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»

Выполнил:

Студент группы 23ВВВ2

Стрельцов А.П.

Федоров Б.М.

Приняли:

Юрова О. В.

Митрохин М. А.

Пенза 2024

**Цель работы:** поработать с динамическими списками, выполнить задания и усвоить материал.

**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).**

**ЛИСТИНГ**

**laba 3-1.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

// Структура для хранения элемента очереди

typedef struct node {

char inf[256]; // Информация об элементе

int priority; // Приоритет элемента

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

} Node;

Node\* head = NULL; // Указатель на голову списка

// Функция для создания нового элемента

Node\* get\_struct(void) {

setlocale(LC\_ALL, "RUS");

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

if (p == NULL) {

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

exit(1);

}

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

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

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

scanf("%d", &(p->priority));

p->next = NULL;

return p;

}

// Функция для добавления элемента в очередь с учетом приоритета

void spstore(void) {

Node\* p = get\_struct();

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

p->next = head;

head = p;

}

else {

Node\* current = head;

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

current = current->next;

}

p->next = current->next;

current->next = p;

}

}

// Функция для просмотра содержимого списка

void review(void) {

setlocale(LC\_ALL, "RUS");

Node\* struc = head;

if (head == NULL) {

printf("Список пуст\n");

return;

}

while (struc) {

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

struc = struc->next;

}

}

int main() {

char choice;

do {

spstore();

printf("Хотите добавить ещё один объект? (y/n): ");

scanf(" %c", &choice);

// Очистка буфера ввода

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

} while (choice == 'y' || choice == 'Y');

review();

return 0;

}

Результат работы программы

На рисунке 1 показана реализация задания №1. Программа позволяет вносить название объекта и указывать его приоритетность. По приоритетности программа расставляет объекты в столбик.



Рисунок 1 - Результат работы программы №1.

1. **\* На основе приведенного кода реализуйте структуру данных *Очередь*.**

**ЛИСТИНГ**

**laba 3-2.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node {

char inf[256];

struct node\* next;

};

struct node\* head = NULL;

struct node\* last = NULL;

struct node\* get\_struct(void) {

struct node\* p = NULL;

char s[256];

setlocale(LC\_ALL, "RUS");

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

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

exit(1);

}

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

scanf("%s", s);

if (\*s == 0) {

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

free(p); // Освобождаем память, если запись не была произведена

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p;

}

void enqueue(void) {

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

if (head == NULL && p != NULL) {

head = p;

last = p;

}

else if (head != NULL && p != NULL) {

last->next = p;

last = p;

}

}

void dequeue(void) {

setlocale(LC\_ALL, "RUS");

struct node\* temp = head;

if (head == NULL) {

printf("Очередь пуста\n");

return;

}

head = head->next; // Удаляем первый элемент

if (head == NULL) {

last = NULL; // Если очередь стала пустой, обнуляем указатель на последний элемент

}

free(temp); // Освобождаем память

}

void review(void) {

setlocale(LC\_ALL, "RUS");

struct node\* struc = head;

if (head == NULL) {

printf("Очередь пуста\n");

return;

}

while (struc) {

printf("%s ", struc->inf);

struc = struc->next;

}

printf("\n");

}

int main() {

char choice;

do {

enqueue(); // Добавление нового элемента в очередь

printf("Хотите добавить ещё один объект? (y/n): ");

scanf(" %c", &choice); // Запрос на продолжение добавления

// Очистка буфера ввода

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

} while (choice == 'y' || choice == 'Y');

printf("Содержимое очереди:\n");

review(); // Вывод содержимого очереди

// Удаление элементов из очереди до её опустошения

if (head != NULL) {

dequeue(); // Удаляем первый элемент

review();

}

else {

printf("Очередь опустошена.\n");

}

return 0;

}

Результат работы программы

На рисунке 2 и 3 показана реализация задания №2. Программа реализует базовые операции с очередью: добавление элементов, удаление элементов и просмотр содержимого очереди.



Рисунок 2 - Результат работы программы №2.



Рисунок 3 - Результат работы программы №2.

