

ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

ассистент

должность, уч. степень, звание

подпись, дата

К.А. Кочин

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Стек и очередь

по курсу: Структуры и алгоритмы обработки данных

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

4136

подпись, дата

Бобрович Н. С.

инициалы, фамилия

Цель работы

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

Задание

Вариант №13.

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.

Номер задачи - 4. Стек - статический. Очередь - статическая.

Листинг программы

```

1  #include <iostream>
2  #define MAXOCH 10
3  #define MAXST 1
4  #define TL 10
5
6  using namespace std;
7
8  struct Task {
9      string name = "";
10     int priority = 0;
11     int type = 0;
12     int durationTime = 0;
13     int startTime = 0;
14 };
15
16 void add_task_list(Task* lst) {
17     int count;
18     cout << "Введите количество задач" << endl;
19     cin >> count;
20     for (int i = 0; i < count; i++) {
21         cout << "Введите имя задача - ";
22         cin >> lst[i].name;
23         cout << "Введите тип задача (0-2)";
24         cin >> lst[i].type;
25         cout << "Введите время начала входа задача - ";
26         cin >> lst[i].startTime;
27         cout << "Введите время обработки задача - ";
28         cin >> lst[i].durationTime;
29         cout << "Введите приоритет задачи - ";
30         cin >> lst[i].priority;
31     }
32 }
33
34 void work_Proc(Task* Proc) {
35     if (Proc->durationTime <= 0) {
36         Proc->name = "";
37     }
38     else
39         Proc->durationTime = Proc->durationTime - 1;
40 }
41
42 void displacement(Task* Ocher) {
43
44     for (int i = 0; i < MAXOCH; i++) {
45         if (i != MAXOCH - 1)
46             Ocher[i] = Ocher[i + 1];
47         else
48             Ocher[MAXOCH - 1].name = "";

```

```

49     }
50 }
51
52 void displacement_st(Task* Stack) {
53     for (int i = 0; i < MAXST; i++) {
54         if (i != MAXST - 1)
55             Stack[i] = Stack[i + 1];
56         else
57             Stack[MAXST - 1].name = "";
58     }
59 }
60
61
62 void displacement_task_list(Task* List) {
63     for (int i = 0; i < TL; i++) {
64         if (i != TL - 1)
65             List[i] = List[i + 1];
66         else
67             List[TL - 1].name = "";
68     }
69 }
70
71 bool Stack_add(Task* Stack, Task Proc) {
72     bool res = false;
73     for (int k = 0; k < MAXST; k++) {
74         if (Stack[k].name == "") {
75             for (int j = k; j > 0; j--) {
76                 Stack[j] = Stack[j - 1];
77             }
78             Stack[0] = Proc;
79             res = true;
80             break;
81         }
82     }
83     if (res == false) {
84         cout << "Стек переполнен" << endl;
85     }
86     return res;
87 }
88
89
90 bool check_for_emptiness(Task* Ocher) {
91     bool tracker = true;
92     for (int i = 0; i < MAXOCH; i++) {
93         if (Ocher[i].name != "")
94             tracker = false;
95     }
96     return tracker;

```

```

97     }
98
99     void print_stack(Task* Stack) {
100         cout << "Стек:" << endl;
101         for (int i = 0; i < MAXST; i++) {
102             if (Stack[i].name != "")
103                 cout << "Задача " << Stack[i].name << " типа " << Stack[i].type << endl;
104         }
105     }
106
107     void print_ocher(Task* Ocher) {
108         cout << "Очередь:" << endl;
109         for (int i = 0; i < MAXOCH; i++) {
110             if (Ocher[i].name != "") {
111                 cout << "Задача " << Ocher[i].name << " типа " << Ocher[i].type << endl;
112             }
113         }
114     }
115
116     void print_proc(Task* Proc) {
117         cout << "Работа процессоров" << endl;
118         if (Proc->name != "") {
119             cout << "Задача " << Proc->name << " типа " << Proc->type << endl;
120         }
121     }
122
123     bool check1(Task* Proc, Task* Proc1, Task* Proc2, Task* Och, Task* Stack, Task* TaskList) {
124         if (Och[0].name == "" && Stack[0].name == "" && TaskList[0].name == "")
125             return false;
126         else
127             return true;
128     }
129
130
131
132     int main()
133     {
134         setlocale(LC_ALL, "Rus");
135         Task Ocher[MAXOCH];
136         int iOch = 0;
137         Task Stack[MAXST];
138         int iSt = 0;
139         Task TaskList[TL];
140         Task Proc0, Proc1, Proc2;
141         add_task_list(TaskList);
142         int timer = 0;
143
144

