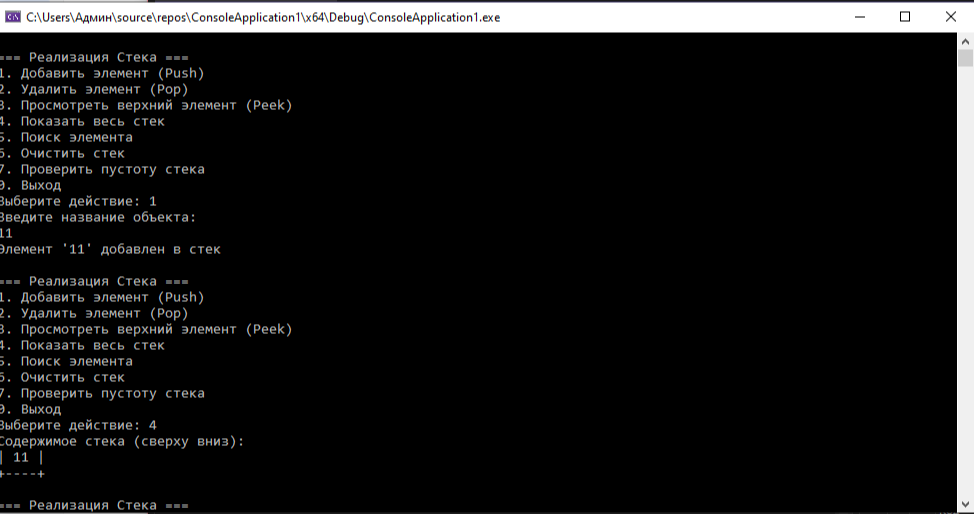
menu();

// Очистка памяти перед выходом

clear\_stack();

return 0;

}



В ходе выполнения лабораторной работы была изучена теория и практика работы со списками как фундаментальной структурой данных. Список представляет собой динамическую структуру данных, организованную в виде последовательности элементов. В отличие от массивов, списки обеспечивают гибкость управления памятью, поскольку их размер может изменяться во время выполнения программы. Это особенно важно в случаях, когда объем данных заранее неизвестен.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

// Структура элемента очереди с приоритетом

struct node {

char inf[256];

int priority; // Приоритет: чем выше число, тем выше приоритет

struct node\* next;

};

// Глобальные указатели для очереди

struct node\* head = NULL;

struct node\* last = NULL;

// Прототипы функций

struct node\* get\_struct(void);

void enqueue(void);

void enqueue\_with\_priority(void);

void dequeue(void);

void review(void);

struct node\* find(char\* name);

void del(char\* name);

void queue\_front(void);

int is\_empty(void);

void clear\_queue(void);

void change\_priority(void);

// Функция создания нового элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL)

{

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

exit(1);

}

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

if (scanf("%255s", s) != 1)

{

printf("Ошибка ввода\n");

free(p);

return NULL;

}

if (\*s == 0)

{

printf("Запись не была произведена\n");

free(p);

return NULL;

}

strcpy(p->inf, s);

p->priority = 0; // Приоритет по умолчанию

p->next = NULL;

return p;

}

// Функция создания нового элемента с приоритетом

struct node\* get\_struct\_with\_priority(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);

}

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

if (scanf("%255s", s) != 1)

{

printf("Ошибка ввода\n");

free(p);

return NULL;

}

if (\*s == 0)

{

printf("Запись не была произведена\n");

free(p);

return NULL;

}

printf("Введите приоритет (целое число, чем выше - тем важнее): \n");

if (scanf("%d", &priority) != 1)

{

printf("Ошибка ввода приоритета\n");

free(p);

return NULL;

}

strcpy(p->inf, s);

p->priority = priority;

p->next = NULL;

return p;

}

// Добавление элемента в конец очереди (без приоритета)

void enqueue(void)

{

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

if (p == NULL) {

return;

}

if (head == NULL)

{

head = p;

last = p;

}

else

{

last->next = p;

last = p;

}

printf("Элемент '%s' добавлен в очередь\n", p->inf);

}

// Добавление элемента с приоритетом (в правильную позицию)

void enqueue\_with\_priority(void)

{

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

if (p == NULL) {

return;

}

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

if (head == NULL || p->priority > head->priority)

{

p->next = head;

head = p;

if (last == NULL) {

last = p;

}

printf("Элемент '%s' с приоритетом %d добавлен в начало очереди\n", p->inf, p->priority);

return;

}

// Поиск правильной позиции для вставки

struct node\* current = head;

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

{

current = current->next;

}

p->next = current->next;

current->next = p;

// Если вставляем в конец, обновляем last

if (p->next == NULL) {

last = p;

}

printf("Элемент '%s' с приоритетом %d добавлен в очередь\n", p->inf, p->priority);

