Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Лабораторная работа № 11

"Информационные динамические структуры"

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

**Постановка задачи**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

Для каждого вариант разработать следующие функции:

1. Создание списка.

2. Добавление элемента в список (в соответствии со своим вариантом).

3. Удаление элемента из списка (в соответствии со своим вариантом).

4. Печать списка.

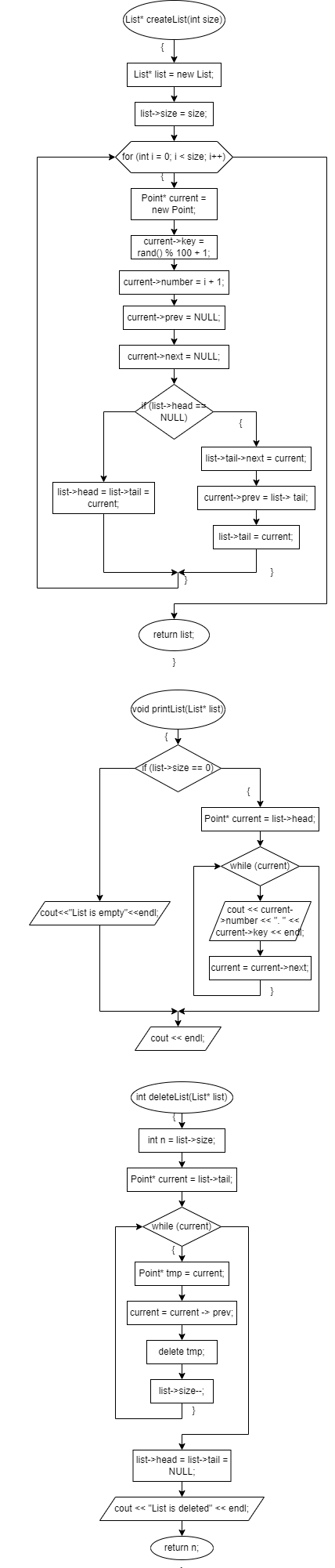
5. Запись списка в файл.

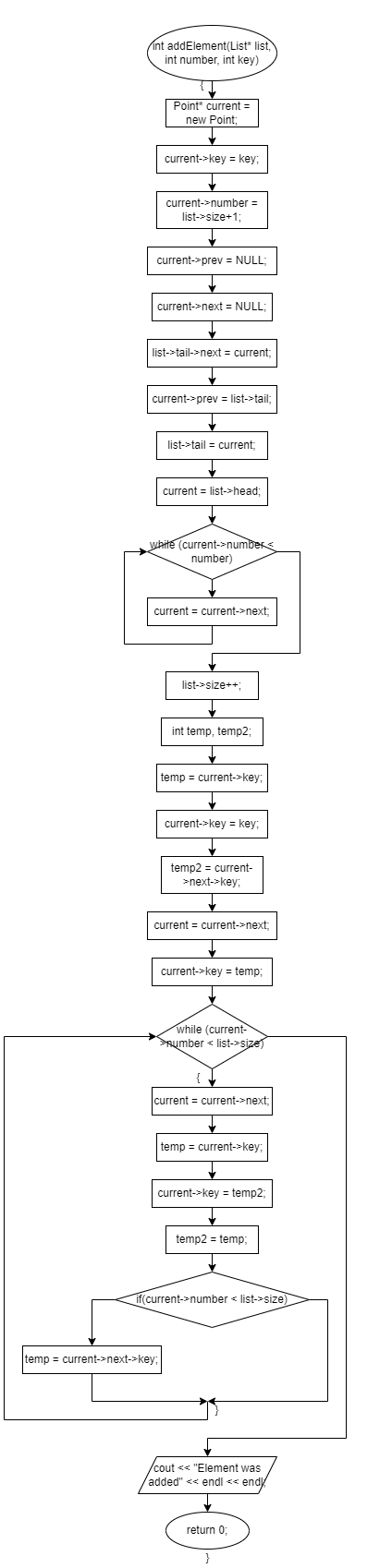
6. Уничтожение списка.

