Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»  
  
  
  
  
  
  
**ОТЧЁТ**

по лабораторной работе № 3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки»

Выполнил:

Студент групп 21ВВ1.3

Самохвалов Я.Д.

Приняли:

к.т.н., доцент Юрова О.А.

д.т.н., профессор, зав. каф. ВТ Митрохин М. А.

Пенза 2022

**Название:** Оценка времени выполнения программ

**Цель работы:** выполнить лабораторные указания 1-2 используя динамические списки.

**Лабораторные задания:**

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.

**Задание 1:**

**Листинг:**

PQueue.h:

#ifndef PQUEUE\_H

#define PQUEUE\_H 1

struct PQUEUE;

typedef struct PQUEUE PQueue;

/\* constructor \*/

extern PQueue\* pq\_new(void);

/\* add data with priority, less number of priority goes first \*/

extern void pq\_push(PQueue\* pq, void\* data, int priority);

/\* add copy of data with priority \*/

extern void pq\_push\_copy(PQueue\* pq, const void\* data, int priority, void\* (\*data\_copy\_func)(const void\*));

/\* return pointer to first elements data \*/

extern void\* pq\_front\_data(PQueue\* pq);

/\* return first elements priority \*/

extern int pq\_front\_priority(PQueue\* pq);

/\* remove first element from queue \*/

extern void pq\_shift(PQueue\* pq);

/\* remove first element destructing data \*/

extern void pq\_shift\_data(PQueue\* pq, void (\*data\_destructor\_func)(void\*));

/\* tests if queue is empty \*/

extern int pq\_is\_empty(PQueue\* pq);

/\* return size of queue \*/

extern unsigned pq\_size(PQueue\* pq);

/\* destructor \*/

extern void pq\_free(PQueue\* pq);

/\* destructor with data destructor function \*/

extern void pq\_free\_data(PQueue\* pq, void (\*data\_destructor\_func)(void\*));

#endif /\* PQUEUE\_H \*/

PQueue.cpp:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <assert.h>

#include <stdio.h>

#include <conio.h>

#include "PQueue.h"

int \_stateMenu;

void Menu()

{

printf("Enter action:\n1 - Print queue\n2 - Find priority in queue\n3 - Find data in queue\n4 - Pop first element with highest priority\n5 - Push new element with priority and data\n6 - Delete queue\n7 - Save queue in file\n0 - Exit\n");

\_stateMenu = \_getch() - 48;

}

typedef struct PQNODE {

void\* data;

int priority;

struct PQNODE\* prev;

struct PQNODE\* next;

} PQNode;

static PQNode\* pqn\_new(void\* data, int priority) {

PQNode\* node = (PQNode\*)malloc(sizeof(PQNode));

assert(node);

node->data = data;

node->priority = priority;

node->prev = NULL;

node->next = NULL;

return node;

}

struct PQUEUE {

PQNode\* head;

PQNode\* tail;

};

PQueue\* pq\_new(void) {

PQueue\* pq = (PQueue\*)malloc(sizeof(PQueue));

assert(pq);

pq->head = NULL;

pq->tail = NULL;

return pq;

}

void pq\_push(PQueue\* pq, void\* data, int priority) {

PQNode\* node = pqn\_new(data, priority);

if (pq\_is\_empty(pq)) {

pq->head = pq->tail = node;

}

else {

PQNode\* last = pq->tail;

while (last && last->priority > priority) {

last = last->prev;

}

while (last && last->priority == priority && last->data < data)

last = last->prev;

if (!last) {

node->next = pq->head;

pq->head->prev = node;

pq->head = node;

}

else if (last == pq->tail) {

node->prev = pq->tail;

pq->tail->next = node;

pq->tail = node;

}

else {

node->prev = last;

node->next = last->next;

last->next->prev = node;

last->next = node;

}

}

}

void pq\_push\_copy(PQueue\* pq, const void\* data, int priority, void\* (\*data\_copy\_func)(const void\*)) {

pq\_push(pq, data\_copy\_func(data), priority);

}

void pq\_findpr(PQueue\* pq, int priority) {

PQNode\* last = pq->head;

int c = 1;

while (last && last->priority <= priority)

{

if (last->priority == priority)

{

printf("%d %d %d\n", (int)last->priority, (int)last->data, c);

}

last = last->next;

c++;

