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»

Выполнила:

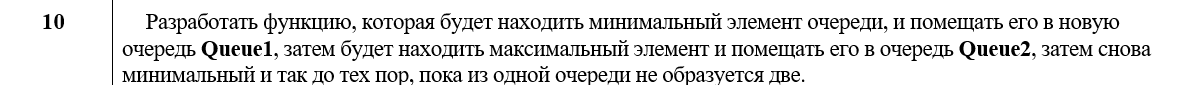
Студентка 1 курса 6 группы

Литвинчук Дарья Валерьевна

Преподаватель: асс. Андронова М.В.

2024, Минск

Вариант 10



#include <iostream>

using namespace std;

struct Queue{

int info;

Queue\* next;

};

void create(Queue\*\* begin, Queue\*\* end, int p); // фнукция добавления элемента в очередь

void view(Queue\* begin); // функция вывода

int fromFIFO(Queue\*\* begin); // функция извлечения

void twoqueue(Queue\*\* begin, Queue\*\* end, Queue\*\* begin1, Queue\*\* end1, Queue\*\* begin2, Queue\*\* end2); //функция разбиения 1 очереди на 2

void view2(Queue\* begin1, Queue\* begin2); //функция вывода 2 очередей

int main()

{

setlocale(LC\_ALL, "rus");

int choice;

//указатели на начало каждой очереди

Queue\* begin = NULL, \* end;

Queue\* begin1 = NULL, \* end1;

Queue\* begin2 = NULL, \* end2;

int size;

do

{

cout << "1 - Добавление элементов в очередь" << endl;

cout << "2 - Извлечение элемента из очереди" << endl;

cout << "3 - Вывод очереди" << endl;

cout << "4 - Функция разбиения" << endl;

cout << "5 - Функция вывода новых очередей" << endl;

cout << "6 - Выход" << endl;

cin >> choice;

//анализируем choice

switch (choice)

{

case 1:

int p;

cout << "Введите размер очереди: "; cin >> size;

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

{

cout << "Введите элемент: "; cin >> p;

create(&begin, &end, p);

}

break;

case 2:

if (begin == NULL)

cout << "Очередь пуста" << endl;

else

cout << "Извлеченный элемент: " << fromFIFO(&begin) << endl;

break;

case 3:

if (begin == NULL) //вывод на экран

cout << "No elements" << endl;

else

view(begin);

break;

case 4:

if (begin == NULL)

cout << "Очередь пуста" << endl;

else

twoqueue(&begin, &end, &begin1, &end1, &begin2, &end2);

break;

case 5:

cout << "Первая очередь: ";

view(begin1);

cout << "Вторая очередь: ";

view(begin2);

break;

}

} while (choice != 6);

return 0;

}

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

{

Queue\* t = new Queue;

t->info = p;

t->next = NULL;

if (\*begin == NULL)

\*begin = \*end = t;

else

{

(\*end)->next = t;

\*end = t;

}

}

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

{

Queue\* t = begin;

while (t != NULL)

{

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

t = t->next;

}

cout << endl;

}

int fromFIFO(Queue\*\* begin) // Извлечение элемента из очереди

{

Queue\* t = \*begin;

int val = t->info;

\*begin = t->next;

delete t;

return val;

}

void twoqueue(Queue\*\* begin, Queue\*\* end, Queue\*\* begin1, Queue\*\* end1, Queue\*\* begin2, Queue\*\* end2) // функция разбиения очереди

{

while (\*begin != nullptr)

{

Queue\* t = \*begin;

Queue\* prev = nullptr;

Queue\* minNode = nullptr;

Queue\* maxNode = nullptr;

int minVal = INT\_MAX;

int maxVal = INT\_MIN;

// Находим минимальный элемент в очереди

while (t != nullptr)

{

if (t->info < minVal)

{

minVal = t->info;

minNode = t;

}

t = t->next;

}

// Помещаем минимальный элемент в Queue1

if (minNode != nullptr)

{

create(begin1, end1, minNode->info);

