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:

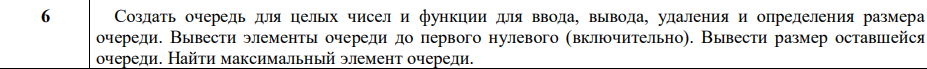
Студент 1 курса 8 группы

Лужецкий Владислав Константинович

Преподаватель: асс. Андронова М.В.

2024, Минск

Основной вариант 6:



#include <iostream>

#include <Windows.h>

using namespace std;

struct Number

{

int info;

Number\* next;

};

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

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

Number\* maxElem(Number\* begin); //определение минимального элемента

void outputBeforeNullPtr(Number\* begin);

void outputAfterNullPtr(Number\* begin, int size);

void determineSizeQueue(Number\* begin, int size);

void deleteQueue(Number\* begin, int size);

int main()

{

setlocale(LC\_ALL, "Rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

t = new Number;

int p, size, choice;

cout << "\nВведите максимальный размер очереди: ";

cin >> size;

do {

cout << "\n\tВыберите пункт меню: \n";

cout << "1. Добавление элемента в очередь: \n";

cout << "2. Просмотр очереди и вывод максимального: \n";

cout << "3. Вывод до первого нулевого элемента: \n";

cout << "4. Вывод после первого нулевого элемента: \n";

cout << "5. Определение размера очереди: \n";

cout << "6. Удаления очереди: \n";

cout << "0. Выход \n";

cout << "Ваш вариант: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Введите число: ";

cin >> p;

t->info = p; //первый элемент

t->next = NULL;

begin = end = t;

for (int i = 1; i < size; i++) //создание очереди

{

cout << "Введите число: ";

cin >> p;

create(&begin, &end, p);

}

break;

case 2:

cout << "\nЭлементы очереди: \n";

if (begin == NULL) //вывод на экран

cout << "No elements" << endl;

else

view(begin);

t = maxElem(begin); //определение максимального

cout << "\nМаксимум: " << t->info << endl;

break;

case 3:

cout << "\nэлементы очереди до первого нулевого (включительно):\n";

outputBeforeNullPtr(begin);

break;

case 4:

cout << "\nэлементы очереди после первого нулевого (включительно):\n";

outputAfterNullPtr(begin, size);

break;

case 5:

cout << "\nРазмер очереди: ";

determineSizeQueue(begin, size);

break;

case 6:

deleteQueue(begin, size);

break;

case 0:

exit(0);

default:

cout << "\nНекорректный ввод!";

break;

}

} while (choice != 0);

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

{

Number\* t = begin;

if (t == NULL)

{

cout << "Number is empty\n";

return;

}

else

while (t != NULL)

{

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

t = t->next;

}

}

Number\* maxElem(Number\* begin) //Определение минимального элемента

{

Number\* t = begin, \* mx = nullptr;

int max;

if (t == NULL)

{

cout << "Number is empty\n"; return 0;

}

else

{

max = t->info;

while (t != NULL)

{

if (t->info >= max)

{

max = t->info;

mx = t;

}

t = t->next;

}

}

return mx;

}

void outputBeforeNullPtr(Number\* begin)

{

Number\* t = begin;

while (t->info != 0)

{

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

t = t->next;

}

if (t->info == 0)

{

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

t = t->next;

return;

}

}

void outputAfterNullPtr(Number\* begin, int size)

{

Number\* t = begin;

int counter = 0;

cout << "\nэлементы очереди после первого нулевого:\n";

while (t->info != 0)

{

counter++;

t = t->next;

}

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

{

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

t = t->next;

}

}

void determineSizeQueue(Number\* begin, int size)

{

Number\* t = begin;

int counter = 0;

if (t == NULL)

{

cout << "Number is empty\n";

return;

}

else

{

while (t != NULL)

{

counter++;

t = t->next;

}

}

cout << counter;

}

void deleteQueue(Number\* begin, int size)

{

Number\* t = begin, \*temp;

if (t == NULL)

{

cout << "Number is empty\n";

return;

}

else

{

while (t != NULL)

{

temp = t;

