Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу “Логика и основа алгоритмизации в ИЗ”

на тему “Динамические списки”

Выполнили студенты группы 21ВВ1.3:

Лукьянова Дарья

Давкин Максим

Михальцова Полина

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2022

**Название:**

Динамические списки.

**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:**

1. Приоритетная очередь:

#include <iostream>

#include <time.h>

#include <windows.h>

#include <queue>

using namespace std;

struct pri\_queue

{

int num;

int priority;

pri\_queue\* next = nullptr;

};

void insert(pri\_queue\*& main, pri\_queue\* new\_el) {

pri\_queue\* tmp = main;

pri\_queue\* left = main;

while (tmp != nullptr && new\_el->priority < tmp->priority)

{

left = tmp;

tmp = tmp->next;

}

if (tmp != left) {

left->next = new\_el;

}

else main = new\_el;

new\_el->next = tmp;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int count;

cout << "Введите число элементов: ";

cin >> count;

if (count <= 1) count = 2;

pri\_queue\* head = nullptr;

while(count){

pri\_queue\* tmp = new pri\_queue;

cout << "\nВведите данные: ";

cin >> tmp->num;

cout << "Введите приоритет: ";

cin >> tmp->priority;

insert(head, tmp);

count--;

}

pri\_queue\* tmp = head;

while (tmp != nullptr) {

cout << tmp->num << endl;

tmp = tmp->next;

}

return 0;

}

1. Очередь:

// '+' - добавить элемент

// '-' - удалить элемент

// 'f' - найти элемент

// '+' , 'print' - вывести очередь на экран

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <windows.h>

#include <malloc.h>

#include <string.h>

struct node

{

char inf[256]; // полезная информация

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

};

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

/\* Функция просмотра всей очереди \*/

int review(struct node\*\* head) {

struct node\* gg = \*head;

if (gg == NULL) {

printf("Очередь пуста!\n");

return 0;

}

printf("\n");

while (gg) {

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

gg = gg->next;

}

printf("\n");

return 0;

}

/\* Функция создания элемента очереди \*/

struct node\* get\_struct()

{

struct node\* p = NULL;

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

char s[256];

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

scanf("%s", &s);

if (\*s == 0) {

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

return 0;

}

if (strcmp("print", s) == 0) {

review(&head);

return 0;

}

if (p != NULL) {

strcpy(p->inf, s); // устанавливаем данные в выделенном узле и возвращаем их

p->next = NULL;

return p;

}

else {

printf("\nHeap Overflow");

exit(1);

}

}

/\* Функция добавления элемента в очередь (в конец) \*/

void spstore()

{

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) // если список уже есть, то вставляем в конец

{

last->next = p;

last = p;

}

return;

}

/\* Функция поиска элемента очереди \*/

struct node\* find()

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста!\n");

return 0;

}

char name[20];

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

scanf("%s", &name);

while (struc)

{

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

{

printf("Элемент найден)\n");

return 0;

}

struc = struc->next;

}

printf("Элемент не найден(\n");

return 0;

}

/\* Функция удаления элемента очереди (первого) \*/

void end() {

if (head == NULL)

{

printf("Очередь пуста!\n");

exit(1);

}

struct node\* tmp = head; // принять к сведению данные верхнего узла

printf("Верхний удаляемый элемент: %s\n", head->inf);

head = head->next; // обновляем верхний указатель, чтобы он указывал на следующий узел

if (head == NULL) { // если список стал пустым

last = NULL;

}

free(tmp);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

while (true) {

int i = \_getch();

switch (i) {

case '+':

spstore();

break;

case '-':

end();

break;

case 'f':

find();

break;

default:

break;

}

}

return 0;

}

1. Стек:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <windows.h>

#include <malloc.h>

#include <string.h>

struct node

{

char inf[256]; // информация

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

};

/\* Функция просмотра всего стека \*/

int review(struct node\*\* top) {

struct node\* gg = \*top;

if (gg == NULL) {

printf("Стек пуст!\n");

return 0;

}

printf("\n");

while (gg) {

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

gg = gg->next;

}

printf("\n");

return 0;

}

/\* Функция для создания элемента стека \*/

struct node\* get\_struct(struct node\*\* top)

{

struct node\* p = NULL;

p = (node\*)malloc(sizeof(struct node)); // создаем новый узел

char s[256];

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

scanf("%s", &s);

if (\*s == 0) {

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

return 0;

}

if (strcmp("print", s) == 0) {

review(top);

return 0;

}

if (p != NULL) {

strcpy(p->inf, s);

p->next = \*top; // устанавливаем указатель next так, чтобы он указывал на текущий верхний узел списка

\*top = p; // обновляем верхний указатель

}

else {

printf("\nHeap Overflow");

exit(1);

}

}

/\* Функция поиска элемента стека \*/

struct node\* find(struct node\*\* top)

{

struct node\* struc = \*top;

if (top == NULL)

{

printf("Стек пуст!\n");

}

char name[20];

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

scanf("%s", &name);

while (struc)

{

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

{

printf("Элемент найден)\n");

return 0;

}

struc = struc->next;

}

printf("Элемент не найден(\n");

return 0;

}

/\* Функция удаления элемента стека (последнего) \*/

void pop(struct node\*\* top)

{

struct node\* node = \*top;

if (\*top == NULL)

{

printf("Стек пуст!\n");

exit(1);

}

printf("Верхний удаляемый элемент: %s\n", (\*top)->inf); // принять к сведению данные верхнего узла

node = \*top;

