Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №11.2**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Информационные и динамические структуры. Очереди

Вариант 24

Выполнила:

Студент группы ИВТ-20-2б

Ананина Арина Юрьевна

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2021**

1. Постановка задачи

Сформировать очередь. Распечатать полученную структуру. Выполнить обработку очереди:

* Удалить из очереди последний чётный элемент.

Распечатать полученный результат. Удалить соответствующую структуру из памяти.

2. Анализ задачи

1. В программе используется структура Queue, int, queue<int>.

Тип int используется для хранения данных, как параметр в цикле, для временного хранения размера динамической структуры.

Int size;

Тип int используется для хранения данных внутри структуры.

cin>> p->data;

Тип queue<int>используется для хранения стека через STL.

2. С данными типа int производятся следующие действия:

математические операции (инкремент в циклах с параметром).

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

С данными типа Queue производятся следующие действия: создание очереди, добавление элементов, вывод на экран.

Данные типа int также используются для хранения информации.

С данными типа queue<int> производятся те же действия, что и с типом Queue, но они реализованы с использованием STL.

3. Очередь реализована в программе с помощью структуры и STL.

4. Поля, используемые в структуре:

struct QueueItem

{

int data;

QueueItem\* prev;

};

5. Добавление элемента в очередь и выбор элемента из очереди организованы с помощью двух функций

void push(Queue\*\* queue, int element)

{

QueueItem\* newElement = newItem(element);

(\*queue)->back->prev = newElement;

(\*queue)->back = newElement;

if (!(\*queue)->front) {

(\*queue)->front = newElement;

}

}

int pop(Queue\*\* queue)

{

int el = (\*queue)->front->data;

QueueItem\* rem = (\*queue)->front;

(\*queue)->front = (\*queue)->front->prev;

delete rem;

return el;

}

Функция выборки элемента удаляет его из очереди и возвращает значение удалённого элемента.

Ввод Очереди организован с клавиатуры, с помощью функции. В качестве параметров в функцию передаётся значение переменной size, которая отвечает за кол-во элементов очереди.

Queue\* make\_queue(int size)

{

int tmp;

Queue\* queue = new Queue;

cout << "Введите элемент 1: ";

cin >> tmp;

QueueItem\* item = newItem(tmp);

queue->front = item;

queue->back = item;

for (int i = 2; i <= size; i++)

{

cout << "Введите элемент " << i << ": ";

cin >> tmp;

push(&queue, tmp);

}

return queue;

}

Вывод массива организован с помощью функции print. Так как стек читается с конца, то изначально в функции из стека все элементы переписываются в массив, а затем уже в правильном порядке с помощью цикла, который перебирает все элементы массива, выводятся на экран.

void print(Queue\*& queue)

{

if (!queue->front) {

cout << "Стек пустой" << endl;

return;

}

vector<int> copy = vector<int>();

while (queue->front)

{

copy.push\_back(pop(&queue));

}

for (int i = 0; i < copy.size(); i++) {

cout << copy[i] << " ";

push(&queue, copy[i]);

}

cout << endl;

}

6. Функция deleteEl удаляет из очереди последний чётный элемент.

С помощью цикла осуществляется проход по очереди с удалением элементов, элементы записываются в массив и запоминается индекс последнего четного элемента. С помощью цикла осуществляется проход по массиву, внутри цикла оператор условия проверяет, равен ли индекс элемента индексу последнего четного элемента, если он не равен, то элемент массива записывается в очередь.

while (queue->front)

{

item = pop(&queue);

copy.push\_back(item);

if (item % 2 == 0) {

tmp = copy.size() - 1;

}

}

for (int i = 0; i < copy.size(); i++) {

if (i != tmp) {

push(&queue, copy[i]);

}

}

3. Код программы

#include <iostream>

#include <queue>

using namespace std;

int a;

queue<int> MakeSTL(int size)