}

// Изменение приоритета элемента

void change\_priority(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

char name[256];

int new\_priority;

printf("Введите название элемента для изменения приоритета: ");

scanf("%255s", name);

printf("Введите новый приоритет: ");

if (scanf("%d", &new\_priority) != 1)

{

printf("Ошибка ввода приоритета\n");

return;

}

// Находим элемент

struct node\* current = head;

struct node\* prev = NULL;

struct node\* target = NULL;

while (current != NULL)

{

if (strcmp(current->inf, name) == 0)

{

target = current;

break;

}

prev = current;

current = current->next;

}

if (target == NULL)

{

printf("Элемент '%s' не найден\n", name);

return;

}

// Удаляем элемент из текущей позиции

if (prev == NULL)

{

head = target->next;

if (target == last) {

last = NULL;

}

}

else

{

prev->next = target->next;

if (target == last) {

last = prev;

}

}

// Обновляем приоритет и вставляем обратно с новым приоритетом

target->priority = new\_priority;

target->next = NULL;

// Вставляем элемент в правильную позицию по новому приоритету

if (head == NULL || target->priority > head->priority)

{

target->next = head;

head = target;

if (last == NULL) {

last = target;

}

}

else

{

struct node\* curr = head;

while (curr->next != NULL && curr->next->priority >= target->priority)

{

curr = curr->next;

}

target->next = curr->next;

curr->next = target;

if (target->next == NULL) {

last = target;

}

}

printf("Приоритет элемента '%s' изменен на %d\n", name, new\_priority);

}

void dequeue(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

struct node\* temp = head;

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

head = head->next;

free(temp);

if (head == NULL) {

last = NULL;

}

}

void queue\_front(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

}

else

{

printf("Первый элемент очереди: %s (приоритет: %d)\n", head->inf, head->priority);

}

}

int is\_empty(void)

{

return head == NULL;

}

// Просмотр содержимого очереди

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

printf("Содержимое очереди (от начала к концу):\n");

int position = 1;

while (struc)

{

printf("%d. %s (приоритет: %d)\n", position++, struc->inf, struc->priority);

struc = struc->next;

}

}

// Поиск элемента по содержимому

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста\n");

return NULL;

}

while (struc)

{

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

{

printf("Элемент '%s' найден (приоритет: %d)\n", name, struc->priority);

return struc;

}

struc = struc->next;

}

printf("Элемент '%s' не найден\n", name);

return NULL;

}

// Удаление всех элементов с указанным значением

void del(char\* name)

{

struct node\* current = head;

struct node\* prev = NULL;

int count = 0;

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

// Проходим по всей очереди и удаляем все вхождения

while (current != NULL)

{

if (strcmp(name, current->inf) == 0)

{

// Найден элемент для удаления

struct node\* temp = current;

if (prev == NULL)

{

// Удаляем первый элемент

head = current->next;

current = head;

}

else

{

// Удаляем элемент из середины или конца

prev->next = current->next;

current = current->next;

}

// Обновляем last если нужно

if (temp == last)

{

last = prev;

}

free(temp);

count++;

}

else

{

// Переходим к следующему элементу

prev = current;

current = current->next;

}

}

if (count > 0)

{

printf("Удалено %d элементов с именем '%s'\n", count, name);

}

else

{

printf("Элементы с именем '%s' не найдены\n", name);

}

}

// Функция очистки всей очереди

void clear\_queue(void)

{

struct node\* current = head;

struct node\* next;

while (current != NULL)

{

next = current->next;

free(current);

current = next;

}

head = NULL;

last = NULL;

printf("Очередь очищена\n");

}

// Меню для тестирования очереди

void menu(void)

{

int choice;

char name[256];

while (1)

{

printf("\n=== Меню очереди с приоритетами ===\n");

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

printf("2. Добавить элемент с приоритетом\n");

printf("3. Удалить элемент по значению\n");

printf("4. Поиск элемента\n");

printf("5. Изменить приоритет элемента\n");

printf("6. Просмотреть всю очередь\n");

printf("0. Выход\n");

printf("Выберите действие: ");

if (scanf("%d", &choice) != 1) {

printf("Ошибка ввода\n");

while (getchar() != '\n'); // очистка буфера

continue;

}

switch (choice)

{

case 1:

enqueue();

break;

case 2:

enqueue\_with\_priority();

break;

case 3:

printf("Введите элемент для удаления: ");

scanf("%255s", name);

del(name);

break;

case 4:

printf("Введите элемент для поиска: ");

scanf("%255s", name);

find(name);

break;

case 5:

change\_priority();

break;

case 6:

review();

break;

case 0:

clear\_queue();

printf("Выход\n");

return;

default:

printf("Неверный выбор\n");

}

}

}

int main(void)

{

setlocale(LC\_ALL, "Russian");

menu();

return 0;

}