\*top = (\*top)->next; // обновляем верхний указатель, чтобы он указывал на следующий узел

free(node);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

struct node\* top = NULL;

while (true) {

int i = \_getch();

switch (i) {

case '+':

get\_struct(&top);

break;

case '-':

pop(&top);

break;

case 'f':

find(&top);

break;

default:

break;

}

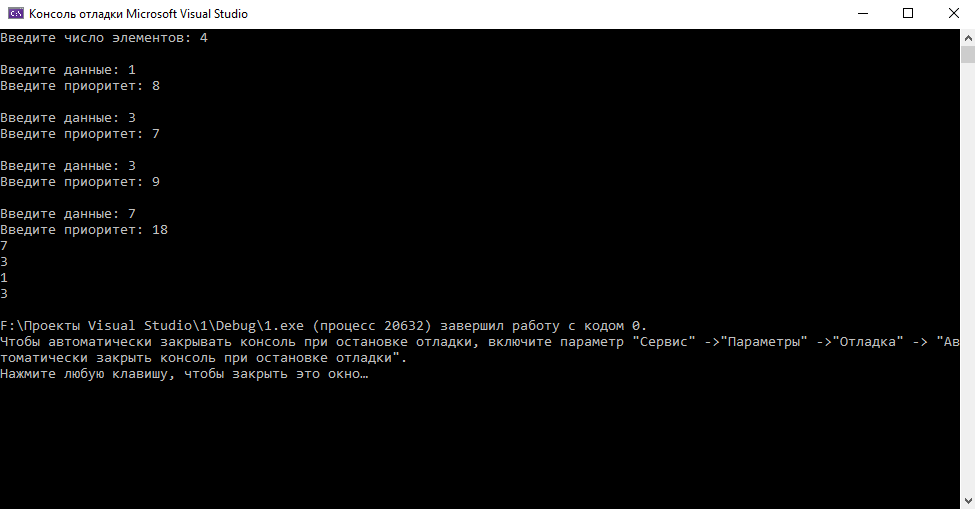
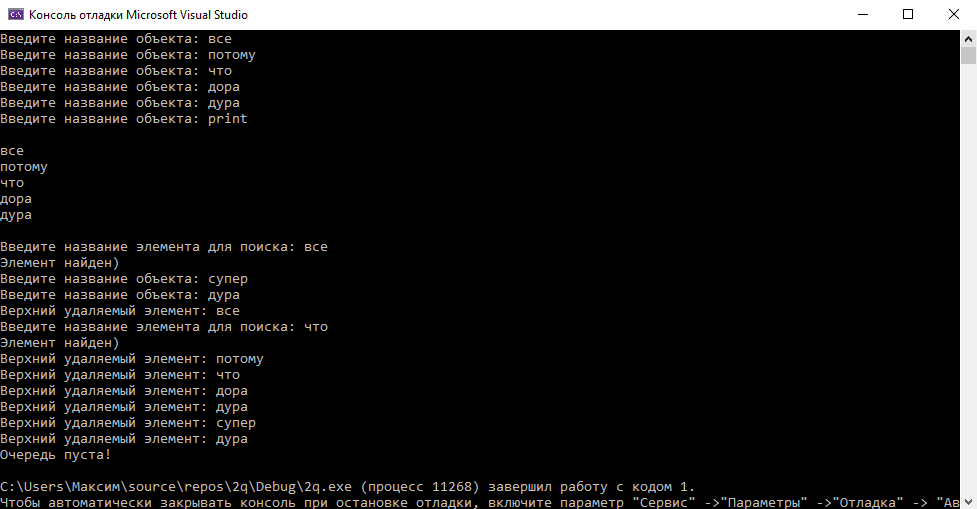
}

return 0;

}

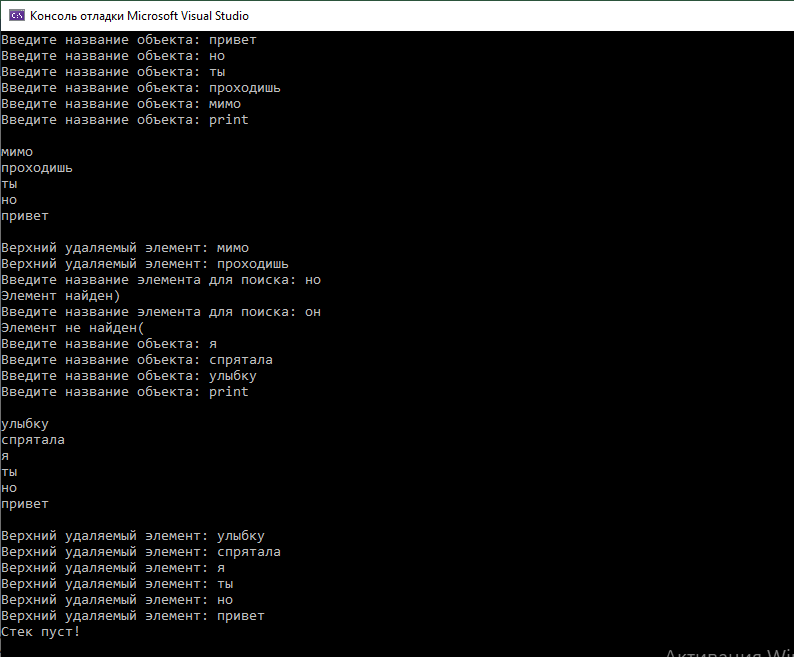
**Результаты выполнения программ:**

**Рисунок 2 – результат выполнения второй программы**



**Рисунок 1 – результат выполнения первой программы**

**Рисунок 3 – результат выполнения третьей программы**



**Вывод:** в ходе выполнения лабораторной работы была реализована приоритетная очередь, структуры данных Очередь и Стек.