Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. Тихонова

Департамент электронной инженерии

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ №5

по дисциплине «Системное программирование»

«Комбинирование разноязыковых модулей»

Вариант 24

Студент гр. БИБ201

Шадрунов Алексей

Дата выполнения: 13 февраля 2023 г.

Преподаватель:

Смирнов Д. В.

«___» _____ 2023 г.

Москва, 2023

Содержание

1 Цель работы

Написать программу, в которой создается одномерный числовой массив. После заполнения значениями (случайными числами) массива в нем нужно выполнить циклическую перестановку элементов. Количество позиций для циклической перестановки вводится пользователем с клавиатуры.

2 Ход работы

2.1 Автомат с одним видом товара

Реализуем алгоритм работы автомата с одним видом товара — напитком за **9 рублей**.

- Начальное состояние — 0. В автомате нет денег. Автомат ожидает внесения монет.

- Вносится монета — 1, 2, 5 или 10 рублей. В зависимости от внесённой суммы автомат переходит в следующее состояние, определяемое по таблице переходов (Таблица 1).

- В случае, если сумма денег в автомате больше или равна 9 рублей, автомат выдаёт напиток и сдачу и переходит в состояние 0.

- В случае, если сумма денег меньше 9, ожидается следующее внесение монеты.

Таблица переходов построена по следующему принципу: в первой колонке указано текущее состояние (количество рублей в автомате), во второй колонке — монета, полученная автоматом, в третьей колонке — следующее состояние. В следующих четырёх колонках указан выходной алфавит автомата. Если в колонке стоит единица, то такую сдачу следует выдать.

Таблица переходов представляет собой таблицу переходов для данного конечного автомата.

Таблица 1 – Таблица переходов для автомата с одним видом товаров

State	Insert	Nextstate	Change 1	Change 2	Change 2 2	Change 5
0	1	1	0	0	0	0
0	2	2	0	0	0	0
0	5	5	0	0	0	0
0	10	0	1	0	0	0
1	1	2	0	0	0	0
1	2	3	0	0	0	0
1	5	6	0	0	0	0
1	10	0	0	1	0	0
2	1	3	0	0	0	0
2	2	4	0	0	0	0

2	5	7	0	0	0	0
2	10	0	1	1	0	0
3	1	4	0	0	0	0
3	2	5	0	0	0	0
3	5	8	0	0	0	0
3	10	0	0	0	1	0
4	1	5	0	0	0	0
4	2	6	0	0	0	0
4	5	0	0	0	0	0
4	10	0	0	0	0	1
5	1	6	0	0	0	0
5	2	7	0	0	0	0
5	5	0	1	0	0	0
5	10	0	1	0	0	1
6	1	7	0	0	0	0
6	2	8	0	0	0	0
6	5	0	0	1	0	0
6	10	0	0	1	0	1
7	1	8	0	0	0	0
7	2	0	1	0	0	0
7	5	0	1	1	0	0
7	10	0	1	1	0	1
8	1	0	0	0	0	0
8	2	0	1	0	0	0
8	5	0	0	0	1	0
8	10	0	0	0	1	1

