ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Рогачев С. А. |
| 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. № | М011 |  |  |  | Борисов С. И. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

**Цель работы**

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

**Задание на лабораторную**

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));

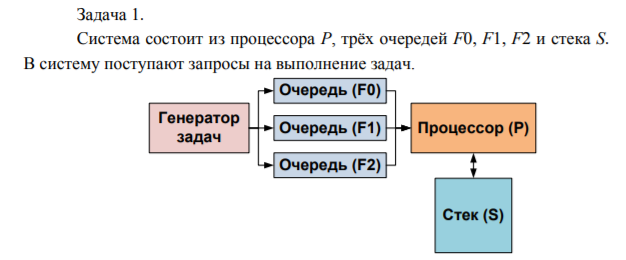
2) Для каждой задачи из списка входных задач должно быть определено время поступления;

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. Варианты задания приведены в таблице 3 (формулировки задач приведены после таблицы).

**Вариант задания: 4 (задача 1)**





Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди F0. Если она пуста, можно обрабатывать задачи из очереди F1. Если и она пуста, то можно обрабатывать задачи из очереди F2. Если все очереди пусты, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если поступает задача с более высоким приоритетом, чем обрабатываемая в данный момент, 3 то обрабатываемая помещается в стек и может обрабатываться тогда и только тогда, когда все задачи с более высоким приоритетом уже обработаны.

**Листинг программы, реализующей алгоритм**

// М011 ЛБ-3 Боривов С. И. Вариант 4.

#include <iostream>

#include <string>

#include <time.h>

using namespace std;

struct task

{

string name;

int proceed\_time;

int needed\_time;

int input\_time;

int priority;

};

struct stack\_element

{

struct task\* value;

struct stack\_element\* next;

};

struct processor

{

struct task\* task;

struct stack\_element\* stack;

};

struct queue\_element

{

struct task\* task;

struct queue\_element\* last;

};

struct task\_queue

{

struct queue\_element\* queue0;

struct queue\_element\* queue1;

struct queue\_element\* queue2;

};

string task\_to\_string(struct task\* task)

{

string name\_pre("\tИмя: ");

string proceed\_pre("\n\tВремя выполнения задачи: ");

string priority\_pre("\n\tПриоритет: ");

string input\_pre("\n\t Время поступления: ");

string needed\_pre("\n\tНеобходимое время: ");

return name\_pre + task->name + proceed\_pre

+ to\_string(task->proceed\_time)

+ needed\_pre + to\_string(task->needed\_time)

+ priority\_pre + to\_string(task->priority)

+ input\_pre + to\_string(task->input\_time) + "\n";

}

int get\_queue\_length(struct queue\_element\* queue)

{

struct queue\_element\* ptr = queue;

int i = 0;

if (queue == 0)

return 0;

while (ptr != NULL)

{

i++;

ptr = ptr->last;

}

return i;

}

struct task\* generate\_task(string name, int needed\_time, int input\_time, int priority)

{

struct task\* ret = new struct task;

ret->name = name;

ret->needed\_time = needed\_time;

ret->input\_time = input\_time;

ret->priority = priority;

ret->proceed\_time = 0;

return ret;

}

struct task\* generate\_random(int time)

{

string name;

cout << "Введите имя: ";

cin >> name;

return generate\_task(name, rand() % 10 + 1, time, rand() % 3);

}

struct stack\_element\* push\_stack(struct stack\_element\* stack, struct task\* task)

{

struct stack\_element\* new\_element = new struct stack\_element;

new\_element->next = stack;

new\_element->value = task;

return new\_element;

}

struct stack\_element\* pop\_stack(struct stack\_element\* stack, struct task\* task)

{

\*task = \*(stack->value);

return stack->next;

}

void push\_task\_queue(struct task\_queue\* queue, struct task\* task)

{

struct queue\_element\* ptr;

struct queue\_element\* new\_element = new struct queue\_element;

new\_element->task = task;

new\_element->last = 0;

switch (task->priority)

{

case 0:

if (queue->queue0 == 0)

{

queue->queue0 = new\_element;

cout << "Задача добавлена: \n" << task\_to\_string(new\_element->task) << endl;

return;

}

ptr = queue->queue0;

while (ptr->last != NULL)

ptr = ptr->last;

ptr->last = new\_element;

break;

case 1:

if (queue->queue1 == 0)

