МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ассистент |  |  |  | М.А.Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3 |
| «СТЕК И ОЧЕРЕДЬ» |
| по курсу: Структуры и алгоритмы обработки данных |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134К |  |  |  | Опарин С.Н. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

1. ***Цель работы***

Целью работы является изучение структур данных «стек» и «очередь», а также получение практических навыков их реализации.

1. ***Задание***

Реализовать структуры данных «стек» и «очередь» в соответствии с заданным вариантом.

Дополнительно программа должна удовлетворять следующим требованиям:

1) Вывод на экран состояния моделируемой системы на каждой итерации работы (содержимое стека(ов), очереди(ей), процессора(ов));

2) Для каждой задачи из списка входных задач должно быть определено время поступления;

3) Необходимо наличие, как автоматического генератора задач, так и возможность ручного добавления задач, с указанием их параметров (в зависимости от задания);

4) Необходимо обработать ситуации, при которых какая-либо структура данных может быть переполнена.

***Вариант задания***



1. Листинг программы, реализующей алгоритм:

#include <iostream>

#define STACK\_MAX\_SIZE 1

using namespace std;

int random(int min, int max){

return min + rand() & (max - min + 1);

}

struct Task{

int priority;//Приоритет задачи

int taskTime;//Время захода в процессор

int durationTime;//Время выполнения в тактах

};

struct Queue{

int qLen = 0;

Task \*queue;

Queue(int q\_L) {queue = new Task[q\_L];}

Task popTask(){

Task t = queue[0];

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

queue[i] = queue[i + 1];

}

qLen--;

return t;

}

int getPriority(){

return queue[0].priority;

}

int getDurationTime(){

return queue[0].durationTime;

}

void pushTask(Task newTask){

queue[qLen].priority = newTask.priority;

queue[qLen].durationTime = newTask.durationTime;

queue[qLen].taskTime = newTask.taskTime;

qLen++;

}

void pushElement(int priority, int durationTime, int taskTime){

queue[qLen].priority = priority;

queue[qLen].durationTime = durationTime;

queue[qLen].taskTime = taskTime;

qLen++;

}

void showQueue(){

cout << "------------------- ДЛИНА = " << qLen << " ------------------------- \n";

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

cout << " " << i + 1 << ". Элемент " << endl;

cout << " - Приоритет: " << queue[i].priority << endl;

cout << " - Время захода в процессор: " << queue[i].taskTime << endl;

cout << " - Продолжительность в тактах: " << queue[i].durationTime << endl;

}

cout << " \n";

}

bool is\_empty(){ return qLen == 0; }

};

void generateRandomTask(Queue &q, int count){

int priority, taskTime, durationTime;

priority = random(0, 1);

taskTime = count;

durationTime = random(2, 4);

while(durationTime == 0){

durationTime = random(2, 4);

}

q.pushElement(priority, durationTime, taskTime);

}

void generateManualTask(Queue &queue, int count){

int priority, taskTime, durationTime;

cout << "Введите приоритет = ";

cin >> priority;

while(priority > 2 or priority < 0){

cout << "Ошибка, нужно вводить приоритет в промежутке от 0 до 2" << endl;

cout << endl;

cout << "Введите новое значение: ";

cin >> priority;

}

cout << "Введите время поступления задачи = ";

cin >> taskTime;

while(taskTime != count + 1){

cout << "Ошибка, нужно вводить время захода в процессор по порядку" << endl;

cout << endl;

cout << "Введите " << count + 1 << ": ";

cin >> taskTime;

}

cout << "Введите количество тактов выполнения = ";

cin >> durationTime;

while(durationTime <= 0){

cout << "Ошибка, количество тактов должно быть больше 0" << endl;

cout << "Введите новое количество тактов выполнения: ";

cin >> durationTime;

}

queue.pushElement(priority, durationTime, taskTime);

}

struct Stack{

private:

Task stack[STACK\_MAX\_SIZE];

int selfLen = 0;

public:

bool is\_empty(){

return selfLen == 0;

}

bool is\_full(){

return selfLen == STACK\_MAX\_SIZE;

}

void pushToStack(Task t){

stack[selfLen].priority = t.priority;

stack[selfLen].taskTime = t.taskTime;

stack[selfLen].durationTime = t.durationTime;

selfLen++;

}

Task popFromStack(){

if(selfLen >= 1) --selfLen;

return stack[selfLen];

}

void showStack(){

int i = 1;

cout << "------------------- Длина = " << selfLen << "-------------------" << endl;

while(selfLen - i != -1){

cout << endl;

cout << "Элемент № " << i << endl;

cout << "Приоритет - " << stack[selfLen - i].priority << endl;

cout << "Время поступления задачи - " << stack[selfLen - i].taskTime << endl;

cout << "Количество тактов - " << stack[selfLen - i].durationTime << endl;

i++;

}

cout << endl;

}

};

struct Processor{

private:

bool isFree = true;

public:

