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

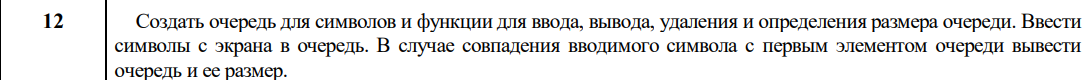
Статько Герман Вячеславович

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 12**

****

#include <iostream>

using namespace std;

// Структура для элемента очереди

struct queue

{

char symbol; // Символ, хранимый в элементе

queue\* next; // Указатель на следующий элемент

};

// Функция для создания нового элемента очереди

void create(queue\*\*, queue\*\*, char);

// Функция для вывода всех элементов очереди

void view(queue\*);

// Функция для удаления элемента по номеру

void remove(queue\*\*, queue\*\*, int);

int size\_count = 1; // Глобальная переменная для хранения размера очереди

int main()

{

setlocale(LC\_CTYPE, "rus");

char choice, p, element;

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

int size, index;

// Вывод меню

cout << "1 - основное задание" << endl

<< "2 - вывести элементы очереди" << endl

<< "3 - удалить элемент" << endl

<< "4 - вывести размер очереди" << endl

<< "5 - выход" << endl;

do

{

cout << "?: ";

cin >> choice;

switch (choice)

{

case '1':

{

t = new queue;

cout << "Введите максимальный размер очереди: ";

cin >> size;

cout << "Введите символ: ";

cin >> p;

t->symbol = p;

t->next = nullptr;

begin = end = t;

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

{

cout << "Введите символ: ";

cin >> p;

create(&begin, &end, p);

if (p == begin->symbol)

{

view(begin);

cout << "Размер: " << size\_count << endl;

break;

}

}

break;

}

case '2':

if (begin == nullptr)

{

cout << "Элементы отсутствуют" << endl;

}

else

{

view(begin);

}

break;

case '3':

cout << "Введите номер элемента, который нужно удалить: ";

cin >> index;

remove(&begin, &end, index);

break;

case '4':

cout << "Размер: " << size\_count << endl;

break;

}

} while (choice != '5');

}

// Функция для добавления нового элемента в конец очереди

void create(queue\*\* begin, queue\*\* end, char p)

{

queue\* t = new queue;

t->next = nullptr;

if (\*begin == nullptr)

{

\*begin = \*end = t; // Устанавливаем начало и конец очереди на новый элемент

}

else

{

t->symbol = p; // Устанавливаем символ в новом элементе

(\*end)->next = t; // Устанавливаем указатель на следующий элемент в текущем последнем элементе

\*end = t; // Обновляем указатель на конец очереди

size\_count++; // Увеличиваем размер очереди

}

}

// Функция для вывода всех элементов очереди

void view(queue\* begin)

{

queue\* t = begin; // Указатель для перебора элементов начинаем с начала очереди

while (t != nullptr)

{

cout << t->symbol << " "; // Выводим символ текущего элемента

t = t->next; // Переходим к следующему элементу

}

cout << endl;

}

// Функция для удаления элемента по номеру

void remove(queue\*\* begin, queue\*\* end, int index)

{

if (\*begin == nullptr)

{

cout << "Очередь пуста" << endl;

return;

}

if (index == 1)

{

queue\* temp = \*begin; // Временный указатель на первый элемент

\*begin = temp->next; // Переназначаем начало очереди на следующий элемент после первого

delete temp; // Освобождаем память первого элемента

size\_count--; // Уменьшаем размер очереди

return;

}

queue\* current = \*begin; // Текущий элемент, начинаем с начала очереди

queue\* prev = nullptr; // Предыдущий элемент, изначально пустой

int count = 1; // Счетчик текущего элемента

while (current != nullptr && count < index)

{

prev = current; // Предыдущий становится текущим

current = current->next; // Переходим к следующему элементу

count++; // Увеличиваем счетчик

}

if (current == nullptr)

{

cout << "Элемент с указанным номером не найден" << endl;

return;