}

}

void pq\_finddt(PQueue\* pq, void\* data) {

PQNode\* last = pq->head;

int c = 1;

while (last && last != pq->tail)

{

if (last->data == data)

{

printf("%d %d %d\n", (int)last->priority, (int)last->data, c);

}

last = last->next;

c++;

}

if (pq->tail->data == data)

{

printf("%d %d %d\n", (int)pq->tail->priority, (int)pq->tail->data, c);

}

}

void\* pq\_front\_data(PQueue\* pq) {

assert(!pq\_is\_empty(pq));

return pq->head->data;

}

int pq\_front\_priority(PQueue\* pq) {

assert(!pq\_is\_empty(pq));

return pq->head->priority;

}

void pq\_shift(PQueue\* pq) {

assert(!pq\_is\_empty(pq));

if (pq->head == pq->tail) {

free(pq->head);

pq->head = pq->tail = NULL;

}

else {

PQNode\* second = pq->head->next;

second->prev = NULL;

free(pq->head);

pq->head = second;

}

}

void pq\_shift\_data(PQueue\* pq, void (\*data\_destructor\_func)(void\*)) {

assert(!pq\_is\_empty(pq));

data\_destructor\_func(pq->head->data);

pq\_shift(pq);

}

int pq\_is\_empty(PQueue\* pq) {

return (pq->head == NULL);

}

unsigned pq\_size(PQueue\* pq) {

unsigned size = 0;

PQNode\* node;

for (node = pq->head; node; node = node->next)

++size;

return size;

}

void pq\_free(PQueue\* pq) {

while (!pq\_is\_empty(pq))

pq\_shift(pq);

}

void pq\_free\_data(PQueue\* pq, void (\*data\_destructor\_func)(void\*)) {

while (!pq\_is\_empty(pq))

pq\_shift\_data(pq, data\_destructor\_func);

}

void pq\_print(PQueue\* pq) {

PQNode\* last = pq->head;

while (last && last != pq->tail)

{

printf("%d %d\n", (int)last->priority, (int)last->data);

last = last->next;

}

printf("%d %d\n", (int)pq->tail->priority, (int)pq->tail->data);

}

void pq\_printf(PQueue\* pq, FILE\* f) {

PQNode\* last = pq->head;

while (last && last != pq->tail)

{

fprintf(f, "%d %d\n", (int)last->priority, (int)last->data);

last = last->next;

}

fprintf(f, "%d %d\n", (int)pq->tail->priority, (int)pq->tail->data);

}

int main() {

int number;

int mas;

char FILE\_NAME[50];

printf("Input file with queue: ");

scanf("%s", FILE\_NAME);

PQueue\* queue = pq\_new();

FILE\* fin = fopen(FILE\_NAME, "r");

if (!fin) {

fprintf(stderr, "Can't open input file!\n");

return 1;

}

while (fscanf(fin, "%d ", &number) == 1) {

fscanf(fin, "%d\n", &mas);

pq\_push(queue, (void\*)mas, number);

}

if (ferror(fin) | fclose(fin))

fprintf(stderr, "Error reading input file!\n");

system("pause");

system("cls");

Menu();

while (\_stateMenu != 0)

{

switch (\_stateMenu)

{

case 1:

system("cls");

if (pq\_size(queue) == 0)

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

pq\_print(queue);

system("pause");

system("cls");

Menu();

break;

case 2:

system("cls");

if (pq\_size(queue) == 0)

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("Enter priority to find: ");

scanf("%d", &number);

pq\_findpr(queue, number);

system("pause");

system("cls");

Menu();

break;

case 3:

system("cls");

if (pq\_size(queue) == 0)

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("Enter data to find: ");

scanf("%d", &mas);

pq\_finddt(queue, (void\*)mas);

system("pause");

system("cls");

Menu();

break;

case 4:

system("cls");

if (pq\_size(queue) == 0)

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("%d %d\n", pq\_front\_priority(queue), pq\_front\_data(queue));

pq\_shift(queue);

system("pause");

system("cls");

Menu();

break;

case 5:

system("cls");

printf("Enter priority and data: ");

scanf("%d %d", &number, &mas);

pq\_push(queue, (void\*)mas, number);

system("pause");

system("cls");

Menu();

break;

case 6:

system("cls");

if (pq\_size(queue) == 0)

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

pq\_free(queue);

system("pause");

system("cls");

Menu();

break;

case 7:

system("cls");

if (pq\_size(queue) == 0)

