Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 8

По дисциплине «Основы алгоритмизации и программирования»

На тему «Полустатические структуры данных: очереди»

Выполнил:

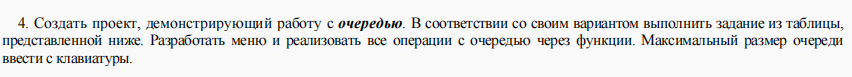
Студент 1 курса 6 группы

Романов Игорь Вячеславович

Преподаватель: асс. Андронова М.В.

2024, Минск

Задание 4



Вариант 12



**queue\_menu.cpp (модуль с меню)**

#include "queue.h"

int main() {

setlocale(LC\_ALL, "Rus");

Queue\* begin = NULL, \*end, \*element; //создание начала и конца очереди и ее элементов

element = new Queue; // Создаем новый элемент очереди

int choice, first, size; //Выбор пользователя, первый элемент очереди и ее размер

cout << "Введите размер очереди: ";

cin >> size;

while (size < 1) {

cout << "Введите значение большее, чем 0: ";

cin >> size;

}

cout << "Ввдедите первый элемент очереди: ";

cin >> first;

element->number = first; //Добавляем первый элемент очереди

element->next = NULL; //Указываем, что следующий элемент NULL

begin = end = element; //Указываем начало и конец очереди

do {

cout << "\nВыберите команду:" << endl;

cout << "1 - Добавление элементов в очередь (при повторении первого элемента выводится очередь ее размер" << endl;

cout << "2 - Вывод элементов очереди" << endl;

cout << "3 - Удаление очереди" << endl;

cout << "0 - Выход" << endl;

cin >> choice;

switch (choice) {

case 1: {

if (size > 1 ) {

createQueue(&begin, &end, size, first);

}

else {

cout << "Очередь не может быть больше заполнена!";

}

break;

}

case 2: {

displayQueue(begin);

break;

}

case 3: {

deleteQueue(&begin);

break;

}

case 0: {

return 0;

break;

}

default: {

cout << "Выбран неверный вариант!" << endl;

}

}

} while (choice != 0);

return 0;

}

**queue\_fnctions.cpp (модуль с функциями программы)**

#include "queue.h"

void createQueue(Queue\*\* begin, Queue\*\* end, int size, int first) {

int n, current\_size = 0; //переменная для ввода числа, текущего размера очереди

cout << "Введите следующие элементы очереди: " << endl;

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

Queue\* temp = new Queue; // Создание указателя на очередь

temp->next = NULL; //указываем, что следующий элемент пустой

cin >> n; // Вводим число с клавиатуры

temp->number = n; // Добавляем значения числа в очередь

(\*end)->next = temp; //Указатель next последнего элемента очереди устанавливается на новый элемент

\*end = temp; //конец очереди указывает на элемент temp

current\_size++; //Увеличиваем размер текущей очереди

if (n == first) {

displayQueueAndSize((\*begin), current\_size);

}

}

cout << "Очередь успешно создана" << endl;

}

void displayQueue(Queue\* begin) {

Queue\* temp = begin; // Создаем временный указатель temp для вывода элементов очереди

if (temp == NULL) {

cout << "Очередь пуста!" << endl;

}

else {

cout << "Элементы очереди: ";

while (temp != NULL) { //Выводим элементы до конца очереди

cout << temp->number << " ";

temp = temp->next; //Перемещаем указатель на следующий элемент

}

cout << endl;

delete temp; //Удаляем temp по окончании работы с очередью

}

}

void deleteQueue(Queue\*\* begin) {

if (begin == NULL) {

cout << "Очередь пуста!" << endl;

}

else {

Queue \* temp = new Queue; // Создаем временный указатель temp для удаления очереди

while (\*begin != NULL) { //Удаляем пока начало очереди не достигнет NULL

temp = \*begin;

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

delete temp;

}

cout << "Очередь успешно удалена!" << endl;

