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 10 группы

Мандрик Алексей Иванович

Преподаватель: асс. Андронова М.В.

Минск, 2024

Создать проект, демонстрирующий работу с ***очередью***. В соответствии со своим вариантом выполнить задание из таблицы, представленной ниже. Разработать меню и реализовать все операции с очередью через функции. Максимальный размер очереди ввести с клавиатуры.

**Вариант 8**



**Код программы:**

stdafx.h:

#pragma once

#include <iostream>

struct Node // Структура

{

int data;

Node\* next;

Node(int data) : data(data), next(nullptr) {}

};

class Queue { // Класс для очереди

private:

Node\* front;

Node\* rear;

int size;

public:

// Прототипы

Queue();

~Queue();

void enqueue(int data);

int dequeue();

bool isEmpty() const;

int getSize() const;

void clear();

void deleteFirstNegative();

void print() const;

};

Функции

#include "stdafx.h"

Queue::Queue() : front(nullptr), rear(nullptr), size(0) {}

Queue::~Queue() {

clear();

}

void Queue::enqueue(int data) { // Функция для ввода

Node\* node = new Node(data);

if (isEmpty()) {

front = node;

}

else {

rear->next = node;

}

rear = node;

size++;

}

int Queue::dequeue() { // Функция для очищения очереди

if (isEmpty()) {

std::cerr << "Очередь пуста." << std::endl;

return -1;

}

Node\* nodeToDelete = front;

int data = nodeToDelete->data;

if (front == rear) {

front = rear = nullptr;

}

else {

front = front->next;

}

delete nodeToDelete;

size--;

return data;

}

bool Queue::isEmpty() const { // Функция для определния пуста ли очереди

return front == nullptr;

}

int Queue::getSize() const { // Функция для определения размера очереди

return size;

}

void Queue::clear() { // Функция для очищения очереди

while (!isEmpty()) {

dequeue();

}

}

void Queue::deleteFirstNegative() { // Функция для удаления первого отрицательного элемента

Node\* node = front;

Node\* prev = nullptr;

while (node != nullptr && node->data >= 0) {

prev = node;

node = node->next;

}

if (node != nullptr) {

if (prev != nullptr) {

prev->next = node->next;

}

else {

front = node->next;

}

if (rear == node) {

rear = prev;

}

delete node;

size--;

}

}

void Queue::print() const { // Функция для вывода очереди

Node\* node = front;

while (node != nullptr) {

std::cout << node->data << " ";

node = node->next;

}

std::cout << std::endl;

}Main

#include <iostream>

#include "stdafx.h" // Файл заголовков

using namespace std;

void menu() // Меню

{

cout << "Сделайте выбор:" << endl;

cout << " 1 - Ввод чисел" << endl;

cout << " 2 - Вывод чисел" << endl;

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

cout << " 4 - Удалить первый отрицательный элемент" << endl;

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

cout << " 0 - Выход" << endl;

}

int main()

{

setlocale(LC\_ALL, "Rus");

int capacity, numElements;

cout << "Введите емкость очереди: ";

cin >> numElements;

Queue q;

int choice;

do

{

menu();

cin >> choice;

switch (choice) {

case 1: { // Ввод элементов очереди

cout << "Введите " << numElements << " элементы:" << endl;

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

int element;

cin >> element;

q.enqueue(element);

}

break;

}

case 2: { // Вывод элементов очереди

cout << "Элементы очереди: ";

q.print();

break;

}

case 3: { // Определение размера очереди

cout << "Размер очереди : " << q.getSize() << endl;

break;

}

case 4: { // Удаление первого отрицательного элемента

q.deleteFirstNegative();

break;