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("File name: ");

scanf("%s", FILE\_NAME);

fin = fopen(FILE\_NAME, "w");

pq\_printf(queue, fin);

printf("Queue saved\n");

fclose(fin);

system("pause");

system("cls");

Menu();

break;

case 0:

exit(EXIT\_SUCCESS);

default:

system("cls");

printf("Wrong number");

system("pause");

system("cls");

Menu();

break;

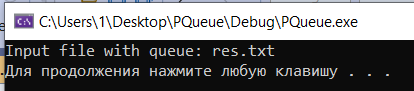
}

}

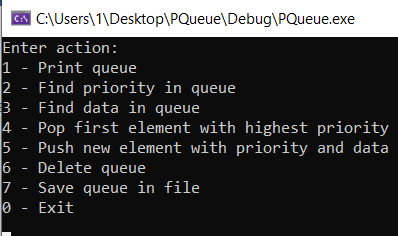
}

**Выводы результатов:**

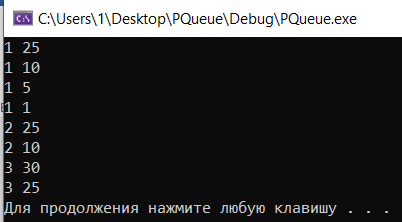
Ввод имени файла, в котором располагается очередь с приоритетом.

****

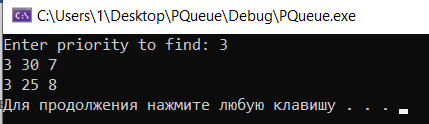
Вывод меню.



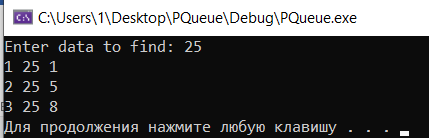
1 – Вывод очереди с приоритетом.



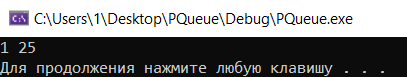
2 – Найдём приоритет в очереди.

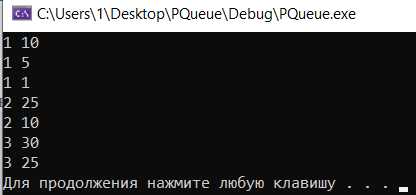


3 – Найдём данные в очереди.

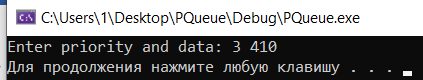


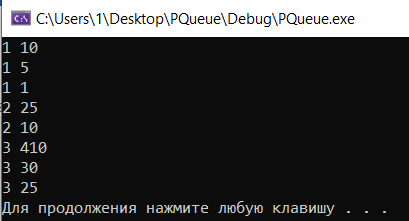
4 – Удалить первый элемент с наибольшим приоритетом.





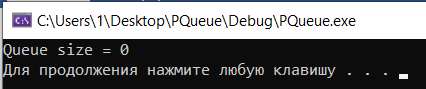
5 – Добавить новый элемент с приоритетом и данным.



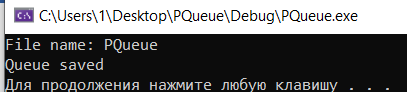


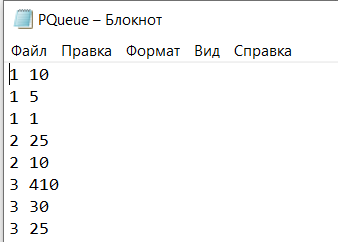
6 – Удалить всю очередь с приоритетом.





7 – Сохранить очередь с приоритетом в файл.



\

**Задание 2\*:**

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <locale.h>

int \_stateMenu;

void Menu()

{

printf("Enter action:\n1 - Print queue\n2 - Find data in queue\n3 - Pop first element in queue\n4 - Push new element in queue\n5 - Delete queue\n6 - Save queue in file\n0 - Exit\n");

\_stateMenu = \_getch() - 48;

}

struct list

{

int el;

list\* next;

};

struct queue

{

list\* beg, \* end;

};

void qpush(queue\* q, int iEl) //добавить в очередь

{

list\* tmp;

tmp = new list;

tmp->next = NULL;

tmp->el = iEl;

if (q->end != NULL)

q->end->next = tmp;

else

q->beg = tmp;

q->end = tmp;