}

}

void displayQueueAndSize(Queue\* begin, int size) {

displayQueue(begin);

cout << "Размер очереди: " << size << endl;

}

**queue.h (заголовочный файл)**

#include <iostream>

using namespace std;

struct Queue { // создаем очередь

int number;

Queue\* next;

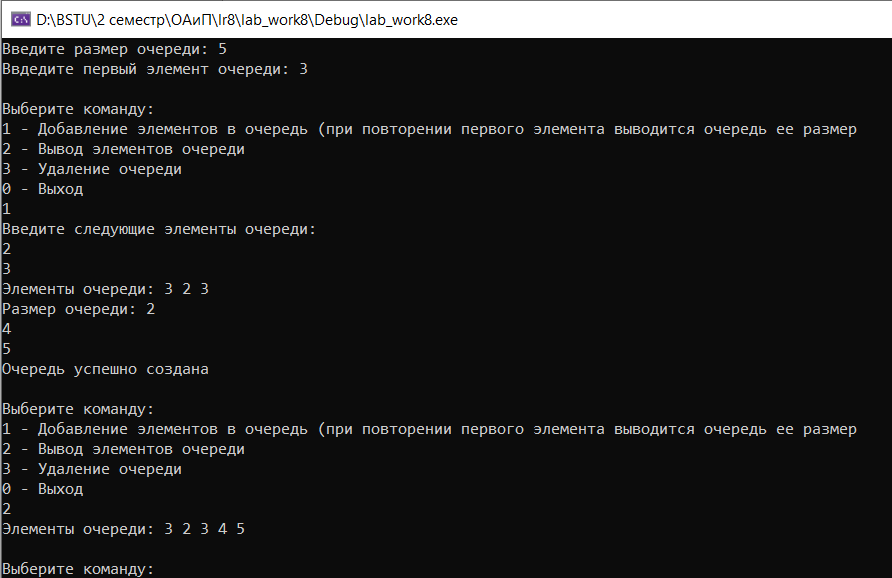
};

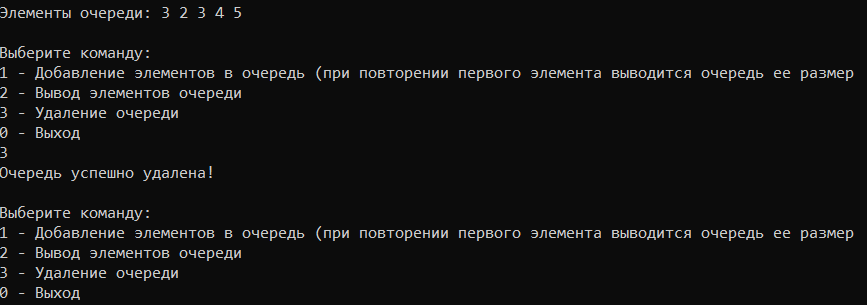
void createQueue( Queue\*\* begin, Queue \*\* end, int size, int first);

void displayQueue(Queue \*begin);

void deleteQueue(Queue\*\* begin);

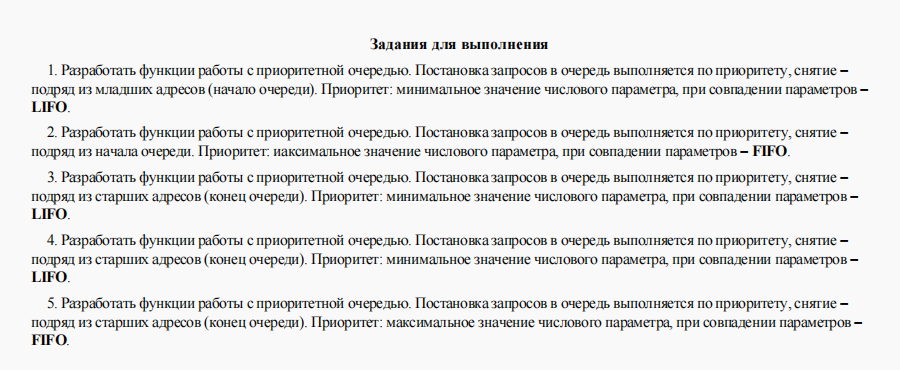
void displayQueueAndSize(Queue\* begin, int size);





Задание 5

1. 



