МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФБГОУ ВО «ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра «Программная инженерия»

**Отчёт**

По лабораторной работе №1

**«Абстрактный тип данных на примере**

**реализации очереди в последовательной и связной памяти»**

**Выполнили:**

Студенты группы 92-ПГ

Погосян Ж.В.

**Проверили:**

Ужаринский А.Ю.

Орёл — 2021

**АТД:**

*АТД: Queue [1 .. MaxEl] of ElType {FIFO}*

*Операторы: InitQueue (Q: Queue)*

*EnQueue (Q: Queue, x: ElType): bool*

*DeQueue (Q: Queue; x: ElType): bool*

*GetAll(Q: Queue) : array[0..MaxEl] of ElType*

*GetFirst(Q: Queue): ^ElType*

*Empty (Q: Queue): bool*

*Full (Q: Queue): bool*

**Код программы:**

**Main:**

#include <iostream>

#include <queue>

#include <string.h>

#include <vector>

#include "queue.h"

//#include "queue2.h"

using namespace std;

struct detail {

char name[5];

unsigned time;

};

int main() {

setlocale(LC\_ALL, "rus");

Queue<detail> queue;

queue.InitQueue();

char input;

while (true) {

cout << "1 - поставить деталь на обработку" << endl;

cout << "2 - сделать шаг модельного времени" << endl;

cout << "3 - снять обрабатывающуюся деталь" << endl;

cout << "4 - вывести список обрабатывающихся деталей" << endl;

cout << "5 - сбросить моделирование" << endl;

cout << "6 - выйти" << endl;

cin >> input;

switch (input) {

case '1': {

detail det;

cout << "Введите номер детали (максимум 4 цифры): ";

cin >> det.name;

cout << "Введите время, требующееся для обработки данной детали: ";

cin >> det.time;

if (queue.EnQueue(det))

cout << "Деталь успешное добавлена" << endl;

else

cout << "Очередь переполнена" << endl;

break;

}

case '2': {

detail\* det;

det = queue.GetFirst();

if (det != nullptr) {

det->time--;

if (det->time == -1) {

detail tmp;

queue.DeQueue(tmp);

cout << "Деталь " << tmp.name << " успешно обработана" << endl;

}

else {

cout << "Деталь " << det->name << " все еще обрабатывается" << endl;

cout << "Осталось " << det->time << " шагов" << endl;

}

}

else {

cout << "Очередь пуста, вы тратите время впустую!!!" << endl;

}

break;

}

case '3': {

detail det;

if (queue.DeQueue(det)) {

cout << "Деталь " << det.name << " была принудительно снята с производства" << endl;

}

else {

cout << "Очередь пуста" << endl;

}

break;

}

case '4': {

vector<detail> container = queue.GetAll();

for (auto det : container)

cout << "Деталь " << det.name << endl;

break;

}

case '5': {

queue.InitQueue();

cout << "Очередь деталей сброшена" << endl;

break;

}

case '6': {

return 0;

}

}

}

}

**Модуль 1:**

#pragma once

#include <vector>

//----------Очередь в последовательной памяти---------

template <typename T>

class Queue {

public:

void InitQueue();

bool EnQueue(T elem);

bool DeQueue(T& result);

std::vector<T> GetAll();

T\* GetFirst();

bool Empty();

bool Full();

private:

T element[6];

int head, tail;

};

//----------Очередь в последовательной памяти---------

template <typename T>

void Queue<T>::InitQueue() {

head = tail = 0;

};

template <typename T>

bool Queue<T>::EnQueue(T elem) {

if (this->Full()) {

return false;

}

else {

this->element[this->tail] = elem;

this->tail = (this->tail + 1) % 6;

}

return true;

};

template <typename T>

bool Queue<T>::DeQueue(T& result) {

if (this->Empty()) {

return false;

}

else {

result = element[head];

this->head = (this->head + 1) % 6;

}

return true;

};

template <typename T>

std::vector<T> Queue<T>::GetAll() {

std::vector<T> container;

int head\_back = head;

int tail\_back = tail;

int counter = 0;

while (tail != head) {

container.push\_back(element[head]);

head = (head + 1) % 6;

