Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах.»

на тему: «Динамические списки.»

Выполнили**:**

студенты группы 21ВВ4

Федоренко Вероника

Роганов Данила

Проверили:

Юрова О.В,

Акифьев И.В.

Пенза 2022

**Название**

Динамические списки.

**Цель работы**

Научиться реализовывать очереди и стеки, функции для работы с ними.

**Лабораторное 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).

Листинг

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <locale.h>

#include <Windows.h>

struct node

{

int priority;

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;

void addElementToStack(node\* element) {

if (head == NULL)

{

head = element;

return;

}

else

{

element->next = head;

head = element;

}

}

node\* getElementFromStack() {

node\* item;

if (head == NULL)

{

return NULL;

}

else

{

item = head;

head = head->next;

return item;

}

}

void addElementToQueueWithPriority(node\* element) { //приоритетная очередь

node\* item = head;

node\* previuseElem = NULL;

while (item != NULL) //вставка между элементами

{

if (item->priority < element->priority) {

if (previuseElem == NULL) {

element->next = item;

head = element;

}

else {

struct node\* temp = previuseElem->next;

previuseElem->next = element;

element->next = item;

}

return;

}

else {

previuseElem = item;

item = item->next;

}

}

if (head == NULL) {

head = element;

}

else {

previuseElem->next = element;

}

}

// Функции добавления элемента, просмотра списка

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

char find\_el[256];

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

struct node\* createStruct(); // функция создания элемента

node\* createStruct()

{

struct node\* p = NULL;

char s[256];

int prioprity = 0;

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);

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

scanf("%d", &prioprity);

p->priority = prioprity;

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 = createStruct();

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 printQueue()

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc != NULL)

{

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("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 main() {

setlocale(LC\_ALL, "Rus");

int countElementsInQueue = 2;

printf("Работа с приоритетной очередью\n");

for (int i = 0; i < countElementsInQueue; i++)

{

node\* inputElement = createStruct();

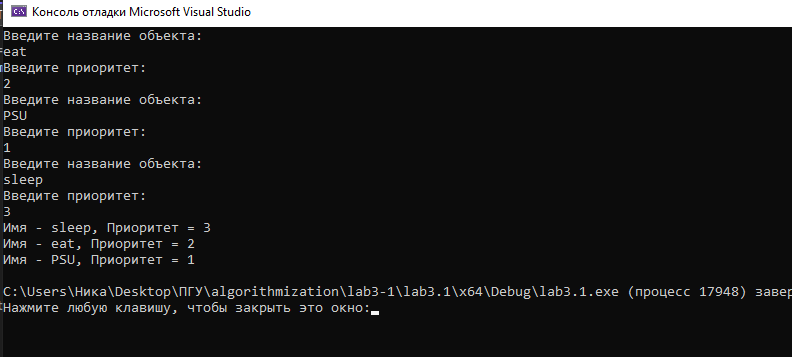
addElementToQueueWithPriority(inputElement);

}

printQueue();

}

Результат



**Задание 2:** На основе приведенного кода реализуйте структуру данных Очередь.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <locale.h>

#include <Windows.h>

struct node

{

int priority;

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;

void addElementToQueue(node\* element) { //очередь

node\* item = head;

if (head == NULL)

{

head = element;

return;

}

while (item->next != NULL)

{

item = item->next;

}

item->next = element;

}

node\* getElementFromQueue() {

node\* item;

if (head == NULL)

{

return NULL;

}

else

{

item = head;

head = head->next;

return item;

}

}

// Функции добавления элемента, просмотра списка

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

char find\_el[256];

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

struct node\* createStruct(); // функция создания элемента

node\* createStruct()

{

struct node\* p = NULL;

char s[256];

int prioprity = 0;

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);

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

scanf("%d", &prioprity);

p->priority = prioprity;

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 = createStruct();

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 printQueue()

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc != NULL)

{

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("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 main() {

setlocale(LC\_ALL, "Rus");

int countElementsInQueue = 2;

head = NULL;

printf("Работа с обычыной очередью\n");

for (int i = 0; i < countElementsInQueue; i++)

{

node\* inputElement = createStruct();

addElementToQueue(inputElement);

}

node\* elementToPrint = head;

while (elementToPrint != NULL)

{

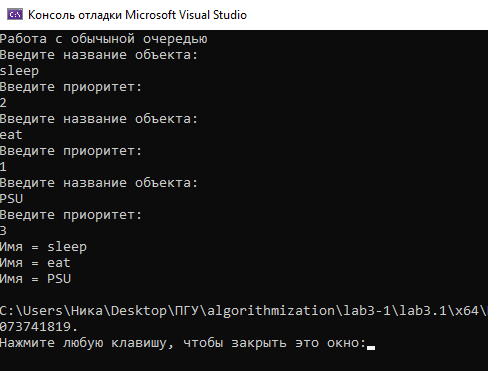
elementToPrint = getElementFromQueue();

printf("Имя = %s \n", elementToPrint->inf);

}

}

Результат



**Задание 3**: На основе приведенного кода реализуйте структуру данных Стек.

Листинг

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <locale.h>

#include <Windows.h>

struct node

{

int priority;

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;

void addElementToStack(node\* element) {

if (head == NULL)

{

head = element;

return;

}

else

{

element->next = head;

head = element;

}

}

node\* getElementFromStack() {

node\* item;

if (head == NULL)

{

return NULL;

}

else

{

item = head;

head = head->next;

return item;

}

}

// Функции добавления элемента, просмотра списка

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

char find\_el[256];

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

struct node\* createStruct(); // функция создания элемента

node\* createStruct()

{

struct node\* p = NULL;

char s[256];

int prioprity = 0;

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);

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

scanf("%d", &prioprity);

p->priority = prioprity;

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 = createStruct();

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 printQueue()

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc != NULL)

{

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("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 main() {

setlocale(LC\_ALL, "Rus");

int countElementsInQueue = 3;

node\* elementToPrint = head;

printf("Работа с стеком\n");

head = NULL;

for (int i = 0; i < countElementsInQueue; i++)

{

node\* inputElement = createStruct();

addElementToStack(inputElement);

}

elementToPrint = head;

while (elementToPrint != NULL)

{

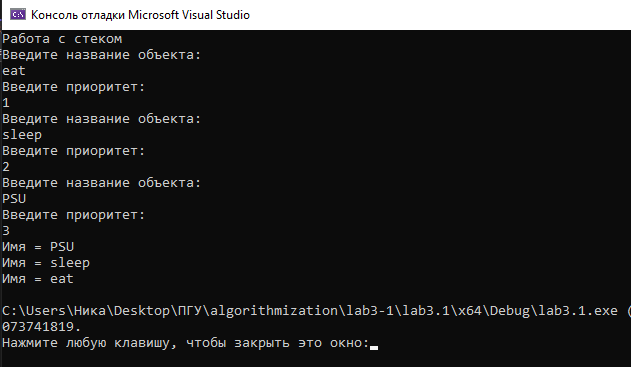
elementToPrint = getElementFromStack();

printf("Имя = %s \n", elementToPrint->inf);

}

}

Результат



Вывод: в ходе выполнения лабораторной работы были разработаны программы, реализующие работу c очередью и стеком.