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

Жамойдо Артём Игоревич

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 4**

Создать очередь для символов и функции для ввода, вывода и определения размера очереди. Ввести символы с экрана в очередь. В случае совпадения вводимого символа с последним элементом очереди удалить первый элемент.

#include <iostream>

using namespace std;

struct Queue

{

char data;

Queue\* next;

};

// Функция для добавления элемента в очередь

void enqueue(char x, Queue\*& front, Queue\*& rear)

{

Queue\* newNode = new Queue;

newNode->data = x;

newNode->next = nullptr;

if (front == nullptr)

{

front = rear = newNode;

}

else

{

rear->next = newNode;

rear = newNode;

}

}

// Функция для удаления элемента из очереди

char dequeue(Queue\*& front, Queue\*& rear)

{

if (front == nullptr)

{

cout << "Очередь пуста.\n";

return '\0';

}

Queue\* temp = front;

char data = front->data;

if (front == rear)

{

front = rear = nullptr; // Очищаем очередь, если в ней был один элемент

}

else

{

front = front->next; // Перемещаем front на следующий элемент

}

delete temp;

return data;

}

// Функция для вывода элементов в очереди

void display(Queue\* front)

{

if (front == nullptr)

{

cout << "Очередь пуста.\n";

return;

}

cout << "Элементы очереди: ";

Queue\* current = front;

while (current != nullptr)

{

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

current = current->next;

}

cout << endl;

}

int main()

{

setlocale(0, "ru");

Queue\* front = nullptr;

Queue\* rear = nullptr;

int maxSize;

cout << "Введите максимальный размер очереди: ";

cin >> maxSize;

char input;

while (maxSize > 0)

{

cout << "Введите элемент очереди: ";

cin >> input;

if (rear != nullptr && input == rear->data)

{

cout << "Совпадение с последним элементом. Удаление первого элемента.\n";

enqueue(input, front, rear); // Добавление нового элемента

dequeue(front, rear); // Удаление первого элемента

}

else

{

enqueue(input, front, rear);

maxSize--;

}

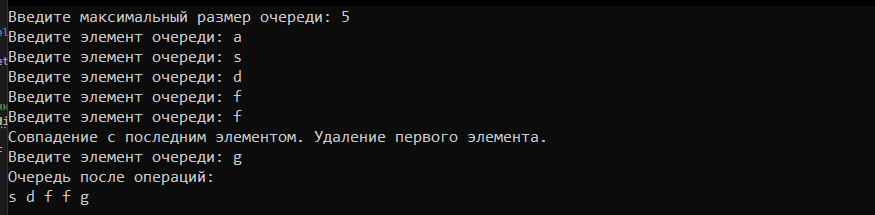
}

cout << "Очередь после операций:\n";

display(front);

return 0;

}

****

Доп. задачи

1. В приложении 1 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.

#include <iostream>

using namespace std;

struct Item

{

int data;

Item\* next;

};

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

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

// Проверка, пуста ли очередь

bool isNull()

{

return (head == nullptr);

}

// Удаление элемента из начала очереди

void deletFirst()

{

if (isNull())

{

cout << "Очередь пуста" << endl;

}

else

{

Item\* p = head;

head = head->next;

delete p;

}

}

// Вывод элемента из начала очереди

void getFromHead()

{

if (isNull())

{

cout << "Очередь пуста" << endl;

}

else {

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

}

}

void insertWithPriority(int x)

{

Item\* p = new Item;

p->data = x;

p->next = nullptr;

if (isNull())

{

head = tail = p; // Вставка в пустую очередь

}

else if (x <= head->data) // Вставка в начало очереди

{

p->next = head;

head = p;

}

else

{

Item\* prev = head;

Item\* current = head->next;

// Находим место для вставки с учетом приоритета

while (current != nullptr && x > current->data)

{

prev = current;

current = current->next;

}

// Вставка элемента

p->next = current;

prev->next = p;

if (current == nullptr)

{

tail = p; // Обновление указателя на конец очереди

}

}

}

void printQueue()

{

Item\* p = head;

if (isNull())

{

cout << "Очередь пуста" << endl;

}

else

{

cout << "Очередь = ";

while (p != nullptr)

{

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

cout << "->";

p = p->next;

