```c
/*@author Michal Pawlowski*/

#define _CRT_SECURE_NO_WARNINGS
#define BUF_SIZE 100
#define OPEN_KEY '1'
#define STOP_KEY '2'
#define CLOSE_KEY '3'
#define EXIT_KEY '4'

#include <stdlib.h>
#include <string.h>
#include <Windows.h>
#include <conio.h>
#include <stdio.h>
#include <stdbool.h>

typedef enum machineState {

    open,
    opening,
    stop,
    closing,
    close

}machineState;

typedef struct circularBuffer {

    char tab[BUF_SIZE];
    int writeIndex;
    int readIndex;
    int bufferLength;

}circularBuffer;

typedef struct gateMachine {

    machineState gateCase;
    int percent;

}gateMachine;

/** Funkcja tworzy bufor cykilczny
@date 2020-05-15
@return wskaznik na bufor cykilczny
*/
circularBuffer createCircularBuffer() {

    circularBuffer buffer;
    buffer.writeIndex = 0;
    buffer.readIndex = 0;
    buffer.bufferLength = 0;

```

```
54      return buffer;
55  }
56
57  /** Funkcja dodaje element do bufora cyklicznego
58  @date 2020-05-15
59  @param buffer wskaznik na bufor
60  @param input dane wejsciowe
61  */
62  void addToCircularBuffer(circularBuffer* buffer, char input) {
63
64      if (input == OPEN_KEY || input == STOP_KEY || input == CLOSE_KEY ||
        input == EXIT_KEY) {
65
66          if (buffer->bufferLength == BUF_SIZE) {
67
68              printf("ERROR: Buffer overflow!\n");
69              return;
70          }
71          else {
72
73              buffer->tab[buffer->writeIndex] = input;
74              buffer->bufferLength++;
75              buffer->writeIndex++;
76              if (buffer->writeIndex == BUF_SIZE) buffer->writeIndex = 0;
77          }
78      }
79  }
80
81  /** Funkcja odczytuje wartosc z bufora oraz niszczy ja
82  @date 2020-05-15
83  @param buffer wskaznik na bufor
84  */
85  char readFromCircularBuffer(circularBuffer* buffer) {
86
87      if (buffer->bufferLength > 0) {
88
89          char temp = buffer->tab[buffer->readIndex];
90
91          buffer->tab[buffer->readIndex] = '\0';
92          buffer->bufferLength--;
93          buffer->readIndex++;
94
95          if (buffer->readIndex == BUF_SIZE) buffer->readIndex = 0;
96          return temp;
97      }
98      else {
99
100         return '\0';
101     }
102 }
103
104 /** Funkcja obslugujaca maszyne stanow
105 @date 2020-05-15
```

```
106 @param buffer wskaznik na bufor
107 @param gate wskaznik na maszyne stanow
108 @param exitStatus zmienna okreslajaca czy zakonczyc program
109 */
110 void caseServis(circularBuffer* buffer, gateMachine* gate, bool* exitStatus) ↵
       {
111
112     switch (readFromCircularBuffer(buffer)) {
113
114     case OPEN_KEY:
115
116         switch (gate->gateCase) {
117
118         case open:
119             break;
120
121         case opening:
122             gate->percent -= 10;
123             if (gate->percent == 0) gate->gateCase = open;
124             break;
125
126         case stop:
127         case closing:
128         case close:
129             gate->percent -= 10;
130             gate->gateCase = opening;
131             break;
132         }
133         break;
134
135     case STOP_KEY:
136         switch (gate->gateCase) {
137
138         case open:
139         case stop:
140             break;
141
142         case opening:
143         case closing:
144         case close:
145             gate->gateCase = stop;
146             break;
147         }
148         break;
149
150     case CLOSE_KEY:
151
152         switch (gate->gateCase) {
153
154         case open:
155         case opening:
156         case stop:
157             gate->percent += 10;
```

```
158             gate->gateCase = closing;
159             break;
160 
161         case closing:
162             gate->percent += 10;
163             if (gate->percent == 100) gate->gateCase = close;
164             break;
165 
166         case close:
167             break;
168         }
169         break;
170 
171     case EXIT_KEY:
172 
173         *exitStatus = true;
174         break;
175 
176     default:
177 
178         switch (gate->gateCase) {
179 
180         case open:
181         case close:
182         case stop:
183             break;
184 
185         case opening:
186 
187             gate->percent -= 10;
188             if (gate->percent == 0) gate->gateCase = open;
189             break;
190 
191         case closing:
192             gate->percent += 10;
193             if (gate->percent == 100) gate->gateCase = close;
194             break;
195         }
196         break;
197         break;
198     }
199 }
200 
201 /** Funkcja tworzy strukture odwzorowujaca stan maszyny stanow
202 @date 2020-05-15
203 @return wskaznik na strukture
204 */
205 gateMachine createGate() {
206 
207     gateMachine tempGate;
208     tempGate.gateCase = close;
209     tempGate.percent = 100;
210 
```

```
211     return tempGate;
212 }
213
214 /** Funkcja dodaje element do bufora cyklicznego
215 @date 2020-05-15
216 @param strumien wyjsciowy
217 @param maszyna stanow
218 */
219 void printState(FILE* outStream, gateMachine gate) {
220
221     switch (gate.gateCase) {
222
223     case open:
224         fprintf(outStream, "Gate status: Open\n");
225         printf("Gate status: Open\n");
226         break;
227     case opening:
228     case closing:
229         fprintf(outStream, "Gate status: Closed in %d%%\n", gate.percent);
230         printf("Gate status: Closed in %d%%\n", gate.percent);
231         break;
232     case stop:
233         fprintf(outStream, "Gate status: Closed in %d%% (Stopped)\n",
              gate.percent);
234         printf("Gate status: Closed in %d%% (Stopped)\n", gate.percent);
235         break;
236     case close:
237         fprintf(outStream, "Gate status: Closed\n");
238         printf("Gate status: Closed\n");
239         break;
240     }
241 }
242
243 int main() {
244
245     circularBuffer buffer = createCircularBuffer();
246     gateMachine gate = createGate();
247     bool exitStatus = false;
248     char input;
249     FILE* logFile = fopen("logfile.txt", "w");
250
251     while (!exitStatus) {
252
253         input = '\0';
254         printState(logFile, gate);
255
256         if (_kbhit())
257             input = getchar();
258
259         addToCircularBuffer(&buffer, input);
260         caseServis(&buffer, &gate, &exitStatus);
261         Sleep(1500);
262     }
```

```
263
264     fprintf(logFile, "INFO: Program Stopped!\n");
265     printf("INFO: Program Stopped!\n");
266     return 0;
267 }
```