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 22ВВВ3:

Байков А. В.

Гераськина Д. А.

Приняли:

Юрова О. В.

Акифьев И. В.

Пенза 2023

**Цель работы**

Повторить и усовершенствовать полученные ранее знания о динамических списках на языке СИ.

**Лабораторное 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**

**Lab3.1**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <locale.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char inputFileName[] = "lear.txt"; // Файл для загрузки очереди

char outputFileName[] = "lear.txt"; // Файл для сохранения очереди

struct node {

char inf[256]; // полезная информация

int priority; // приоритет

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

};

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

int len = 0;

struct node\* get\_struct(void) {

struct node\* p = NULL;

char s[256];

int priority;

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL) { // выделяем память под новый элемент списка

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

exit(1);

}

printf("Введите название объекта: \n"); // вводим данные

scanf("%s", s);

if (\*s == 0) {

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

return NULL;

}

strcpy(p->inf, s);

printf("\nВведите приоритет объекта: \n"); // вводим приоритет

scanf("%d", &priority);

p->priority = priority;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в соответствии с приоритетом) \*/

void spstore(void) {

struct node\* p = NULL;

p = get\_struct();

if (head == NULL && p != NULL) { // если списка нет, то устанавливаем голову списка

head = p;

last = p;

}

else if (head != NULL && p != NULL) { // список уже есть, то вставляем в соответствии с приоритетом

struct node\* current = head;

struct node\* prev = NULL;

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

prev = current;

current = current->next;

}

if (prev == NULL) {

// Вставляем в начало списка

p->next = head;

head = p;

}

else {

// Вставляем между prev и current

prev->next = p;

p->next = current;

if (current == NULL) {

last = p; // Если p вставлен в конец списка, обновляем last

}

}

}

}

/\* Просмотр содержимого списка. \*/

void review(void) {

struct node\* struc = head;

if (head == NULL) {

printf("\nСписок пуст\n");

}

while (struc) {

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

struc = struc->next;

}

}

/\* Поиск элемента по содержимому. \*/

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

struct node\* struc = head;

if (head == NULL) {

printf("\nСписок пуст\n");

}

while (struc) {

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

return struc;

}

struc = struc->next;

}

printf("\nЭлемент не найден\n");

return NULL;

}

/\* Удаление элемента по содержимому. \*/

void del(char\* name) {

struct node\* struc = head; // указатель, проходящий по списку установлен на начало списка

struct node\* prev = NULL; // указатель на предшествующий удаляемому элементу

int flag = 0; // индикатор отсутствия удаляемого элемента в списке

if (head == NULL) // если голова списка равна NULL, то список пуст

{

printf("\nСписок пуст\n");

return;

}

if (strcmp(name, struc->inf) == 0) // если удаляемый элемент - первый

{

flag = 1;

head = struc->next; // устанавливаем голову на следующий элемент

free(struc); // удаляем первый элемент

struc = head; // устанавливаем указатель для продолжения поиска

}

else {

prev = struc;

struc = struc->next;

}

while (struc) // проход по списку и поиск удаляемого элемента

{

if (strcmp(name, struc->inf) == 0) // если нашли, то

{

flag = 1; // выставляем индикатор

if (struc->next) // если найденный элемент не последний в списке

{

prev->next = struc->next; // меняем указатели

free(struc); // удаляем элемент

struc = prev->next; // устанавливаем указатель для продолжения поиска

}

else // если найденный элемент последний в списке

{

prev->next = NULL; // обнуляем указатель предшествующего элемента

free(struc); // удаляем элемент

last = prev; // обновляем last

return;

}

}

else // если не нашли, то

{

prev = struc; // устанавливаем указатели для продолжения поиска

struc = struc->next;

}

}

if (flag == 0) // если флаг = 0, значит нужный элемент не найден

{

printf("\nЭлемент не найден\n");

return;

}

}

void SaveFile(const char\* filename) {

FILE\* file = fopen(filename, "w");

if (file == NULL) {

printf("\nОшибка при открытии файла для записи\n");

return;

}

struct node\* struc = head;

while (struc) {

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

struc = struc->next;