}

int qpop(queue\* q, int\* iEl) // взять из очереди

{

if (q->beg == NULL) return 0;

list\* tmp;

tmp = q->beg;

\*iEl = tmp->el;

q->beg = tmp->next;

delete tmp;

if (q->beg == NULL) q->end = NULL;

return \*iEl;

}

queue\* CreateQueue() //создать очередь

{

queue\* q;

q = new queue;

q->beg = NULL;

q->end = NULL;

return q;

}

int isQueueEmpty(queue\* q) //проверка очереди на пустоту

{

if (q->beg == NULL) return 1;

return 0;

}

int ClearQueue(queue\* q) //очистка очереди

{

if (q->beg == NULL) return 0;

list\* tmp, \* t;

tmp = q->beg;

while (tmp->next != NULL)

{

t = tmp;

tmp = t->next;

delete t;

}

q->beg = NULL;

q->end = NULL;

return 1;

}

void finddt(queue\* q, int data)

{

queue\* tmp = CreateQueue();

int iEl, c = 1;

while (!isQueueEmpty(q))

{

qpop(q, &iEl);

if (iEl == data) printf("%d %d\n", iEl, c);

qpush(tmp, iEl);

c++;

}

while (!isQueueEmpty(tmp))

{

qpop(tmp, &iEl);

qpush(q, iEl);

}

}

void PrintQueueF(queue\* q, FILE\* F) //вывестии очередь на экран

{

queue\* tmp = CreateQueue();

int iEl;

while (!isQueueEmpty(q))

{

qpop(q, &iEl);

fprintf(F, "%d ", iEl);

qpush(tmp, iEl);

}

while (!isQueueEmpty(tmp))

{

qpop(tmp, &iEl);

qpush(q, iEl);

}

}

void PrintQueue(queue\* q) //вывестии очередь на экран

{

queue\* tmp = CreateQueue();

int iEl;

while (!isQueueEmpty(q))

{

qpop(q, &iEl);

printf("%d ", iEl);

qpush(tmp, iEl);

}

while (!isQueueEmpty(tmp))

{

qpop(tmp, &iEl);

qpush(q, iEl);

}

}

int main()

{

setlocale(LC\_ALL, "rus");

int number;

char FILE\_NAME[50];

printf("Input file with queue: ");

scanf("%s", FILE\_NAME);

queue\* q = CreateQueue();

FILE\* fin = fopen(FILE\_NAME, "r");

if (!fin) {

fprintf(stderr, "Can't open input file!\n");

return 1;

}

while (fscanf(fin, "%d ", &number) == 1)

qpush(q, number);

if (ferror(fin) | fclose(fin))

fprintf(stderr, "Error reading input file!\n");

system("pause");

system("cls");

Menu();

while (\_stateMenu != 0)

{

switch (\_stateMenu)

{

case 1:

system("cls");

if (isQueueEmpty(q))

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

PrintQueue(q);

printf("\n");

system("pause");

system("cls");

Menu();

break;

case 2:

system("cls");

if (isQueueEmpty(q))

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("Enter data to find: ");

scanf("%d", &number);

finddt(q, number);

system("pause");

system("cls");

Menu();

break;

case 3:

system("cls");

if (isQueueEmpty(q))

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("%d\n", qpop(q, &number));

system("pause");

system("cls");

Menu();

break;

case 4:

system("cls");

printf("Enter data to push: ");

scanf("%d", &number);

qpush(q, number);

system("pause");

system("cls");

Menu();

break;

case 5:

system("cls");

if (isQueueEmpty(q))

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

ClearQueue(q);

printf("Queue cleared\n");

system("pause");

system("cls");

Menu();

break;

case 6:

system("cls");

if (isQueueEmpty(q))

{

printf("Queue size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("File name: ");

scanf("%s", FILE\_NAME);

fin = fopen(FILE\_NAME, "w");

PrintQueueF(q, fin);

printf("Queue saved\n");

fclose(fin);

system("pause");

system("cls");

Menu();

break;

case 0:

exit(EXIT\_SUCCESS);

default:

system("cls");

printf("Wrong number");

system("pause");

system("cls");

Menu();

break;

}

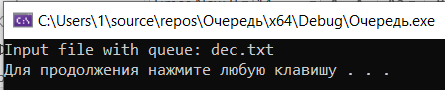
}

return 0;

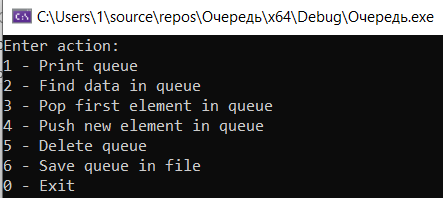
}

**Выводы результатов:**

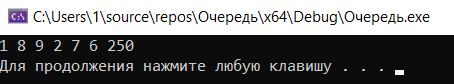
Ввод имени файла, в котором располагается очередь с приоритетом.

