**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“

Выполнили студенты группы 23ВВВ1 :

Макаров А.С.

Приняли:

д.т.н. доцент Митрохин М.А.

к.т.н. доцент Юрова О.В.

Пенза 2024

**Цель работы.**

Вспомнить работу с динамическими списками в Си и закрепить на практике.

**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:**

**1)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

struct node

{

char inf[256];

struct node\* next;

int priority = 0;

};

struct node\* head = NULL;

int dlinna = 0;

// 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 priority = 0;

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

{

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

exit(1);

}

printf("Введите название объекта: ");

scanf("%s", s);

printf("\n");

if (\*s == 0)

{

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

return NULL;

}

strcpy(p->inf, s);

// Добавляем приоритет

printf("Введите приоритет объекта: ");

scanf("%d",&priority);

printf("\n");

p->priority = priority;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (по приоритету)\*/

void spstore()

{

node\* p = get\_struct();

if (p == NULL)

return;

if (head == NULL) // если списка нет, то устанавливаем голову списка

{

head = p;

}

else // список уже есть, то вставляем в конец

{

node\* current = head;

if (p->priority > current->priority) // если приоритет нового элемента больше, чем у текущего элемента

{

p->next = current;

head = p;

}

else // если приоритет нового элемента меньше или равен, чем у текущего элемента

{

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

{

current = current->next;

}

p->next = current->next;

current->next = p;

}

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

int i = 1;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

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

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

{

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;

}

}

void delete\_element()

{

char name[256];

printf("Введите имя элемента: ");

scanf("%s", name);

printf("\n");

del(name);

}

void print\_menu()

{

printf("Выберете действие:\n");

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

printf("2. Удалить элемент \n");

printf("3. Вывести список \n");

printf("4. Выход \n");

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

bool exit = false;

while(!exit)

{

print\_menu();

int option = 0;

scanf("%d", &option);

switch(option)

{

case 1: spstore(); break;

case 2: delete\_element(); break;

case 3: review(); break;

case 4: exit = true; break;

default: std::cin.get(); break;

}

}

return 0;

}

2)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

#include <iostream>

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

int dlinna = 0;

// 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: "); // вводим данные

scanf("%s", s);

printf("\n");

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;

}

node\* get\_element()

{

node\* res = head;

head = head->next;

res->next = NULL;

return res;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

int i = 1;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("%d) Имя - %s \n", i++, 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;

}

void print\_menu()

{

printf("Выберете действие:\n");

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

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

printf("3. Вывести очередь \n");

printf("4. Выход \n");

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

bool exit = false;

while (!exit)

{

print\_menu();

int option = 0;

scanf("%d", &option);

switch (option)

{

case 1: spstore(); break;

case 2: printf("Полученный элемент: %s\n", get\_element()->inf); break;

case 3: review(); break;

case 4: exit = true; break;

default: std::cin.get(); break;

}

}

return 0;

}

**3)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

#include <iostream>

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

int dlinna = 0;

// 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;

}

node\* get\_element()

{

node\* curr = head;

node\* prev = curr;

while(curr->next != NULL)

{

prev = curr;

curr = curr->next;

}

prev->next = NULL;

return curr;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

int i = 1;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("%d) %s \n", i++, 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(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;

}

}

void print\_menu()

{

printf("Выберете действие:\n");

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

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

printf("3. Вывести очередь \n");

printf("4. Выход \n");

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

bool exit = false;

while (!exit)

{

print\_menu();

int option = 0;

scanf("%d", &option);

switch (option)

{

case 1: spstore(); break;

case 2: printf("Полученный элемент: %s\n", get\_element()->inf); break;

case 3: review(); break;

case 4: exit = true; break;

default: std::cin.get(); break;