{

queue->queue1 = new\_element;

cout << "Задача добавлена: \n" << task\_to\_string(new\_element->task) << endl;

return;

}

ptr = queue->queue1;

while (ptr->last != NULL)

ptr = ptr->last;

ptr->last = new\_element;

break;

case 2:

if (queue->queue2 == 0)

{

queue->queue2 = new\_element;

cout << "Задача добавлена: \n" << task\_to\_string(new\_element->task) << endl;

return;

}

ptr = queue->queue2;

while (ptr->last != NULL)

ptr = ptr->last;

ptr->last = new\_element;

break;

}

cout << "Задача добавлена: \n" << task\_to\_string(new\_element->task) << endl;

return;

}

struct task\* pop\_task\_queue(struct task\_queue\* queue)

{

struct queue\_element\* ret;

if (get\_queue\_length(queue->queue0) != 0)

{

ret = queue->queue0;

queue->queue0 = queue->queue0->last;

return ret->task;

}

if (get\_queue\_length(queue->queue1) != 0)

{

ret = queue->queue1;

queue->queue1 = queue->queue1->last;

return ret->task;

}

if (get\_queue\_length(queue->queue2) != 0)

{

ret = queue->queue2;

queue->queue2 = queue->queue2->last;

return ret->task;

}

return 0;

}

int get\_queue\_priority(struct task\_queue\* queue)

{

if (get\_queue\_length(queue->queue0) == 0)

{

if (get\_queue\_length(queue->queue1) == 0)

{

if (get\_queue\_length(queue->queue2) == 0)

{

return 3;

}

return 2;

}

else

return 1;

}

else return 0;

}

void add\_random(struct task\_queue\* queue, int time)

{

push\_task\_queue(queue, generate\_random(time));

}

string queues\_to\_string(struct task\_queue\* queue)

{

string ret("");

struct queue\_element\* ptr = queue->queue0;

ret += "\nОчередь 0:";

while (ptr != 0)

{

ret += "\n" + task\_to\_string(ptr->task);

ptr = ptr->last;

}

ptr = queue->queue1;

ret += "\n\nОчередь 1:";

while (ptr != 0)

{

ret += "\n" + task\_to\_string(ptr->task);

ptr = ptr->last;

}

ptr = queue->queue2;

ret += "\n\nОчередь 2:";

while (ptr != 0)

{

ret += "\n" + task\_to\_string(ptr->task);

ptr = ptr->last;

}

return ret;

}

string stack\_to\_string(struct stack\_element\* stack)

{

string ret("");

struct stack\_element\* ptr = stack;

ret += "\nСтек:";

while (ptr != 0)

{

ret += "\n" + task\_to\_string(ptr->value);

ptr = ptr->next;

}

return ret;

}

void add\_element(struct task\_queue\* queue, int time)

{

int needed;

int priority;

string name;

cout << "Введите имя: ";

cin >> name;

cout << "\nВведите необходимое время: ";

cin >> needed;

cout << "\nВведите приоритет: ";

cin >> priority;

push\_task\_queue(queue, generate\_task(name, needed, time, priority));

}

void iterate(struct task\_queue\* queue, struct processor\* processor, int\* time)

{

if (processor->task != 0)

{

int new\_priority;

new\_priority = get\_queue\_priority(queue);

if (new\_priority < processor->task->priority)

{

processor->stack = push\_stack(processor->stack, processor->task);

processor->task = pop\_task\_queue(queue);

}

}

else

{

if (processor->stack != 0)

{

if (processor->stack->value->priority <= get\_queue\_priority(queue))

{

struct task\* new\_task = new struct task;

processor->stack = pop\_stack(processor->stack, new\_task);

processor->task = new\_task;

}

}

else

if ((processor->task = pop\_task\_queue(queue)) == 0)

{

cout << "Нет задачи" << endl;

return;

}

}

(\*time)++;

processor->task->proceed\_time += 1;

cout << "Все время процесса: " << (\*time) << endl;

if (processor->task->proceed\_time >= processor->task->needed\_time)

{

cout << "Задача выполнена: \n" << task\_to\_string(processor->task) << endl;

processor->task = 0;

}

else

cout << "В процессор: \n" << task\_to\_string(processor->task) << endl;

cout << queues\_to\_string(queue) << endl;

cout << stack\_to\_string(processor->stack) << endl;