```



```

145 while (check1(&Proc0, &Proc1, &Proc2, Ocher, Stack, TaskList)) {
146     timer++;
147     cout << "Идет " << timer << " такт" << endl;
148     work_Proc(&Proc0);
149     work_Proc(&Proc1);
150     work_Proc(&Proc2);
151     if (timer == TaskList[0].startTime) {
152         for (int j = 0; j < MAXOCH; j++) {
153             if (Ocher[j].name == "") {
154                 Ocher[j] = TaskList[0];
155                 displacement_task_list(TaskList);
156             }
157             break;
158         }
159     }
160
161     bool cheker = true;
162     switch (Ocher[0].type) {
163     case 0:
164         if (Proc0.durationTime <= 0) {
165             Proc0 = Ocher[0];
166         }
167         else {
168             if
169                 (Proc0.priority < Ocher[0].priority) {
170                 if (Stack_add(Stack, Proc0) == false) {
171                     cheker = false;
172                     break;
173                 }
174                 Proc0 = Ocher[0];
175             }
176             else {
177                 if (Stack_add(Stack, Ocher[0]) == false) {
178                     cheker = false;
179                     break;
180                 }
181             }
182         }
183         displacement(Ocher);
184         break;
185     case 1:
186         if (Proc1.durationTime <= 0) {
187             Proc1 = Ocher[0];
188         }
189         else {
190             if (Proc1.priority < Ocher[0].priority) {
191                 if (Stack_add(Stack, Proc1) == false) {
192                     cheker = false;

```



```

193         break;
194     }
195     Proc1 = Ocher[0];
196 }
197 else {
198     if (Stack_add(Stack, Ocher[0]) == false) {
199         cheker = false;
200         break;
201     }
202 }
203 }
204 displacement(Ocher);
205 break;
206 case 2:
207     if (Proc2.durationTime <= 0) {
208         Proc2 = Ocher[0];
209     }
210 }
211 else {
212     if (Proc2.priority < Ocher[0].priority) {
213         if (Stack_add(Stack, Proc2) == false) {
214             cheker = false;
215             break;
216         }
217         Proc2 = Ocher[0];
218     }
219 }
220 else {
221     if (Stack_add(Stack, Ocher[0]) == false) {
222         cheker = false;
223         break;
224     }
225 }
226 }
227 displacement(Ocher);
228 break;
229 }
230
231 if (cheker == false) {
232     cout << "Стек переполнен" << endl;
233     break;
234 }
235 if (check_for_emptiness(Ocher)) {
236     switch (Stack[0].type) {
237     case 0:
238         if (Proc0.durationTime <= 0) {
239             Proc0 = Stack[0];
240             displacement_st(Stack);