}

fclose(file);

printf("\nОчередь успешно сохранена в файл %s\n", filename);

}

void LoadFile(const char\* filename) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

printf("\nОшибка при открытии файла для чтения\n");

return;

}

// Очистка существующей очереди

struct node\* current = head;

while (current != NULL) {

struct node\* next = current->next;

free(current);

current = next;

}

head = last = NULL;

char inf[256];

int priority;

while (fscanf(file, "%s %d", inf, &priority) != EOF) {

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

if (p == NULL) {

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

fclose(file);

exit(1);

}

strcpy(p->inf, inf);

p->priority = priority;

p->next = NULL;

// Вставка элемента в отсортированную очередь

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

// Вставка в начало

p->next = head;

head = p;

}

else {

struct node\* current = head;

struct node\* prev = NULL;

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

prev = current;

current = current->next;

}

// Вставка между prev и current

prev->next = p;

p->next = current;

if (current == NULL) {

last = p; // Обновление last, если p вставлен в конец

}

}

}

fclose(file);

printf("\nОчередь загружена из файла %s\n", filename);

}

int main() {

setlocale(LC\_ALL, "Rus");

int choice;

char name[256];

struct node\* found = NULL;

while (true)

{

printf("\n1. Просмотреть очередь\n");

printf("2. Добавить элемент в очередь\n");

printf("3. Найти элемент\n");

printf("4. Удалить элемент\n");

printf("5. Сохранить очередь в файл\n");

printf("6. Загрузить очередь из файла\n");

printf("7. Выход\n");

printf("\nВыберите операцию: ");

scanf("%d", &choice);

printf("\n");

switch (choice)

{

case 1:

review();

break;

case 2:

spstore();

break;

case 3:

printf("Введите название объекта для поиска: ");

scanf("%s", name);

found = find(name);

if (found != NULL) {

printf("Найденный элемент: Имя - %s, Приоритет - %d\n", found->inf, found->priority);

}

break;

case 4:

printf("Какой объект удалит?: ");

scanf("%s", name);

del(name);

break;

case 5:

SaveFile(outputFileName); // Function to save the queue to a file

break;

case 6:

LoadFile(inputFileName); // Function to load the queue from a file

break;

case 7:

exit(0);

default:

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

}

}

return 0;

}

**Lab3.2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node

{

char inf[256]; // Полезная информация

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

};

struct node\* front = NULL, \* rear = NULL; // Указатели на начало и конец очереди

void enqueue(void); // Добавление элемента в очередь

void dequeue(void); // Удаление элемента из очереди

void display(void); // Просмотр содержимого очереди

void saveToFile(void); // Сохранение данных очереди в файл

void loadFromFile(void); // Загрузка данных очереди из файла

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL) // Выделяем память под новый элемент очереди

{

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

exit(1);

}

printf("Введите название объекта: \n"); // Вводим данные

scanf("%s", s);

if (\*s == 0)

{

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

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p; // Возвращаем указатель на созданный элемент

}

/\* Добавление элемента в очередь \*/

void enqueue(void)

{

struct node\* p = NULL;

p = get\_struct();

if (rear == NULL && p != NULL) // Если очередь пуста, то устанавливаем начало и конец очереди

{

front = p;

rear = p;

}

else if (rear != NULL && p != NULL) // В очереди уже есть элементы, добавляем в конец

{

rear->next = p;

rear = p;

}

return;

}

/\* Удаление элемента из очереди (из начала) \*/

void dequeue(void)

{

if (front == NULL) // Если начало очереди равно NULL, то очередь пуста

{

printf("Очередь пуста\n");

return;

}

struct node\* temp = front;

front = front->next;

free(temp);

if (front == NULL) // Если после удаления элемента начало очереди стало NULL, то это был последний элемент

{

rear = NULL; // Устанавливаем и конец очереди в NULL

}

}

/\* Просмотр содержимого очереди \*/

void display(void)

{

struct node\* struc = front;

if (front == NULL)

{

printf("Очередь пуста\n");

}

while (struc)

{

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

struc = struc->next;

}

return;

}

/\* Сохранение данных очереди в файл \*/

void saveToFile(void)

