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(а):

Студент(ка) 1 курса 8 группы

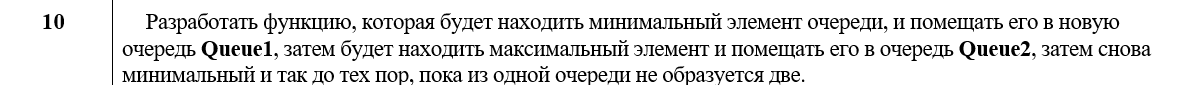
Семёнов Даниил Вячеславович

Преподаватель: асс. Андронова М.В.

Минск, 2024

4. 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 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;

}

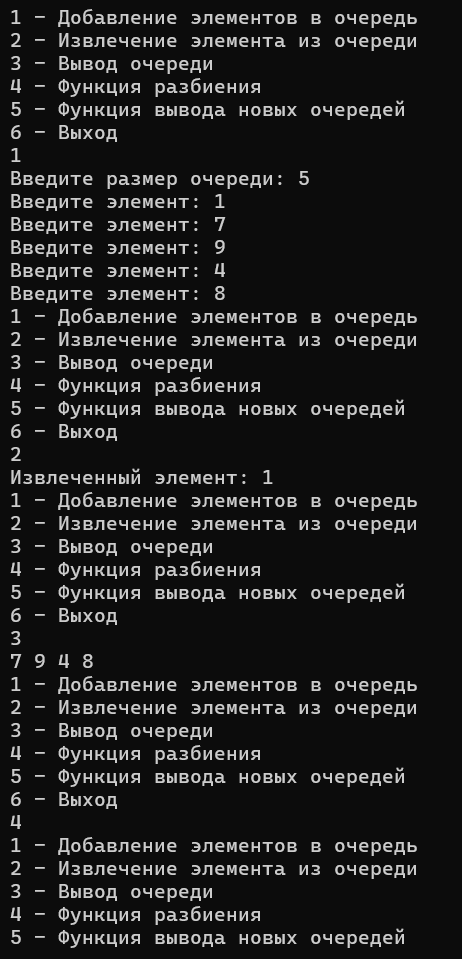
// Освобождаем узел

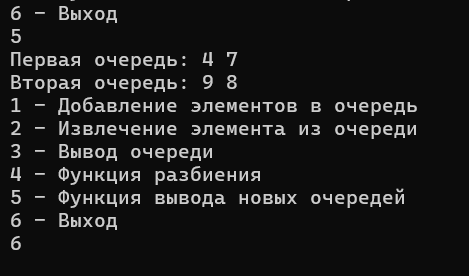
delete maxNode;

}

}

}





Доп. Задачи

2



//код, который я забыл добавить

#include <iostream>

using namespace std;

struct Queue

{

double info;

Queue\* next;

};

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

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

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

void cycle(Queue\*\* begin, Queue\*\* end); //функция циклического сдвига

int main()

{

setlocale(LC\_ALL, "rus");

int choice;

//указатели на начало каждой очереди

Queue\* begin = NULL, \* end;

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 циклического сдвига" << endl;

cout << "5 - Выход" << endl;

cin >> choice;

//анализируем choice

switch (choice)

{

case 1:

double 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

cycle(&begin, &end);

break;

}

} while (choice != 5);

return 0;

}

void create(Queue\*\* begin, Queue\*\* end, double 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;

}

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

{

Queue\* t = \*begin;

double val = t->info;

\*begin = t->next;

delete t;

return val;

}

void cycle(Queue\*\* begin, Queue\*\* end) // функция циклического сдвига

{

Queue\* t = \*begin;

Queue\* MAX = NULL;

int max = -DBL\_MIN;

while (t != NULL)

{

//ищем макисмальный элемент

if (max < t->info)

{

max = t->info;

MAX = t;

}

t = t->next;

}

if (MAX != nullptr)

{

// Перемещаем максимальный элемент в начало очереди

while (\*begin != MAX)

{

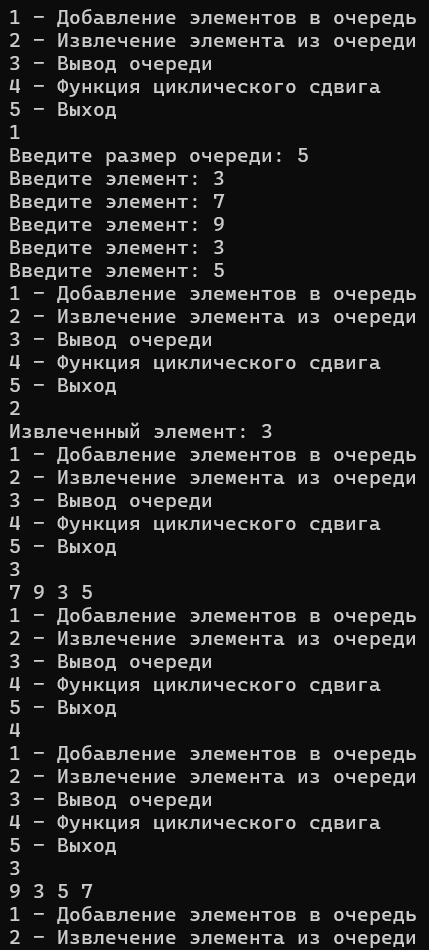
double element = fromFIFO(begin);

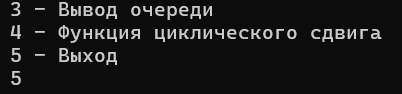
create(begin, end, element);