// Удаляем минимальный элемент из очереди

if (\*begin == minNode)

{

\*begin = minNode->next;

}

else

{

prev = \*begin;

while (prev->next != minNode)

{

prev = prev->next;

}

prev->next = minNode->next;

}// Освобождаем узел

delete minNode;

}

// Проверяем, остался ли второй элемент

if (\*begin == nullptr)

break;

// Находим максимальный элемент в очереди

t = \*begin;

while (t != nullptr)

{

if (t->info > maxVal)

{

maxVal = t->info;

maxNode = t;

}

t = t->next;

}

// Помещаем максимальный элемент в Queue2

if (maxNode != nullptr)

{

create(begin2, end2, maxNode->info);

// Удаляем максимальный элемент из очереди

if (\*begin == maxNode)

{

\*begin = maxNode->next;

}

else

{

prev = \*begin;

while (prev->next != maxNode)

{

prev = prev->next;

}

prev->next = maxNode->next;

}

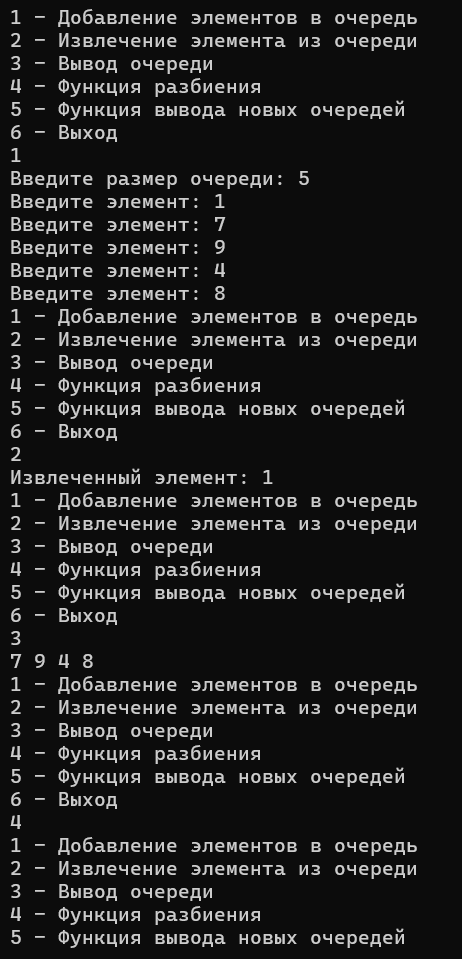
// Освобождаем узел

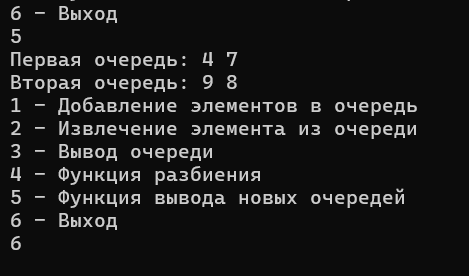
delete maxNode;

}

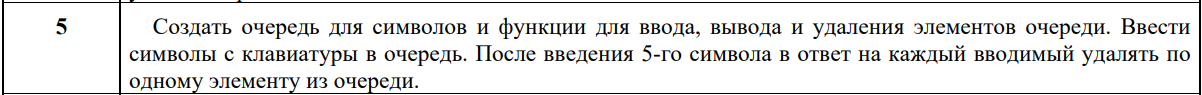
}

}





Доп. Задачи

Вариант 5  
  
#include <iostream>

using namespace std;

// Структура элемента очереди

struct QueueNode {

char data;

QueueNode\* next;

};

// Структура очереди

struct Queue {

QueueNode\* front;

QueueNode\* rear;

};

// Инициализация очереди

void initQueue(Queue& q) {

q.front = nullptr;

q.rear = nullptr;

}

// Проверка на пустоту очереди

bool isEmpty(const Queue& q) {

return (q.front == nullptr);

}

// Добавление элемента в очередь

void enqueue(Queue& q, char value) {

// Создаем новый узел

QueueNode\* newNode = new QueueNode;

newNode->data = value;

newNode->next = nullptr;