В листинге 1 приведена реализация данного алгоритма с помощью таблицы переходов.

```c
#include <stdio.h>
#include <stdlib.h>
/*
    * single row of FSM:
    * contains current state, input (a coin),
    * next state and output (the change)
    */
typedef struct row
{
    int state;      // states from 0 to 8
    int insert;     // an inserted coin (1, 2, 5, 10)
    int next_state; // state after coin
    int change[4];  // should a machine return 1, 2, 2x2 or 5 coin of change
} row;
// manually add rows to table
void create_table(row *pointer)
{
    pointer[0] = (struct row){0, 1, 1, {0, 0, 0, 0}};
    pointer[1] = (struct row){0, 2, 2, {0, 0, 0, 0}};
    pointer[2] = (struct row){0, 5, 5, {0, 0, 0, 0}};
    pointer[3] = (struct row){0, 10, 0, {1, 0, 0, 0}};
    pointer[4] = (struct row){1, 1, 2, {0, 0, 0, 0}};
    pointer[5] = (struct row){1, 2, 3, {0, 0, 0, 0}};
    pointer[6] = (struct row){1, 5, 6, {0, 0, 0, 0}};
```

```
25      pointer[7] = (struct row){1, 10, 0, {0, 1, 0, 0}};
26      pointer[8] = (struct row){2, 1, 3, {0, 0, 0, 0}};
27      pointer[9] = (struct row){2, 2, 4, {0, 0, 0, 0}};
28      pointer[10] = (struct row){2, 5, 7, {0, 0, 0, 0}};
29      pointer[11] = (struct row){2, 10, 0, {1, 1, 0, 0}};
30      pointer[12] = (struct row){3, 1, 4, {0, 0, 0, 0}};
31      pointer[13] = (struct row){3, 2, 5, {0, 0, 0, 0}};
32      pointer[14] = (struct row){3, 5, 8, {0, 0, 0, 0}};
33      pointer[15] = (struct row){3, 10, 0, {0, 0, 1, 0}};
34      pointer[16] = (struct row){4, 1, 5, {0, 0, 0, 0}};
35      pointer[17] = (struct row){4, 2, 6, {0, 0, 0, 0}};
36      pointer[18] = (struct row){4, 5, 0, {0, 0, 0, 0}};
37      pointer[19] = (struct row){4, 10, 0, {0, 0, 0, 1}};
38      pointer[20] = (struct row){5, 1, 6, {0, 0, 0, 0}};
39      pointer[21] = (struct row){5, 2, 7, {0, 0, 0, 0}};
40      pointer[22] = (struct row){5, 5, 0, {1, 0, 0, 0}};
41      pointer[23] = (struct row){5, 10, 0, {1, 0, 0, 1}};
42      pointer[24] = (struct row){6, 1, 7, {0, 0, 0, 0}};
43      pointer[25] = (struct row){6, 2, 8, {0, 0, 0, 0}};
44      pointer[26] = (struct row){6, 5, 0, {0, 1, 0, 0}};
45      pointer[27] = (struct row){6, 10, 0, {0, 1, 0, 1}};
46      pointer[28] = (struct row){7, 1, 8, {0, 0, 0, 0}};
47      pointer[29] = (struct row){7, 2, 0, {1, 0, 0, 0}};
48      pointer[30] = (struct row){7, 5, 0, {1, 1, 0, 0}};
49      pointer[31] = (struct row){7, 10, 0, {1, 1, 0, 1}};
50      pointer[32] = (struct row){8, 1, 0, {0, 0, 0, 0}};
51      pointer[33] = (struct row){8, 2, 0, {1, 0, 0, 0}};
52      pointer[34] = (struct row){8, 5, 0, {0, 0, 1, 0}};
53      pointer[35] = (struct row){8, 10, 0, {0, 0, 1, 1}};
54  }
55
56  int main()
57  {
58      // init FSM with precalculated values
59      int states_num = 36;
60      row *fsm = malloc(sizeof(row) * states_num);
61      create_table(fsm);
62
63      // start a machine
64      printf("Price is 9 \n\n");
65
66      int cur_state = 0;
67      int input;
68
69      while (1)
70      {
71          // new iteration
72          printf("\nCurrent state: %d\n", cur_state);
73          printf("Enter coin (1, 2, 5 or 10)? \n");
74          scanf("%d", &input);
75
76          if (input != 1 && input != 2 && input != 5 && input != 10)
77              continue;  // wrong coin
78
79          int i = 0;
80          while (fsm[i].state != cur_state || fsm[i].insert != input)
81              i++;  // find current row in table
82
83          if (fsm[i].next_state == 0)  // the machine returned to S0
84          {
85              // so we give a drink!
86              printf("Here's your drink! \n");
87              printf("And change: \n");
88              printf(" - 1 ruble:  %d \n", fsm[i].change[0]);
89              printf(" - 2 rubles: %d \n", fsm[i].change[1]);
90              printf(" - 4 rubles: %d \n", fsm[i].change[2]);
91              printf(" - 5 rubles: %d \n\n", fsm[i].change[3]);
92          }
93          // go to next state
```

```
94          printf("going to state %d \n", fsm[i].next_state);
95          cur_state = fsm[i].next_state;
96      }
97      return 0;
98  }
```

Листинг 1 – Реализация первого конечного автомата

На рисунке 1 приведён граф переходов для данной реализации автомата. На рисунке 2 показана работа программы.

