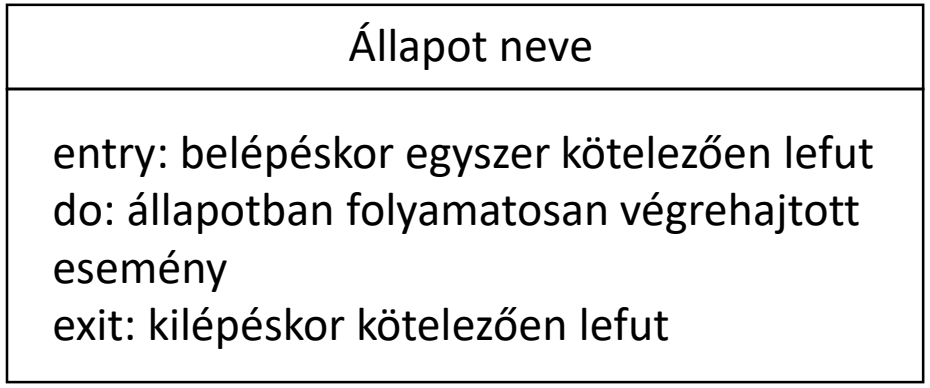
ÓBUDAI EGYETEM
NEUMANN JÁNOS INFORMATIKAI KAR

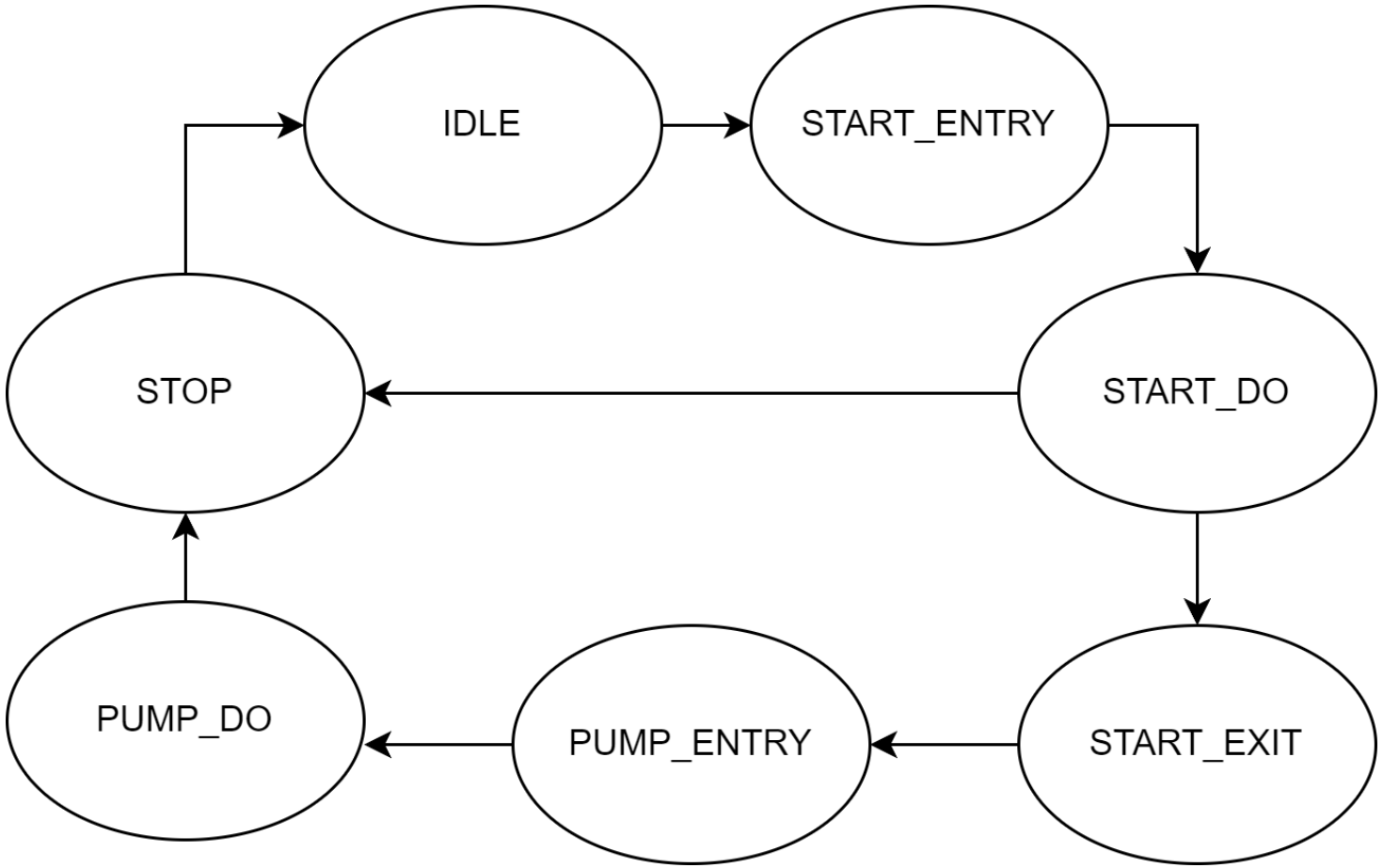
FSM – Finite State Machine

UML – State Machine



- **Entry:** Annak a tevékenységnek a leírása, ami az állapotba való belépés esetén elvégzendő. Ez a tevékenység mindig végrehajtódik, attól függetlenül, hogy milyen átmenet miatt lépett be az objektum az állapotba.
- **Do:** Annak a tevékenységnek a leírása, amely az adott állapothoz tartozik. Ha több ilyen is van, azokat a végrehajtásuk sorrendjében soroljuk fel. Ezt a tevékenységet egy esemény félbeszakíthatja. Az esemény lehet:
- **Esemény:** Belső esemény bekövetkezéséhez kapcsolt tevékenység leírása.
- **Exit:** Annak a tevékenységnek a leírása, amelyet bármely átmenet bekövetkezése esetén még az állapotból való kilépés előtt végre kell hajtani.