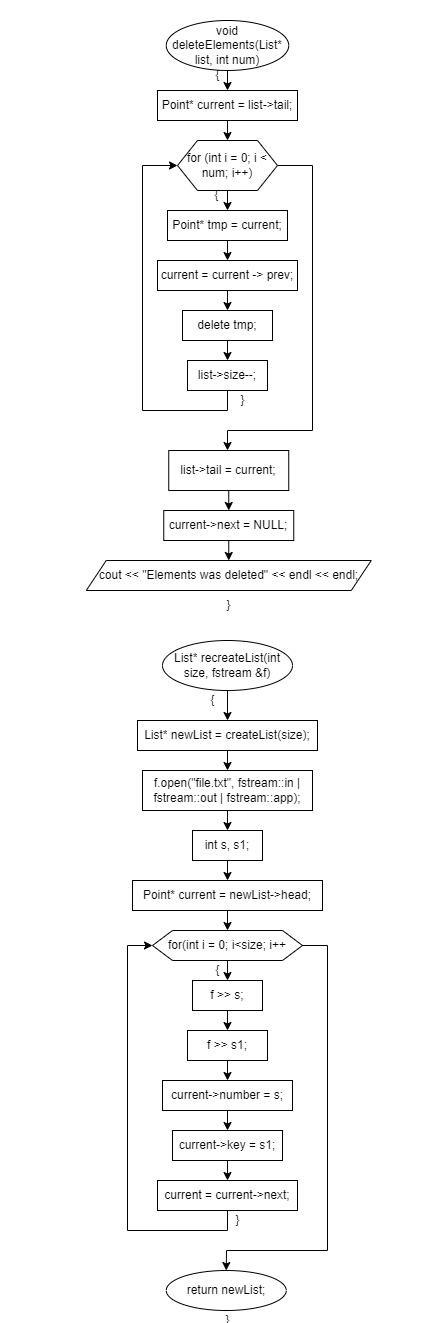
7. Восстановление списка из файла.

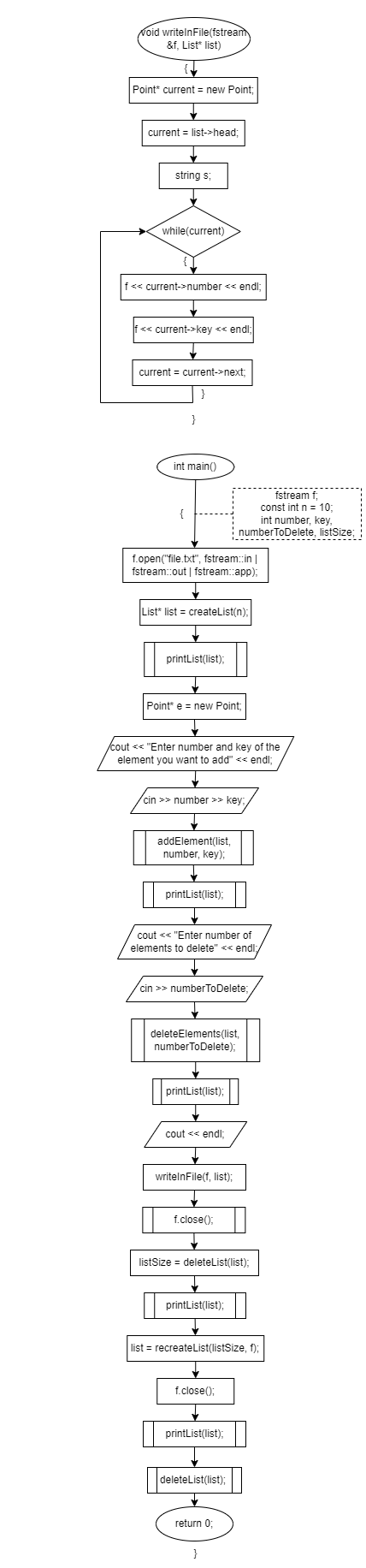
Вариант 10: Записи в линейном списке содержат ключевое поле типа int. Сформировать двунаправленный список. Добавить в него элемент с заданным номером, удалить К элементов из конца списка.