1.1

#include<iostream>

using namespace std;

struct Item {

int data;

Item\* next;

};

Item\* head, \* tail;

bool isNull(void) { //Проверка на пустоту

return (head == NULL);

}

void deletFirst() { //Извлечение элемента из начала

if (isNull()) {

cout << "Очередь пуста" << endl;

}

else {

Item\* p = head;

head = head->next;

delete p;

}

}

void getFromHead() { //Получение элемента из начала

if (isNull()) {

cout << "Очередь пуста" << endl;

}

else {

cout << "Начало = " << head->data << endl;

}

}

void insertToQueue(int x) { //Добавление элемента в очередь

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

p->data = x;

p->next = NULL;

//Если очередь пустая, то в конец и начало очереди добавляем элемент

if (isNull()) {

head = tail = p;

}

//Иначе перебираем все элементы очереди для установки нового

else {

Item\* previous = NULL;

Item\* current = head;

//Пока новый элемент больше текущих или очередь не закончилась

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

previous = current; //перебираем указатель до NULL, чтобы установить жлемент в конец строки

current = current->next;

}

//Если элемент с более высоким приоритетом,то устанавливаем в начало очереди

if (previous == NULL) {

p->next = head;

head = p;

}

else {

//Если число больше всех текущих, то устанавливаем его в конец

if (current == NULL) {

tail->next = p;

tail = p;

}

//Если число где-то между

else {

p->next = current;

previous->next = p;

}

}

}

}

void printQueue() { //Вывод очереди

Item\* p = new Item;

if (isNull()) {

cout << "Очередь пуста" << endl;

}

else {

cout << "Очередь = ";

p = head;

while (!isNull()) {

if (p != NULL) {

cout << p->data << " "; cout << "->";

p = p->next;

}

else {

cout << "NULL" << endl;

return;

}

}

}

}

void clrQueue() { //Очистка очереди

while (!isNull()) deletFirst();

}

int main() {

setlocale(LC\_CTYPE, "Russian");

int choice = 1, z; head = NULL; tail = NULL;

while (choice != 0) {

cout << "1 - добавить элемент" << endl;

cout << "2 - получить элемент с начала" << endl;

cout << "3 - извлечь элемент с начала" << endl;

cout << "4 - вывести элементы" << endl;

cout << "5 - очистить очередь" << endl;

cout << "0 - выход" << endl;

cout << "Выберите действие "; cin >> choice;

switch (choice) {

case 1: {

cout << "Введите элемент: ";

cin >> z;

insertToQueue(z);

printQueue();

break;

}

case 2: {

getFromHead();

break;

}

case 3: {

deletFirst();

break;

}

case 4: {

printQueue();

break;

}

case 5: {

clrQueue();

break;

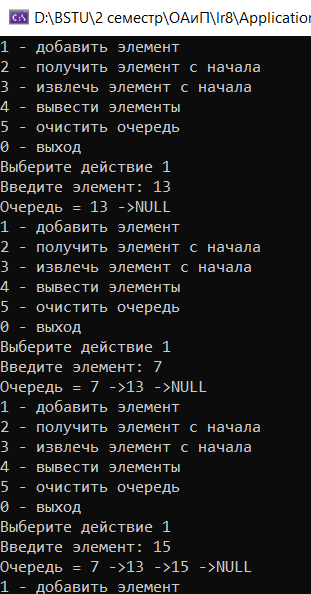
}

}

}

return 0;

}



1.2

Изменим функцию insertToQueue()

void insertToQueue(int x) { //Добавление элемента в очередь

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

p->data = x;

p->next = NULL;

