```
1
2  #include "Command_Handler.h"
3  #include "nrf24.h"
4  #include <stdbool.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include <stdint.h>
8  #include <avr/io.h>
9  #include <util/delay.h>
10
11
12 const commandType availableCommand[AVAILABLE_COMMANDS] = {
13     { .commandBase = "TURN_RELAY_ON", .nParameters = 1, .handlerFunction =
       &TURN_RELAY_ON_HANDLE},
14     { .commandBase = "TURN_RELAY_OFF", .nParameters = 1, .handlerFunction =
       &TURN_RELAY_OFF_HANDLE},
15     { .commandBase = "BUILT_IN_LED_TEST", .nParameters = 0, .handlerFunction =
       &BUILT_IN_LED_TEST_HANDLER},
16     { .commandBase = "TURN_EVERYTHING_ON", .nParameters = 0, .handlerFunction =
       &TURN_EVERYTHING_ON_HANDLER},
17     { .commandBase = "TURN_EVERYTHING_OFF", .nParameters = 0, .handlerFunction =
       &TURN_EVERYTHING_OFF_HANDLER},
18     { .commandBase = "CALL_NURSE", .nParameters = 0, .handlerFunction =
       &CALL_NURSE_HANDLE}
19 };
20
21 bool initliazeMemory(){
22     if(memoryInitialized) return false;
23     parameter[0] = (void*)calloc(28,1);
24     parameter[1] = (void*)calloc(28,1);
25     parameter[2] = (void*)calloc(28,1);
26     command_buffer = (uint8_t*)calloc(32,1);
27     if(parameter[0]==nullptr||parameter[1]==nullptr||parameter[2]==nullptr||
       command_buffer==nullptr) return false;
28     memoryInitialized = true;
29     return true;
30 }
31
32
33 void composeCommand(void* output_buffer, commandType* commandT, void**
     inputParameter){
34     strcpy(output_buffer, commandT->commandBase);
35     char* startParamPTR = (char*)(output_buffer+strlen(commandT->commandBase));
36     char* endParamPTR = (char*)(startParamPTR+1+strlen(*inputParameter));
37
38     for (uint8_t index = 0; index < commandT->nParameters; index++){
39         *startParamPTR='[';
40         strcpy(startParamPTR+1, *inputParameter);
41         *endParamPTR=']';
42         startParamPTR=(endParamPTR+1);
43         if (index!=(commandT->nParameters-1)){
44             inputParameter++;
```

```
45              uint8_t len = strlen(*inputParameter);
46              endParamPTR = (char*)(startParamPTR+len+1);
47          }
48      }
49      *startParamPTR='\0';
50  }
51
52  bool decomposeCommand(void* input_buffer, commandType* commandT, void**
      outputParameter){
53
54      for (uint8_t index = 0; index < AVAILABLE_COMMANDS; index++){
55          if (memmem(input_buffer, COMMAND_BUFFER_SIZE, availableCommand
              [index].commandBase, strlen(availableCommand[index].commandBase))!
              =nullptr)
56          {
57              *commandT = availableCommand[index]; break;
58          }
59          else if (index==(AVAILABLE_COMMANDS-1)) { return false;}
60      }
61
62      for (uint8_t x = 0; x < commandT->nParameters; x++){
63          uint8_t* startNumPTR = memchr(input_buffer, '[', COMMAND_BUFFER_SIZE);
64          uint8_t* endNumPTR = memchr(input_buffer, ']', COMMAND_BUFFER_SIZE);
65          if (startNumPTR==nullptr||endNumPTR==nullptr) { if(x==0) return false;
              break; }
66          (*startNumPTR) = 0x20;
67          (*endNumPTR) = 0x20;
68          startNumPTR++;
69          uint32_t bytes = ((endNumPTR)) - ((startNumPTR));
70          if (bytes>PARAMETER_BUFFER_SIZE) return false;
71          memcpy(outputParameter[x], startNumPTR, bytes);
72      }
73
74      return true;
75  }
76
77  void TURN_RELAY_ON_HANDLE() {
78      uint8_t relayIndex =  atoi(parameter[0]);
79      switch (relayIndex) {
80          case 0:
81          bit_set(PORTD, BIT(3));
82          break;
83          case 1:
84          bit_set(PORTD, BIT(2));
85          break;
86          case 2:
87          bit_set(PORTD, BIT(6));
88          break;
89          case 3:
90          bit_set(PORTD, BIT(5));
91          break;
92      }
```

```
 93  }
 94
 95  void TURN_RELAY_OFF_HANDLE() {
 96      uint8_t relayIndex =  atoi(parameter[0]);
 97      switch (relayIndex) {
 98          case 0:
 99          bit_clear(PORTD, BIT(3));
100          break;
101          case 1:
102          bit_clear(PORTD, BIT(2));
103          break;
104          case 2:
105          bit_clear(PORTD, BIT(6));
106          break;
107          case 3:
108          bit_clear(PORTD, BIT(5));
109          break;
110      }
111  }
112
113  void BUILT_IN_LED_TEST_HANDLER(){
114      for (uint8_t x = 0; x < 8; x++) {
115          bit_flip(PORTD, BIT(7));
116          bit_flip(PORTB, BIT(0));
117          _delay_ms(250);
118      }
119      bit_clear(PORTD, BIT(7));
120      bit_clear(PORTB, BIT(0));
121  }
122
123  void TURN_EVERYTHING_ON_HANDLER(){
124      bit_set(PORTD, BIT(3));
125      bit_set(PORTD, BIT(2));
126      bit_set(PORTD, BIT(6));
127      bit_set(PORTD, BIT(5));
128  }
129
130  void TURN_EVERYTHING_OFF_HANDLER(){
131      bit_clear(PORTD, BIT(3));
132      bit_clear(PORTD, BIT(2));
133      bit_clear(PORTD, BIT(6));
134      bit_clear(PORTD, BIT(5));
135  }
136
137  void CALL_NURSE_HANDLE(){
138      bit_set(PORTD, BIT(5));
139      _delay_ms(500);
140      bit_clear(PORTD, BIT(5));
141      _delay_ms(500);
142      bit_set(PORTD, BIT(5));
143      _delay_ms(500);
144      bit_clear(PORTD, BIT(5));
```

```
145     _delay_ms(500);
146     bit_set(PORTD, BIT(5));
147     _delay_ms(500);
148     bit_clear(PORTD, BIT(5));
149 }
150
```