****

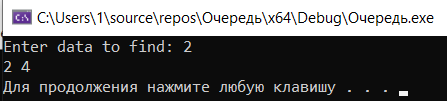
Вывод меню.



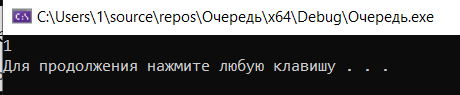
1 – Вывод очереди.

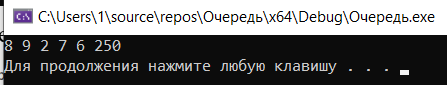
****

2 – Поиск данных в очереди.

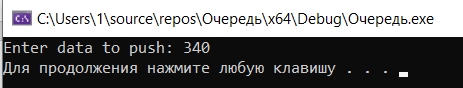


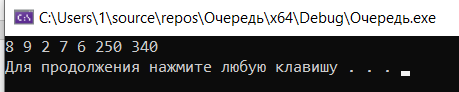
3 – Удаление первого элемента в очереди.



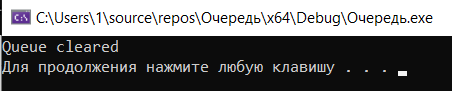


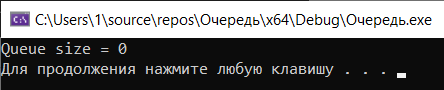
4 – Добавление нового элемента в очередь.



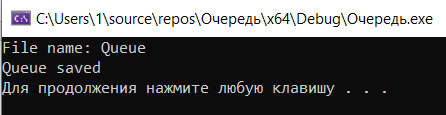


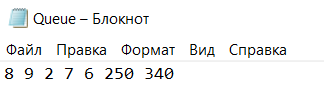
5 – Удаление всей очереди.





6 – Сохранение в файл.





**Задание 3\*:**

**Листинг:**

// Стэк.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

int \_stateMenu;

void Menu()

{

printf("Enter action:\n1 - Print stack\n2 - Find data in stack\n3 - Pop last element in stack\n4 - Push new element in stack\n5 - Delete stack\n6 - Save stack in file\n0 - Exit\n");

\_stateMenu = \_getch() - 48;

}

#define STACK\_OVERFLOW -100

#define STACK\_UNDERFLOW -101

#define OUT\_OF\_MEMORY -102

typedef int T;

typedef struct Node\_tag {

T value;

struct Node\_tag\* next;

} Node\_t;

void push(Node\_t\*\* head, T value)

{

Node\_t\* tmp = (Node\_t\*)malloc(sizeof(Node\_t));

if (tmp == NULL) {

exit(STACK\_OVERFLOW);

}

tmp->next = \*head;

tmp->value = value;

\*head = tmp;

}

T pop(Node\_t\*\* head)

{

Node\_t\* out;

T value;

if (\*head == NULL) {

exit(STACK\_UNDERFLOW);

}

out = \*head;

\*head = (\*head)->next;

value = out->value;

free(out);

return value;

}

T peek(const Node\_t\* head)

{

if (head == NULL) {

exit(STACK\_UNDERFLOW);

}

return head->value;

}

void printStack(const Node\_t\* head)

{

printf("stack > ");

while (head) {

printf("%d ", head->value);

head = head->next;

}

}

void printStackF(const Node\_t\* head, FILE\* F)

{

while (head) {

fprintf(F, "%d ", head->value);

head = head->next;

}

}

void finddt(const Node\_t\* head, int data)

{

int c = 1;

while (head)

{

if (head->value == data) printf("%d %d\n", head->value, c);

head = head->next;

c++;

}

}

size\_t getSize(const Node\_t\* head)

{

size\_t size = 0;

while (head) {

size++;

head = head->next;

}

return size;

}

int main() {

int number;

Node\_t\* head = NULL;

char FILE\_NAME[50];

printf("Input file with stack: ");

scanf("%s", FILE\_NAME);