//Если очередь пустая, то в конец и начало очереди добавляем элемент

if (isNull()) {

head = tail = p;

}

//Иначе перебираем все элементы очереди для установки нового

else {

Item\* previous = NULL;

Item\* current = head;

//Пока новый элемент меньше или равно текущих или очередь не закончилась

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

previous = current; //перебираем указатель до NULL, чтобы установить жлемент в конец строки

current = current->next;

}

//Если элемент с более высоким приоритетом,то устанавливаем в начало очереди

if (previous == NULL) {

p->next = head;

head = p;

}

else {

//Если число меньше всех текущих, то устанавливаем его в конец

if (current == NULL) {

tail->next = p;

tail = p;

}

//Если число где-то между

else {

p->next = current;

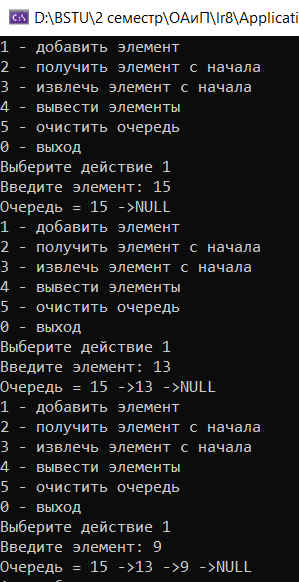
previous->next = p;

}

}

}

}



1.3, 1.4 (они идентичны)

Изменим из 1.1 функции getFromHead() и deleteFirst()

void deletFirst() { //Извлечение элемента из конца

if (isNull()) {

cout << "Очередь пуста" << endl;

}

else {

Item\* temp = NULL;

Item\* current = head;

while (!isNull()) { //Перебираем указатель, пока не доберемся до конца очереди

if (current->next == tail) { //если добрались, то через временную переменную удаляем последний элемент

current->next = NULL;

temp = tail;

tail = current;

break;

}

current = current->next; //Перемещаем указатель на следующий элемент

}

delete temp;

}

}

void getFromHead() { //Получение элемента из конца

if (isNull()) {

cout << "Очередь пуста" << endl;

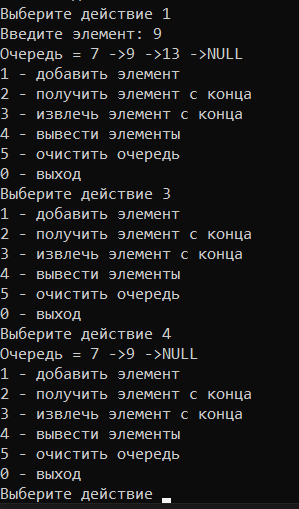
}

else {

cout << "Конец = " << tail->data << endl;

}

}



1.5

Подставим функцию из 1.2 в 1.4

void deletFirst() { //Извлечение элемента из конца

if (isNull()) {

cout << "Очередь пуста" << endl;

}

else {

Item\* temp = NULL;

Item\* current = head;

while (!isNull()) { //Перебираем указатель, пока не доберемся до конца очереди

if (current->next == tail) { //если добрались, то через временную переменную удаляем последний элемент

current->next = NULL;

temp = tail;

tail = current;

break;

}

current = current->next; //Перемещаем указатель на следующий элемент

}

delete temp;

}

}

void getFromHead() { //Получение элемента из конца

if (isNull()) {

cout << "Очередь пуста" << endl;

}

else {

cout << "Конец = " << tail->data << endl;

}

}

void insertToQueue(int x) { //Добавление элемента в очередь

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

p->data = x;

p->next = NULL;

//Если очередь пустая, то в конец и начало очереди добавляем элемент

if (isNull()) {

head = tail = p;

}

//Иначе перебираем все элементы очереди для установки нового

else {

Item\* previous = NULL;

Item\* current = head;