}

case 5: { // Очищение

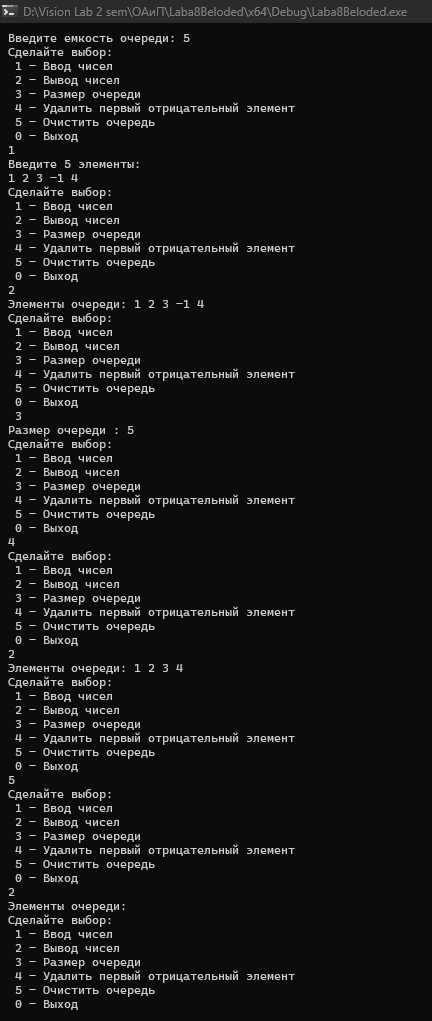
q.clear();

}

}

} while (choice != 0);

}**Результат выполнения:**



**Дополнительные задания:**

**Доп 1:**

В [приложении 1](file:///C:\Users\User\Downloads\ПрактикумОАП-2часть2021.doc#prilogenie1) приведен проект, в котором реализована очередь на основе односвязного *списка* с *приоритетным* включением. На основе данного проекта разработать функции, которые предлагается создать в данном приложении.

**Код программы:**

#include<iostream>

using namespace std;

struct Item // Структура

{

int data;

Item\* next;

};

Item\* head = nullptr, \* tail = nullptr;

bool isNull(void) //Проверка на пустоту

{

return (head == nullptr);

}

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; //новый указатель

p->data = x;

p->next = nullptr;

Item\* v = new Item; //указатель для нового числа

Item\* p1 = new Item;

Item\* p2 = new Item;

int i = 0;

if (isNull())

head = tail = p;

else

{

p2 = head; p1 = head;

while (p1 != nullptr) //пока очередь не закончится

{

if (i == 1)

{

if (x <= p1->data) //число меньше, чем в очереди

{

v->data = x;

v->next = p1;

p2->next = v;

return;

}

p2 = p2->next; // следующее число

}

else

{

if (x <= p1->data)

{

v->data = x;

v->next = p1;

head = v;

return;

}

}

p1 = p1->next;

i = 1;

}

if (p1 == nullptr)

{

tail->next = p;

tail = p;

}

}

}

void printQueue() //Вывод очереди

{

Item\* p = new Item;

if (isNull())

cout << "Очередь пуста" << endl;

else

{

cout << "Очередь = ";

p = head;

while (!isNull())

{

if (p != nullptr)

{

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

p = p->next;

}

else

{

cout << "NULL" << endl;

return;

}

}

}

}

void clrQueue() //Очистка очереди

{

while (!isNull()) deletFirst();

}

// Функции работы с приоритетной очередью

// 1. Приоритет: минимальное значение числового параметра, при совпадении параметров - LIFO

void insertPriorityQueueMinLIFO(int x) {

insertToQueue(x);

}

void deletePriorityQueueMinLIFO() {

deletFirst();

}

// 2. Приоритет: максимальное значение числового параметра, при совпадении параметров - FIFO

void insertPriorityQueueMaxFIFO(int x) {

// Поскольку приоритет основан на максимальном значении, поменяем логику вставки,

// чтобы самое большое значение оказалось в начале

Item\* p = new Item; // Создание нового узла

p->data = x;

p->next = nullptr;

if (isNull()) {

head = tail = p;

}

else {

p->next = head;

head = p;

}

}

void deletePriorityQueueMaxFIFO() {

deletFirst();

}

// 3. Приоритет: минимальное значение числового параметра, при совпадении параметров - LIFO

void insertPriorityQueueMinLIFORear(int x) {

insertToQueue(x);