}

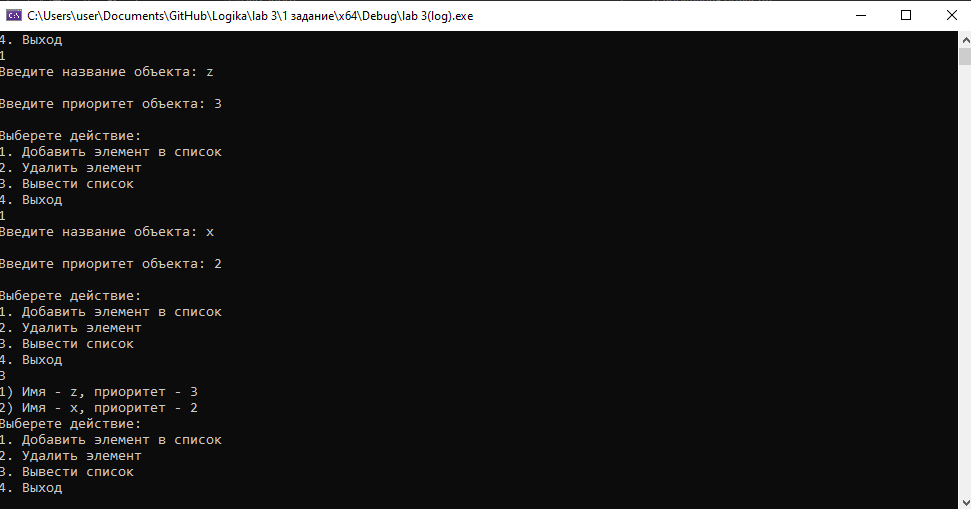
}

return 0;

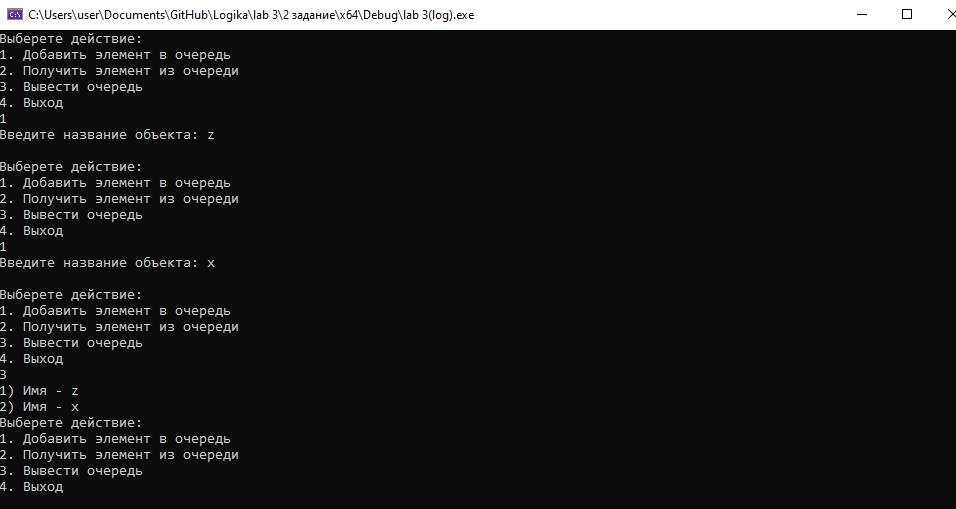
}

**Результат:**

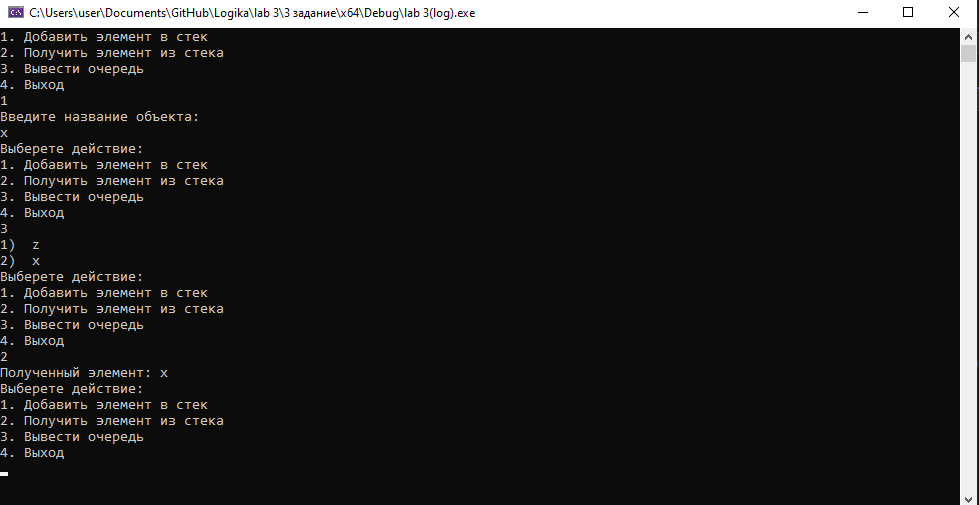
**1)**



**2)**

****

**3)**

****

**Вывод:**

В ходе лабораторной работы мы вспомнили работу с динамическими списками в Си и закрепили на практике.