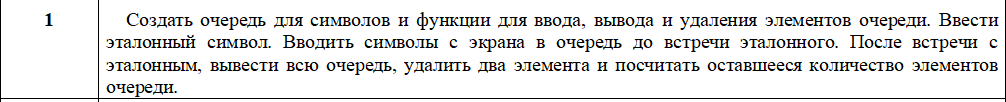
Банкузов Михаил 7 группа Лабораторная работа №8 Вариант 1



#ifndef QUEUE\_H

#define QUEUE\_H

class Queue {

public:

Queue(int size); // конструктор

~Queue(); // деструктор

bool isFull() const; // проверяет, заполнена ли очередь

bool isEmpty() const; // проверяет, пуста ли очередь

void enqueue(char c); // добавляет элемент в очередь

char dequeue(); // удаляет элемент из очереди

void printQueue() const; // выводит содержимое очереди в консоль

int front; // индекс первого элемента в очереди

int rear; // индекс последнего элемента в очереди

private:

char\* buffer; // массив символов

int capacity; // максимальный размер очереди

};

#endif

#include "queue.h"

#include <iostream>

// Конструктор класса Queue

Queue::Queue(int size) {

buffer = new char[size]; // Выделение памяти под буфер

front = 0; // Указатель на первый элемент очереди устанавливается на ноль

rear = -1; // Указатель на последний элемент очереди устанавливается на -1 (так как очередь пуста)

capacity = size; // Запоминаем максимальный размер очереди

}

// Деструктор класса Queue

Queue::~Queue() {

delete[] buffer; // Освобождение памяти

}

// Проверка, заполнена ли очередь до максимального размера

bool Queue::isFull() const {

return (rear == capacity - 1);

}

// Проверка, пуста ли очередь

bool Queue::isEmpty() const {

return (front > rear);

}

// Добавление элемента в очередь

void Queue::enqueue(char c) {

if (isFull()) {

std::cout << "Queue is full\n"; // Если очередь заполнена, выводим сообщение об ошибке

return; // И завершаем выполнение метода

}

rear++; // Увеличиваем указатель на последний элемент очереди

buffer[rear] = c; // Добавляем элемент в очередь

}

// Удаление элемента из очереди

char Queue::dequeue() {

if (isEmpty()) {

std::cout << "Queue is empty\n"; // Если очередь пуста, выводим сообщение об ошибке

return '\0'; // И возвращаем нулевой символ

}

char c = buffer[front]; // Получаем первый элемент из очереди

front++; // Увеличиваем указатель на первый элемент очереди

return c; // Возвращаем удаленный элемент

}

// Вывод очереди на экран

void Queue::printQueue() const {

std::cout << "Queue: ";

for (int i = front; i <= rear; i++) { // Проходимся по всей очереди

std::cout << buffer[i] << " "; // Выводим каждый элемент очереди на экран

}

std::cout << "\n"; // Переводим строку

}

#include "queue.h"

#include <iostream>

int main() {

int maxSize;

char target;

std::cout << "Enter maximum size of queue: ";

std::cin >> maxSize;

std::cout << "Enter target character: ";

std::cin >> target;

// Создаем очередь заданного размера

Queue q(maxSize);

char c;

bool foundTarget = false;

std::cout << "Enter characters:\n";

// Читаем символы из входного потока и добавляем их в очередь

while (std::cin >> c) {

// Если нашли целевой символ, останавливаем чтение и помечаем его как найденный

if (c == target) {

foundTarget = true;

break;

}

q.enqueue(c);

}

if (foundTarget) {

// Если нашли целевой символ, выводим текущее состояние очереди и удаляем два первых элемента

std::cout << "Queue before deletion:\n";

q.printQueue();

char c1 = q.dequeue();

char c2 = q.dequeue();

std::cout << "Deleted characters: " << c1 << " " << c2 << "\n";

// Выводим состояние очереди после удаления элементов

std::cout << "Queue after deletion:\n";

q.printQueue();

// Вычисляем количество оставшихся элементов в очереди

int remaining = q.isEmpty() ? 0 : (q.rear - q.front + 1);

std::cout << "Remaining elements in queue: " << remaining << "\n";