}

void deletePriorityQueueMinLIFORear() {

// Поскольку удаление происходит с конца (LIFO), обычная функция извлечения будет достаточна

deletFirst();

}

// 4. Приоритет: минимальное значение числового параметра, при совпадении параметров - LIFO

void insertPriorityQueueMaxLIFO(int x) {

// Поскольку приоритет основан на минимальном значении, поменяем логику вставки,

// чтобы самое большое значение оказалось в начале

Item\* p = new Item; // Создание нового узла

p->data = x;

p->next = nullptr;

if (isNull()) {

head = tail = p;

}

else {

p->next = head;

head = p;

}

}

void deletePriorityQueueMaxLIFO() {

// Поскольку удаление происходит с начала (LIFO), обычная функция извлечения будет достаточна

deletFirst();

}

// 5. Приоритет: максимальное значение числового параметра, при совпадении параметров - FIFO

void insertPriorityQueueMaxFIFORear(int x) {

insertToQueue(x);

}

void deletePriorityQueueMaxFIFORear() {

deletFirst();

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int choice = 1, z;

while (choice != 0) // Меню

{

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: // Ввод элементов очереди

cout << "Введите элемент: ";

cin >> z;

insertPriorityQueueMinLIFO(z);

printQueue();

break;

case 2: // Получение элемента из начала очереди

getFromHead();

break;

case 3: // Удаление элемента из начала очереди

deletePriorityQueueMinLIFO();

break;

case 4: // Вывод

printQueue();

break;

case 5: // Очистка

clrQueue();

break;

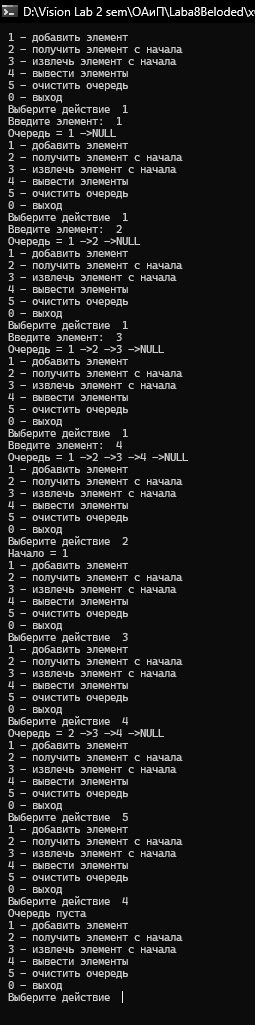
}

}

return 0;

}

**Результат выполнения:**

****

**Доп 2:**

Создать очередь с вещественными числами, и заполнить ее с клавиатуры. Выполнить циклический сдвиг элементов в очереди так, чтобы в ее начале был расположен наибольший элемент.

**Код программы:**

#include <iostream>

#include <string>

using namespace std;

struct Number // Структура

{

double info;

Number\* next;

};

void create(Number\*\* begin, Number\*\* end, double p); // Формирование элементов очереди

void view(Number\* begin); // Функция вывода элементов очереди

void Del(Number\*\* begin, Number\*\* p); // Удаление элементов очереди

void toRight(Number\*\* begin, Number\*\* end); // Перемещение элементов очереди вправо

double MAX(Number\* begin); // Нахождение максимального значения в очереди

int main()

{

setlocale(LC\_ALL, "Rus");

Number\* begin = NULL, \* end, \* t;

t = new Number;

double p; int size;

string s;

cout << "Введите длину очереди: " << endl;

cin >> size;

cout << "Введите элементы очереди: ";

cin >> p;

t->info = p; // Первый элемент

t->next = NULL;

begin = end = t;

for (int i = 0; i < size - 1; i++)

{

double temp;

cin >> temp;

create(&begin, &end, temp);

}

cout << "Исходная очередь: ";

view(begin);

toRight(&begin, &end);

cout << "Очередь после сдвига: ";

view(begin);

return 0;

}

void create(Number\*\* begin, Number\*\* end, double p) // Формирование элементов очереди