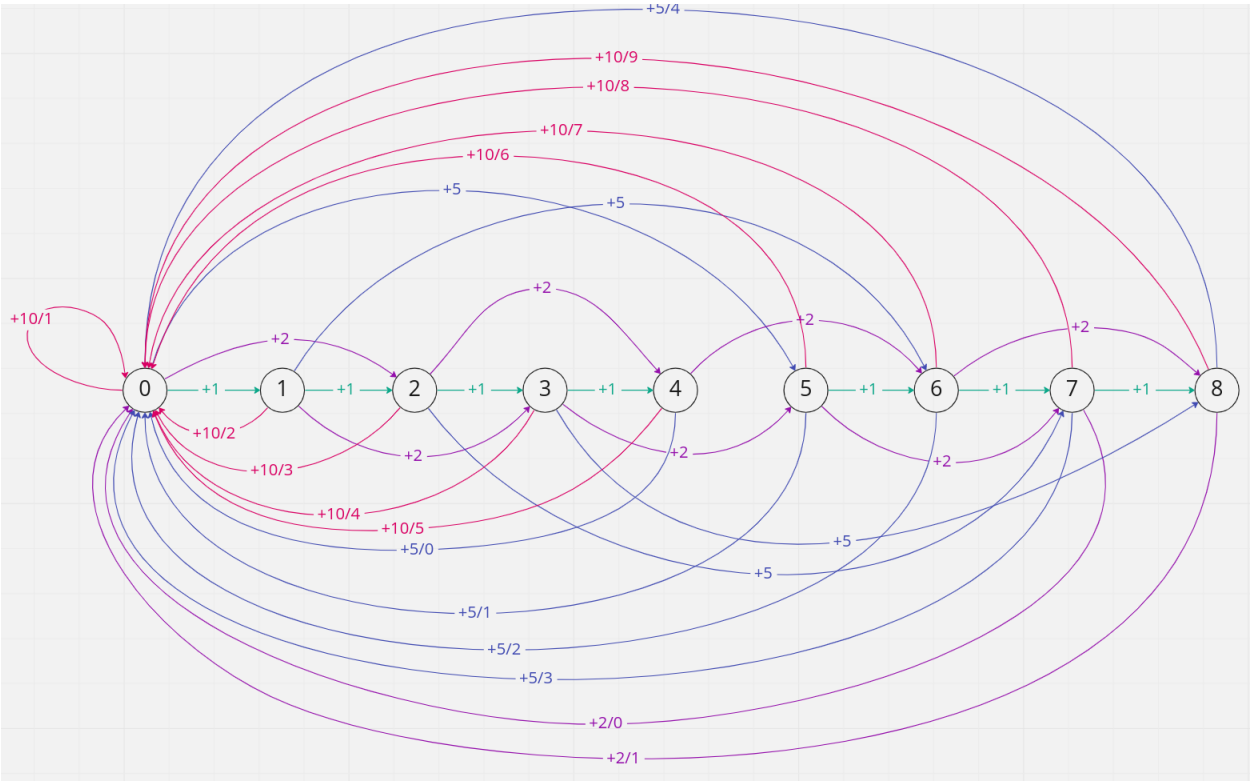

Рисунок 1 – Граф переходов первого автомата

```
alex@alex-nb ~/D/y/h/fsm (master)> ./a.out
Price is 9


Current state: 0
Enter coin (1, 2, 5 or 10)?
5
going to state 5

Current state: 5
Enter coin (1, 2, 5 or 10)?
2
going to state 7

Current state: 7
Enter coin (1, 2, 5 or 10)?
5
Here's your drink!
And change:
 - 1 ruble:  1
 - 2 rubles: 1
 - 4 rubles: 0
 - 5 rubles: 0

going to state 0

Current state: 0
Enter coin (1, 2, 5 or 10)?
```

Рисунок 2 – Работа конечного автомата

2.2 Автомат Мили

Модифицируем автомат для обработки двух видов товаров. Первый товар, как и раньше, стоит 9 рублей, а второй — 3 рубля. Алгоритм:

- Первый шаг алгоритма — выбрать напиток. Автомат находится в состоянии 0, затем в зависимости от выбранного товара — 1 или 2 — автомат переходит в состояние 10 или 20 соответственно.