}

else {

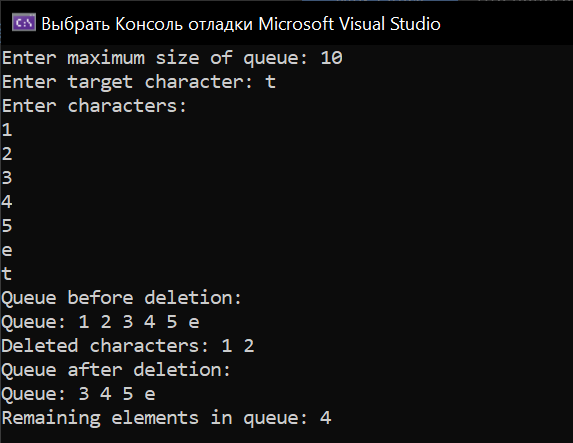
// Если целевой символ не найден, сообщаем об этом

std::cout << "Target character not found\n";

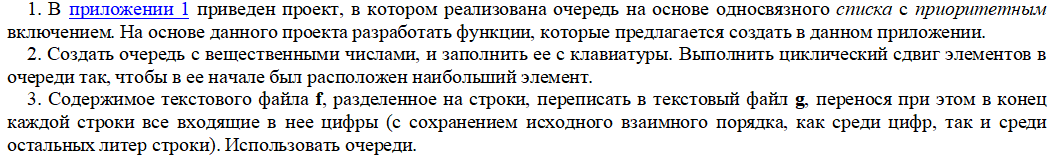
}

return 0;

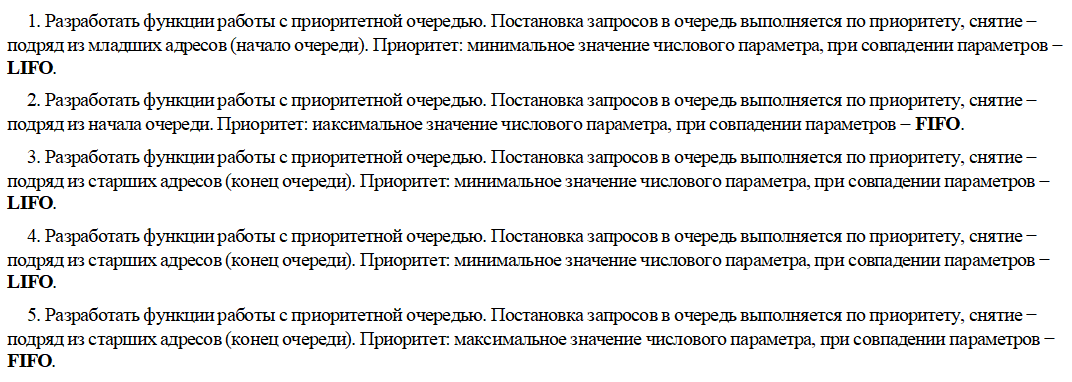
}



Дополнительные задания



Задание 1



Пункт 1

void insertToQueue(int x) {

Item\* p = new Item; //новый указатель

p->data = x;

p->next = NULL;

if (isNull()) {

head = tail = p;

}

else {

// Найдем место, куда нужно вставить элемент

Item\* prev = NULL;

Item\* cur = head;

while (cur != NULL && cur->data <= x) {

prev = cur;

cur = cur->next;

}

// Вставим элемент на найденное место

if (prev == NULL) { // Вставляем в начало очереди

p->next = head;

head = p;

}

else {

p->next = prev->next;

prev->next = p;

}

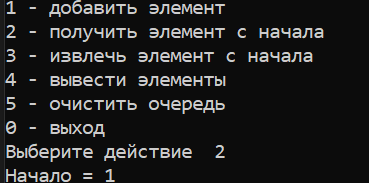
if (p->next == NULL) { // Вставляем в конец очереди