1. **\* На основе приведенного кода реализуйте структуру данных *Стек*.**

**ЛИСТИНГ**

**laba 3-3.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node {

char inf[256]; // Данные элемента

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

};

struct node\* top = NULL; // Указатель на верхний элемент стека

// Функция для создания нового элемента

struct node\* get\_struct(void) {

struct node\* p = NULL;

char s[256];

setlocale(LC\_ALL, "RUS");

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

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

exit(1);

}

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

scanf("%s", s);

if (\*s == 0) {

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

free(p); // Освобождаем память, если запись не была произведена

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p; // Возвращаем указатель на созданный элемент

}

// Функция для добавления элемента в стек

void push(void) {

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

if (p != NULL) {

p->next = top; // Новый элемент указывает на текущий верхний элемент

top = p; // Обновляем верхний элемент

}

}

// Функция для удаления элемента из стека

void pop(void) {

setlocale(LC\_ALL, "RUS");

if (top == NULL) {

printf("Стек пуст\n");

return;

}

struct node\* temp = top; // Сохраняем указатель на верхний элемент

top = top->next; // Обновляем верхний элемент

free(temp); // Освобождаем память

}

// Функция для просмотра содержимого стека

void review(void) {

setlocale(LC\_ALL, "RUS");

struct node\* struc = top;

if (top == NULL) {

printf("Стек пуст\n");

return;

}

while (struc) {

printf("%s ", struc->inf);

struc = struc->next;

}

printf("\n");

}

int main() {

char choice;

setlocale(LC\_ALL, "RUS");

do {

printf("Выберите действие:\n");

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

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

printf("3. Просмотреть содержимое стека\n");

printf("4. Выход\n");

printf("Ваш выбор: ");

scanf(" %c", &choice); // Запрос на выбор действия

switch (choice) {

case '1':

push(); // Добавление элемента в стек

break;

case '2':

pop(); // Удаление элемента из стека

break;

case '3':

review(); // Просмотр содержимого стека

break;

case '4':

printf("Выход из программы.\n");

break;

default:

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

}

// Очистка буфера ввода

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

} while (choice != '4'); // Продолжаем, пока пользователь не выберет выход

return 0;

}

Результат работы программы

На рисунке 4, 5 и 6 показана реализация задания №3. Программа реализует базовые операции со стеком: добавление элементов, удаление элементов и просмотр содержимого стека.

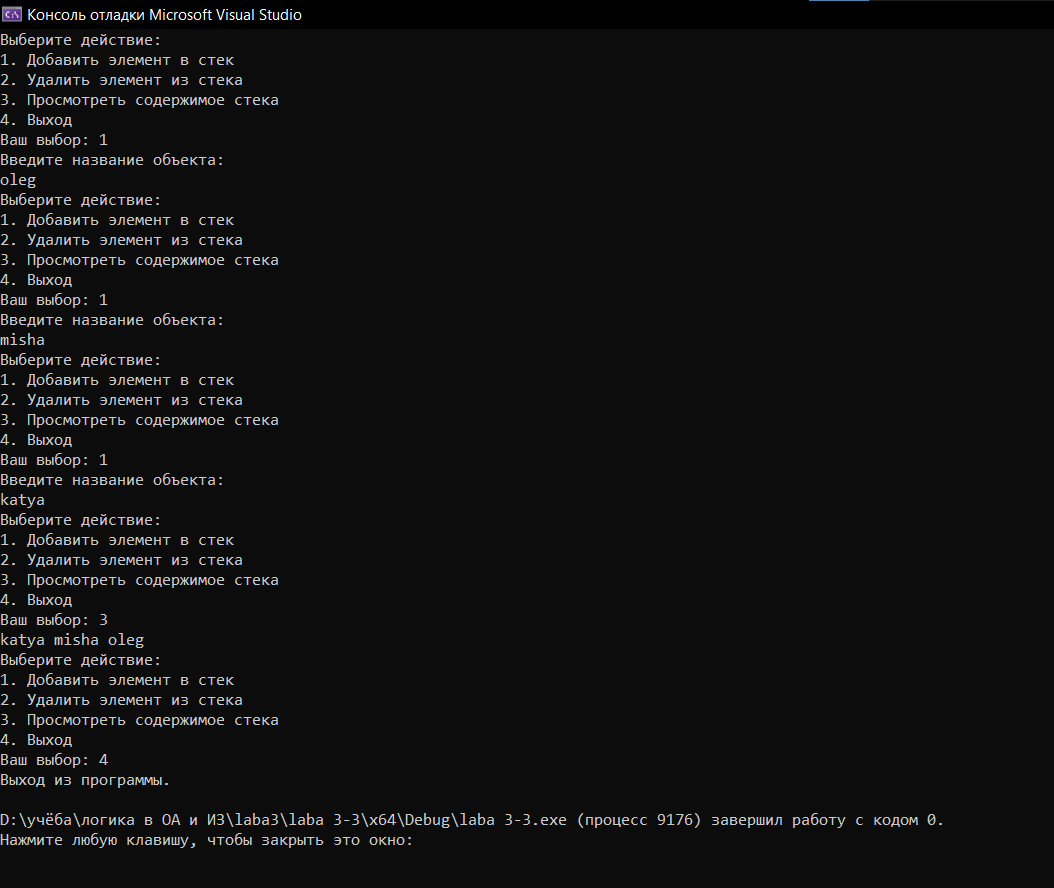


Рисунок 4 - Результат работы программы №3.