FILE\* fin = fopen(FILE\_NAME, "r");

if (!fin) {

fprintf(stderr, "Can't open input file!\n");

return 1;

}

while (fscanf(fin, "%d ", &number) == 1)

push(&head, number);

if (ferror(fin) | fclose(fin))

fprintf(stderr, "Error reading input file!\n");

system("pause");

system("cls");

Menu();

while (\_stateMenu != 0)

{

switch (\_stateMenu)

{

case 1:

system("cls");

if (getSize(head) == 0)

{

printf("Stack size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printStack(head);

printf("\n");

system("pause");

system("cls");

Menu();

break;

case 2:

system("cls");

if (getSize(head) == 0)

{

printf("Stack size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("Enter data to find: ");

scanf("%d", &number);

finddt(head, number);

system("pause");

system("cls");

Menu();

break;

case 3:

system("cls");

if (getSize(head) == 0)

{

printf("Stack size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("%d\n", pop(&head));

system("pause");

system("cls");

Menu();

break;

case 4:

system("cls");

printf("Enter data to push: ");

scanf("%d", &number);

push(&head, number);

system("pause");

system("cls");

Menu();

break;

case 5:

system("cls");

if (getSize(head) == 0)

{

printf("Stack size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

while (getSize(head) >0) pop(&head);

printf("Stack cleared\n");

system("pause");

system("cls");

Menu();

break;

case 6:

system("cls");

if (getSize(head) == 0)

{

printf("Stack size = 0\n");

system("pause");

system("cls");

Menu();

break;

}

printf("File name: ");

scanf("%s", FILE\_NAME);

fin = fopen(FILE\_NAME, "w");

printStackF(head, fin);

printf("Stack saved\n");

fclose(fin);

system("pause");

system("cls");

Menu();

break;

case 0:

exit(EXIT\_SUCCESS);

default:

system("cls");

printf("Wrong number");

system("pause");

system("cls");

Menu();

break;

}

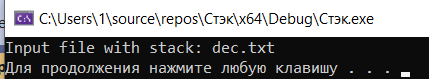
}

return 0;

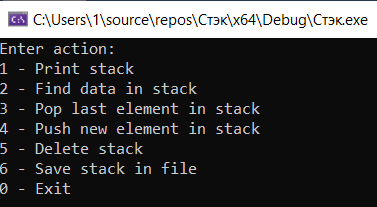
}

**Выводы результатов:**

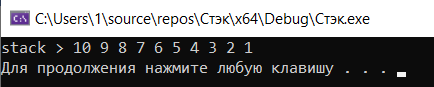
Ввод имени файла, в котором располагается очередь с приоритетом.

****

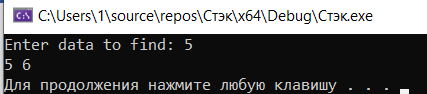
Вывод меню.



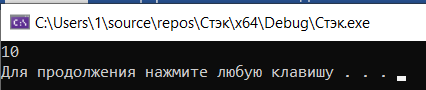
1 – Вывод стека.

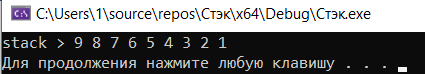


2 – Поиск данных в стека.

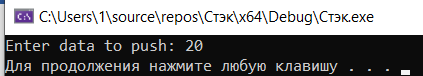


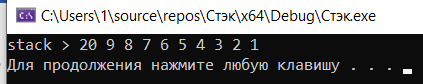
3 – Удаление первого элемента в стека.



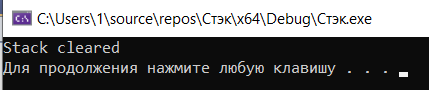


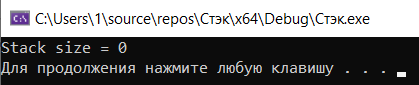
4 – Добавление нового элемента в стек.



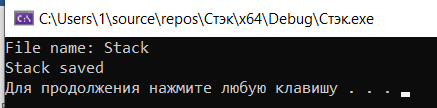


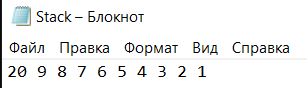
5 – Удаление всей стека.





6 – Сохранение в файл.





**Вывод:** мы выполнили лабораторные задания 1-3, используя двухсвязные (в очереди и в очереди с приоритетом) и односвязные (в стеке) списки. Программы работают так, как было задуманно.