- Далее работа автомата повторяет автомат из первого пункта, только состояния кодируются двумя цифрами, первая из которых обозначает номер напитка.

- После выдачи напитка и сдачи автомат возвращается в состояние 0.

В листинге 2 приведена реализация данного алгоритма с помощью таблицы переходов.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /*
5      * single row of FSM:
6      * contains current state, input (a coin),
7      * next state and output (the change)
8      */
9  typedef struct row
10 {
11     int state;      // states from 0 to 8
12     int insert;     // an inserted coin (1, 2, 5, 10)
13     int next_state; // state after coin
14     int change[4];  // should a machine return 1, 2, 2x2 or 5 coin of change
15     int item;       // an item to give (1 or 2)
16 } row;
17
18 // manually add rows to table
19 void create_table(row *pointer)
```

```
20 {
21     pointer[0] = (struct row){0, 1, 10, {0, 0, 0, 0}, 0};
22     pointer[1] = (struct row){0, 2, 20, {0, 0, 0, 0}, 0};
23     pointer[2] = (struct row){10, 1, 11, {0, 0, 0, 0}, 0};
24     pointer[3] = (struct row){10, 2, 12, {0, 0, 0, 0}, 0};
25     pointer[4] = (struct row){10, 5, 15, {0, 0, 0, 0}, 0};
26     pointer[5] = (struct row){10, 10, 0, {1, 0, 0, 0}, 1};
27     pointer[6] = (struct row){11, 1, 12, {0, 0, 0, 0}, 0};
28     pointer[7] = (struct row){11, 2, 13, {0, 0, 0, 0}, 0};
29     pointer[8] = (struct row){11, 5, 16, {0, 0, 0, 0}, 0};
30     pointer[9] = (struct row){11, 10, 0, {0, 1, 0, 0}, 1};
31     pointer[10] = (struct row){12, 1, 13, {0, 0, 0, 0}, 0};
32     pointer[11] = (struct row){12, 2, 14, {0, 0, 0, 0}, 0};
33     pointer[12] = (struct row){12, 5, 17, {0, 0, 0, 0}, 0};
34     pointer[13] = (struct row){12, 10, 0, {1, 1, 0, 0}, 1};
35     pointer[14] = (struct row){13, 1, 14, {0, 0, 0, 0}, 0};
36     pointer[15] = (struct row){13, 2, 15, {0, 0, 0, 0}, 0};
37     pointer[16] = (struct row){13, 5, 18, {0, 0, 0, 0}, 0};
38     pointer[17] = (struct row){13, 10, 0, {0, 0, 1, 0}, 1};
39     pointer[18] = (struct row){14, 1, 15, {0, 0, 0, 0}, 0};
40     pointer[19] = (struct row){14, 2, 16, {0, 0, 0, 0}, 0};
41     pointer[20] = (struct row){14, 5, 0, {0, 0, 0, 0}, 1};
42     pointer[21] = (struct row){14, 10, 14, {0, 0, 0, 1}, 0};
43     pointer[22] = (struct row){15, 1, 16, {0, 0, 0, 0}, 0};
44     pointer[23] = (struct row){15, 2, 17, {0, 0, 0, 0}, 0};
45     pointer[24] = (struct row){15, 5, 0, {1, 0, 0, 0}, 1};
46     pointer[25] = (struct row){15, 10, 0, {1, 0, 0, 1}, 1};
47     pointer[26] = (struct row){16, 1, 17, {0, 0, 0, 0}, 0};
48     pointer[27] = (struct row){16, 2, 18, {0, 0, 0, 0}, 0};
49     pointer[28] = (struct row){16, 5, 0, {0, 1, 0, 0}, 1};
50     pointer[29] = (struct row){16, 10, 0, {0, 1, 0, 1}, 1};
51     pointer[30] = (struct row){17, 1, 18, {0, 0, 0, 0}, 0};
52     pointer[31] = (struct row){17, 2, 0, {1, 0, 0, 0}, 1};
53     pointer[32] = (struct row){17, 5, 0, {1, 1, 0, 0}, 1};
54     pointer[33] = (struct row){17, 10, 0, {1, 1, 0, 1}, 1};
55     pointer[34] = (struct row){18, 1, 0, {0, 0, 0, 0}, 1};
56     pointer[35] = (struct row){18, 2, 0, {1, 0, 0, 0}, 1};
57     pointer[36] = (struct row){18, 5, 0, {0, 0, 1, 0}, 1};
58     pointer[37] = (struct row){18, 10, 0, {0, 0, 1, 1}, 1};
59     pointer[38] = (struct row){20, 1, 21, {0, 0, 0, 0}, 0};
60     pointer[39] = (struct row){20, 2, 22, {0, 0, 0, 0}, 0};
61     pointer[40] = (struct row){20, 5, 0, {0, 1, 0, 0}, 2};
62     pointer[41] = (struct row){20, 10, 0, {0, 1, 0, 1}, 2};
63     pointer[42] = (struct row){21, 1, 22, {0, 0, 0, 0}, 0};
64     pointer[43] = (struct row){21, 2, 0, {0, 0, 0, 0}, 2};
65     pointer[44] = (struct row){21, 5, 0, {1, 1, 0, 0}, 2};
66     pointer[45] = (struct row){21, 10, 0, {1, 1, 0, 1}, 2};
67     pointer[46] = (struct row){22, 1, 0, {0, 0, 0, 0}, 2};
68     pointer[47] = (struct row){22, 2, 0, {1, 0, 0, 0}, 2};
69     pointer[48] = (struct row){22, 5, 0, {0, 0, 1, 0}, 2};
70     pointer[49] = (struct row){22, 10, 0, {1, 1, 1, 1}, 2};
71 }
72
73 int main()
74 {
75     // init FSM with precalculated values
76     int states_num = 50;
77     row *fsm = malloc(sizeof(row) * states_num);
78     create_table(fsm);
79
80     // start a machine
81     printf("Price of first item:   9 \n");
82     printf("Price of second item:  3 \n\n");
83
84     int cur_state = 0;
85     int input;
86
87     while (1)
88     {
```

