ФЕДЕРАЛЬНОЕ Государственное АВТОНОМНОЕ образовательное учреждение

высшего образования

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

Кафедра математического и программного обеспечения информационных систем

**Отчет по лабораторной работе №5**

**по дисциплине: «Структуры и алгоритмы компьютерной обработки данных»**

**Тема работы «Структура данных “стек” и “очередь”»**

студента очного отделения

2 курса 12001801 группы

Скрипниченко Илья Игоревич

Проверил(а):

Курлов Василий Васильевич

Белгород 2019

**Лабораторная работа №5**

**Цель работы:** изучение СД типа”стек” и “очередь”, их программная реализация и использование.

**Характеристика СД типа “стек”**

Стек – динамическая структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека.

По определению, элементы извлекаются из стека в порядке, обратном их добавлению в эту структуру, т.е. действует принцип "последний пришёл — первый ушёл".

На абстрактном уровне стек представляет собой линейную структуру – последовательность.

На физическом уровне стек может быть реализован последовательной или связной схемой хранения. Располагаться стек может в статической или динамической памяти. Стек, по сути, является линейным списком с ограниченным доступом к элементам, поэтому он может быть реализован на основе СД ЛС. Достаточно ввести ограничения на операции и обеспечить доступ к элементу, являющемуся вершиной стека. В качестве такого элемента может быть либо первый, либо последний элемент в ЛС. Рассмотрим особенности выполнения операций включения и исключения в зависимости от типа ЛС, на основе которого реализуется стек, и выбора вершины стека.

Возможны три операции со стеком: добавление элемента (иначе проталкивание, push), удаление элемента (pop) и чтение головного элемента (peek).

При проталкивании (push) добавляется новый элемент, указывающий на элемент, бывший до этого головой. Новый элемент теперь становится головным.

При удалении элемента (pop) убирается первый, а головным становится тот, на который был указатель у этого объекта (следующий элемент). При этом значение убранного элемента возвращается.

**Характеристика СД типа “очередь”**

**Очередь** – это структура данных, добавление и удаление элементов в которой происходит путём операций push и pop соответственно. Притом первым из очереди удаляется элемент, который был помещен туда первым, то есть в очереди реализуется принцип «первым вошел — первым вышел» (*FIFO*). У очереди имеется голова и хвост . Когда элемент ставится в очередь, он занимает место в её хвосте. Из очереди всегда выводится элемент, который находится в ее голове. Очередь поддерживает следующие операции:

* проверка очереди на наличие в ней элементов,
* операция вставки нового элемента,
* операция удаления нового элемента,
* операция получения количества элементов в очереди.

На физическом уровне очередь может быть реализована последовательной или связной схемой хранения. Располагаться очередь может в статической или динамической памяти. Очередь, по сути, является линейным списком с ограниченным доступом к элементам, поэтому он может быть реализован на основе структуры данных линейного списка. Достаточно ввести ограничения на операции и обеспечить доступ к элементам, расположенным в начале и в конце очереди.

**Индивидуальное задание**

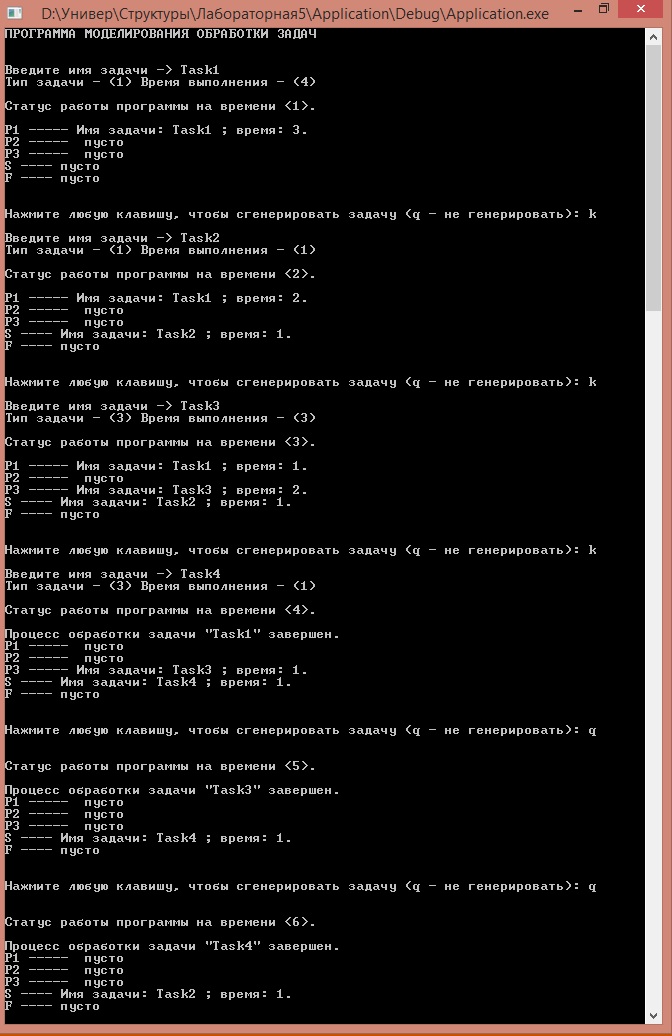
5. Система состоит из трёх процессоров P1, P2, P3, очереди F, стека S и распределителя R . В систему поступают запросы на выполнение задач трёх ти-пов – Т1, Т2 и Т3, каждая для своего процессора.

