Описание АТД:

Queue [1 .. max\_el] of ElType {FIFO}

Операторы класса Queue:

clear() : void

is\_empty() : bool

is\_full() : bool

push(element : ElType) : bool

pop(element : ElType&) : bool

front(element : ElType\*&) : bool

size() : int

get\_all(elements : ElType\*) : void

operator string()

**LAB1.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

template <typename T>

struct QueueData;

template <typename T>

class Queue {

private:

//Данные варьируются в зависимости от подключаемого модуля

QueueData<T> data;

public:

void clear();

bool is\_empty();

bool is\_full();

bool push(T element);

bool pop(T& element);

bool front(T\*& element);

int size();

void get\_all(T\* elements);

operator string();

};

**LAB1.cpp (подключена очередь в последовательной памяти module1.cpp)**

#include "module1.cpp"

#include "Windows.h"

struct ElType {

char code[4 + 1] = { '\0', '\0', '\0', '\0', '\0'};//Одну доп. ячейку для символа конца строки

unsigned int time;

operator string() {

return "Деталь " + string(code) + " -> Время: " + to\_string(time);

}

};

Queue<ElType> queue;

ElType input() {

ElType elem;

string input\_code\_string = "";

while (input\_code\_string.empty()) {

cout << "Код детали (до 4-ёх символов) >> ";

cin.ignore();

getline(cin, input\_code\_string);

}

for (int i = 0; i < 4 && i < input\_code\_string.size(); i++)

elem.code[i] = input\_code\_string[i];

cout << "Модельное время до изготовления >> ";

cin >> elem.time;

return elem;

}

int main() {

setlocale(LC\_ALL, "ru");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int cmd;

while (true) {

cout << "\033[0m";//Сделать текст стандартным

cout <<

"\

--------------\n\

Список команд:\n\

1. Постановка детали на обработку.\n\

2. Переход к следующему моменту модельного времени.\n\

3. Снятие детали с обработки до ее завершения.\n\

4. Вывод списка обрабатываемых деталей на экран в порядке очереди.\n\

5. Сброс процесса моделирования.\n\

0. Выход из программы.\n\

";

cout << "Введите команду >> ";

cin >> cmd;

cout << "\033[1m";//Сделать текст жирным

ElType\* element\_ptr;

ElType element;

string answer;

bool success;

switch (cmd) {

case 1:

success = queue.push(input());

if (success)

answer = "Готово!";

else

answer = "Не удалось поставить деталь. Очередь переполнена.";

break;

case 2:

success = queue.front(element\_ptr);

if (success) {

element\_ptr->time--;

element = \*element\_ptr;

if (element\_ptr->time <= 0) {

queue.pop(element);

answer = "Обработка детали " + string(element.code) + " полностью завершена.";

}

else

answer = "Деталь " + string(element.code) + " обрабатывается (" + to\_string(element.time) + ").";

}

else

answer = "Не удалось извлечь элемент. Очередь пуста.";

break;

case 3:

success = queue.pop(element);

if (success) {

answer = "Обработка детали " + string(element.code) + " прервана.";

}

else

answer = "Не удалось извлечь элемент. Очередь пуста.";

break;

case 4:

cout << string(queue) << endl;

break;

case 5:

queue.clear();

cout << "Процесс моделирования сброшен. ";

break;

case 0:

cout << "Программа завершена." << endl;

exit(0);

default:

cout << "Нет команды с таким номером" << endl;

}

cout << answer << endl;

}

}

**module1.cpp**

#include "LAB1.h"

//Очередь в последовательной памяти

//Данные для очереди

template <typename T>

struct QueueData {

const int max\_el = 5 + 1;//Одну доп. ячейку для определения переполнения

T\* element = new T[max\_el];

int head = 0;

int tail = 0;

~QueueData() {

delete[] element;

}

};

template <typename T>

void Queue<T>::clear() {

data.head = 0;

data.tail = 0;

}

template <typename T>

bool Queue<T>::is\_empty() {

return data.tail == data.head;

}

template <typename T>

bool Queue<T>::is\_full() {

return (data.tail + 1) % data.max\_el == data.head;

}

template <typename T>

bool Queue<T>::push(T element) {

if (is\_full()) return false;

data.element[data.tail] = element;

data.tail = (data.tail + 1) % data.max\_el;

return true;