FMS – Finite State Machine (pump machine)



NC
C
NO
D1
Q1
R1
SRD-05VDC-SL-C
10A 250VAC 10A 125VAC
10A 30VDC 10A 28VDC
Keyes_SR1y
ON_Led
S

5V GND D5

AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)

```
#define RELAY 5      //RELAY modul
#define PUMPLED 4    //pump status LED
#define READYLED 3   //ready status LED

//state machine def
typedef enum {
  IDLE,
  START_ENTRY,
  START_DO,
  START_EXIT,
  PUMP_ENTRY,
  PUMP_DO,
  STOP
} State_Type;
```

```
16  //timer def
17  typedef struct {
18    unsigned long prevTime;
19    int deltaTime;
20  } sTimer;
21
22  sTimer StartLedTimer = {
23    .prevTime = 1,
24    .deltaTime = 500
25  };
26
```

```
27  sTimer PumpLedTimer = {
28    .prevTime = 0,
29    .deltaTime = 1000
30  };
31
32  State_Type curr_state;
```

```
34  unsigned char CheckTime(sTimer *timer);
35  String CheckSerialData(void);
36  void IdleStateHandler(void);
37  void StartEntryStateHandler(void);
38  void StartDoStateHandler(void);
39  void StartExitStateHandler(void);
40  void PumpEntryStateHandler(void);
41  void PumpDoStateHandler(void);
42  void StopStateHandler(void);
```

```
44  void setup() {
45    //init serial monitor
46    Serial.begin(9600);
47
48    //led init
49    pinMode(RELAY, OUTPUT);
50    pinMode(PUMPLED, OUTPUT);
51    pinMode(READYLED, OUTPUT);
52    digitalWrite(RELAY, LOW);
53    digitalWrite(PUMPLED, LOW);
54    digitalWrite(READYLED, HIGH);
55
56    //set firt state
57    curr_state = STOP;
58  }
```

```
60  void loop() {
61    switch (curr_state) {
62      case IDLE:
63        IdleStateHandler();
64        break;
65
66      case START_ENTRY:
67        StartEntryStateHandler();
68        break;
69
70      case START_DO:
71        StartDoStateHandler();
72        break;
73
74      case START_EXIT:
75        StartExitStateHandler();
76        break;
77
78      case PUMP_ENTRY:
79        PumpEntryStateHandler();
80        break;
81
82      case PUMP_DO:
83        PumpDoStateHandler();
84        break;
85
86      case STOP:
87        StopStateHandler();
88        break;
89    }
90  }
91
```

```
92  //************ IDLE STATE HANDLER ************
93  void IdleStateHandler(void) {
94    if (CheckSerialData() == "start") {
95      curr_state = START_ENTRY;
96    }
97  }
98
99  //************ START STATES HANDLERS ************
100 unsigned long StartTime = 0;
101 void StartEntryStateHandler(void) {
102   StartTime = millis();
103
104   digitalWrite(READYLED, LOW);
105   digitalWrite(RELAY, LOW);
106   digitalWrite(PUMPLED, LOW);
107
108   curr_state = START_DO;
109 }
```

```
110
111 void StartDoStateHandler(void) {
112   //check incoming commad
113   if (CheckSerialData() == "stop") {
114     curr_state = STOP;
115   }
116
117   if (CheckTime(&StartLedTimer)) {
118     digitalWrite(RELAY, !digitalRead(RELAY));
119     digitalWrite(PUMPLED, !digitalRead(PUMPLED));
120     digitalWrite(READYLED, !digitalRead(READYLED));
121   }
122
123   if (millis() - StartTime > 10000) {
124     curr_state = START_EXIT;
125   }
126 }
```

```
128  void StartExitStateHandler(void) {
129    digitalWrite(RELAY, LOW);
130    digitalWrite(PUMPLED, LOW);
131    digitalWrite(READYLED, LOW);
132
133    curr_state = PUMP_ENTRY;
134  }
135
136  //************ PUMP STATES HANDLERS ************
137  unsigned long StartPumpTime = 0;
138  void PumpEntryStateHandler(void) {
139    StartPumpTime = millis();
140    //pump start
141    digitalWrite(RELAY, HIGH);
142    curr_state = PUMP_DO;
143  }
```

```
145  void PumpDoStateHandler(void) {
146    //check incoming commad
147    if (CheckSerialData() == "stop") {
148      curr_state = STOP;
149    }
150
151    if (CheckTime(&PumpLedTimer)) {
152      digitalWrite(PUMPLED, !digitalRead(PUMPLED));
153    }
154
155    if (millis() - StartPumpTime > 20000) {
156      curr_state = STOP;
157    }
158  }
```

```
160  //************ STOP STATE HANDLER ************
161  void StopStateHandler(void) {
162    digitalWrite(RELAY, LOW);
163    digitalWrite(PUMPLED, LOW);
164    digitalWrite(READYLED, HIGH);
165
166    curr_state = IDLE;
167  }
168
169  //************ OTHER FUNCTIONS ************
170  unsigned char CheckTime(sTimer *timer) {
171    unsigned long actTime = millis();
172    if (actTime > ((*timer).prevTime + (*timer).deltaTime)) {
173      (*timer).prevTime = actTime;
174      return 1;
175    }
176    return 0;
177  }
```

```
179  String CheckSerialData(void) {
180    if (Serial.available()) {
181      String data = Serial.readString();
182      data.trim();  // remove \r \n
183      return data;
184    }
185    return "\0";
186  }
187
```