//Пока новый элемент меньше или равно текущих или очередь не закончилась

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

previous = current; //перебираем указатель до NULL, чтобы установить жлемент в конец строки

current = current->next;

}

//Если элемент с более высоким приоритетом, то устанавливаем в начало очереди

if (previous == NULL) {

p->next = head;

head = p;

}

else {

//Если число меньше всех текущих, то устанавливаем его в конец

if (current == NULL) {

tail->next = p;

tail = p;

}

//Если число где-то между

else {

p->next = current;

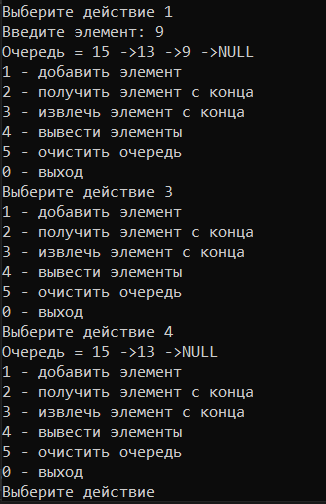
previous->next = p;

}

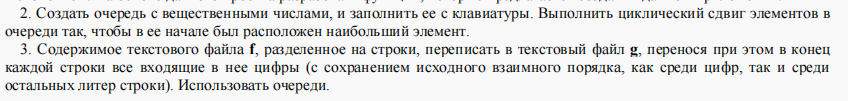
}

}

}



2.



#include <iostream>

using namespace std;

struct Queue { // создаем очередь

double number;

Queue\* next;

};

void createQueue(Queue\*\* begin, Queue\*\* end, int size);

void displayQueue(Queue\* begin);

void deleteQueue(Queue\*\* begin);

void ringQueue(Queue\*\* begin);

void createQueue(Queue\*\* begin, Queue\*\* end, int size) {

Queue\* temp;

double n; //переменная для ввода числа

cout << "Введите следующие элементы очереди: " << endl;

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

temp = new Queue; // Создание нового элемента очереди

cin >> n; // Ввод значения с клавиатуры

temp->number = n; // Запись значения в элемент

temp->next = NULL; // Указываем, что следующий элемент пустой

if (\*begin == NULL) {

\*begin = temp; // Если начало очереди пустое, устанавливаем начало

}

else {

(\*end)->next = temp; // Иначе устанавливаем связь с последним элементом

}

\*end = temp; // Устанавливаем конец очереди на новый элемент

}

cout << "Очередь успешно создана" << endl;

}

void displayQueue(Queue\* begin) {

Queue\* temp = begin; // Создаем временный указатель temp для вывода элементов очереди

if (temp == NULL) {

cout << "Очередь пуста!" << endl;

}

else {

while (temp != NULL) { //Выводим элементы до конца очереди

cout << temp->number << " ";

temp = temp->next; //Перемещаем указатель на следующий элемент

}

cout << endl;

}

}

void deleteQueue(Queue\*\* begin) {

Queue\* temp;

while (\*begin != NULL) { //Удаляем пока начало очереди не достигнет NULL

temp = \*begin;

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

delete temp;

}

}

void ringQueue(Queue\*\* begin) {

if (\*begin == NULL) {

return; // Проверка на пустую очередь

}

Queue\* temp = \*begin;

Queue\* current = \*begin;

double max = (\*begin)->number; // Переменная для максимального значения в очереди

// Находим максимальное значение в очереди

while (temp != NULL) {

if (temp->number > max) {

max = temp->number;

}

temp = temp->next;

}

// Пока максимальный элемент не находится в начале и очередь не состоит из одного элемента

while ((\*begin)->number != max) {

temp = \*begin;

current = temp->next;

\*begin = current;

temp->next = nullptr;

while (current->next != nullptr) {

current = current->next;

}

current->next = temp;

}

}

int main() {

setlocale(LC\_ALL, "Rus");