```
89          // new iteration
90          printf("\nCurrent state: %d\n", cur_state);
91          if (cur_state == 0)
92              printf("Select item (1 or 2)? \n");
93          else
94              printf("Enter coin (1, 2, 5 or 10)? \n");
95          scanf("%d", &input);
96
97          if (cur_state == 0 && input != 1 && input != 2)
98              continue; // wrong item
99
100         if (cur_state != 0 && input != 1 && input != 2 && input != 5 && input
        != 10)
101             continue; // wrong coin
102
103         int i = 0;
104         while (fsm[i].state != cur_state || fsm[i].insert != input)
105             i++; // find current row in table
106
107         if (fsm[i].next_state == 0) // the machine returned to S0
108         {
109             // so we give a drink!
110             printf("Here's your item #%d! \n", fsm[i].item);
111             printf("And change: \n");
112             printf(" - 1 ruble:  %d \n", fsm[i].change[0]);
113             printf(" - 2 rubles: %d \n", fsm[i].change[1]);
114             printf(" - 4 rubles: %d \n", fsm[i].change[2]);
115             printf(" - 5 rubles: %d \n\n", fsm[i].change[3]);
116         }
117
118         // go to next state
119         printf("going to state %d \n", fsm[i].next_state);
120         cur_state = fsm[i].next_state;
121     }
122
123     return 0;
124 }
```

Листинг 2 – Фрагмент функции main

На рисунке 3 приведён граф переходов для данной реализации автомата.

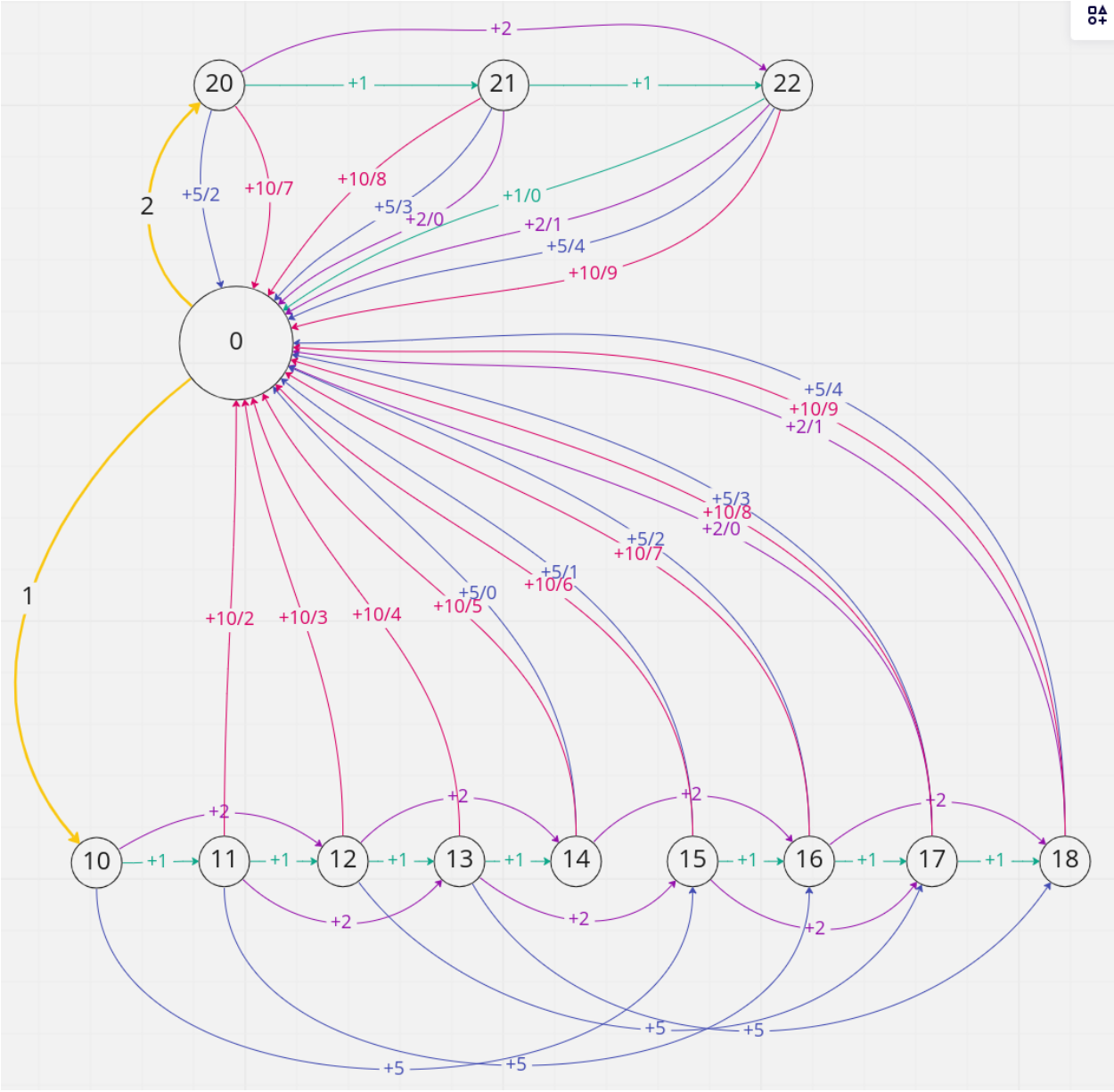

Рисунок 3 – Граф переходов автомата с двумя товарами

Таблица переходов построена аналогично. Последняя колонка показывает номер товара, который выдаёт автомат (Таблица 2).

Таблица 2 – Таблица переходов для автомата с двумя видами товаров