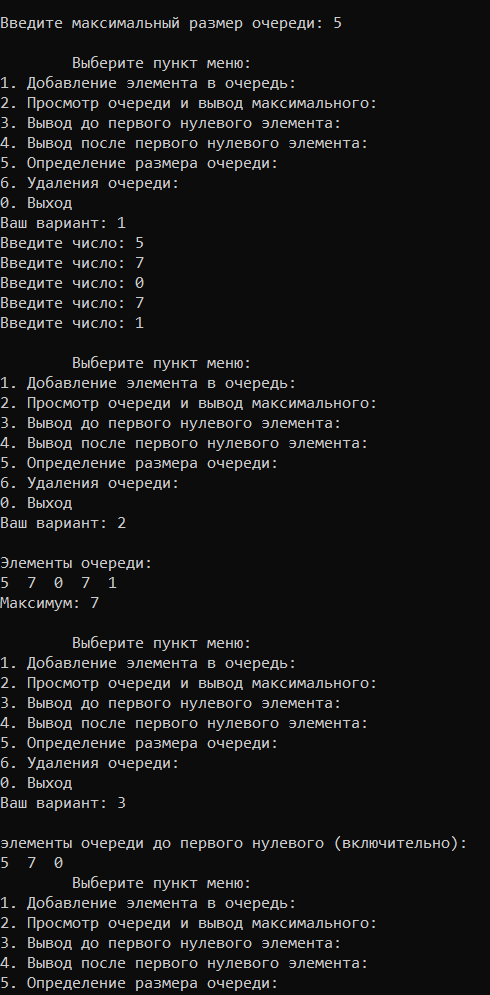
t = t->next;

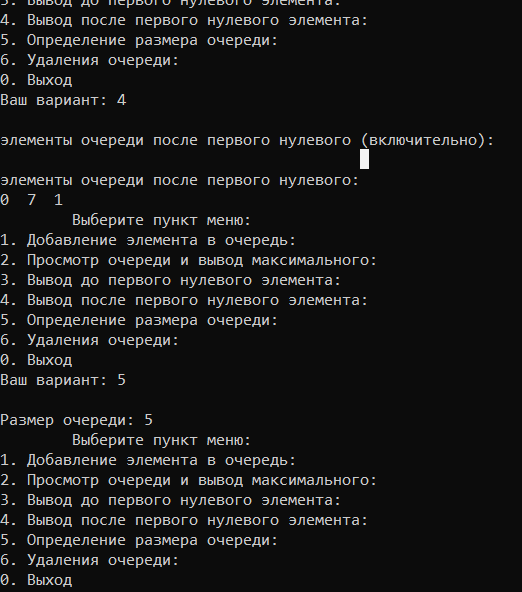
delete temp;

}

}

}

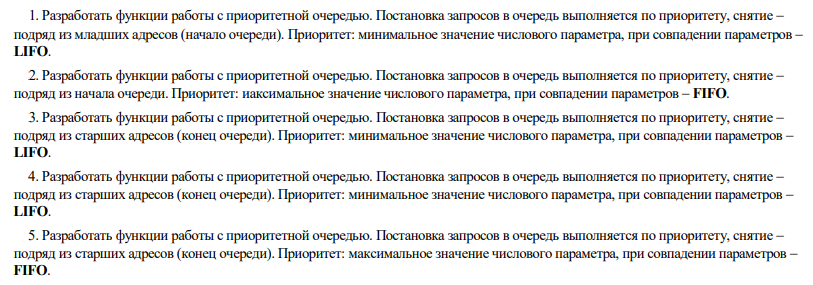


****

**Дополнительные задания:**

**Доп1:**

****

****

**1)**

#include<iostream>//Далее буду кидать только изменившуюся часть кода. И почему 5 заданий ради 3 баллов? Это нечестно

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 с наименьшим приоритетом

{

if (isNull())

{

cout << "Очередь пуста" << endl;

return;

}

Item\* minPrev = NULL; // Предыдущий элемент перед элементом с наименьшим приоритетом

Item\* minCurr = head; // Элемент с наименьшим приоритетом

Item\* prev = head; // Предыдущий элемент

Item\* curr = head->next; // Текущий элемент

while (curr != NULL)

{

if (curr->data < minCurr->data)

{

minPrev = prev;

minCurr = curr;

}

prev = curr;

curr = curr->next;

}

if (minPrev == NULL)

{

head = head->next;

}

else

{

minPrev->next = minCurr->next;

}

delete minCurr;

}

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

{

Item\* p = new Item;

p->data = x;

p->next = NULL;

if (isNull())

head = tail = p;

else

{

p->next = head;

head = 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();