{

queue<int> res;

int tmp;

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

{

cout << "Введите элемент " << i + 1 << ": ";

cin >> tmp;

res.push(tmp);

}

return res;

}

void PrintSTL(queue<int> st)

{

if (st.empty())

{

cout << "Стек пустой" << endl;

return;

}

while (!st.empty())

{

cout << st.front() << " ";

st.pop();

}

cout << endl;

}

queue<int> DeleteSTL(queue<int> st)

{

int n = st.size();

queue<int> per;

int\* ar = new int[n];

int tmp = -1;

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

{

ar[i] = st.front();

st.pop();

if (ar[i] % 2 == 0) {

tmp = i;

}

}

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

{

if (i != tmp) {

per.push(ar[i]);

}

}

return per;

}

struct QueueItem

{

int data;

QueueItem\* prev;

};

struct Queue

{

QueueItem\* front;

QueueItem\* back;

};

QueueItem\* newItem(int element)

{

QueueItem\* newElement = new QueueItem;

newElement->data = element;

newElement->prev = 0;

return newElement;

}

void push(Queue\*\* queue, int element)

{

QueueItem\* newElement = newItem(element);

(\*queue)->back->prev = newElement;

(\*queue)->back = newElement;

if (!(\*queue)->front) {

(\*queue)->front = newElement;

}

}

int pop(Queue\*\* queue)

{

int el = (\*queue)->front->data;

QueueItem\* rem = (\*queue)->front;

(\*queue)->front = (\*queue)->front->prev;

delete rem;

return el;

}

Queue\* make\_queue(int size)

{

int tmp;

Queue\* queue = new Queue;

cout << "Введите элемент 1: ";

cin >> tmp;

QueueItem\* item = newItem(tmp);

queue->front = item;

queue->back = item;

for (int i = 2; i <= size; i++)

{

cout << "Введите элемент " << i << ": ";

cin >> tmp;

push(&queue, tmp);

}

return queue;

}

void print(Queue\*& queue)

{

if (!queue->front) {

cout << "Стек пустой" << endl;

return;

}

vector<int> copy = vector<int>();

while (queue->front)

{

copy.push\_back(pop(&queue));

}

for (int i = 0; i < copy.size(); i++) {

cout << copy[i] << " ";

push(&queue, copy[i]);

}

cout << endl;

}

void deleteEl(Queue\*& queue)

{

if (!queue->front)

{

cout << "Стек пустой" << endl;

return;

}

vector<int> copy = vector<int>();

int tmp = -1;

int item;

while (queue->front)

{

item = pop(&queue);

copy.push\_back(item);

if (item % 2 == 0) {

tmp = copy.size() - 1;

}

}

for (int i = 0; i < copy.size(); i++) {

if (i != tmp) {

push(&queue, copy[i]);

}

}

}

int main()

{

system("color F0");

setlocale(LC\_ALL, "RUS");

cout << "Очередь через STL библиотеку." << endl;

int size;

do

{

cout << "Введите кол-во элементов очереди: ";

cin >> size;

} while (size <= 0);

queue<int> st1 = MakeSTL(size);

PrintSTL(st1);

queue<int> st2 = DeleteSTL(st1);

PrintSTL(st2);

cout << "Очередь через структуру." << endl;

int size2;

do

{

cout << "Введите кол-во элементов очереди: ";

cin >> size2;

} while (size2 <= 0);

Queue\* st3 = make\_queue(size2);

print(st3);

deleteEl(st3);