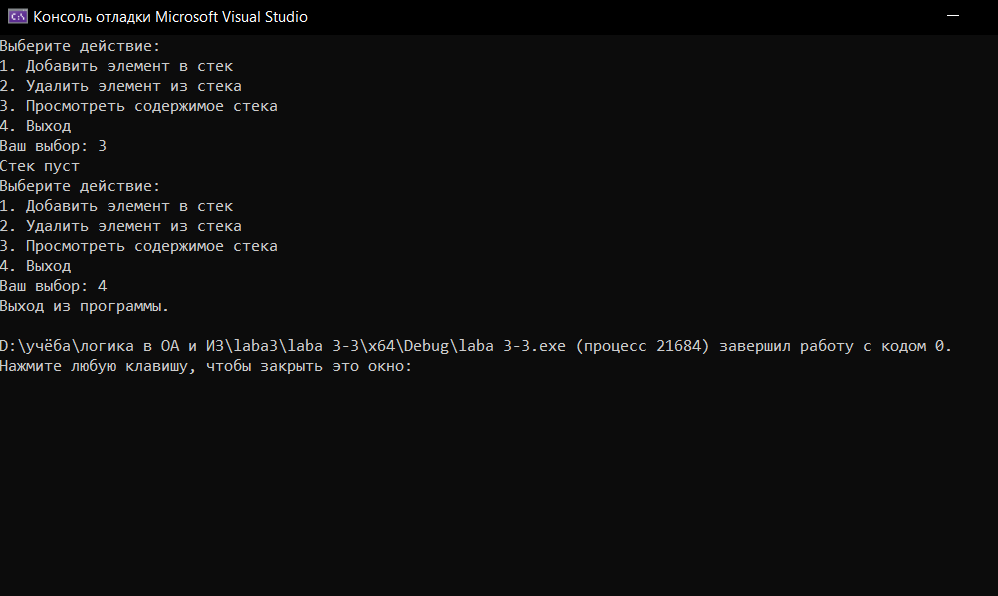


Рисунок 5 - Результат работы программы №3.

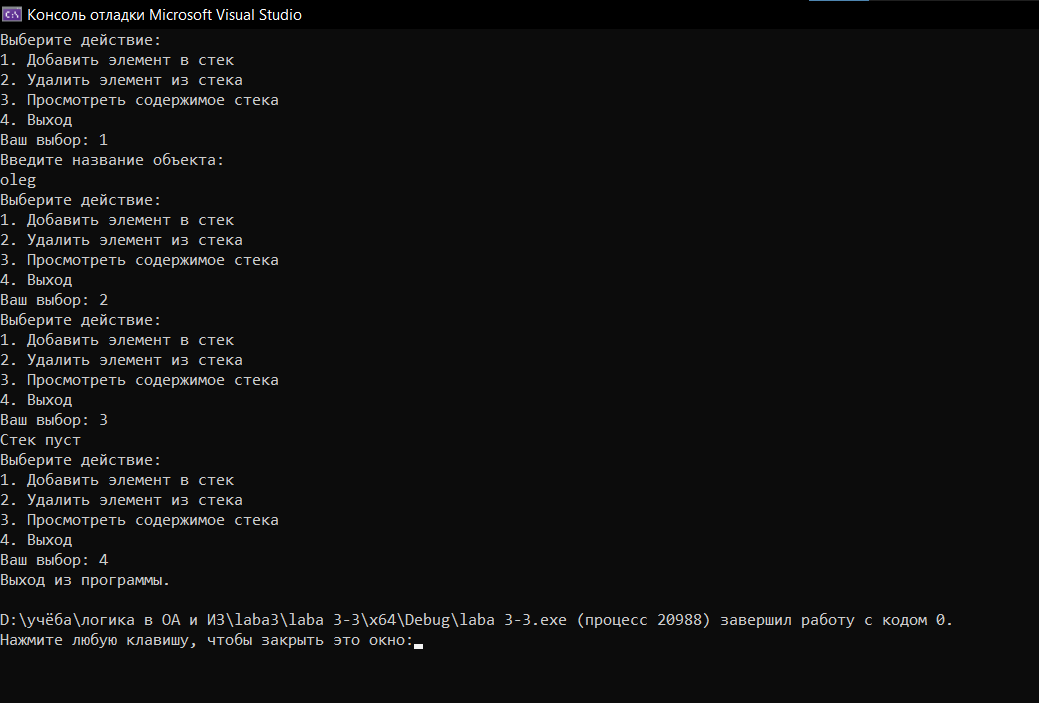


Рисунок 6 - Результат работы программы №3.

**Вывод**

изучили новые функции, применили их в лабораторной работе и оценили время выполнения данных нам программ.