**Блок-схема**

****

****

****

****

**Код программы (Односвязный список)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <windows.h>

#include <fstream>

using namespace std;

const int MAXLINE = 255; // макс длина строки

struct Point

{

int number;

int key;

Point\* next;

};

struct List

{

size\_t size = 0;

Point\* head = NULL;

Point\* tail = NULL;

};

void printList(List\* list)

{

if (list->size == 0)

cout << "List is empty" << endl;

else

{

Point\* current = list->head;

for (int i = 0; i < list->size; i++)

{

cout << current->number << ". " << current->key << endl;

current = current->next;

}

}

cout << endl;

}

List\* createList(int size)

{

List\* list = new List;

list->size = size;

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

{

Point\* current = new Point;

current->key = rand() % 100 + 1;

current->number = i + 1;

current->next = NULL;

if (list->head == NULL)

list->head = list->tail = current;

else

{

list->tail->next = current;

list->tail = current;

}

}

return list;

}

List\* recreateList(int size, fstream& f)

{

List\* newList = createList(size);

f.open("file.txt", fstream::in | fstream::out | fstream::app);

int s, s1;

Point\* current = newList->head;

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

{

f >> s;

f >> s1;

current->number = s;

current->key = s1;

current = current->next;

}

return newList;

}

int deleteList(List\* list)

{

int n = list->size;

Point\* temp;

for (int i = 0; i < list->size; i++)

{

temp = list->head->next;

delete list->head;

list->head = temp;

}

list->size = 0;

list->tail = nullptr;

cout << "List is deleted" << endl;

return n;

}

int addElement(List\* list, int number, int key)

{

Point\* current = new Point;

current->key = key;

current->number = list->size + 1;

current->next = NULL;

list->tail->next = current;

list->tail = current;

current = list->head;

while (current->number < number)

current = current->next;

list->size++;

int temp, temp2;

temp = current->key;

current->key = key;

temp2 = current->next->key;

current = current->next;

current->key = temp;

while (current->number < list->size)

{

current = current->next;

temp = current->key;

current->key = temp2;

temp2 = temp;

if (current->number < list->size) temp = current->next->key;

}

cout << "Element was added" << endl << endl;

return 0;

}

void writeInFile(fstream& f, List\* list)

{

Point\* current = new Point;

current = list->head;

string s;

for (int i = 0; i < list->size; i++)

{

f << current->number << endl;

f << current->key << endl;

current = current->next;

}

}

void deleteElements(List\* list, int num)

{

Point\* current = list->head;

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

{

current = current->next;

}

list->tail = current;

current = current->next;

int tempSize = list->size - (list->size-num);

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

{

Point\* tmp = current;

current = current->next;

delete tmp;

list->size--;

}

cout << "Elements was deleted" << endl << endl;

}

int main()

{

fstream f;

f.open("file.txt", fstream::in | fstream::out | fstream::app);

const int n = 10;

int number, key, numberToDelete, listSize;

List\* list = createList(n);

printList(list);

Point\* e = new Point;

cout << "Enter number and key of the element you want to add" << endl;

cin >> number >> key;

addElement(list, number, key);

printList(list);

cout << "Enter number of elements to delete" << endl;

cin >> numberToDelete;

deleteElements(list, numberToDelete);

printList(list);

cout << endl;

writeInFile(f, list);

f.close();

listSize = deleteList(list);

printList(list);

list = recreateList(listSize, f);

f.close();

printList(list);

deleteList(list);

return 0;

}

**Код программы (Двусвязный список)**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct Point

{

int number;

Point\* prev;

int key;

Point\* next;

};

struct List

{

size\_t size = 0;

Point\* head = NULL;

Point\* tail = NULL;

};

List\* createList(int size)

{

List\* list = new List;

list->size = size;

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

{

Point\* current = new Point;

current->key = rand() % 100 + 1;

current->number = i + 1;

current->prev = NULL;

current->next = NULL;

if (list->head == NULL)

list->head = list->tail = current;

else

{

list->tail->next = current;

current->prev = list-> tail;

list->tail = current;