Поступающие запросы ставятся в очередь. Если в “голове” очереди находится задача Ti и процессор Ti свободен, то распределитель ставит задачу на выполнение в процессор Pi, а если процессор Pi занят, то распределитель отправляет задачу в стек и из очереди извлекается следующая задача. Если в вершине стека находится задача, процессор которой в данный момент свободен, то эта задача извлекается и отправляется на выполнение.

Стек реализовать на ОЛС. Вершина стека – первый элемент ОЛС.

Очередь реализовать на ОЛС. “Хвост” очереди – последний, а “голова” - первый элемент ОЛС.

**Результаты работы программы**

****

|  |  |  |
| --- | --- | --- |
| **Время** | **Объекты** | **Задачи** |
| 1 | P1 | (Task1, 3) |
| P2 | Пусто |
| P3 | Пусто |
| S | Пусто |
| F | Пусто |
| 2 | P1 | (Task1, 2) |
| P2 | Пусто |
| P3 | Пусто |
| S | (Task2, 1) |
| F | Пусто |
| 3 | P1 | (Task1, 1) |
| P2 | Пусто |
| P3 | (Task3, 2) |
| S | (Task2, 1) |
| F | Пусто |
| 4 | P1 | Пусто |
| P2 | Пусто |
| P3 | (Task3, 1) |
| S | (Task4, 1) (Task2, 1) |
| F | Пусто |
| 5 | P1 | Пусто |
| P2 | Пусто |
| P3 | (Task4, 1) |
| S | (Task2, 1) |
| F | Пусто |
| 6 | P1 | Пусто |
| P2 | Пусто |
| P3 | (Task2, 1) |
| S | Пусто |
| F | Пусто |
| 7 | P1 | Пусто |
| P2 | Пусто |
| P3 | Пусто |
| S | Пусто |
| F | Пусто |

**Листинг программы**

#include "pch.h"

#include <iostream>

#include <string>

#include <ctime>

//////////////////////Односвязный линейный список

template<typename T>

class SLLList

{

private:

template<typename T>

class Node

{

public:

Node\* next;

T data;

Node(T data = T(), Node\* next = nullptr)

{

this->data = data;

this->next = next;

}

};

int size;

Node<T>\* head;

public:

SLLList();

~SLLList();

T pop\_front();

T pop\_back();

T check() { return head->data; }

void push\_front(T data);

void push\_back(T data);

int get\_size() { return size; }

void clear() { while (size) pop\_front(); }

};

template<typename T>

SLLList<T>::SLLList()

{

size = 0;

head = nullptr;

}

template<typename T>

SLLList<T>::~SLLList()

{

clear();

}

template<typename T>

T SLLList<T>::pop\_front()

{

Node<T>\* temp = head;

head = head->next;

T data = temp->data;

delete temp;

size--;

return data;

}

template<typename T>

T SLLList<T>::pop\_back()

{

Node<T>\* current = this->head;

while (current->next) current = current->next;

T data = current->data;

current = NULL;

size--;

return data;

}

template<typename T>

void SLLList<T>::push\_front(T data)

{

head = new Node<T>(data, head);

size++;

}

template<typename T>

void SLLList<T>::push\_back(T data)

{

if (head == nullptr) head = new Node<T>(data);

else

{

Node<T>\* current = this->head;

while (current->next != nullptr) current = current->next;

current->next = new Node<T>(data);

}

size++;

}

//////////////////////Стек

template<typename T>

class Stack

{

private:

SLLList<T> block;

public:

void push(T data) { block.push\_front(data); }

T pop() { return block.pop\_front(); }

bool isEmpty() { return block.get\_size() == 0; }

T check\_top() { return block.check(); }

};

//////////////////////Очередь

template<typename T>

class Queue

{

private:

SLLList<T> block;

public:

void push(T data) { block.push\_back(data); }

T pop() { return block.pop\_front(); }

bool isEmpty() { return block.get\_size() == 0; }

T check\_top() { return block.check(); }

};