}

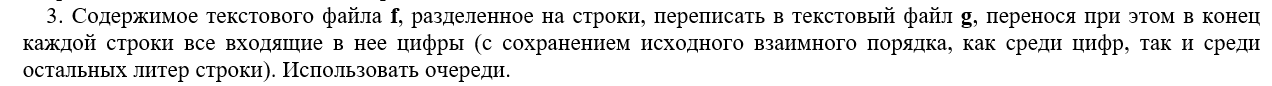
}

}





3



//в допах не сказано делать меню

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node

{

char data;

Node\* next;

};

struct Queue

{

// указатели на начало и конец очереди

Node\* head;

Node\* tail;

};

void CreateQueue(Queue\* q) // создаём пустую очередь

{

q->head = NULL;

q->tail = NULL;

}

void Enqueue(Queue\* q, char data) //функция для добавления элемента в конец очрееди

{

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (q->tail == NULL)

{

q->head = newNode;

q->tail = newNode;

}

else

{

q->tail->next = newNode;

q->tail = newNode;

}

}

void Symbol(Queue\* Letter, Queue\* Number) // функция для отеделения цифр

{

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

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

string line;

char symbol;

getline(f, line);

cout << "Исходная строка: " << line << endl;

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

{

symbol = line[i];

if (symbol >= 48 && symbol <= 57) // если символ цифра

{

Enqueue(Number, symbol); // в конец очереди с цифрами

}

else //иначе

{

Enqueue(Letter, symbol); // в конец очереди с буквами

}

}

Node\* line1 = Letter->head;

Node\* line2 = Number->head;

cout << "Буквы: " << endl;

while (line1)

{

// вывод букв и запись в файл

cout << line1->data;

g << line1->data;

line1 = line1->next;

}

cout << endl;

cout << "Цифры: " << endl;

while (line2)

{

// вывод цифр и запись в файл

cout << line2->data;

g << line2->data;

line2 = line2->next;

}

cout << endl;

f.close();

g.close();

}

int main()

{

setlocale(LC\_ALL, "RU");

// создаём очереди для букв и цифр

Queue Letter;

Queue Number;

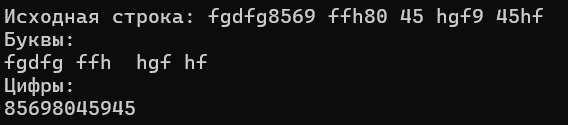
CreateQueue(&Letter);

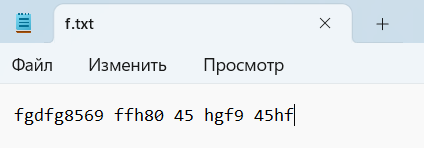
CreateQueue(&Number);

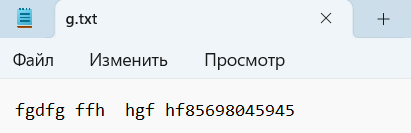
Symbol(&Letter, &Number); // функция разделения на буквы и цифры

return 0;

}







1



#include<iostream>

using namespace std;

struct Item

{

int data;

Item\* next;

};

Item\* head, \* tail;

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 deletlast() //Извлечение элемента из конца

{

if (isNull())

cout << "Очередь пуста" << endl;

else

{

Item\* p;

tail->data = NULL;

}

}

void getFromHead() //Получение элемента из начала

{

if (isNull())

cout << "Очередь пуста" << endl;

else

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

}

void getFromTail() //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; //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; //флажок

if (isNull())

head = tail = p;

else

{

p2 = head; p1 = head; // !!!Чтобы реализовать ввод LiFo необходимо чтобы указатель p2 был позади, FiFo наоборот

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

{

if (i == 1) //ТАКЖЕ НЕОБХОДИМЫ ДЛЯ РЕАЛИЗАЦИИ LiFo

{

if (i == 1)

{

if (x > p1->data) //число больше, чем в очереди

{

v->data = x;

v->next = p1;

p2->next = v;

return;

}

if (x == p1->data)

{

v->data = x;

v->next = p1->next;

p1->next = v;

return;

}

p2 = p2->next;

}

if (x > p1->data)

{

v->data = x;

v->next = p1;

head = v;

return;

}

if (p1->next == nullptr)

{

v->data = x;

v->next = nullptr;

tail->next = v;

tail = v;

return;

}

i = 1;

p1 = p1->next; // следующее число

}

else // else необходимое для реализации вставки LiFo

{

if (x >= p1->data)

{

v->data = x;

v->next = p2;

head = v;

return;

}

}

p1 = p1->next; //ТАКЖЕ НЕОБХОДИМЫ ДЛЯ РЕАЛИЗАЦИИ LiFo

i = 1;

}

if (p1 == NULL) // Выражение ДЛЯ РЕАЛИЗЦИИ LiFo

{

v->data = x;

v->next = tail;

p2->next = v;

}

}

}

void printQueue() //Вывод очереди

{

int g;

Item\* p = new Item;

if (isNull())

cout << "Очередь пуста" << endl;

else

{

cout << "Очередь = ";

p = head;

while (!isNull())

{

if (p != nullptr)

{

g = p->data;

if (g == 0) {

}

else { 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;

int x;

head = nullptr; tail = nullptr;

do

{

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 << "6 - получить элемент с конца" << endl;

cout << "7 - извлечь элемент с конца" << endl;

cout << "8 - выход" << endl;

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

switch (choice)

{

case 1: cout << "Введите элемент: "; cin >> x;

insertToQueue(x); printQueue(); break;

case 2: getFromHead(); break;

case 3: deletFirst(); break;

case 4: printQueue(); break;

case 5: clrQueue(); break;

case 6: getFromTail(); break;

case 7: deletlast(); break;

}

}while (choice != 8);

return 0;

}