{

FILE\* file;

struct node\* struc = front;

if (front == NULL)

{

printf("Очередь пуста, нет данных для сохранения\n");

return;

}

file = fopen("queue.txt", "w");

if (file == NULL)

{

printf("Ошибка при открытии файла\n");

return;

}

while (struc)

{

fprintf(file, "%s\n", struc->inf);

struc = struc->next;

}

fclose(file);

printf("Данные очереди успешно сохранены в файле\n");

}

/\* Загрузка данных очереди из файла \*/

void loadFromFile(void)

{

FILE\* file;

char s[256];

file = fopen("queue.txt", "r");

if (file == NULL)

{

printf("Файл не найден или ошибка при открытии\n");

return;

}

while (fscanf(file, "%s", s) != EOF)

{

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

if (p == NULL)

{

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

exit(1);

}

strcpy(p->inf, s);

p->next = NULL;

if (rear == NULL)

{

front = p;

rear = p;

}

else

{

rear->next = p;

rear = p;

}

}

fclose(file);

printf("Данные из файла успешно загружены в очередь\n");

}

int main()

{

setlocale(LC\_ALL, "Rus");

int choice;

while (1)

{

printf("\nОперации над очередью:\n");

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

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

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

printf("4. Сохранить данные очереди в файл\n");

printf("5. Загрузить данные очереди из файла\n");

printf("6. Выход\n");

printf("Введите выбор: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

saveToFile();

break;

case 5:

loadFromFile();

break;

case 6:

exit(0);

default:

printf("Неправильный выбор\n");

}

}

return 0;

}

**Lab3.3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node

{

char inf[256]; // Полезная информация

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

};

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

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

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

void display(void); // Просмотр содержимого стека

void saveToFile(void); // Сохранение данных стека в файл

void loadFromFile(void); // Загрузка данных стека из файла

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL) // Выделяем память под новый элемент стека

{

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

exit(1);

}

printf("Введите название объекта: \n"); // Вводим данные

scanf("%s", s);

if (\*s == 0)

{

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

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p; // Возвращаем указатель на созданный элемент

}

/\* Добавление элемента в стек \*/

void push(void)

{

struct node\* p = NULL;

p = get\_struct();

if (p != NULL)

{

p->next = top;

top = p;

}

}

/\* Удаление элемента из стека (с вершины) \*/

void pop(void)

{

if (top == NULL)

{

printf("Стек пуст\n");

return;

}

struct node\* temp = top;

top = top->next;

free(temp);

}

/\* Просмотр содержимого стека \*/

void display(void)

{

struct node\* struc = top;

if (top == NULL)

{

printf("Стек пуст\n");

}

while (struc)

{

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

struc = struc->next;

}

}

/\* Сохранение данных стека в файл \*/

void saveToFile(void)

{

FILE\* file;

struct node\* struc = top;

if (top == NULL)

{

printf("Стек пуст, нет данных для сохранения\n");

return;

}

file = fopen("stack.txt", "w");

if (file == NULL)

{

printf("Ошибка при открытии файла\n");

return;

}

while (struc)

{

fprintf(file, "%s\n", struc->inf);

struc = struc->next;

}

fclose(file);

printf("Данные стека успешно сохранены в файле\n");

}

/\* Загрузка данных стека из файла \*/

void loadFromFile(void)

{

FILE\* file;

char s[256];

file = fopen("stack.txt", "r");

if (file == NULL)

{

printf("Файл не найден или ошибка при открытии\n");

return;

}

while (fscanf(file, "%s", s) != EOF)

{

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

if (p == NULL)

{

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

exit(1);

}

strcpy(p->inf, s);

p->next = top;

top = p;

}

fclose(file);

printf("Данные из файла успешно загружены в стек\n");

}

int main()

{

setlocale(LC\_ALL, "Rus");

int choice;

while (1)

{

printf("\nОперации над стеком:\n");

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

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

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

printf("4. Сохранить данные стека в файл\n");

printf("5. Загрузить данные стека из файла\n");

printf("6. Выход\n");

printf("Введите выбор: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

saveToFile();

break;

case 5:

loadFromFile();

break;

case 6:

exit(0);

default:

printf("Неправильный выбор\n");