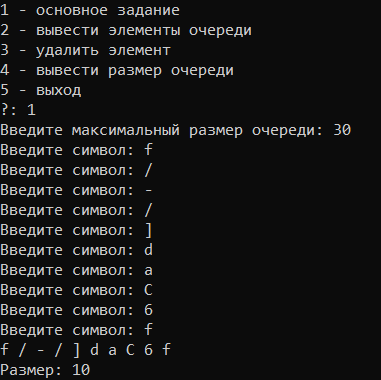
}

prev->next = current->next; // Пропускаем текущий элемент при связывании предыдущего и следующего

delete current; // Освобождаем память текущего элемента

size\_count--; // Уменьшаем размер очереди

}

****

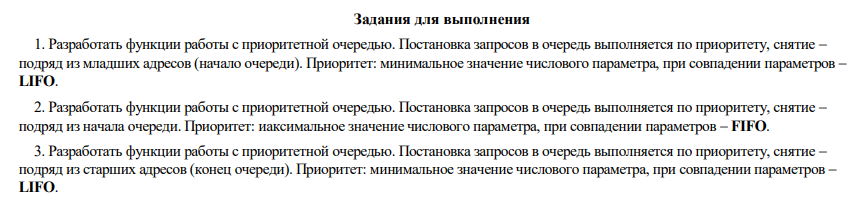
****

****

****

**Доп. Задания**

**1**

****

**Сразу уточню, я очень надеюсь, что выполнил задание правильно**

**НО если бы методичка была чуть понятнее, то жить было бы проще**

**Если программа работает неправильно, я бы попросил вас пояснить, что я сделал не так**

#include <iostream>

#include <locale>

using namespace std;

// Структура элемента очереди

struct Item {

int data; // Данные элемента

Item\* next; // Указатель на следующий элемент

};

// Прототипы функций

void insertAscending(int); // Вставка элемента в порядке возрастания

void insertDescending(int); // Вставка элемента в порядке убывания

void printQueue(); // Вывод очереди

void removeBeginningMinLIFO(); // Удаление элемента из начала по минимальному значению с соблюдением LIFO

void removeBeginningMaxFIFO(); // Удаление элемента из начала по максимальному значению с соблюдением FIFO

void removeEndMinLIFO(); // Удаление элемента из конца по минимальному значению с соблюдением LIFO

Item\* head = nullptr; // Указатель на начало очереди

Item\* tail = nullptr; // Указатель на конец очереди

int main() {

setlocale(LC\_CTYPE, "rus"); // Установка локали для поддержки кириллицы

char choice; // Переменная для выбора действия

int num; // Переменная для ввода числа

// Вывод меню на экран

cout << "1 - стандартное добавление элемента" << endl

<< "2 - вывести элементы очереди" << endl

<< "3 - получить элемент - начало" << endl

<< "4 - получить элемент - конец" << endl

<< "5 - снятие - начало, постановка - минимальное значение, совпадение - LIFO" << endl

<< "6 - снятие - начало, постановка - максимальное значение, совпадение - FIFO" << endl

<< "7 - снятие - конец, постановка - минимальное значение, совпадение - LIFO" << endl

<< "8 - выход" << endl;

do {

cout << "?: ";

cin >> choice;

switch (choice) {

case '1':

cout << "Введите число: ";

cin >> num;

insertAscending(num); // Вставка элемента в порядке возрастания

break;

case '2':

printQueue(); // Вывод очереди

break;

case '3':

if (head != nullptr) {

cout << "Начало: " << head->data << endl; // Вывод начала очереди

}

else {

cout << "Очередь пуста." << endl;

}

break;

case '4':

if (tail != nullptr) {

cout << "Конец: " << tail->data << endl; // Вывод конца очереди

}

else {

cout << "Очередь пуста." << endl;

}

break;

case '5':

cout << "Введите число: ";

cin >> num;

insertDescending(num); // Вставка элемента в порядке убывания

removeBeginningMinLIFO(); // Удаление элемента из начала по минимальному значению с соблюдением LIFO

break;

case '6':

cout << "Введите число: ";

cin >> num;

insertDescending(num); // Вставка элемента в порядке убывания

removeBeginningMaxFIFO(); // Удаление элемента из начала по максимальному значению с соблюдением FIFO

break;

case '7':

cout << "Введите число: ";

cin >> num;

insertDescending(num); // Вставка элемента в порядке убывания

removeEndMinLIFO(); // Удаление элемента из конца по минимальному значению с соблюдением LIFO

break;

}

} while (choice != '8'); // Выход из цикла при выборе опции

"выход"

}

// Функция вставки элемента в порядке возрастания

void insertAscending(int x) {

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

p->data = x; // Присвоение данных новому элементу

p->next = nullptr; // Инициализация указателя на следующий элемент

if (head == nullptr) {

head = tail = p; // Если очередь пуста, новый элемент становится началом и концом очереди