Task pr = {};

void pushTask(Task newTask){

pr = newTask;

isFree = false;

}

void doCurrentTask(){

if(pr.durationTime > 0){

pr.durationTime -= 1;

if(pr.durationTime == 0){

isFree = true;

}

}

else{

isFree = true;

}

}

void showProcessor(){

if((isFree == 0) && (pr.durationTime != 0)){

cout << "Приоритет - " << pr.priority << endl;

cout << "Время поступления задачи - " << pr.taskTime << endl;

cout << "Количество тактов - " << pr.durationTime << endl;

}

else{

cout << endl << "Процессор свободен";

cout << "--------------------------------------" << endl;

}

}

Task getCurrentTask(){

isFree = true;

return pr;

}

bool is\_free(){

return isFree;

}

};

int main(){

Stack stack;//Стек задач

Processor processor\_;

int mode, n, t = 0;

cout << "1 - Автоматический" << endl;

cout << "2 - Ручной" << endl;

cout << "Выберите способ заполнения задач: ";

cin >> mode;

cout << "Введите количество задач: ";

while(true){

cin >> n;

if(n <= 15) break;

else cout << "Введите количество задач < 20";

}

Queue tasks(n); //Список задач

Queue queue0(n), queue1(n), queue2(n);//Очереди задач

while(t != n){

if(mode == 1){

generateRandomTask(tasks, t);

}

else{

generateManualTask(tasks, t);

}

t += 1;

}

cout << "\n\nСгенерированные задачи: " << endl;

tasks.showQueue();

cout << "------------------------------------------------------- " << endl;

t = 0;

while (!tasks.is\_empty() || !queue0.is\_empty() || !queue1.is\_empty() || !queue2.is\_empty() || !stack.is\_empty() || !processor\_.is\_free()){ // Если где-то что-то есть

int taskTimeToCompare;

cout << "\n \t\t <<ТАКТ " << t + 1 << ">>\n\n";

if (!tasks.is\_empty()){ // Если задачи есть

if(tasks.getPriority() == 0){ //Если первая в очереди задача с приоритетом 0

taskTimeToCompare = tasks.getDurationTime();

queue0.pushTask(tasks.popTask());

}

else if(tasks.getPriority() == 1){ //Если первая в очереди задача с приоритетом 1

taskTimeToCompare = tasks.getDurationTime();

queue1.pushTask(tasks.popTask());

}

else{ //Если первая в очереди задача с приоритетом 2

taskTimeToCompare = tasks.getDurationTime();

queue2.pushTask(tasks.popTask());

}

}

if(!queue0.is\_empty()){

if(processor\_.is\_free()){

processor\_.pushTask(queue0.popTask());

processor\_.doCurrentTask();

}

else{

if(processor\_.pr.priority > queue0.getPriority()){

if(processor\_.pr.durationTime < taskTimeToCompare \* 0.5){

processor\_.doCurrentTask();

}

else{

if(stack.is\_full()){

if(processor\_.getCurrentTask().priority == 1){

queue1.pushTask(processor\_.getCurrentTask());

}

else{

queue2.pushTask(processor\_.getCurrentTask());

}

processor\_.pushTask(queue0.popTask());

processor\_.doCurrentTask();

}

else{

stack.pushToStack(processor\_.getCurrentTask());

processor\_.pushTask(queue0.popTask());

processor\_.doCurrentTask();

}

}

}

else{

processor\_.doCurrentTask();

}

}

}

else if(!queue1.is\_empty()){

if(processor\_.is\_free()){

processor\_.pushTask(queue1.popTask());

processor\_.doCurrentTask();

}

else{

if(processor\_.pr.priority > queue1.getPriority()){

if(processor\_.pr.durationTime < taskTimeToCompare \* 0.5){

processor\_.doCurrentTask();

}

else{

if(stack.is\_full()){

if(processor\_.getCurrentTask().priority == 2){

queue2.pushTask(processor\_.getCurrentTask());

}

processor\_.pushTask(queue1.popTask());

processor\_.doCurrentTask();

}

else{

stack.pushToStack(processor\_.getCurrentTask());

processor\_.pushTask(queue1.popTask());

processor\_.doCurrentTask();

}

}

}

else{

processor\_.doCurrentTask();

}

}

}

else if(!queue2.is\_empty()){

if(processor\_.is\_free()){

processor\_.pushTask(queue2.popTask());

processor\_.doCurrentTask();

}

else{

if(processor\_.pr.priority > queue2.getPriority()){

if(processor\_.pr.durationTime < taskTimeToCompare \* 0.5){

processor\_.doCurrentTask();

}

else{

if(stack.is\_full()){

queue2.pushTask(processor\_.getCurrentTask());

processor\_.pushTask(queue2.popTask());

processor\_.doCurrentTask();

}

else{

stack.pushToStack(processor\_.getCurrentTask());

processor\_.pushTask(queue2.popTask());

processor\_.doCurrentTask();