// Если очередь пуста, устанавливаем front и rear на новый узел

if (isEmpty(q)) {

q.front = newNode;

q.rear = newNode;

}

else {

// Иначе добавляем новый узел в конец очереди и обновляем rear

q.rear->next = newNode;

q.rear = newNode;

}

}

// Удаление элемента из очереди

void dequeue(Queue& q) {

// Если очередь пуста, ничего не делаем

if (isEmpty(q)) {

cout << "Очередь пуста. Невозможно удалить элемент." << endl;

return;

}

// Удаляем первый элемент и обновляем front

QueueNode\* temp = q.front;

q.front = q.front->next;

// Если удаляемый элемент был последним, обновляем rear

if (q.front == nullptr) {

q.rear = nullptr;

}

delete temp;

}

// Вывод содержимого очереди

void displayQueue(const Queue& q) {

// Если очередь пуста, выводим сообщение об этом

if (isEmpty(q)) {

cout << "Очередь пуста." << endl;

return;

}

// Выводим элементы очереди

QueueNode\* current = q.front;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

// Освобождение памяти, занятой элементами очереди

void clearQueue(Queue& q) {

while (!isEmpty(q)) {

dequeue(q);

}

}

int main() {

setlocale(LC\_ALL, "rus");

Queue chars;

initQueue(chars);

int n;

cout << "Введите длину очереди: "; cin >> n;

cout << "Введите очередь: ";

for (int i = 0; i < n; i++)//Ввод очереди;

{

char a;

cin >> a;

enqueue(chars, a);

if (i + 1 > 5)

{

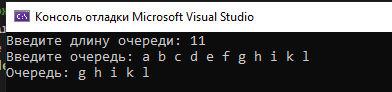
dequeue(chars);

}

}

cout << "Очередь: "; displayQueue(chars);

return 0;}

  
  
Вариант 4  
#include <iostream>

using namespace std;

// Структура элемента очереди

struct QueueNode {

char data;

QueueNode\* next;

};

// Структура очереди

struct Queue {

QueueNode\* front;

QueueNode\* rear;

};

// Инициализация очереди

void initQueue(Queue& q) {

q.front = nullptr;

q.rear = nullptr;

}

// Проверка на пустоту очереди

bool isEmpty(const Queue& q) {

return (q.front == nullptr);

}

// Добавление элемента в очередь

void enqueue(Queue& q, char value) {

// Создаем новый узел

QueueNode\* newNode = new QueueNode;

newNode->data = value;

newNode->next = nullptr;

// Если очередь пуста, устанавливаем front и rear на новый узел

if (isEmpty(q)) {

q.front = newNode;

q.rear = newNode;

}

else {

// Иначе добавляем новый узел в конец очереди и обновляем rear

q.rear->next = newNode;

q.rear = newNode;

}

}

// Удаление элемента из очереди

void dequeue(Queue& q) {

// Если очередь пуста, ничего не делаем

if (isEmpty(q)) {

cout << "Очередь пуста. Невозможно удалить элемент." << endl;

return;

}

// Удаляем первый элемент и обновляем front

QueueNode\* temp = q.front;

q.front = q.front->next;

// Если удаляемый элемент был последним, обновляем rear

if (q.front == nullptr) {

q.rear = nullptr;

}

delete temp;

}

// Вывод содержимого очереди

void displayQueue(const Queue& q) {

// Если очередь пуста, выводим сообщение об этом

if (isEmpty(q)) {

cout << "Очередь пуста." << endl;

return;

}

// Выводим элементы очереди

QueueNode\* current = q.front;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

// Освобождение памяти, занятой элементами очереди

void clearQueue(Queue& q) {

while (!isEmpty(q)) {

dequeue(q);

}

}

int main() {

setlocale(LC\_ALL, "rus");

Queue chars;

initQueue(chars);

int n;

cout << "Введите длину очереди: "; cin >> n;

cout << "Введите очередь: ";

for (int i = 0; i < n; i++)//Ввод очереди;