}

else {

tail->next = p; // Вставка элемента в конец очереди

tail = p; // Обновление указателя на конец очереди

}

}

// Функция вставки элемента в порядке убывания

void insertDescending(int x) {

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

p->data = x; // Присвоение данных новому элементу

p->next = nullptr; // Инициализация указателя на следующий элемент

if (head == nullptr) {

head = tail = p; // Если очередь пуста, новый элемент становится началом и концом очереди

}

else {

if (x > head->data) { // Если новый элемент больше начала очереди

p->next = head; // Новый элемент становится началом очереди

head = p; // Обновление указателя на начало очереди

}

else {

Item\* prev = nullptr;

Item\* curr = head;

while (curr != nullptr && x < curr->data) { // Поиск места для вставки нового элемента

prev = curr;

curr = curr->next;

}

p->next = curr; // Вставка нового элемента

prev->next = p;

if (curr == nullptr) {

tail = p; // Обновление указателя на конец очереди

}

}

}

}

// Функция вывода очереди

void printQueue() {

if (head == nullptr) {

cout << "Очередь пуста" << endl;

}

else {

cout << "Очередь: ";

Item\* temp = head;

while (temp != nullptr) { // Перебор всех элементов очереди

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

}

// Функция удаления элемента из начала по минимальному значению с соблюдением LIFO

void removeBeginningMinLIFO() {

if (head != nullptr) {

Item\* temp = head;

Item\* minNode = head;

Item\* prev = nullptr;

Item\* prevMinNode = nullptr;

while (temp->next != nullptr) { // Поиск минимального элемента и его предыдущего элемента

if (temp->next->data < minNode->data) {

minNode = temp->next;

prevMinNode = temp;

}

temp = temp->next;

}

if (prevMinNode != nullptr) {

prevMinNode->next = minNode->next; // Удаление минимального элемента

if (minNode == tail) {

tail = prevMinNode;

}

}

else {

head = head->next; // Если минимальный элемент является началом очереди

if (head == nullptr) {

tail = nullptr;

}

}

delete minNode;

}

else {

cout << "Очередь пуста." << endl;

}

}

// Функция удаления элемента из начала по максимальному значению с соблюдением FIFO

void removeBeginningMaxFIFO() {

if (head != nullptr) {

Item\* temp = head;

Item\* maxNode = head;

Item\* prev = nullptr;

while (temp->next != nullptr) { // Поиск максимального элемента

if (temp->next->data > maxNode->data) {

maxNode = temp->next;

prev = temp;

}

temp = temp->next;

}

if (prev != nullptr) {

prev->next = maxNode->next; // Удаление максимального элемента

if (maxNode == tail) {

tail = prev;

}

}

else {

head = head->next; // Если максимальный элемент является началом очереди

if (head == nullptr) {

tail = nullptr;

}

}

delete maxNode;

}

else {

cout << "Очередь пуста." << endl;

}

}

// Функция удаления элемента из конца по минимальному значению с соблюдением LIFO

void removeEndMinLIFO() {

if (head != nullptr) {

Item\* prev = nullptr;

Item\* temp = head;

while (temp->next != nullptr) { // Поиск последнего элемента

prev = temp;

temp = temp->next;

}

if (prev != nullptr) {

prev->next = nullptr; // Удаление последнего элемента

tail = prev;

}

else {

head = tail = nullptr; // Если очередь содержит только один элемент

}

delete temp;

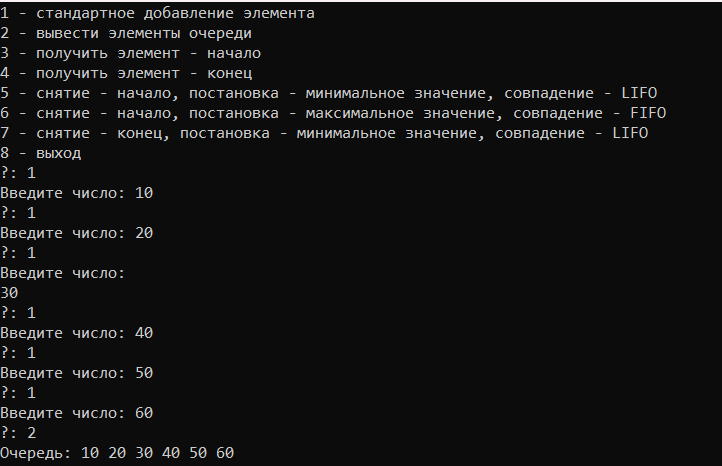
}

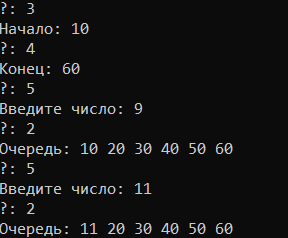
else {

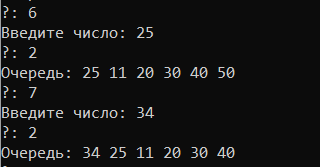
cout << "Очередь пуста." << endl;