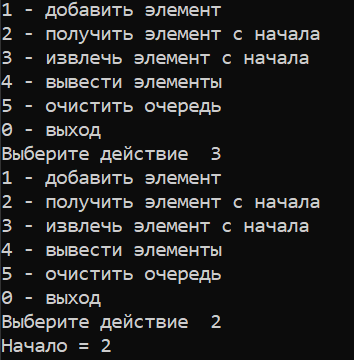
tail = p;

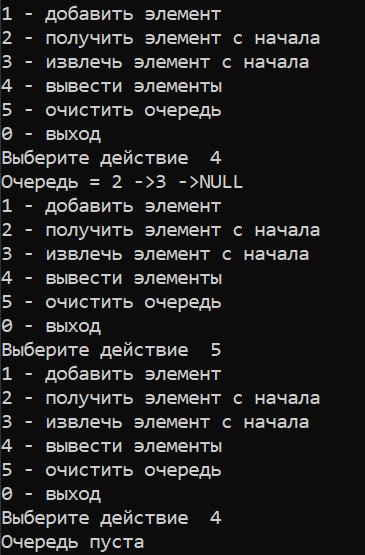
}

}

}







Пункт 2

void insertToQueue(int x) {

Item\* p = new Item;

p->data = x;

p->next = NULL;

if (isNull()) {

head = tail = p;

}

else {

// Найдем место, куда нужно вставить элемент

Item\* prev = NULL;

Item\* cur = head;

while (cur != NULL && cur->data > x) {

prev = cur;

cur = cur->next;

}

// Вставим элемент на найденное место

if (prev == NULL) { // Вставляем в начало очереди

p->next = head;

head = p;

}

else {

p->next = prev->next;

prev->next = p;

}

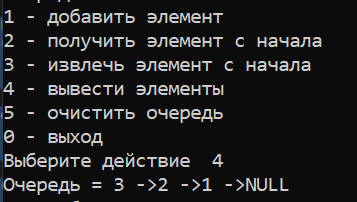
if (p->next == NULL) { // Вставляем в конец очереди

tail = p;

}

}

}



Пункт 3

void insertToQueue(int x, int p) {

Item\* pNew = new Item; //новый указатель

pNew->data = x;

pNew->next = NULL;

if (isNull()) {

head = tail = pNew;

}

else {

Item\* pPrev = NULL;

Item\* pCur = head;

// Ищем место для вставки элемента

while (pCur != NULL && (pCur->data < pNew->data || (pCur->data == pNew->data && pCur != tail))) {

pPrev = pCur;

pCur = pCur->next;

}

// Вставляем элемент на найденное место

if (pPrev == NULL) { // Вставляем в начало очереди

pNew->next = head;

head = pNew;

}

else {

pNew->next = pPrev->next;

pPrev->next = pNew;

}

if (pNew->next == NULL) { // Вставляем в конец очереди

tail = pNew;

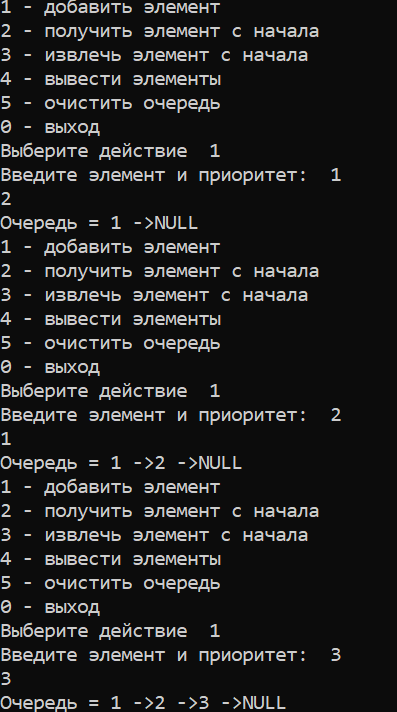
}

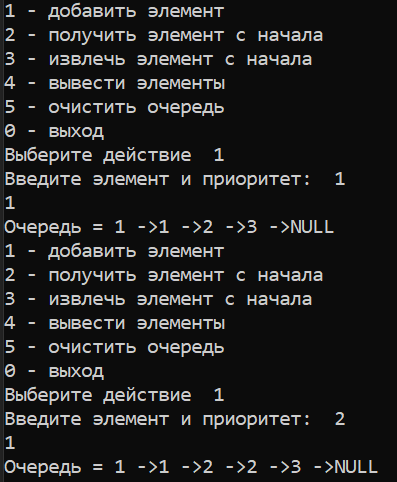
}

}

case 1: cout << "Введите элемент и приоритет: "; cin >> z >> i;

insertToQueue(z, i); printQueue(); break;





