МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №3  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Динамические списки»

Выполнили:

Студенты группы 23ВВВ2

Лисов Е.А.

Кочегин В.В..

Приняли:

Митрохин М. А.  
Юрова О.В.

Пенза 2024

**Цель работы**

Разработка и реализация структур данных, таких как список с приоритетом элементов, очередь и стек, с использованием базовых принципов программирования.

**Задание**

**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).

**Задание 2**:

На основе приведенного кода реализуйте структуру данных *Очередь*.

**Задание 3**:

На основе приведенного кода реализуйте структуру данных *Стек*.

**Ход работы**

**Задание 1.**

С использованием заранее приведённых функций внесли изменения для создания рабочей программы.

Использовали int prior; для приоритетного расположения элементов в списке.

struct node

{

char inf[256];

int prior;

struct node\* next;

};

Изменили функцию spstore(void) теперь она сортирует список по приоритету, приоритет задается вручную в функции get\_struct(void).

void spstore(void)

{

struct node\* p = NULL;

p = get\_struct();

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

head = p;

last = p;

}

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

if (head->prior < p->prior) {

p->next = head;

head = p;

}

else{

struct node\* struc = head;

while (struc->next != NULL && struc->next->prior >= p->prior)

{

struc = struc->next;

}

p->next = struc->next;

struc->next = p;

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

}

}

return;

}

**Задание 2.**

Структура данных *Очередь* была реализована на базе предоставленной программы, с изменением функции удаления и записи элементов.

Запись элемента производится только в конец в функция spstore(void) (изменений не было).

При вызове функции del() удаляется только первый элемент.

void del()

{

struct node\* struc = head;

struct node\* prev;

int flag = 0;

if (head == NULL) {

printf("Список пуст\n");

return;

}

prev = struc->next;

head = prev;

free(struc);

}

**Задание 3.**

Для реализации структуры данных *Стек* новый элемент добавляется в начало списка, функция spstore(void), и удаление осуществляется только последнего добавленного элемента, функция del().

Добавление:  
 else if (head != NULL && p != NULL)

{

p->next = head;

head = p;

}

Удаление:

prev = struc->next;

head = prev;

free(struc);