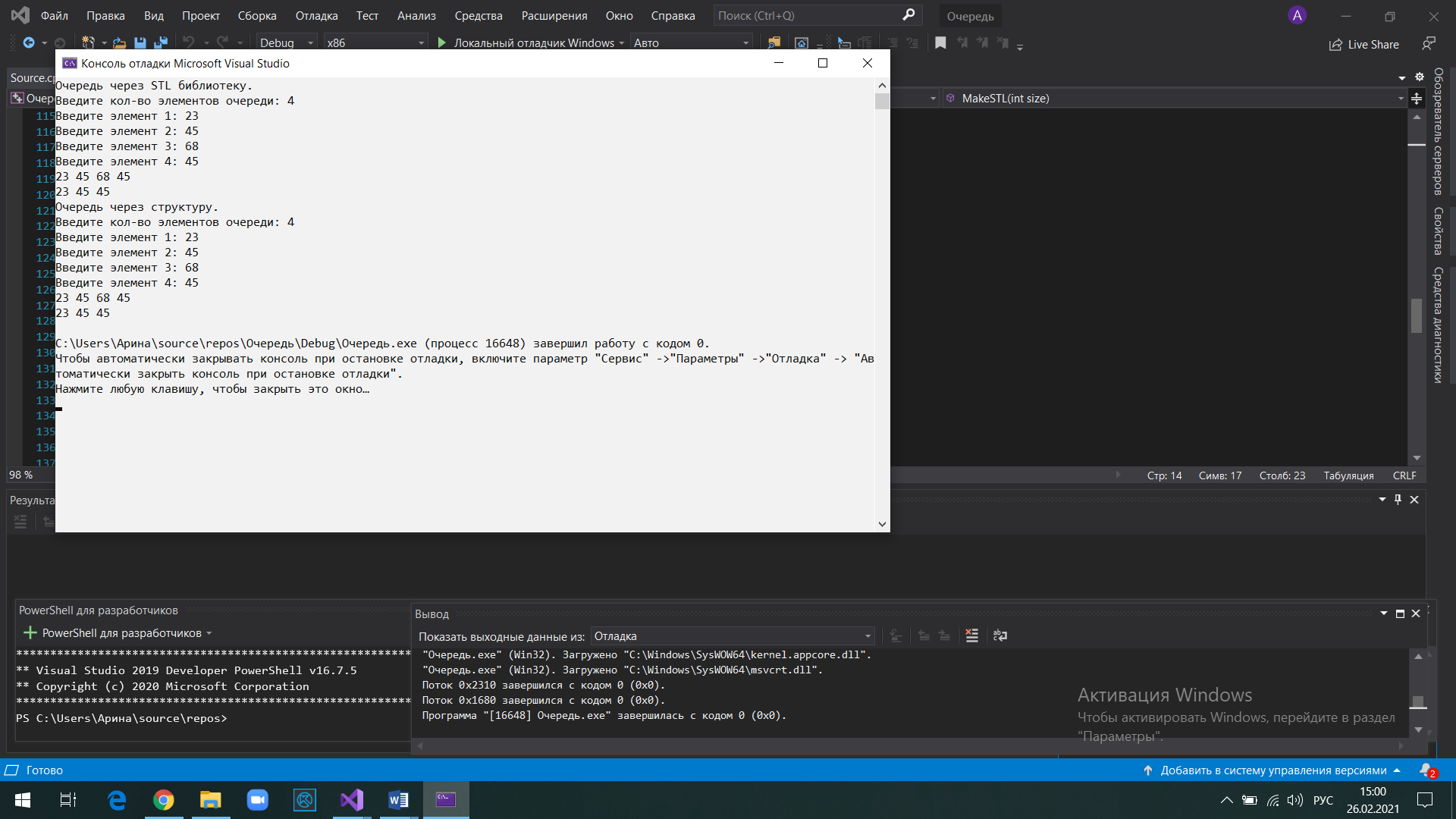
print(st3);

return 0;