Пункт 5

void insertToQueue(int x, int p) {

Item\* pNew = new Item; //новый указатель

pNew->data = x;

pNew->next = NULL;

if (isNull()) {

head = tail = pNew;

}

else {

Item\* pPrev = NULL;

Item\* pCur = head;

// Ищем место для вставки элемента

while (pCur != NULL && (pCur->data < pNew->data || (pCur->data == pNew->data && pCur != tail))) {

pPrev = pCur;

pCur = pCur->next;

}

// Вставляем элемент на найденное место

if (pPrev == NULL) { // Вставляем в начало очереди

pNew->next = head;

head = pNew;

}

else {

pNew->next = pPrev->next;

pPrev->next = pNew;

}

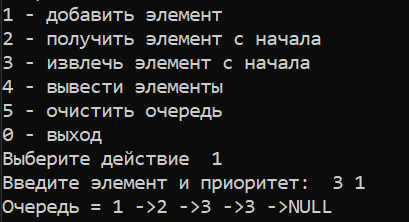
if (pNew->next == NULL || pNew->next->data != pNew->data) { // Вставляем в конец очереди или после элементов с таким же приоритетом

tail = pNew;

}

}

}



Задание 2

#include <iostream>

#include <queue>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

queue<double> q;

int n;

cout << "Введите количество элементов в очереди: ";

cin >> n;

// заполнение очереди с клавиатуры

double x;

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

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

cin >> x;

q.push(x);

}

// поиск максимального элемента

double max\_elem = q.front();

int max\_index = 0;

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

if (q.front() > max\_elem) {

max\_elem = q.front();

max\_index = i;

}

q.push(q.front());

q.pop();

}

// циклический сдвиг элементов в очереди

for (int i = 0; i < (max\_index + n - 1) % n; i++) {

q.push(q.front());

q.pop();

}

// вывод очереди

cout << "Новый порядок элементов в очереди: ";

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

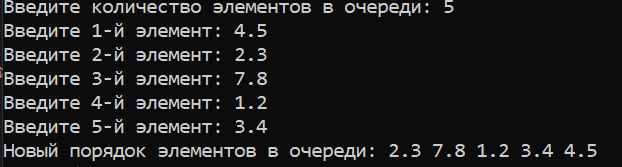
cout << q.front() << " ";

q.pop();

}

return 0;

}



Задание 3

#include <iostream>

#include <fstream>

#include <queue>

#include <cctype>

using namespace std;

int main() {

// Открываем файлы для чтения и записи

ifstream f("f.txt");

ofstream g("g.txt");

// Проверяем, что файлы открылись успешно

if (!f || !g) {

cout << "Ошибка при открытии файлов" << endl;

return 1;

}

// Создаем две очереди для хранения символов и цифр

queue<char> chars;

queue<char> digits;

// Читаем файл f посимвольно

char c;

while (f.get(c)) {

// Если символ - цифра, добавляем его в очередь digits

if (isdigit(c)) {

digits.push(c);

}

// Иначе, добавляем его в очередь chars

else {

chars.push(c);

}

// Если символ - перевод строки или конец файла, записываем содержимое очередей в файл g

if (c == '\n' || f.eof()) {

// Пока очередь chars не пуста, записываем ее первый элемент в файл g и удаляем его из очереди

while (!chars.empty()) {

g << chars.front();

chars.pop();

}

// Пока очередь digits не пуста, записываем ее первый элемент в файл g и удаляем его из очереди

while (!digits.empty()) {

g << digits.front();

digits.pop();

}

// Записываем перевод строки в файл g

g << '\n';

}

}

// Закрываем файлы

f.close();

g.close();

return 0;

}