}

}

return list;

}

void printList(List\* list)

{

if (list->size == 0)

cout<<"List is empty"<<endl;

else

{

Point\* current = list->head;

while (current)

{

cout << current->number << ". " << current->key << endl;

current = current->next;

}

}

cout << endl;

}

int deleteList(List\* list)

{

int n = list->size;

Point\* current = list->tail;

while (current)

{

Point\* tmp = current;

current = current -> prev;

delete tmp;

list->size--;

}

list->head = list->tail = NULL;

cout << "List is deleted" << endl;

return n;

}

int addElement(List\* list, int number, int key)

{

Point\* current = new Point;

current->key = key;

current->number = list->size+1;

current->prev = NULL;

current->next = NULL;

list->tail->next = current;

current->prev = list->tail;

list->tail = current;

current = list->head;

while (current->number < number)

current = current->next;

list->size++;

int temp, temp2;

temp = current->key;

current->key = key;

temp2 = current->next->key;

current = current->next;

current->key = temp;

while (current->number < list->size)

{

current = current->next;

temp = current->key;

current->key = temp2;

temp2 = temp;

if(current->number < list->size) temp = current->next->key;

}

cout << "Element was added" << endl << endl;

return 0;

}

void deleteElements(List\* list, int num)

{

Point\* current = list->tail;

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

{

Point\* tmp = current;

current = current->prev;

delete tmp;

list->size--;

}

list->tail = current;

current->next = NULL;

cout << "Elements was deleted" << endl << endl;

}

List\* recreateList(int size, fstream &f)

{

List\* newList = createList(size);

f.open("file.txt", fstream::in | fstream::out | fstream::app);

int s, s1;

Point\* current = newList->head;

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

{

f >> s;

f >> s1;

current->number = s;

current->key = s1;

current = current->next;

}

return newList;

}

void writeInFile(fstream &f, List\* list)

{

Point\* current = new Point;

current = list->head;

string s;

while(current)

{

f << current->number << endl;

f << current->key << endl;

current = current->next;

}

}

int main()

{

fstream f;

f.open("file.txt", fstream::in | fstream::out | fstream::app);

const int n = 10;

int number, key, numberToDelete, listSize;

List\* list = createList(n);

printList(list);

Point\* e = new Point;

cout << "Enter number and key of the element you want to add" << endl;

cin >> number >> key;

addElement(list, number, key);

printList(list);

cout << "Enter number of elements to delete" << endl;

cin >> numberToDelete;

deleteElements(list, numberToDelete);

printList(list);

cout << endl;

writeInFile(f, list);

f.close();

listSize = deleteList(list);

printList(list);

list = recreateList(listSize, f);

f.close();

printList(list);

deleteList(list);

return 0;

}

**Код программы (Стек)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

using namespace std;

const int MAXLINE = 255;

struct Point

{

int key;

Point\* next = nullptr;

};

struct Stack

{

Point\* head = nullptr;

size\_t size = 0;

};

bool stack\_is\_empty(Stack& stack)

{

return stack.head == nullptr;

}

void output(Point\* obj)

{

if (obj == nullptr) return;

output(obj->next);

cout << obj->key << endl;

}

void printStack(Stack& stack)

{

if (stack.head == nullptr) cout << "Стек пуст" << endl;

else output(stack.head);

}

void push(Stack& stack, int key)

{

stack.size++;

Point\* obj = new Point;

obj->key=key;

obj->next = stack.head;

stack.head = obj;

}

void pop(Stack& stack)

{

if (stack.head != nullptr)

{

Point\* temp = stack.head;

stack.head = stack.head->next;

delete temp;

stack.size--;

}

}

Stack\* createStack(int size)

{

Stack\* stack = new Stack;

if (size > 0 && stack->head == nullptr)

{

int key;

cout << "Введите ключи " << size << " элементов стека: " << endl;

for (int i = 1; i <= size; i++)

{

cout << i << ". ";

cin >> key;

push(\*stack, key);

}

}

return stack;