}

template<typename T>

bool Queue<T>::pop(T& element) {

if (is\_empty()) return false;

element = data.element[data.head];

data.head = (data.head + 1) % data.max\_el;

return true;

}

template<typename T>

bool Queue<T>::front(T\*& element) {

if (is\_empty()) return false;

element = &data.element[data.head];

return true;

}

template <typename T>

int Queue<T>::size() {

//Размер вычисляется как разность хвоста и головы с учётом перехода хвоста в начало массива

return (data.tail - data.head + data.max\_el) % data.max\_el;

}

template <typename T>

void Queue<T>::get\_all(T\* elements) {

int i = 0;

int cursor = data.head;

while (cursor != data.tail) {

elements[i++] = data.element[cursor];

cursor = (cursor + 1) % data.max\_el;

}

}

template<typename T>

Queue<T>::operator string() {

//ОТОБРАЖЕНИЕ СОСТОЯНИЯ ОЧЕРЕДИ

string res = "Очередь в последовательной памяти (head = " + to\_string(data.head) + ", tail = " + to\_string(data.tail) + "):";

//ОТОБРАЖЕНИЕ ЭЛЕМЕНТОВ ОЧЕРЕДИ

res += "\nСписок элементов очереди:";

T\* elements = new T[size()];

get\_all(elements);

for (int i = 0; i < size(); i++) {

res += "\n" + to\_string(i+1) + ". " + string(elements[i]);

}

delete[] elements;

res += "\nКонец очереди.";

return res;

}

**module2.cpp**

#include "LAB1.h"

//Очередь в связной памяти на основе линейного односвязного списка

//Данные для очереди

template <typename T>

struct Node {

T value;

Node<T>\* next = nullptr;

};

template <typename T>

struct QueueData {

int size = 0;

Node<T>\* head = nullptr;

Node<T>\* tail = nullptr;

Node<T>\* reserved\_node = nullptr;//поддержка is\_full() путём резервного элемента

~QueueData() {

delete reserved\_node;

Node<T>\* node = head;

while (node) {

Node<T>\* next = node->next;

delete node;

node = next;

}

}

};

template <typename T>

void Queue<T>::clear() {

data.~QueueData();

data = QueueData<T>();

}

template <typename T>

bool Queue<T>::is\_empty() {

return data.head == nullptr;

}

template <typename T>

bool Queue<T>::is\_full() {

if (data.reserved\_node == nullptr) {

try {

data.reserved\_node = new Node<T>();

return false;

}

catch (bad\_alloc e) {

return true;

}

}

else return false;

}

template <typename T>

bool Queue<T>::push(T element) {

if (is\_full()) return false;

Node<T>\* node = data.reserved\_node;

data.reserved\_node = nullptr;

if (!data.head)

// Указатель храним в переменной head

data.head = node;

else

// Указатель храним в последнем узле списка

data.tail->next = node;

data.size++;

data.tail = node;

node->value = element;

return true;

}

template<typename T>

bool Queue<T>::pop(T& element) {

if (is\_empty()) return false;

Node<T>\* node = data.head;

element = node->value;

data.size--;

data.head = node->next;

delete node;

return true;

}

template<typename T>

bool Queue<T>::front(T\*& element) {

if (is\_empty()) return false;

element = &data.head->value;

return true;

}

template <typename T>

int Queue<T>::size() {

return data.size;

}

template <typename T>

void Queue<T>::get\_all(T\* elements) {

int i = 0;

Node<T>\* node = data.head;

while (node) {

elements[i++] = node->value;

node = node->next;

}

}

template<typename T>

Queue<T>::operator string() {

//ОТОБРАЖЕНИЕ СОСТОЯНИЯ ОЧЕРЕДИ

string res = "Очередь в связной памяти (size = " + to\_string(data.size) + ", reserved\_node? = " + to\_string(data.reserved\_node != nullptr) + "):";

//ОТОБРАЖЕНИЕ ЭЛЕМЕНТОВ ОЧЕРЕДИ

res += "\nСписок элементов очереди:";

T\* elements = new T[size()];

get\_all(elements);

for (int i = 0; i < size(); i++) {

res += "\n" + to\_string(i + 1) + ". " + string(elements[i]);

}

delete[] elements;

res += "\nКонец очереди.";

return res;

}