```

```

241         }
242         break;
243     case 1:
244         if (Proc1.durationTime <= 0) {
245             Proc1 = Stack[0];
246             displacement_st(Stack);
247         }
248         break;
249     case 2:
250         if (Proc2.durationTime <= 0) {
251             Proc2 = Stack[0];
252             displacement_st(Stack);
253         }
254         break;
255     }
256 }
257 if (Ocher[0].name != "") {
258     print_ocher(Ocher);
259     cout << "\n\n";
260 }
261 if (Proc0.name != "") {
262     cout << "Процессор #1" << endl;
263     print_proc(&Proc0);
264     cout << "\n\n";

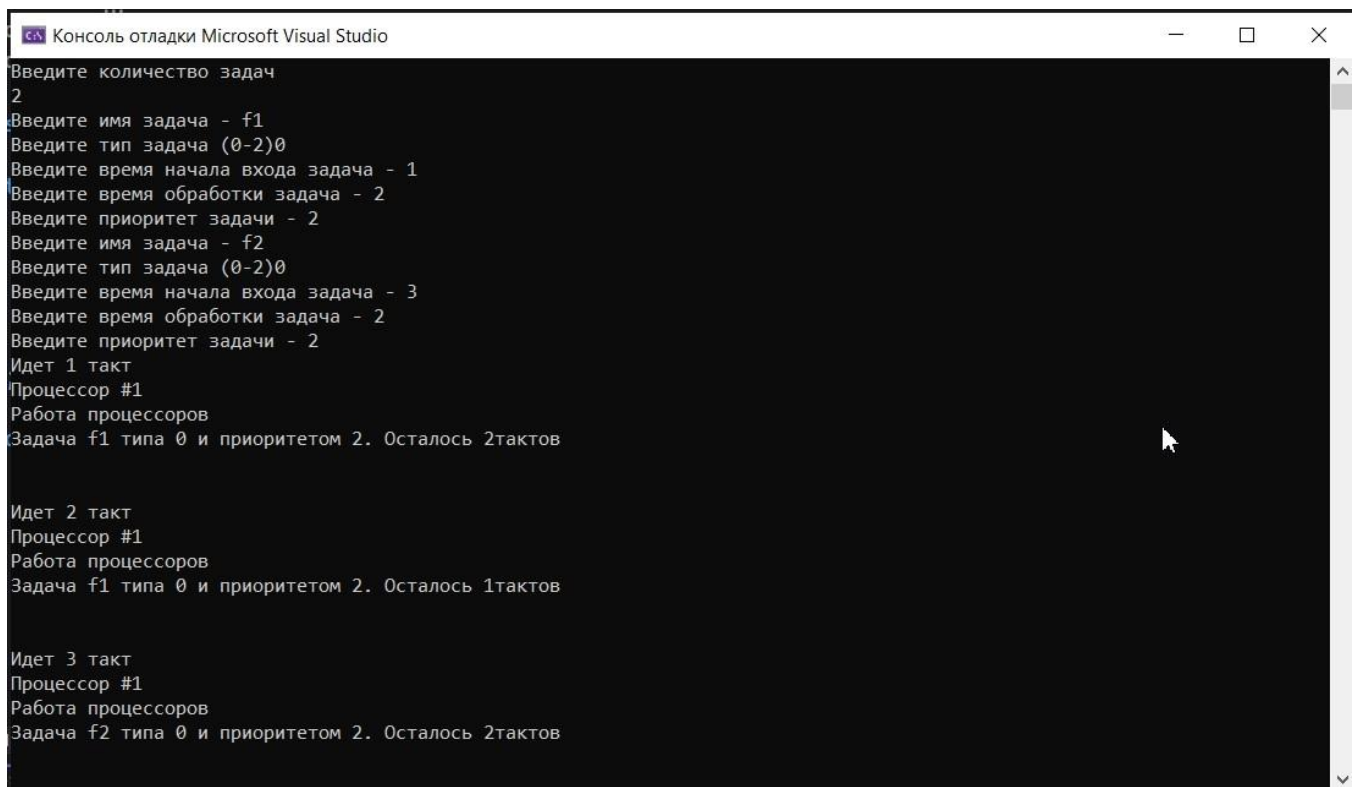
```

```

265 }
266 if (Proc1.name != "") {
267     cout << "Процессор #2" << endl;
268     print_proc(&Proc1);
269     cout << "\n\n";
270 }
271 if (Proc2.name != "") {
272     cout << "Процессор #3" << endl;
273     print_proc(&Proc2);
274     cout << "\n\n";
275 }
276 if (Stack[0].name != "") {
277     print_stack(Stack);
278     cout << "\n\n";
279 }
280
281 }
282
283 }
284

```

Контрольные примеры:

The image shows a screenshot of the 'Консоль отладки Microsoft Visual Studio' (Visual Studio Debug Console) window. The window has a title bar with the Visual Studio logo and standard minimize, maximize, and close buttons. The console text is as follows:

```
Введите количество задач
2
Введите имя задача - f1
Введите тип задача (0-2)0
Введите время начала входа задача - 1
Введите время обработки задача - 2
Введите приоритет задачи - 2
Введите имя задача - f2
Введите тип задача (0-2)0
Введите время начала входа задача - 3
Введите время обработки задача - 2
Введите приоритет задачи - 2
Идет 1 такт
Процессор #1
Работа процессоров
Задача f1 типа 0 и приоритетом 2. Осталось 2тактов

Идет 2 такт
Процессор #1
Работа процессоров
Задача f1 типа 0 и приоритетом 2. Осталось 1тактов

Идет 3 такт
Процессор #1
Работа процессоров
Задача f2 типа 0 и приоритетом 2. Осталось 2тактов
```

Вывод

Изучил стеки и очереди и получил практические навыки их реализации. Как итог написал программу, характеристики которой соответствуют поставленному заданию.