}

void addElement(Stack& stack, int number, int k)

{

int new\_index = stack.size - number + 1;

if (new\_index < 0 || stack\_is\_empty(stack)) cout << "Введен неверный индекс/стек пуст" << endl;

else

{

int key;

Stack\* stack2 = new Stack;

for (int i = 1; i <= new\_index; i++)

{

push(\*stack2, stack.head->key);

pop(stack);

}

cout << "Введите " << k << " элементов: " << endl;

cin.get();

for (int i = 1; i <= k; i++)

{

cin >> key;

push(stack, key);

}

for (int i = 0; i < new\_index; i++)

{

push(stack, stack2->head->key);

pop(\*stack2);

}

}

}

void deleteElement(Stack& stack, int key)

{

int stack\_size = stack.size;

Stack\* stack2 = createStack(0);

for (int i = 1; i <= stack\_size; i++)

{

if (stack.head->key != key)

{

push(\*stack2, stack.head->key);

}

pop(stack);

}

cout << endl;

int stack\_size2 = stack2->size;

for (int i = 1; i <= stack\_size2; i++)

{

push(stack, stack2->head->key);

pop(\*stack2);

}

}

void deleteStack(Stack& stack)

{

Point\* temp;

while (stack.head != nullptr)

{

temp = stack.head->next;

delete stack.head;

stack.head = temp;

}

stack.size = 0;

cout << "Стек удален!" << endl;

}

void toFile\_obj(Point\* obj, FILE\* file)

{

if (obj == nullptr) return;

toFile\_obj(obj->next, file);

fprintf(file, "%d\n", obj->key);

}

void writeInFile(Stack& stack, const char\* filename)

{

FILE\* file;

if ((file = fopen(filename, "wb")) == NULL) //ошибка открытия файла

{

cout << "Ошибка открытия файла" << endl;

exit(2);

}

toFile\_obj(stack.head, file);

fclose(file);

}

void recreateStack(Stack& stack, const char\* filename)

{

FILE\* file;

int key;

if ((file = fopen(filename, "rb")) == NULL)

{

cout << "Ошибка открытия файла" << endl;

exit(2);

}

while (!feof(file))

{

fscanf(file, "%d\n", &key);

push(stack, key);

}

fclose(file);

}

int main()

{

setlocale(LC\_ALL, "ru");

const int n = 10;

int number, key, keyToDelete;

Stack\* stack = createStack(n);

printStack(\*stack);

Point\* e = new Point;

cout << "Введите позицию и количество элементов для добавления" << endl;

cin >> number >> key;

addElement(\*stack, number, key);

printStack(\*stack);

cout << "Введите ключ элемента для удаления" << endl;

cin >> keyToDelete;

deleteElement(\*stack, keyToDelete);

printStack(\*stack);

cout << endl;

writeInFile(\*stack, "file.txt");

deleteStack(\*stack);

printStack(\*stack);

recreateStack(\*stack, "file.txt");

printStack(\*stack);

deleteStack(\*stack);

return 0;

}

**Код программы (Очередь)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

using namespace std;

const int MAXLINE = 255;

struct Point

{

int key;

Point\* next = nullptr;

};

Point\* head = nullptr;

Point\* top = nullptr;

size\_t qSize = 0;

void push(int key)

{

if (qSize == 0)

{

head = new Point;

head->key = key;

top = head;

}

else

{

Point\* obj = new Point;

obj->key = key;

top->next = obj;

top = obj;

}

qSize++;

}

void pop()

{

Point\* temp = head;

head = head->next;

qSize--;

delete temp;

}

void createQueue(int size)

{

if (size > 0 && qSize == 0)

{

int key;

cout << "Введите ключи " << size << " элементов очереди: " << endl;

for (int i = 1; i <= size; i++)

{

cout << i << ". ";

cin >> key;

push(key);

}

}

}

void addElement(int index, int k)

{

if (!qSize == 0 && index >= 1 && index <= qSize && k > 0)

{

int i = 1;

while (i < index)

{

push(head->key);

pop();

i++;