### Результаты работы программы

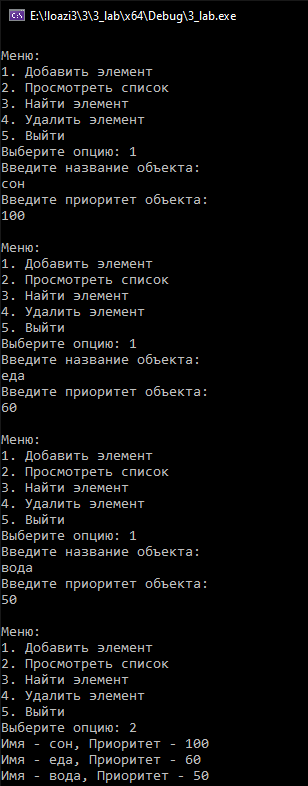


Рисунок 1 — Результаты работы программы пункта №1

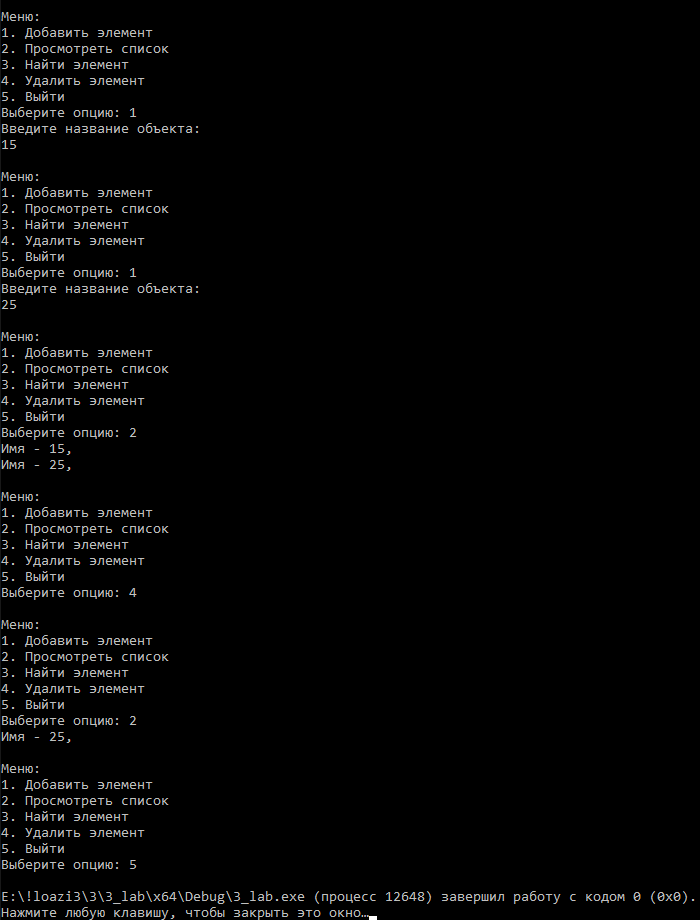


Рисунок 2 — Результаты работы программы пункта №2

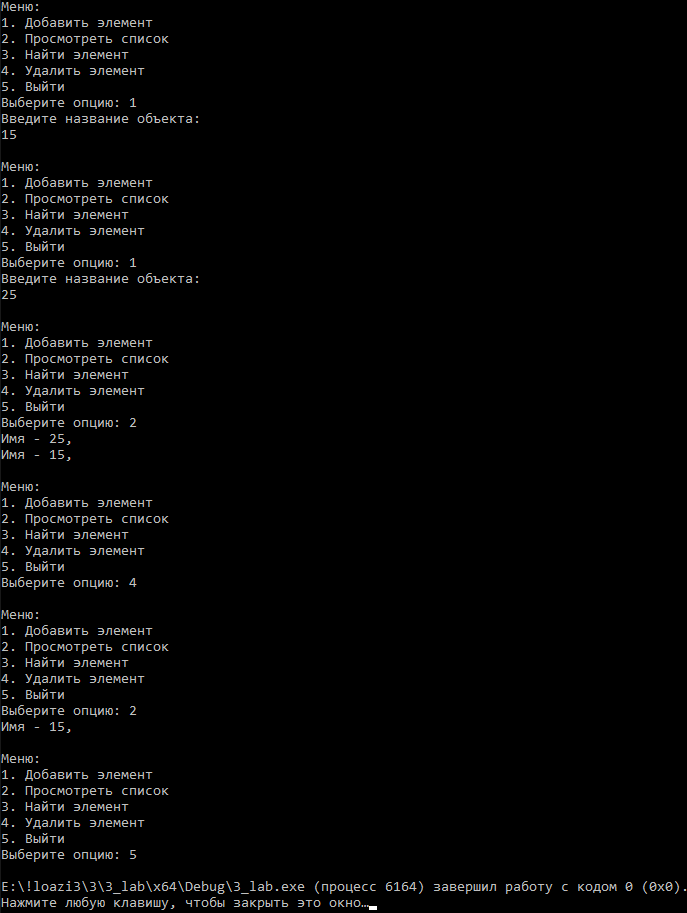


Рисунок 3 — Результаты работы программы пункта №3

### Вывод

В результате выполнения работы были успешно разработаны и реализованы три ключевые структуры данных: приоритетная очередь, очередь и стек. Каждая из этих структур имеет свои уникальные свойства и области применения, что делает их важными инструментами в арсенале программиста. Освоение этих структур данных помогает лучше понимать алгоритмы и оптимизировать решения для различных задач.

**Листинг**

**Задание 1:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <Windows.h>

struct node

{

char inf[256]; // полезная информация

int prior;

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

};

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

// Функции добавления элемента, просмотра списка

void spstore(void), review(void), del(char\* name);

char find\_el[256];

struct node\* find(char\* name); // функция нахождения элемента

struct node\* get\_struct(void); // функция создания элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

int prior;

if ((p = (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("Введите приоритет объекта: \n");

scanf("%d", &prior);

p->prior = prior;

p->next = NULL;

return p;

}

void spstore(void)

{

struct node\* p = NULL;

p = get\_struct();

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

{

head = p;

last = p;

}

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

{

if (head->prior < p->prior) {

p->next = head;

head = p;

}

else{

struct node\* struc = head;

while (struc->next != NULL && struc->next->prior >= p->prior)

{

struc = struc->next;

}

p->next = struc->next;

struc->next = p;

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

}

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

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

struc = struc->next;