}

cout << "NULL" << endl;

}

}

void clrQueue()

{

while (!isNull())

{

deletFirst(); // Очистка очереди путем удаления всех элементов

}

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int choice = 1, z;

head = nullptr;

tail = nullptr;

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;

insertWithPriority(z); // Вставка элемента с учетом приоритета

printQueue(); // Вывод элементов после вставки

break;

case 2:

getFromHead(); // Вывод элемента из начала очереди

break;

case 3:

deletFirst(); // Извлечение элемента из начала очереди

break;

case 4:

printQueue(); // Вывод всех элементов очереди

break;

case 5:

clrQueue(); // Очистка очереди

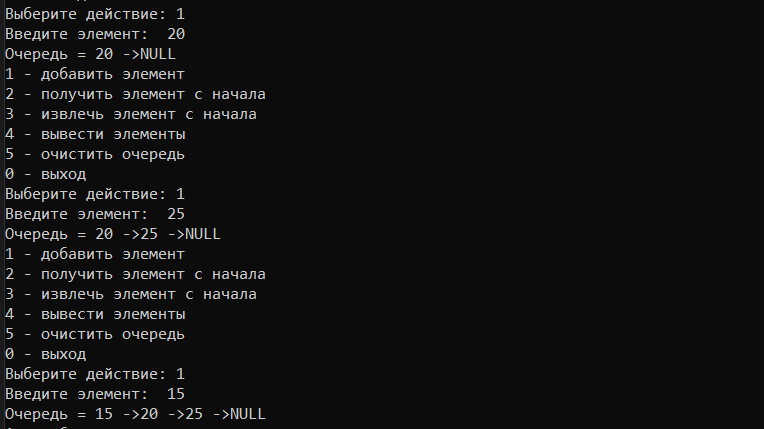
break;

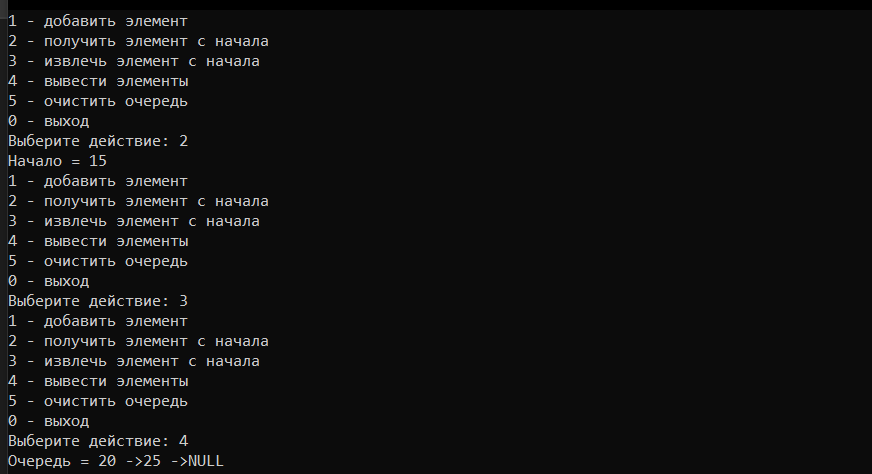
}

}

return 0;

}





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.

#include <iostream>

using namespace std;

struct Item

{

double data;

Item\* next;

};

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

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

bool isNull()

{

return (head == nullptr); // Проверка, пуста ли очередь

}

// Удаление элемента из начала очереди

void deleteFirst()

{

if (isNull())

{

cout << "Очередь пуста" << endl;

}

else

{

Item\* p = head;

head = head->next;

delete p;

}

}

// Вывод элемента из начала очереди

void getFromHead()

{

if (isNull())

{

cout << "Очередь пуста" << endl;

}

else

{

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

}

}

void insertWithPriority(double x)

{

Item\* p = new Item;

p->data = x;

p->next = nullptr;

if (isNull()) // Вставка в пустую очередь

{

head = tail = p;

}

else if (x >= head->data) // Вставка в начало очереди

{

p->next = head;

head = p;

}

else

{

Item\* prev = head;

Item\* current = head->next;

// Находим место для вставки с учетом приоритета

while (current != nullptr && x < current->data)

{

prev = current;

current = current->next;