State	Insert	Nextstate	Change 1	Change 2	Change 2 2	Change 5	Item
0	1	10	0	0	0	0	0
0	2	20	0	0	0	0	0
10	1	11	0	0	0	0	0
10	2	12	0	0	0	0	0
10	5	15	0	0	0	0	0
10	10	0	1	0	0	0	1
11	1	12	0	0	0	0	0

11	2	13	0	0	0	0	0
11	5	16	0	0	0	0	0
11	10	0	0	1	0	0	1
12	1	13	0	0	0	0	0
12	2	14	0	0	0	0	0
12	5	17	0	0	0	0	0
12	10	0	1	1	0	0	1
13	1	14	0	0	0	0	0
13	2	15	0	0	0	0	0
13	5	18	0	0	0	0	0
13	10	0	0	0	1	0	1
14	1	15	0	0	0	0	0
14	2	16	0	0	0	0	0
14	5	0	0	0	0	0	1
14	10	14	0	0	0	1	0
15	1	16	0	0	0	0	0
15	2	17	0	0	0	0	0
15	5	0	1	0	0	0	1
15	10	0	1	0	0	1	1
16	1	17	0	0	0	0	0
16	2	18	0	0	0	0	0
16	5	0	0	1	0	0	1
16	10	0	0	1	0	1	1
17	1	18	0	0	0	0	0
17	2	0	1	0	0	0	1
17	5	0	1	1	0	0	1
17	10	0	1	1	0	1	1
18	1	0	0	0	0	0	1
18	2	0	1	0	0	0	1
18	5	0	0	0	1	0	1
18	10	0	0	0	1	1	1
20	1	21	0	0	0	0	0
20	2	22	0	0	0	0	0
20	5	0	0	1	0	0	2
20	10	0	0	1	0	1	2
21	1	22	0	0	0	0	0
21	2	0	0	0	0	0	2
21	5	0	1	1	0	0	2
21	10	0	1	1	0	1	2