}

void getFromHead() //Получение элемента из начала

{

if (isNull())

cout << "Очередь пуста" << endl;

else

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

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int i = 1, 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;

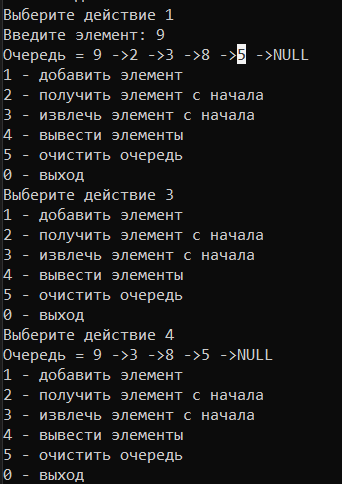
case 5: clrQueue(); break;

}

}

return 0;

}

****

**2)**

void deletFirst()

{

if (isNull())

{

cout << "Очередь пуста" << endl;

return;

}

Item\* maxPrev = NULL; // Предыдущий элемент перед элементом с наибольшим приоритетом

Item\* maxCurr = head; // Элемент с наибольшим приоритетом

Item\* prev = head; // Предыдущий элемент

Item\* curr = head->next; // Текущий элемент

while (curr != NULL)

{

if (curr->data > maxCurr->data)

{

maxPrev = prev;

maxCurr = curr;

}

prev = curr;

curr = curr->next;

}

if (maxPrev == NULL)

{

head = head->next;

}

else

{

maxPrev->next = maxCurr->next;

}

delete maxCurr;

}

**3)**

void delQueue()

{

if (isNull())

{

cout << "Очередь пуста" << endl;

return;

}

Item\* minPrev = NULL; // Предыдущий элемент перед элементом с наименьшим приоритетом

Item\* minCurr = head; // Элемент с наименьшим приоритетом

Item\* prev = head; // Предыдущий элемент

Item\* curr = head->next; // Текущий элемент

while (curr != NULL)

{

if (curr->data < minCurr->data)

{

minPrev = prev;

minCurr = curr;

}

prev = curr;

curr = curr->next;

}

if (minPrev == NULL)

{

head = head->next;

}

else

{

minPrev->next = minCurr->next;

}

delete minCurr;

}

**4)**

const int MAX\_SIZE = 100;

int priorityQueue[MAX\_SIZE];

int size = 0;

void deletFirst()

{

if (size == 0)

{

cout << "Очередь пуста" << endl;

return;

}

int minPriorityIndex = 0;

int minPriority = priorityQueue[0];

// Находим индекс элемента с наименьшим приоритетом

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

{

if (priorityQueue[i] < minPriority)

{

minPriorityIndex = i;

minPriority = priorityQueue[i];

}

}

// Сдвигаем остальные элементы влево, чтобы удалить элемент с наименьшим приоритетом

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

{

priorityQueue[i] = priorityQueue[i + 1];

}

size--;

cout << "Извлечен элемент: " << minPriority << endl;

}

**5)**

const int MAX\_SIZE = 100;

QueueElement priorityQueue[MAX\_SIZE];

int size = 0;

void deletFirst()

{

if (size == 0)

{

cout << "Очередь пуста" << endl;

return;

}

int maxPriorityIndex = 0;

int maxPriority = priorityQueue[0].priority;

// Находим индекс элемента с наибольшим приоритетом

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

{

if (priorityQueue[i].priority > maxPriority)

{

maxPriorityIndex = i;

maxPriority = priorityQueue[i].priority;

}

}

QueueElement firstElement = priorityQueue[maxPriorityIndex];

// Сдвигаем остальные элементы влево, чтобы удалить элемент с наибольшим приоритетом

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

{

priorityQueue[i] = priorityQueue[i + 1];

}

size--;

cout << "Извлечен элемент: " << firstElement.data << endl;

}

**2 доп:**



#include<iostream>//Далее буду кидать только изменившуюся часть кода. И почему 5 заданий ради 3 баллов? Это нечестно

using namespace std;

struct Item

{

float data;

Item\* next;

};

Item\* head, \* tail;

bool isNull(void) //Проверка на пустоту

{

return (head == NULL);

}

void deletFirst() // Извлечение элемента из начала с наименьшим приоритетом

{

if (isNull())

{

cout << "Очередь пуста" << endl;

return;