}

// Вставка элемента

p->next = current;

prev->next = p;

if (current == nullptr)

{

tail = p; // Обновление указателя на конец очереди

}

}

}

void printQueue()

{

Item\* p = head;

if (isNull())

{

cout << "Очередь пуста" << endl;

}

else

{

cout << "Очередь = ";

while (p != nullptr)

{

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

cout << "->";

p = p->next;

}

cout << "NULL" << endl;

}

}

void clearQueue()

{

while (!isNull())

{

deleteFirst(); // Очистка очереди путем удаления всех элементов

}

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int choice = 1;

double z;

head = nullptr;

tail = nullptr;

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;

insertWithPriority(z); // Вставка элемента с учетом приоритета

printQueue(); // Вывод элементов после вставки

break;

case 2:

getFromHead(); // Вывод элемента из начала очереди

break;

case 3:

deleteFirst(); // Извлечение элемента из начала очереди

break;

case 4:

printQueue(); // Вывод всех элементов очереди

break;

case 5:

clearQueue(); // Очистка очереди

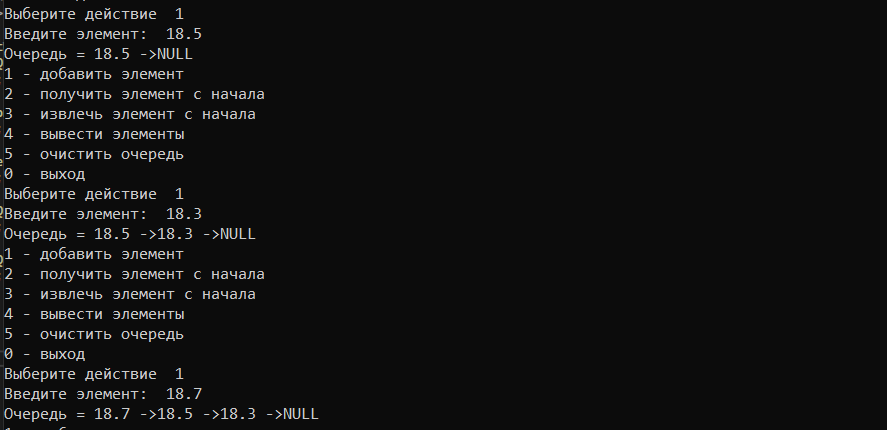
break;

}

}

return 0;

}



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.

#include <iostream>

#include <fstream>

using namespace std;

struct Queue

{

char data[255]; // Массив данных

int head; // Указатель на начало

};

void createQueue(Queue\* Q) // Создание очереди Q

{

Q->head = NULL;

}

bool isEmpty(Queue\* Q) // Проверка очереди на пустоту

{

if (Q->head == 0)

return true;

else

return false;

}

void showQueue(Queue\* Q)

{

ofstream fout("g.txt", ios\_base::app); // Открытие файла для записи

// Вывод содержимого очереди в файл

for (int i = 0; i < Q->head; i++)

{

fout << Q->data[i];

}

Q->head = 0;

fout.close(); // Закрытие файла

}

void addToQueue(Queue\* Q, char value) // Добавление элемента в очередь

{

Q->data[Q->head++] = value;

}

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

int main()

{

ofstream fout("g.txt", ios\_base::trunc); // Открытие файла для записи

fout.close(); // Закрытие файла

setlocale(LC\_ALL, "Rus");

Queue Q1, Q2;

createQueue(&Q1); // Создание очереди Q1

createQueue(&Q2); // Создание очереди Q2

while (!fin.eof())

{

char s[255] = "";

fin.getline(s, 253); // Чтение строки из файла

// Разделение символов на две очереди

for (int i = 0; i < strlen(s); i++)

{

if ((s[i] < '0') || (s[i] > '9')) // Если символ не является цифрой

{

addToQueue(&Q1, s[i]);

}

else

{

addToQueue(&Q2, s[i]);

}

}

showQueue(&Q1); // Вывод содержимого очереди Q1 в файл

showQueue(&Q2); // Вывод содержимого очереди Q2 в файл

ofstream fout("g.txt", ios\_base::app);

fout << endl;

fout.close();

}

fin.close();

return 0;

}