}

void iterate\_multiple(struct task\_queue\* queue, struct processor\* processor, int\* time)

{

int count;

cout << "Введите n: ";

cin >> count;

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

{

iterate(queue, processor, time);

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

struct task\_queue\* queue = new struct task\_queue;

queue->queue0 = 0;

queue->queue1 = 0;

queue->queue2 = 0;

struct processor\* processor = new struct processor;

processor->task = 0;

processor->stack = 0;

int sys\_time = 0;

while (true)

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "|1| Одна итерация. |\n";

cout << "|2| Итерация n кол-во раз. |\n";

cout << "|3| Рандомная задача. |\n";

cout << "|4| Самостоятельный ввод задачи|\n";

cout << "|0| Выход. |\n";

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|"<< endl;

cout << "Ввод: ";

int choice;

cin >> choice;

switch (choice)

{

case 1:

iterate(queue, processor, &sys\_time);

break;

case 2:

iterate\_multiple(queue, processor, &sys\_time);

break;

case 3:

add\_random(queue, sys\_time);

break;

case 4:

add\_element(queue, sys\_time);

break;

case 0:

return 0;

default:

cout << "Ошибка ввода\n";

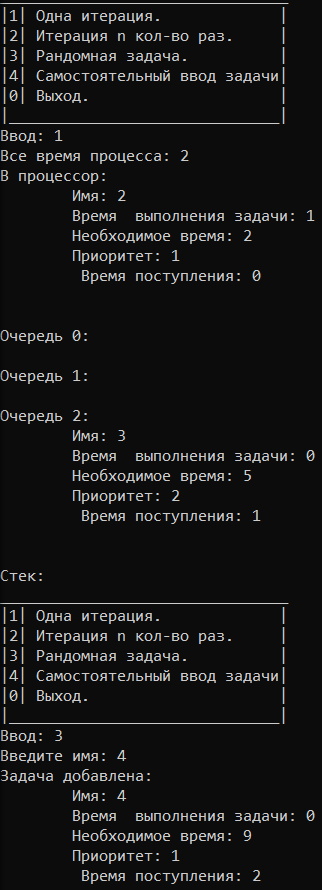
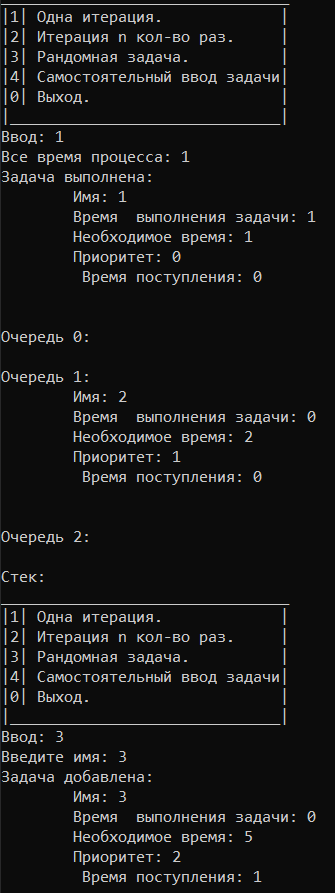
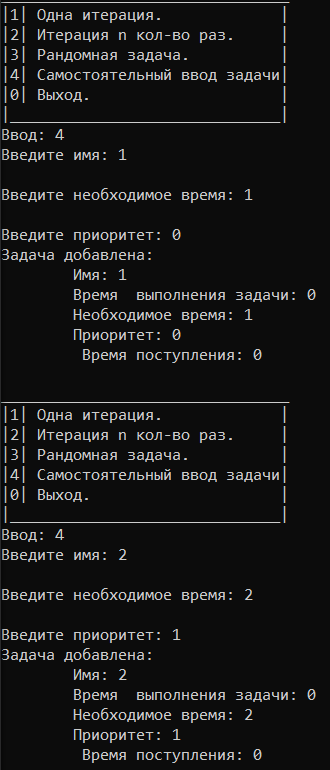
}

}

}

**Примеры**

|  |  |  |
| --- | --- | --- |
| **Необходимое время** | **Приоритет** | **Время поступления** |
| **1** | **0** | **0** |
| **2** | **1** | **0** |
| **5** | **2** | **1** |
| **9** | **1** | **2** |



**Выводы**

На основе этих расчетов можно сделать вывод, что был разработан алгоритм, характеристики которого соответствуют поставленному заданию.