}

Item\* minPrev = NULL; // Предыдущий элемент перед элементом с наименьшим приоритетом

Item\* minCurr = head; // Элемент с наименьшим приоритетом

Item\* prev = head; // Предыдущий элемент

Item\* curr = head->next; // Текущий элемент

while (curr != NULL)

{

if (curr->data < minCurr->data)

{

minPrev = prev;

minCurr = curr;

}

prev = curr;

curr = curr->next;

}

if (minPrev == NULL)

{

head = head->next;

}

else

{

minPrev->next = minCurr->next;

}

delete minCurr;

}

void insertToQueue(float x) //Добавление элемента в очередь

{

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

p->data = x;

p->next = NULL;

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

Item\* p1 = new Item;

Item\* p2 = new Item;

int i = 0; //флажок

if (isNull())

head = tail = p;

else

{

p2 = head; p1 = head;

while (p1 != NULL) //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 == NULL)

{

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 != NULL)

{

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

p = p->next;

}

else

{

cout << "NULL" << endl;

return;

}

}

}

}

void clrQueue() //Очистка очереди

{

while (!isNull()) deletFirst();

}

void getFromHead() //Получение элемента из начала

{

if (isNull())

cout << "Очередь пуста" << endl;

else

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

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int i = 1, choice = 1;

float 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;

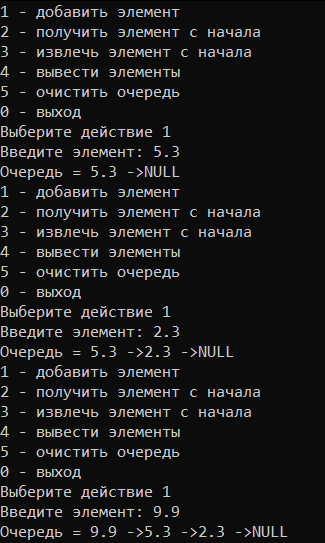
case 5: clrQueue(); break;

}

}

return 0;

}



**3 доп:**

****

#include <iostream>

#include <fstream>

#include <Windows.h>

#include <string>

using namespace std;

// Структура для представления узла в очереди

struct Node

{

char symbol; // Символ

Node\* next = nullptr; // Указатель на следующий узел

};

// Очередь

struct queue

{

Node\* first = nullptr; // Указатель на первый узел

Node\* last = nullptr; // Указатель на последний узел

};

// Функция для добавления элемента в очередь

void push(queue\*& q, char n)

{

if (q->first == nullptr)

{

q->first = q->last = new Node;

q->first->symbol = q->last->symbol = n;

return;

}

q->last = q->last->next = new Node;

q->last->symbol = n;

}

int main()

{

//setlocale(LC\_ALL, "Rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

queue\* Symbols = new queue; // Очередь для символов

queue\* Numbers = new queue; // Очередь для чисел

string buf;

ifstream Fin("f.txt"); // Открытие файла для чтения

ofstream Fout("g.txt"); // Открытие файла для записи

while (!Fin.eof())

{

getline(Fin, buf); // Чтение строки из файла

// Разбор строки на символы и добавление их в соответствующие очереди

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

{

if (buf[i] >= '0' && buf[i] <= '9')

{

push(Numbers, buf[i]); // Добавление цифры в очередь чисел

}

else

{

push(Symbols, buf[i]); // Добавление символа в очередь символов

}

}

// Запись символов из очереди символов в файл

while (Symbols->first != nullptr)

{

Fout.write((char\*)&Symbols->first->symbol, sizeof(char));

Symbols->first = Symbols->first->next;

}

Fout << ' '; // Добавление пробела между символами и числами

// Запись чисел из очереди чисел в файл

while (Numbers->first != nullptr)

{

Fout.write((char\*)&Numbers->first->symbol, sizeof(char));

Numbers->first = Numbers->first->next;

}

Fout << '\n'; // Переход на новую строку в файле

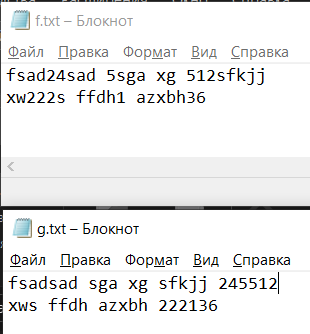
}

Fin.close(); // Закрытие файла чтения

Fout.close(); // Закрытие файла записи

return 0;

}

****