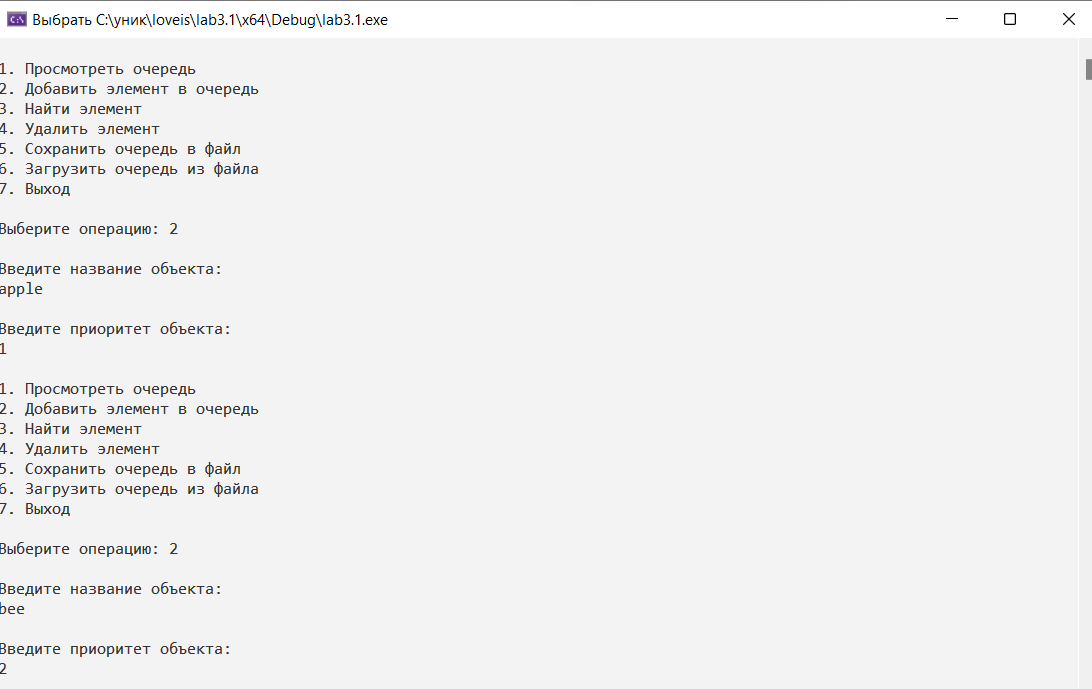
}

}

return 0;

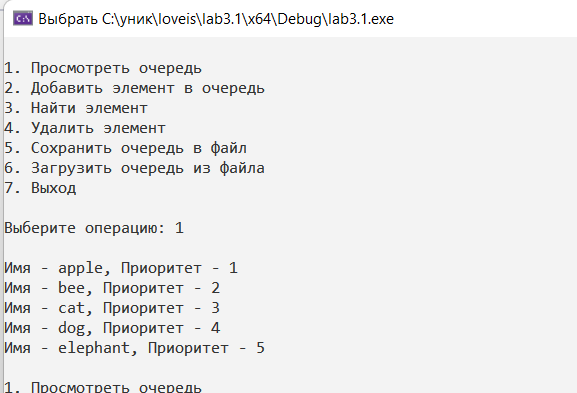
}

**Результаты работы программ**



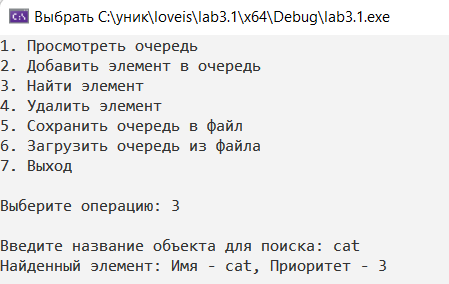
**Рисунок 1 - Результат работы программы Lab3.1**