{

char a;

cin >> a;

enqueue(chars, a);

if (i + 1 == n)

{

char sigma;

cout << "Введите символ: "; cin >> sigma;

if (a == sigma) {

dequeue(chars);

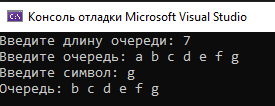
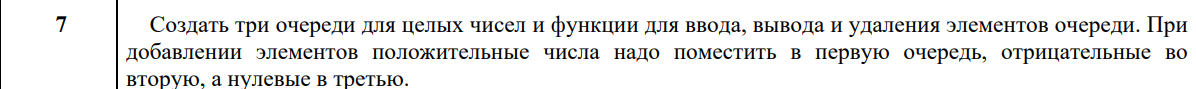
}

}

}

cout << "Очередь: "; displayQueue(chars);

return 0;

}  
  
  
  
Вариант 7  
  
  
#include <iostream>

using namespace std;

// Структура элемента очереди

struct QueueNode {

int data;

QueueNode\* next;

};

// Структура очереди

struct Queue {

QueueNode\* front;

QueueNode\* rear;

};

// Инициализация очереди

void initQueue(Queue& q) {

q.front = nullptr;

q.rear = nullptr;

}

// Проверка на пустоту очереди

bool isEmpty(const Queue& q) {

return (q.front == nullptr);

}

// Добавление элемента в очередь

void enqueue(Queue& q, int value) {

// Создаем новый узел

QueueNode\* newNode = new QueueNode;

newNode->data = value;

newNode->next = nullptr;

// Если очередь пуста, устанавливаем front и rear на новый узел

if (isEmpty(q)) {

q.front = newNode;

q.rear = newNode;

}

else {

// Иначе добавляем новый узел в конец очереди и обновляем rear

q.rear->next = newNode;

q.rear = newNode;

}

}

// Удаление элемента из очереди

void dequeue(Queue& q) {

// Если очередь пуста, ничего не делаем

if (isEmpty(q)) {

cout << "Очередь пуста. Невозможно удалить элемент." << endl;

return;

}

// Удаляем первый элемент и обновляем front

QueueNode\* temp = q.front;

q.front = q.front->next;

// Если удаляемый элемент был последним, обновляем rear

if (q.front == nullptr) {

q.rear = nullptr;

}

delete temp;

}

int pop(Queue& q) {

// Если очередь пуста, ничего не делаем

if (isEmpty(q)) {

cout << "Очередь пуста. Невозможно удалить элемент." << endl;

return 0;

}

// Удаляем первый элемент и обновляем front

QueueNode\* temp = q.front;

int p = temp->data;

q.front = q.front->next;

// Если удаляемый элемент был последним, обновляем rear

if (q.front == nullptr) {

q.rear = nullptr;

}

delete temp;

return p;

}

// Вывод содержимого очереди

void displayQueue(const Queue& q) {

// Если очередь пуста, выводим сообщение об этом

if (isEmpty(q)) {

cout << "Очередь пуста." << endl;

return;

}

// Выводим элементы очереди

QueueNode\* current = q.front;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

// Освобождение памяти, занятой элементами очереди

void clearQueue(Queue& q) {

while (!isEmpty(q)) {

dequeue(q);

}

}

int main() {

setlocale(LC\_ALL, "rus");

Queue positive;

initQueue(positive);

Queue negative;

initQueue(negative);

Queue zero;

initQueue(zero);// Создаем очереди

int n;

cout << "Введите длину очереди: "; cin >> n;

cout << "Введите очередь: ";

for (int i = 0; i < n; i++) // Сортируем элементы в разные очереди

{

int a;

cin >> a;

if (a > 0) {

enqueue(positive, a);

}

else if (a < 0) {

enqueue(negative, a);

}

else {

enqueue(zero, a);

}

}

cout << "Очередь 1: "; displayQueue(positive);

cout << "Очередь 2: "; displayQueue(negative);

cout << "Очередь 3: "; displayQueue(zero);

return 0;

}