}

head = head\_back;

tail = tail\_back;

return container;

};

template <typename T>

T\* Queue<T>::GetFirst() {

if (Empty())

return nullptr;

return &element[head];

}

template <typename T>

bool Queue<T>::Empty() {

if (this->head == this->tail)

return true;

else

return false;

};

template <typename T>

bool Queue<T>::Full() {

if (this->head == (this->tail + 1) % 6)

return true;

else

return false;

};

**Модуль 2:**

//----------Очередь в связной памяти---------

#include <vector>

template <typename T>

class Queue {

public:

void InitQueue();

bool EnQueue(T elem);

bool DeQueue(T& result);

std::vector<T> GetAll();

T\* GetFirst();

bool Empty();

~Queue();

private:

template <typename K>

class Node {

public:

Node(K \_data, Node<K>\* \_next = nullptr) :

data(\_data),

next(\_next) {};

K data;

Node<K>\* next;

};

Node<T>\* head;

Node<T>\* tail;

};

//----------Очередь в связной памяти---------

template <typename T>

void Queue<T>::InitQueue() {

head = tail = nullptr;

}

template <typename T>

bool Queue<T>::EnQueue(T elem) {

if (Empty()) {

head = tail = new Node<T>(elem);

return true;

}

else {

tail->next = new Node<T>(elem);

tail = tail->next;

}

return true;

}

template <typename T>

bool Queue<T>::DeQueue(T& result) {

if (Empty())

return false;

result = head->data;

Node<T>\* to\_delete = head;

head = head->next;

delete to\_delete;

if (head == nullptr)

tail = nullptr;

return true;

}

template <typename T>

std::vector<T> Queue<T>::GetAll() {

std::vector<T> container;

Node<T>\* current = head;

while (current) {

container.push\_back(current->data);

current = current->next;

}

return container;

}

template <typename T>

T\* Queue<T>::GetFirst() {

if (Empty())

return nullptr;

return &head->data;

}

template <typename T>

bool Queue<T>::Empty() {

if (head == tail && head == nullptr)

return true;

else

return false;

}

**Ответы на контрольные вопросы:**

1. Тип данных - множество значений, которые может принимать переменная, и множество операцией над ней. Абстрактный тип данных - математическая модель данных с совокупностью операторов, определенных в рамках этой модели. Основное отличие - АТД является более сложной структурой и может включать другие типы данных. Сходства - и типы данных и АТД могут принимать значения из определённого множества и разрешают операции над элементами;
2. Структура данных (СД) - реализация АТД. Основное отличие от АТД заключается в определённости связей между элементами на физическом уровне (СД описывает логику взаимодействия). Сходство заключается в общем наборе операций и множестве допустимых значений;
3. Использование концепции АТД при нисходящем проектировании позволяет на этапе разработки общих алгоритмов не уделять внимание реализации будущей структуры данных, реализующей используемый АТД;
4. Это обусловлено тем, что разработчики не могут быть уверены, что модуль очереди будет использован только ими и только один раз;
5. При проектировании СД очередь в последовательной памяти предполагается, что элементы будут расположены последовательно (на физическом уровне) и доступ к ним будет осуществляться произвольно (по индексу). СД очереди в связной памяти реализуется так, что блоки могут располагаться в случайных местах физической памяти и доступ к ним может осуществляться исключительно последовательно, т.е. начиная с первого элемента и заканчивая последним (идём по связям);
6. Применения кольцевой очереди позволяет избежать её переорганизации (перемещения элементов в начало после заполнения очереди), что резко увеличивает эффективность алгоритма;
7. В очереди хранятся специализированные указатели на первый и последний элементы (узлы списка). При добавлении в очередь будет создан новый узел и указатель на него поместиться в последний узел очереди. При удалении очереди указатель на первый элемент (узел) перезаписывается новым значения (узлом, следующим за первым), после чего первый узел удаляется из памяти;
8. Данные модули являются универсальными, т.е. они позволяют хранить данных любых типов (встроенных, пользовательских и т.д.), что позволяет использовать их для решения широкого спектра задач. Единственным ограничением, о котором нельзя забывать, является ограниченность размера очереди, реализованном в последовательной памяти (максимум 5 элементов).