**(операция 2 – «добавить элемент в очередь»)**

****

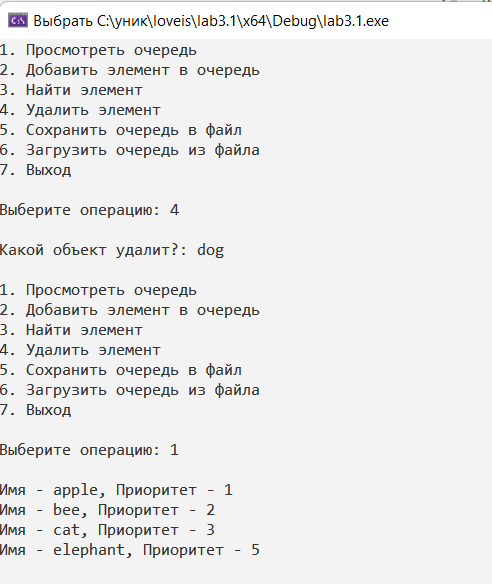
**Рисунок 2 - Результат работы программы Lab3.1**

**(операция 1 – «посмотреть очередь»)**

****

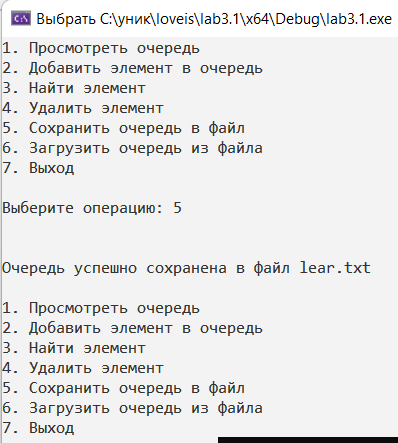
**Рисунок 3 - Результат работы программы Lab3.1**

**(операция 3 – «найти элемент»)**

****

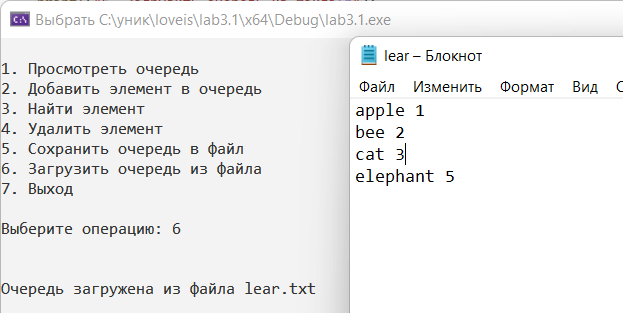
**Рисунок 4 - Результат работы программы Lab3.1**

**(операция 4 – «удалить элемент»)**

****

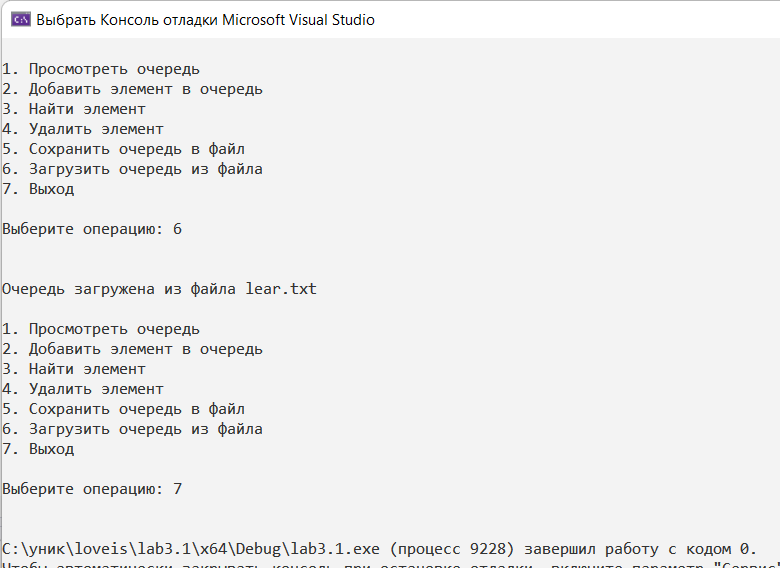
**Рисунок 5 - Результат работы программы Lab3.1**