{

Number\* t = new Number;

t->next = NULL;

if (\*begin == NULL)

\*begin = \*end = t;

else

{

t->info = p;

(\*end)->next = t;

\*end = t;

}

}

void view(Number\* begin) // Вывод элементов очереди

{

Number\* t = begin;

if (t == NULL)

{

cout << "Очередь пуста!\n";

return;

}

else

while (t != NULL)

{

cout << t->info << " ";

t = t->next;

}

cout << endl;

}

void Del(Number\*\* begin, Number\*\* p) // Удаление элементов очереди

{

Number\* t;

t = new Number;

while (\*begin != \*p)

{

t = \*begin;

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

delete t;

}

}

void toRight(Number\*\* begin, Number\*\* end) { // Перемещение элементов очереди вправо

Number\* t = \*begin;

Number\* t2 = \*begin;

double temp;

while (t2->info != MAX(\*begin)) {

temp = t->info;

while (t->next != NULL) {

swap(temp, t->next->info);

t = t->next;

}

t2->info = temp;

}

}

double MAX(Number\* begin) { // Нахождение максимального значения в очереди

Number\* t = begin;

double m = -900;

if (t == NULL)

{

cout << "Очередь пуста!\n";

return -1;

}

else

while (t != NULL)

{

if (t->info >= m) {

m = t->info;

}

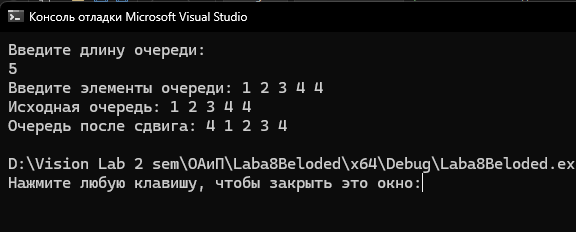
t = t->next;

}

return m;

}

**Результат выполнения:**



**Доп 3:**

Содержимое текстового файла **f**, разделенное на строки, переписать в текстовый файл **g**, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка, как среди цифр, так и среди остальных литер строки). Использовать очереди.

**Код программы:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Number // Структура

{

int info;

Number\* next;

};

void create(Number\*\* begin, Number\*\* end, int p); //формирование элементов очереди

void view(Number\* begin); //функция вывода элементов очереди

void Del(Number\*\* begin, Number\*\* p);

ifstream frm("f.txt");

ofstream out("g.txt");

int main()

{

setlocale(LC\_ALL, "Rus");

Number\* begin = NULL, \* end, \* t;

t = new Number;

int p, size;

string s;

size = 1000;

p = -1000;

t->info = p; //первый элемент

t->next = NULL;

begin = end = t;

if (frm.fail())

{

cout << "\n Ошибка открытия файла";

exit(1);

}

while (!frm.eof())

{

getline(frm, s);

for (int i = 0; i < s.length(); i++) {

if (isdigit(s[i])) {

create(&begin, &end, s[i] - '0');

}

else {

out << s[i];

}

}

view(begin);

Del(&begin, &end);

}

frm.close();

return 0;

}

void create(Number\*\* begin, Number\*\* end, int p) //Формирование элементов очереди

{

Number\* t = new Number;

t->next = NULL;

if (\*begin == NULL)

\*begin = \*end = t;

else

{

t->info = p;

(\*end)->next = t;

\*end = t;

}

}

void view(Number\* begin) //Вывод элементов очереди

{

static int count;

Number\* t = begin;

if (count != 0) {

t = t->next;

}

if (t == NULL)

{

out << "Очередь пуста!\n";

return;

}

else

while (t != NULL)

{

if (t->info == -1000) {

t = t->next;

continue;

}

out << t->info;

t = t->next;

}

out << endl;

count++;

}

void Del(Number\*\* begin, Number\*\* p) //Удаление очереди

{

Number\* t;

t = new Number;

while (\*begin != \*p)

{

t = \*begin;

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

delete t;

}

}

**Результат выполнения:**