22	1	0	0	0	0	0	2
22	2	0	1	0	0	0	2
22	5	0	0	0	1	0	2
22	10	0	1	1	1	1	2

На рисунке 4 показана работа программы.

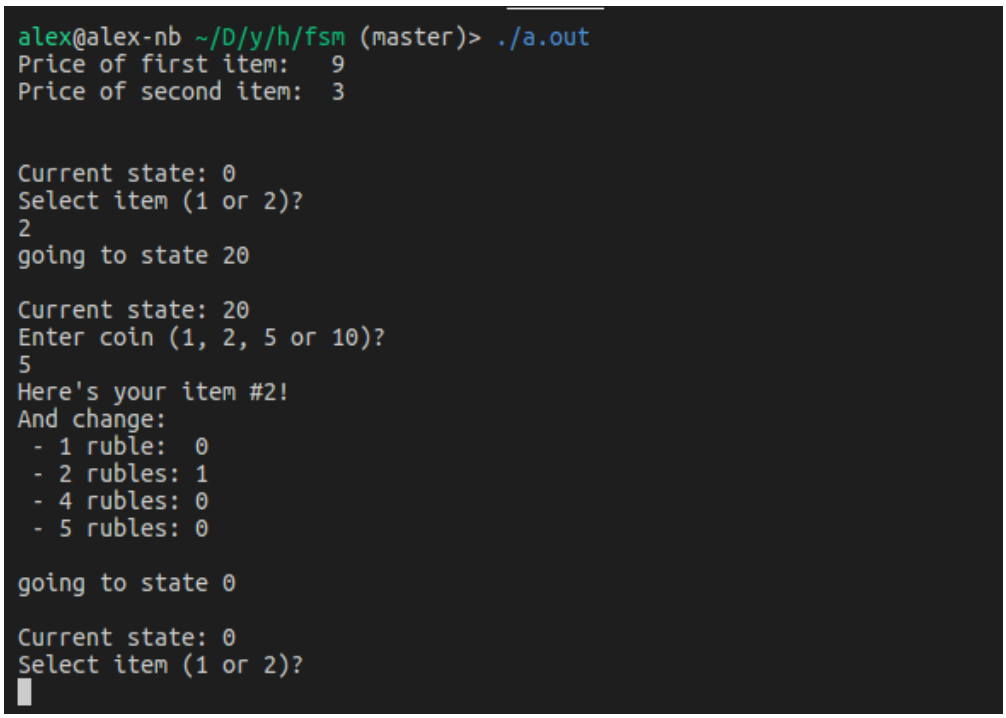

Рисунок 4 – Работа конечного автомата

2.3 Автомат Мура

Для данного автомата продемонстрируем, как будет выглядеть граф Мура. Особенность автомата Мура заключается в том, что выход автомата определяется только текущим состоянием. Это значит, что информацию о внесённых деньгах приходится хранить в текущих состояниях. Граф представлен на рисунке 5.

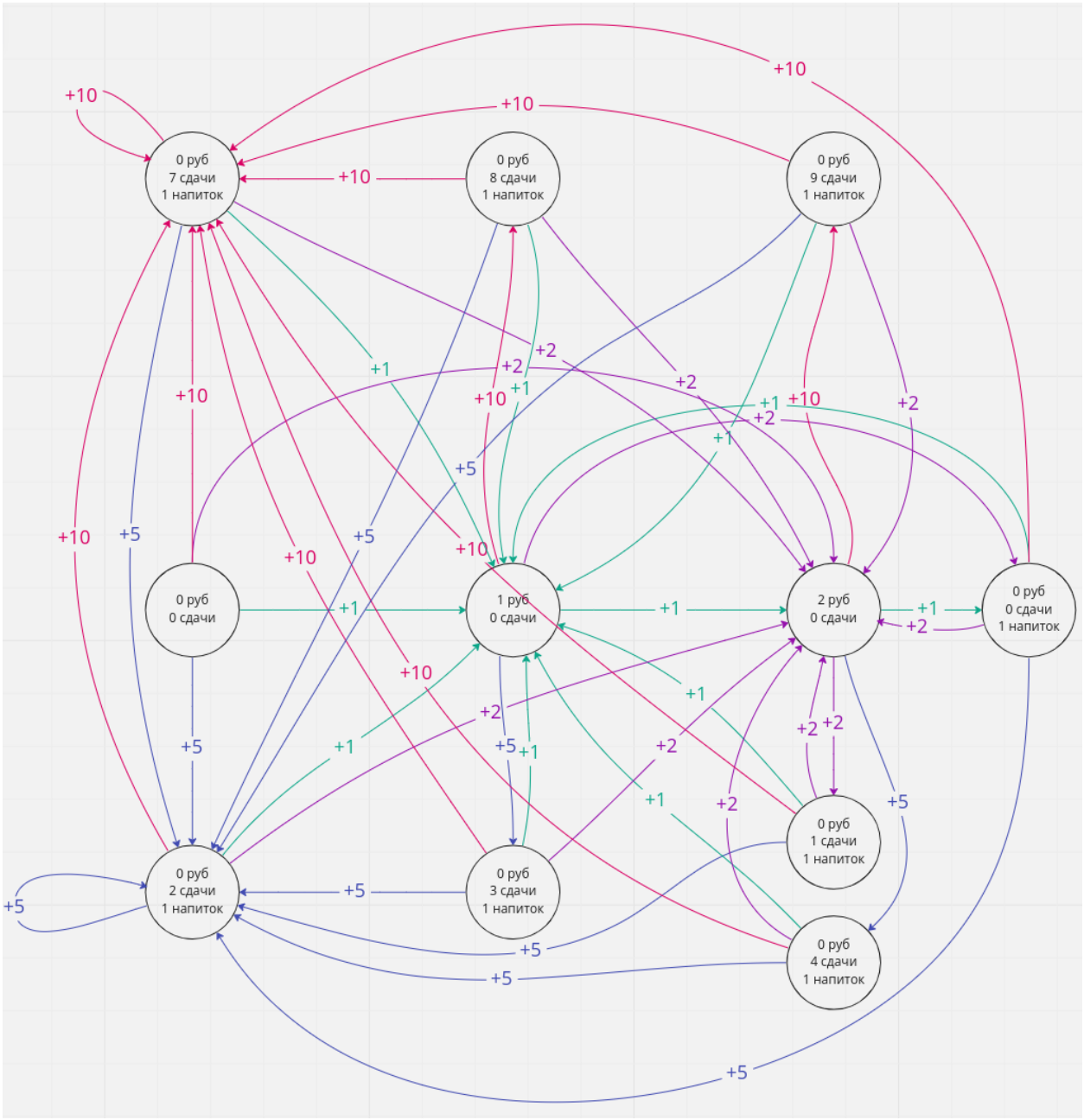

Рисунок 5 – Автомат Мура

3 Выводы о проделанной работе

В рамках данной работы я познакомился с архитектурой программ на основе конечных автоматов и принципом реализации алгоритма устройства на основе конечных автоматов. Разобрал различия между автоматами Мили и Мура. Реализовал логику работы устройства «Вендинговый автомат» на языке C.