**(операция 5 – «сохранить очередь в файл»)**

****

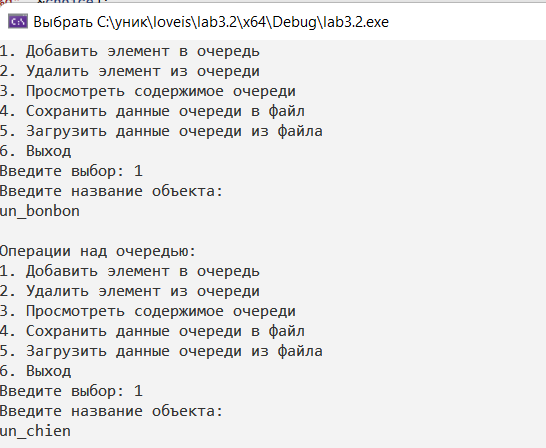
**Рисунок 6 - Результат работы программы Lab3.1**

**(операция 6 – «загрузить очередь из файла»**

****

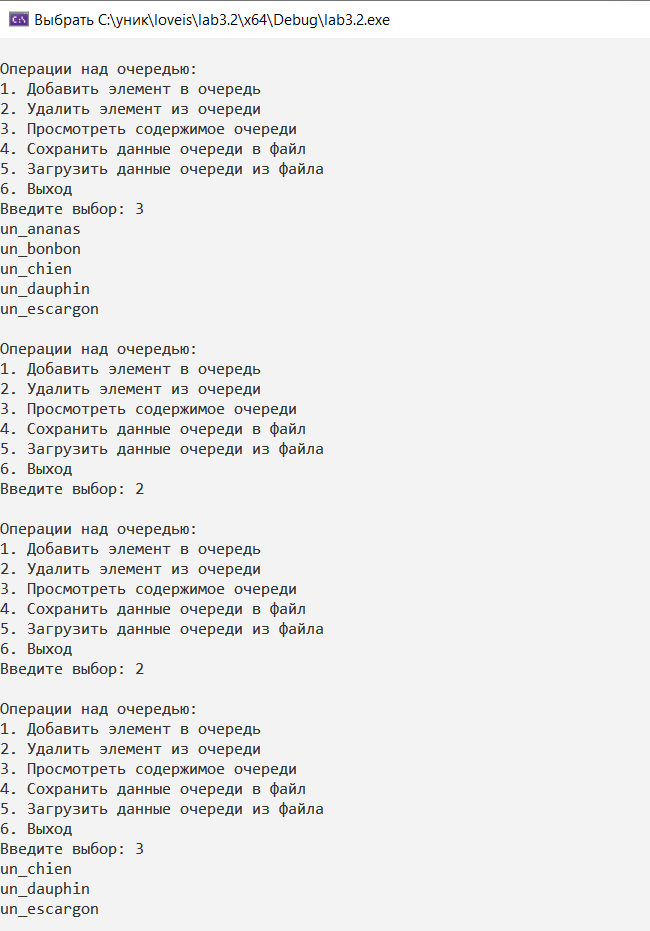
**Рисунок 7 - Результат работы программы Lab3.1**

**(операция 7 – «выход»**



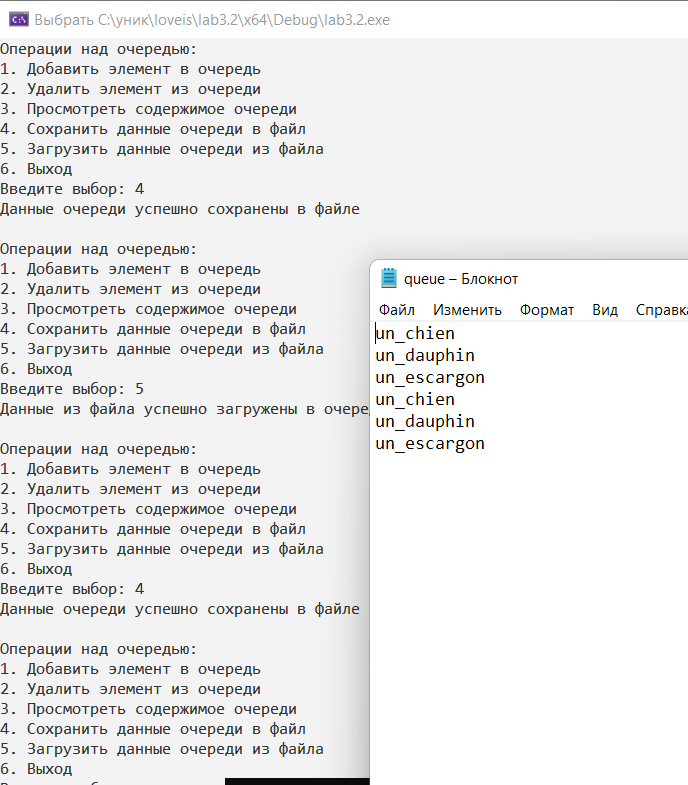
**Рисунок 8 - Результат работы программы Lab3.2**