Queue\* begin = NULL, \* end = NULL; //создание начала и конца очереди

int size; //Размер очереди

cout << "Введите размер очереди: ";

cin >> size;

if (size > 0) {

createQueue(&begin, &end, size);

}

else {

cout << "Очередь не может быть больше заполнена!";

}

cout << "Очередь до циклического сдвига: ";

displayQueue(begin);

cout << "Очередь после циклического сдвига: ";

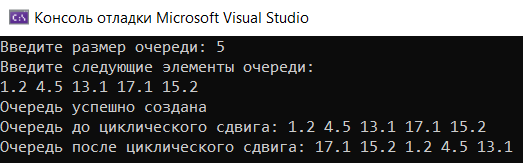
ringQueue(&begin);

displayQueue(begin);

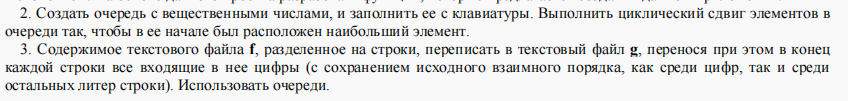
deleteQueue(&begin);

return 0;

}



3.



#include <iostream>

#include <fstream>

#include <string>

#include <cctype> // Для использования isdigit()

using namespace std;

struct Queue { // создаем очередь

int data;

Queue\* next;

};

void createQueueFromFile(Queue\*\* begin, Queue\*\* end, ifstream& inFile, ofstream& outFile);

void writeQueueToFile(Queue\* begin, ofstream& outFile);

void deleteQueue(Queue\*\* begin);

void createQueueFromFile(Queue\*\* begin, Queue\*\* end, ifstream& inFile, ofstream& outFile) {

string line;

while (getline(inFile, line)) { // Читаем строку из файла

string symbols, digits;

for (char c : line) {

if (isdigit(c)) {

digits += c; // Сохраняем цифры

}

else if (!isspace(c)) {

symbols += c; // Сохраняем символы, игнорируя пробелы

}

}

if (!digits.empty()) {

Queue\* temp = new Queue; // Создаем новый элемент очереди

temp->data = stoi(digits); // Преобразуем строку с цифрами в число и записываем в элемент

temp->next = nullptr; // Указываем, что следующий элемент пустой

if (\*begin == nullptr) {

\*begin = temp; // Если начало очереди пустое, устанавливаем начало

}

else {

(\*end)->next = temp; // Иначе устанавливаем связь с последним элементом

}

\*end = temp; // Устанавливаем конец очереди на новый элемент

symbols += " " + digits; // Добавляем цифры в конец строки

}

symbols += "\n"; // Добавляем символ перевода строки

outFile << symbols << " " << endl; // Записываем строку с цифрами в конец файла

}

cout << "Очередь успешно создана" << endl;

}

void writeQueueToFile(Queue\* begin, ofstream& outFile) {

Queue\* temp = begin;

while (temp != nullptr) { // Проходим по очереди и записываем числа в файл

outFile << temp->data << " ";

temp = temp->next;

}

outFile << endl;

cout << "Числа из очереди успешно записаны в файл" << endl;

}

void deleteQueue(Queue\*\* begin) {

Queue\* temp;

while (\*begin != nullptr) { // Удаляем элементы очереди

temp = \*begin;

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

delete temp;

}

cout << "Очередь успешно удалена" << endl;

}

int main() {

setlocale(LC\_ALL, "Rus");

Queue\* begin = nullptr, \* end = nullptr; // Создаем начало и конец очереди

ifstream inFile("f.txt"); // Открываем файл для чтения

ofstream outFile("g.txt"); // Открываем файл для записи

if (!inFile.is\_open() || !outFile.is\_open()) {

cerr << "Ошибка открытия файлов!" << endl;

return 1;

}

createQueueFromFile(&begin, &end, inFile, outFile); // Создаем очередь из чисел файла f

inFile.close();

outFile.close();

deleteQueue(&begin); // Удаляем очередь

return 0;

}