}

int key;

cout << "Введите элементы для вставки: " << endl;

cin.get();

for (int j = 1; j <= k; j++)

{

cin >> key;

push(key);

}

for (i; i <= qSize - k; i++)

{

push(head->key);

pop();

}

}

else cout << "Очередь пуста или введен неверный индекс" << endl;

}

void output(Point\* obj)

{

cout << obj->key << endl;

if (obj == top) return;

output(obj->next);

}

void printQueue()

{

if (qSize == 0) cout << "Очередь пуста" << endl;

else output(head);

}

void deleteElement(int key)

{

if (qSize == 0) cout << "Очередь пуста, нечего удалять!" << endl;

int i = 1;

while (i <= qSize)

{

if (head->key == key)

{

pop();

}

else

{

push(head->key);

pop();

i++;

}

}

}

void deleteQueue()

{

while (head != nullptr)

{

pop();

}

cout << " Очередь удалена!" << endl;

}

void toFile\_obj(Point\* obj, FILE\* file)

{

if (obj == nullptr) return;

fprintf(file, "%d\n", obj->key);

toFile\_obj(obj->next, file);

}

void writeInFile(const char\* filename)

{

FILE\* file;

if ((file = fopen(filename, "wb")) == NULL)

{

cout << "Ошибка открытия файла" << endl;

exit(1);

}

toFile\_obj(head, file);

fclose(file);

}

void recreateQueue(const char\* filename)

{

FILE\* file;

int key;

if ((file = fopen(filename, "rb")) == NULL)

{

cout << "Ошибка открытия файла" << endl;

exit(2);

}

while (!feof(file))

{

fscanf(file, "%d\n", &key);

push(key);

}

fclose(file);

}

int main()

{

setlocale(LC\_ALL, "ru");

const int n = 10;

int number, key, keyToDelete;

createQueue(n);

printQueue();

Point\* e = new Point;

cout << "Введите позицию и количество элементов для добавления" << endl;

cin >> number >> key;

addElement(number, key);

printQueue();

cout << "Введите ключ элемента для удаления" << endl;

cin >> keyToDelete;

deleteElement(keyToDelete);

printQueue();

cout << endl;

writeInFile("file.txt");

deleteQueue();

printQueue();

recreateQueue("file.txt");

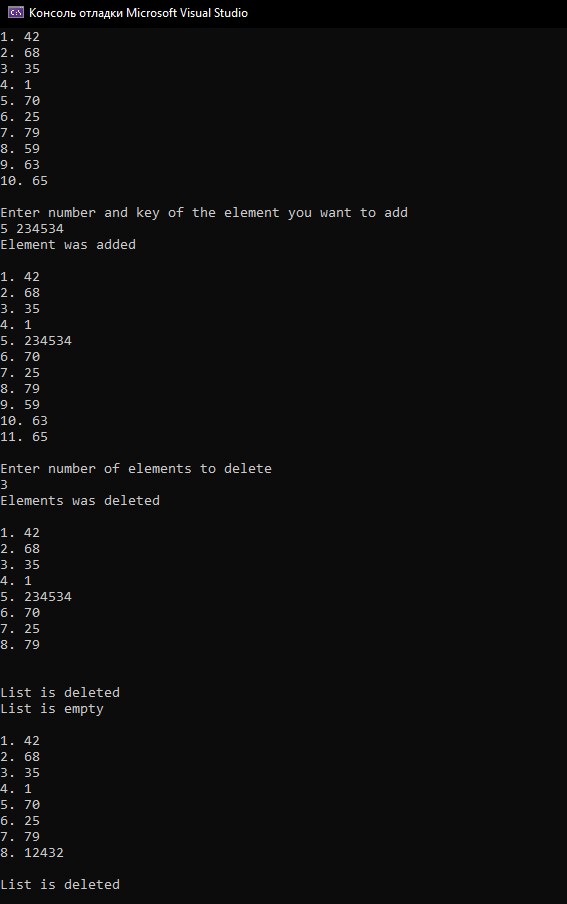
printQueue();

deleteQueue();

return 0;

}

**Результат работы программы**

****