**(операция 1 – «добавить элемент в очередь»)**

****

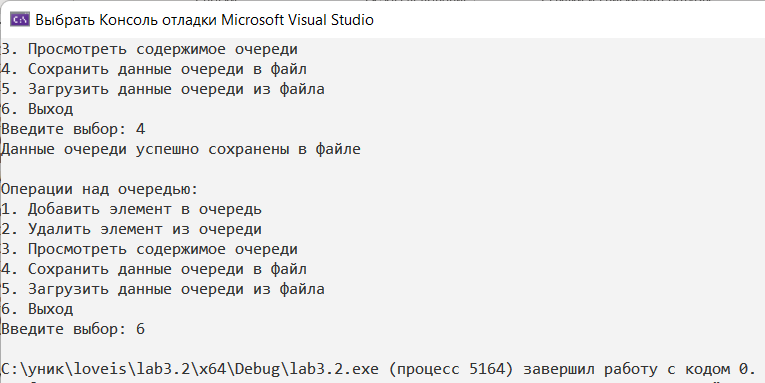
**Рисунок 9 - Результат работы программы Lab3.2**

**(операции 2 и 3 – «удалить элемент из очереди» и «посмотреть содержимое очереди»)**

****

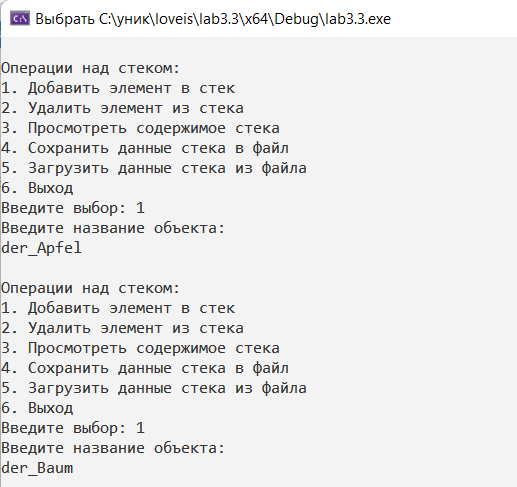
**Рисунок 10 - Результат работы программы Lab3.2**

**(операции 4 и 5 – «сохранить/загрузить данные очереди в файл/из файла»)**

****

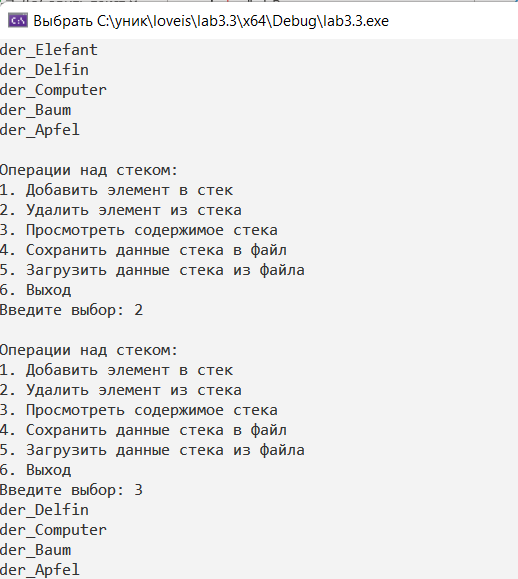
**Рисунок 11 - Результат работы программы Lab3.2**