}

}

}

else{

processor\_.doCurrentTask();

}

}

}

else{

if(!stack.is\_empty()){

if(processor\_.is\_free()){

processor\_.pushTask(stack.popFromStack());

processor\_.doCurrentTask();

}

else{

processor\_.doCurrentTask();

}

}

else{

processor\_.doCurrentTask();

}

}

cout << "-------------------- ЗАДАЧИ ------------------------ " << endl;

tasks.showQueue();

cout << "---------------------- СТЕК ------------------------ " << endl;

if(!stack.is\_empty()){

stack.showStack();

}

else{

cout << endl;

}

cout << "-------------------- ОЧЕРЕДЬ №0 ----------------------- " << endl;

queue0.showQueue();

cout << "-------------------- ОЧЕРЕДЬ №1 ----------------------- " << endl;

queue1.showQueue();

cout << "-------------------- ОЧЕРЕДЬ №2 ----------------------- " << endl;

queue2.showQueue();

cout << "-------------------------------------------------------\n";

cout << "---------------- П Р О Ц Е С С О Р -------------------" << endl;

processor\_.showProcessor();

t += 1;

}

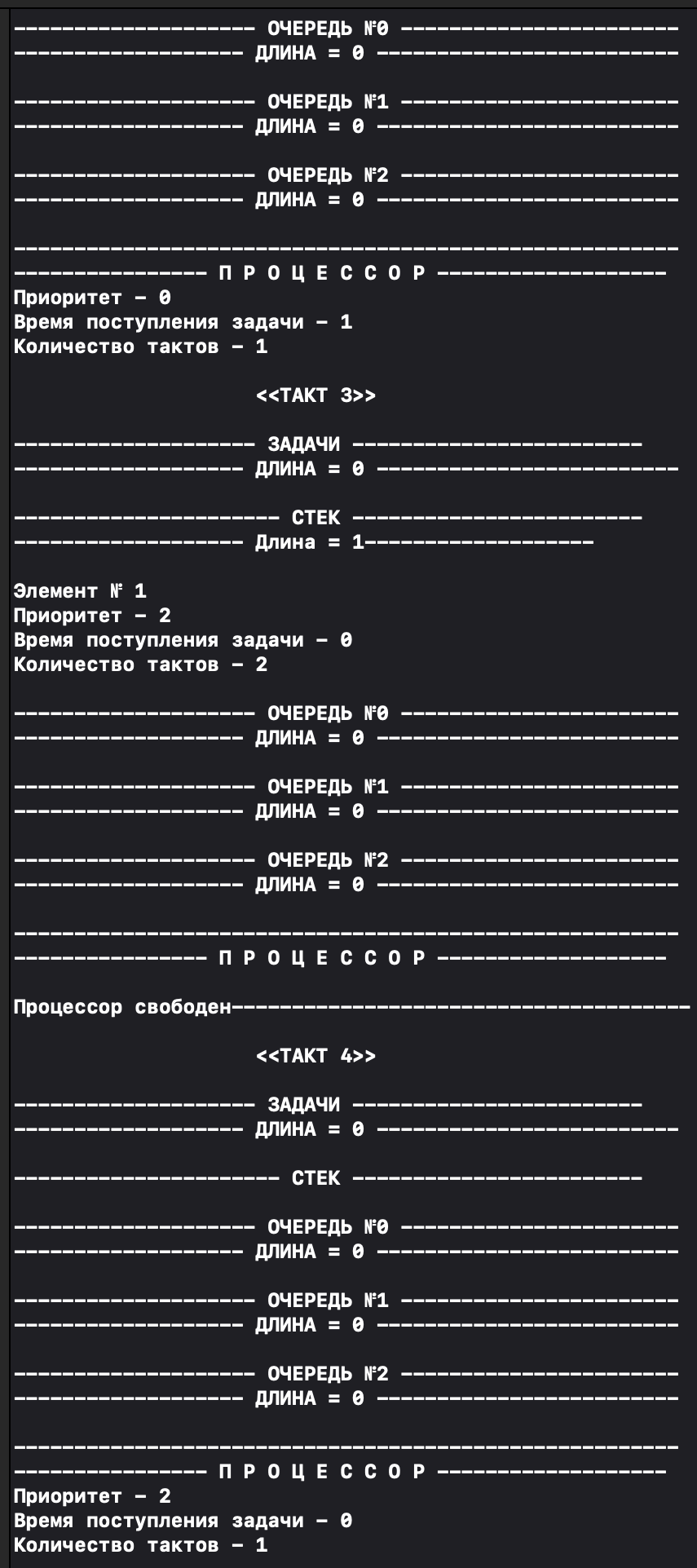
cout << "\n\n----------------------------------------------------\nКОНЕЦ \n";

return 0;

}

1. ***Примеры***

******

******



1. ***Вывод***

В ходе лабораторной работы я изучил структуры данных «стек» и «очередь», а также получил практические навыкы их реализации