}

}

****

****

****

**2**

****

#include <iostream>

using namespace std;

//Структура очереди

struct queue

{

int num;

queue\* next;

};

void create(queue\*\*, queue\*\*, int);//Прототип для создания очереди

void view(queue\*);//Прототип для просмотра очереди

void findMax(queue\*\*);//Прототип для нахождения максимального числа и сдвига

int main()

{

setlocale(LC\_CTYPE, "rus");

queue\* begin = nullptr, \* end, \* t;

int size, num;

cout << "Введите размер очереди: ";

cin >> size;

t = new queue;

cout << "Введите число: ";

cin >> num;

t->num = num;

t->next = nullptr;

begin = end = t;

//Заполение очереди

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

{

cout << "Введите число: ";

cin >> num;

create(&begin, &end, num);

}

view(begin);

findMax(&begin);

view(begin);

// Освобождаем память

while (begin != nullptr)

{

queue\* temp = begin;

begin = begin->next;

delete temp;

}

return 0;

}

void create(queue\*\* begin, queue\*\* end, int x)

{

queue\* t = new queue;

t->num = x;

t->next = nullptr;

if (\*begin == nullptr)

{

\*begin = \*end = t;

}

else

{

(\*end)->next = t;

\*end = t;

}

}

void view(queue\* begin)

{

queue\* t = begin;

cout << "Очередь: ";

if (t == nullptr)

{

cout << "Очередь пуста." << endl;

}

else

{

while (t != nullptr)

{

cout << t->num << " -> ";

t = t->next;

}

cout << endl;

}

}

void findMax(queue\*\* begin)

{

if (\*begin == nullptr)

{

cout << "Очередь пуста." << endl;

return;

}

int max = (\*begin)->num;

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

queue\* prevMax = \*begin;

while (current != nullptr)

{

if (current->num > max)

{

max = current->num;

prevMax = current;

}

current = current->next;

}

if (prevMax != \*begin)

{

// Удалим максимальный элемент из середины

queue\* temp = prevMax->next;

prevMax->next = temp->next;

delete temp;

// Вставим его в начало

queue\* newMax = new queue;

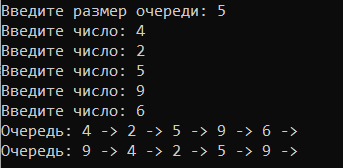
newMax->num = max;

newMax->next = \*begin;

\*begin = newMax;

}

}

****

**3**

****

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

// Определение структуры для связанного списка

struct Item {

char data;

Item\* next;

};

// Инициализация указателей head и tail

Item\* head = nullptr;

Item\* tail = nullptr;

// Проверка, пустой ли список

bool isNull() {

return (head == nullptr);

}

// Вставка элемента в очередь

void insertToQueue(char x) {

Item\* p = new Item;

p->data = x;

p->next = nullptr;

if (isNull()) {

head = tail = p;

}

else {

tail->next = p;

tail = p;

}

}

// Обработка строки ввода и вставка цифр в очередь

void processLine(string& line) {

for (char ch : line) {

if (isdigit(ch)) {

insertToQueue(ch);

}

}

}

// Печать содержимого очереди в выходной файл

void printQueue(ofstream& out) {

if (!isNull()) {

Item\* p = head;

while (p != nullptr) {

out << p->data;

p = p->next;

}

}

}

int main() {

setlocale(LC\_CTYPE, "Russian");

// Открытие входного и выходного файлов

ifstream inFile("f.txt");

ofstream outFile("g.txt");

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

cerr << "Ошибка открытия файла." << endl;

return 1;

}

string line;

while (getline(inFile, line)) {

processLine(line);

outFile << line;

printQueue(outFile);

outFile << endl;

}

// Закрытие файлов

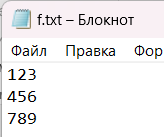
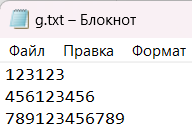
inFile.close();

outFile.close();

cout << "Данные успешно записаны." << endl;

return 0;

}

** **