**(операция 6 – «выход»)**

****

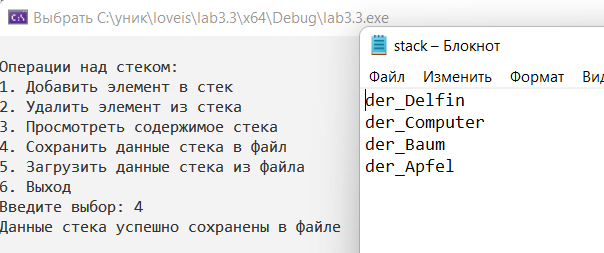
**Рисунок 12 - Результат работы программы Lab3.3**

**(операция 1 – «добавить элемент в стек»)**

****

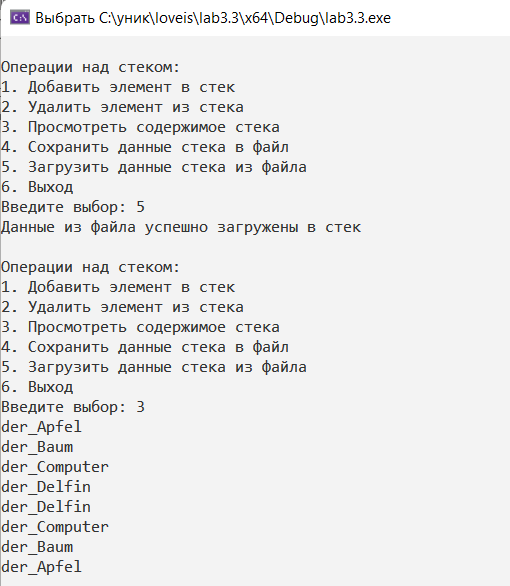
**Рисунок 13 - Результат работы программы Lab3.3**

**(операции 2 и 3 – «удалить элемент из стека» и «посмотреть содержимое стека»)**

****

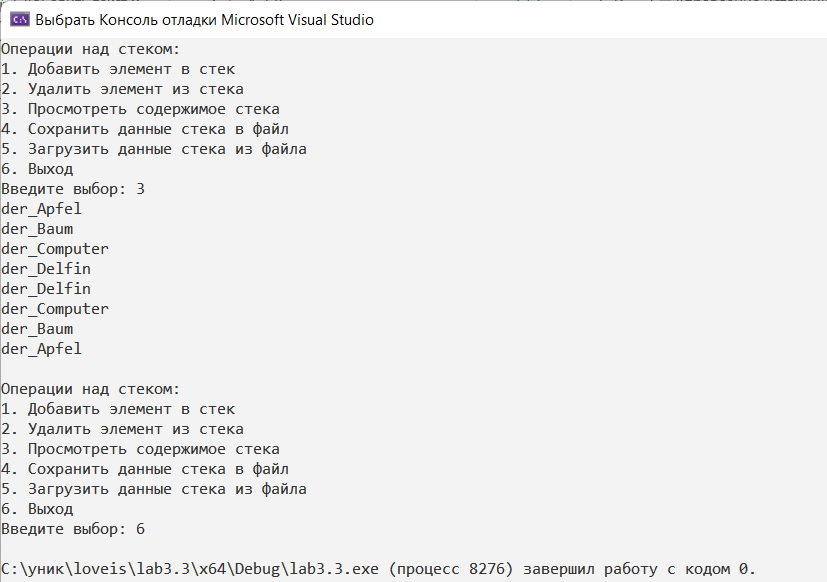
**Рисунок 14 - Результат работы программы Lab3.3**

**(операция 4 – «сохранить данные стека в файл»)**

****

**Рисунок 15 - Результат работы программы Lab3.3**

**(операция 5 – «загрузить данные стека из файла»)**

****

**Рисунок 16 - Результат работы программы Lab3.3**

**(операция 6 – «выход»**

**Вывод**

В ходе работы, удалось повторить и усовершенствовать полученные ранее знания о динамических списках на языке СИ.