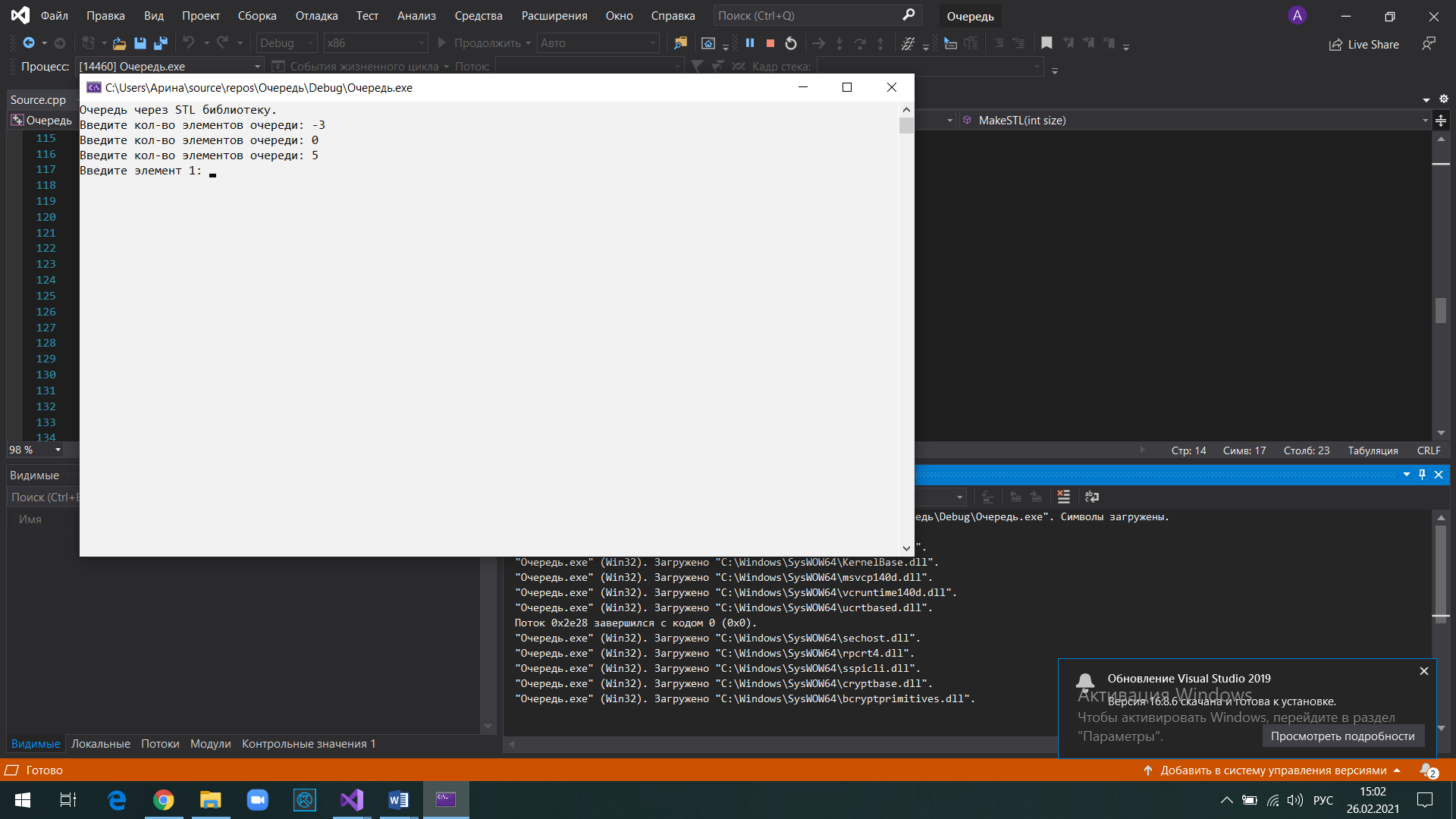
}

4. Выполнение программы.

Работа программы, когда все данные введены корректно.



Если пользователь вводит в качестве кол-ва элементов очереди число, меньшее нуля, то программа запрашивает это число до тех пор, пока не будет введено корректное значение.



5. Блок схема