//////////////////////Задача

class Task

{

private:

std::string name;

unsigned time;

unsigned type;

public:

Task(std::string name = "", unsigned time = 0, unsigned type = 0)

{

this->name = name;

this->time = time;

this->type = type;

}

void generate()

{

std::cout << "Введите имя задачи -> ";

std::getline(std::cin, name);

time = (rand() % 4) + 1;

type = (rand() % 3) + 1;

}

std::string get\_name() { return name; }

unsigned get\_time() { return time; }

unsigned get\_type() { return type; }

void countdown() { --time; }

bool isFinished() { return time == 0; }

void output\_status()

{

if (this->get\_time() > 0) std::cout << "Имя задачи: " << name << " ; время: " << time << ".\n";

else std::cout << " пусто\n";

}

};

//////////////////////Процессор

class Processor

{

private:

Task processed\_task;

public:

Processor() { processed\_task = Task(); }

void push(Task task) { processed\_task = task; }

void delete\_task() { processed\_task = Task(); }

bool check\_task() { return processed\_task.get\_time() > 0; }

void task\_processing()

{

if (this->check\_task())

{

processed\_task.countdown();

if (processed\_task.isFinished())

{

std::cout << "Процесс обработки задачи \"" << processed\_task.get\_name() << "\" завершен.\n";

this->delete\_task();

}

}

}

void show\_task\_status()

{

if (processed\_task.get\_time() > 5 || processed\_task.get\_time() == 0) std::cout << " пусто\n";

else processed\_task.output\_status();

}

};

//////////////////////Распределитель

class Distributor

{

private:

Task received\_task;

public:

Distributor() { received\_task = Task(); }

void push(Task task) { received\_task = task; }

void clear() { received\_task = Task(); }

Task get\_task() { return received\_task; }

};

int main()

{

using namespace std;

setlocale(0, "");

srand(time(NULL));

Processor P1;

Processor P2;

Processor P3;

Processor pArr[3];

pArr[0] = P1;

pArr[1] = P2;

pArr[2] = P3;

Distributor R;

bool R\_tumblr;

Queue<Task> F;

Stack<Task> S;

unsigned dec\_time = 0;

cout << "ПРОГРАММА МОДЕЛИРОВАНИЯ ОБРАБОТКИ ЗАДАЧ\n\n";

char ask = 's';

cout << endl;

while (ask != 'q' || pArr[0].check\_task() || pArr[1].check\_task() || pArr[2].check\_task() || !S.isEmpty() || !F.isEmpty())

{

R\_tumblr = true;

dec\_time++;

Task new\_task;

if (ask != 'q')

{

new\_task.generate(); //Генерируем задачу

cout << "Тип задачи - (" << new\_task.get\_type() << ") Время выполнения - (" << new\_task.get\_time() << ")\n";

F.push(new\_task); //Ставим задачу в очередь

}

if (!S.isEmpty() && !pArr[(S.check\_top()).get\_type() - 1].check\_task()) //Проверяем, освободился ли процессор для обработки верхнего элемента стека

{

R.push(S.pop());

pArr[(S.check\_top()).get\_type() - 1].push(R.get\_task());

R.clear();

}

while (!F.isEmpty() && R\_tumblr)

{

R.push(F.pop()); //Передаем задачу из очереди в распределитель

if (!pArr[(R.get\_task()).get\_type() - 1].check\_task()) //Если нужный процессор свободен - ставим задачу на обработку

{

pArr[(R.get\_task()).get\_type() - 1].push(R.get\_task());

R.clear();

R\_tumblr = false;

}

else //Если занят - кладем задачу в стек и извлекаем из очереди следующую (если она есть), но перед этим проверяем процессор для вершины стека

{

if (pArr[(R.get\_task()).get\_type() - 1].check\_task())

{

S.push(R.get\_task());

R.clear();

R\_tumblr = true;

}

}

}

cout << "\nСтатус работы программы на времени <" << dec\_time << ">.\n\n";

for (int i = 0; i < 3; i++) //Обработка задачи в процессоре

{

if (pArr[i].check\_task()) pArr[i].task\_processing();

}

for (int i = 0; i < 3; i++)

{

cout << "P" << i + 1 << " ----- ";

pArr[i].show\_task\_status();

}

if (!S.isEmpty())

{

cout << "S ---- ";

(S.check\_top()).output\_status();

}

else cout << "S ---- пусто\n";

if (!F.isEmpty())

{

cout << "F ---- ";

(F.check\_top()).output\_status();

}

else cout << "F ---- пусто\n";

cout << "\n\nНажмите любую клавишу, чтобы сгенерировать задачу (q - не генерировать): ";

cin >> ask;

while (cin.get() != '\n') continue;

cout << endl;

}

return 0;

}