}

return;

}

/\* Поиск элемента по содержимому. \*/

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

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

{

return struc;

}

struc = struc->next;

}

printf("Элемент 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 элемент

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

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

{

printf("Список пуст\n");

return;

}

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

{

flag = 1;

head = struc->next; // установливаем 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

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("Элемент не найден\n");

return;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char choice = '0';

char name[256];

while (choice!='5') {

printf("\nМеню:\n");

printf("1. Добавить элемент\n");

printf("2. Просмотреть список\n");

printf("3. Найти элемент\n");

printf("4. Удалить элемент\n");

printf("5. Выйти\n");

printf("Выберите опцию: ");

scanf(" %c", &choice);

switch (choice) {

case '1':

spstore();

break;

case '2':

review();

break;

case '3':

printf("Введите имя для поиска: ");

scanf("%s", name);

if (find(name)) {

printf("Элемент найден: Имя - %s\n", name);

}

break;

case '4':

printf("Введите имя для удаления: ");

scanf("%s", name);

del(name);

break;

case '5':

exit(0);

break;

default:

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

}

}

return 0;

}

**Задание 2:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <Windows.h>

struct node

{

char inf[256]; // полезная информация

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

};

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

// Функции добавления элемента, просмотра списка

void spstore(void), review(void), del(char\* name);

char find\_el[256];

struct node\* find(char\* name); // функция нахождения элемента

struct node\* get\_struct(void); // функция создания элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (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);

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

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 конец

{

last->next = p;

last = p;

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("Имя - %s, \n", struc->inf);

struc = struc->next;

}

return;

}

/\* Поиск элемента по содержимому. \*/

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

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

{

return struc;

}

struc = struc->next;

}

printf("Элемент не найден\n");

return NULL;

}

/\* Удаление элемента по содержимому. \*/

void del()

{

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

struct node\* prev;// указатель на предшествующий удаляемому элемент

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

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

{

printf("Список пуст\n");

return;

}

prev = struc->next;

head = prev;

free(struc);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char choice = '0';

char name[256];

while (choice != '5') {

printf("\nМеню:\n");

printf("1. Добавить элемент\n");

printf("2. Просмотреть список\n");

printf("3. Найти элемент\n");

printf("4. Удалить элемент\n");

printf("5. Выйти\n");

printf("Выберите опцию: ");

scanf(" %c", &choice);

switch (choice) {

case '1':

spstore();

break;

case '2':

review();

break;

case '3':

printf("Введите имя для поиска: ");

scanf("%s", name);

if (find(name)) {

printf("Элемент найден: Имя - %s\n", name);

}

break;

case '4':

del();

break;

case '5':

exit(0);

break;

default:

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

}

}

return 0;

}

**Задание 3:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <Windows.h>

struct node

{

char inf[256]; // полезная информация

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

};

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

// Функции добавления элемента, просмотра списка

void spstore(void), review(void), del(char\* name);

char find\_el[256];

struct node\* find(char\* name); // функция нахождения элемента

struct node\* get\_struct(void); // функция создания элемента

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (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);

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

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)

{

p->next = head;

head = p;

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("Имя - %s, \n", struc->inf);

struc = struc->next;

}

return;

}

/\* Поиск элемента по содержимому. \*/

struct node\* find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

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

{

return struc;

}

struc = struc->next;

}

printf("Элемент не найден\n");

return NULL;

}

/\* Удаление элемента по содержимому. \*/

void del()

{

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

struct node\* prev;// указатель на предшествующий удаляемому элемент

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

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

{

printf("Список пуст\n");

return;

}

prev = struc->next;

head = prev;

free(struc);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char choice = '0';

char name[256];

while (choice != '5') {

printf("\nМеню:\n");

printf("1. Добавить элемент\n");

printf("2. Просмотреть список\n");

printf("3. Найти элемент\n");

printf("4. Удалить элемент\n");

printf("5. Выйти\n");

printf("Выберите опцию: ");

scanf(" %c", &choice);

switch (choice) {

case '1':

spstore();

break;

case '2':

review();

break;

case '3':

printf("Введите имя для поиска: ");

scanf("%s", name);

if (find(name)) {

printf("Элемент найден: Имя - %s\n", name);

}

break;

case '4':

del();

break;

case '5':

exit(0);

break;

default:

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

}

}

return 0;

}