Assignment 4: Finite-State-Machine Generator

Administrivia

This is a substantial assignment (instructor's rough solution is over 200 lines, dashed off on a Sunday afternoon when no-one showed up for office hours). It will be worth 20% of your final grade.

Introduction

There are many systems in which behavior may be modeled using a finite-state machine (FSM). Essentially, a finite-state machine has a finite set of states and a finite set of events. The machine starts in some initial state. When an event occurs, the machine switches into a new (or possibly the current) state, strictly determined by the current state and the event.
The state machine can be viewed as a graph with states as vertices and events as edges.
To make this model useful, you may execute code at a vertex or at an edge.
There are several good ways to implement a state machine. One way is to use enum values for states and events and code the machine as a giant switch statement like this:

```
while (!done) {
  switch(state) {
  case S0:
    //code for state S0
    event = get_next_event_from_somewhere();
    switch (event) {
      case E0:
        // code for (S0,E0) -> S3
        state = S3;
        break;
      case E1:
        //...
        state = ...
        break;
      //...
    }
    break;
  case S1:
    //...
    event = get_next_event_from_somewhere();
    switch(event) {
      case E0:
```

```
        state = ...
      }
      break;
    ...
    }
  }
```

Clearly, implementing a finite-state machine can be tedious with lots of boilerplate code. Chances of an error increase dramatically as the number of states and events increases.

Generator Program

A generator program takes a text description and produces a program as output.[1]
For this assignment, your script will take as input[2] a description of a state machine and generate as output[3] C++ source file containing a giant switch statement. Like this:

```
C++ code at the beginning of the file (i.e. #include etc)
%machine machine_name
  %state state_name
    //code for this state
    %event event_name1 next_state1
    //code for this event
    %event event_name2 next_state2
    ...
  ...
%end_machine
code at end of file
```

```
// C++ code at the beginning of the file (i.e. #include etc)
//Additional declarations for the machine generation
enum State {
  state1_STATE,
  state2_STATE,
  //...
};


enum Event {
  INVALID_EVENT,
  event1_EVENT,
  event2_EVENT,
  ...
};
```

```
Event GetNextEvent();


Event string_to_event(string event_string) {
  // code to return event enum
};


int machine_name(State initial_state) {
  Event event;
  State state = initial_state;
  while (true) {
    switch(state) {
    case state1_STATE:
      // code for state1
      event = GetNextEvent();
      switch(event) {
      case event1_name_EVENT:
        // code for edge (state1, event1)
        state = next_state_STATE;
        //code for this event
        break;
      case event_name2_EVENT:
        state = next_state2_STATE;
        break;
      //...
      default:
        cerr << "invalid event in state state1: " << event << endl;
        return -1;
      }
      break;
    }
    case //...
  }
}


code at end of file
```

Example 1: Hours of Service (Trucking)

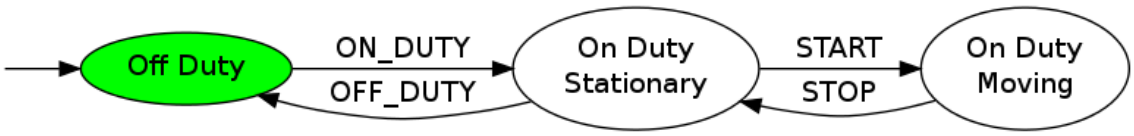
To comply with the Hours of Service (HOS) regulations, an Electronic On-Board Recorder (EOBR)[4] emit a stream of event records as the drivers comes on/off duty and while the truck starts/stops moving.

1. off-duty
2. on-duty and stationary
3. on-duty and moving

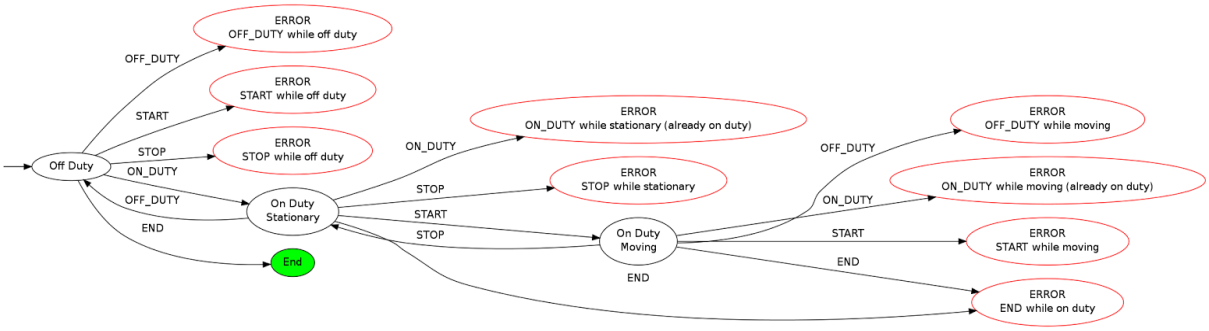
An event causes a change of state.[5] For HOS rules, there are 4 possible events:

1. off-duty
2. on-duty
3. start
4. stop

The simplified state machine for the EOBR looks like this:



Certain events should not occur in given states. For example, the driver cannot go off-duty while the truck is still moving. A sad fact of life is that data can get corrupted. Here is an expanded state machine with error states added:



Machine decription file (**.mdf**) and generated source code:

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <string>
4  #include <time.h>
5
6  using namespace std;
7
8  char * now() {
9    time_t rawtime;
10   time(&rawtime);
11   return ctime (&rawtime);
12 }
13
14 %machine HOS
```

```
15
16 %state OFF_DUTY
17   // code for OFF_DUTY_STATE
18 %event END END
19   cout << "th-th-that's all, folks." << endl;
20   exit(EXIT_SUCCESS);
21 %event ON_DUTY ON_DUTY_STATIONARY
22   cout << "driver coming on duty " << now();
23
24 %state ON_DUTY_STATIONARY
25   cout << "driver is stationary" << endl;
26 %event OFF_DUTY OFF_DUTY
27   cout << "driver coming off duty " << now();
28 %event START ON_DUTY_MOVING
29
30 %state ON_DUTY_MOVING
31   cout << "driver is moving" << endl;
32 %event STOP ON_DUTY_STATIONARY
33
34 %end_machine
35
36
37 Event GetNextEvent() {
38    string event_string;
39    cin >> event_string;
40    return string_to_event(event_string);
41 }
42
43
44 int main() {
45     int result = HOS(OFF_DUTY_STATE);
46     return result >= 0 ? EXIT_SUCCESS : EXIT_FAILURE;
47 }
```

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <string>
4  #include <time.h>
5
6  using namespace std;
7
8  char * now() {
```

```
 9    time_t rawtime;
10    time(&rawtime);
11    return ctime (&rawtime);
12 }
13
14
15
16 #include <iostream>
17 using namespace std;
18
19
20 enum State {
21     END_STATE,
22     OFF_DUTY_STATE,
23     ON_DUTY_MOVING_STATE,
24     ON_DUTY_STATIONARY_STATE,
25 };
26
27
28 enum Event {
29     INVALID_EVENT,
30     END_EVENT,
31     OFF_DUTY_EVENT,
32     ON_DUTY_EVENT,
33     START_EVENT,
34     STOP_EVENT,
35 };
36
37
38 Event GetNextEvent();
39
40
41 Event string_to_event(string event_string) {
42     if (event_string == "END") {
43         return END_EVENT;
44     }
45     if (event_string == "OFF_DUTY") {
46         return OFF_DUTY_EVENT;
47     }
48     if (event_string == "ON_DUTY") {
49         return ON_DUTY_EVENT;
50     }
51     if (event_string == "START") {
52         return START_EVENT;
```

```
53    }
54    if (event_string == "STOP") {
55       return STOP_EVENT;
56    }
57    return INVALID_EVENT;
58 }
59
60
61 int HOS(State initial_state) {
62
63
64    State state = initial_state;
65    Event event;
66    while (true) {
67       switch(state) {
68       case END_STATE:
69          event = GetNextEvent();
70          switch(event) {
71          default:
72             cerr << "invalid event in state END: " << event << endl;
73             return -1;
74           }
75          break;
76       case OFF_DUTY_STATE:
77    // code for OFF_DUTY_STATE
78          event = GetNextEvent();
79          switch(event) {
80          case END_EVENT:
81    cout << "th-th-that's all, folks." << endl;
82    exit(EXIT_SUCCESS);
83             state = END_STATE;
84             break;
85          case ON_DUTY_EVENT:
86    cout << "driver coming on duty " << now();
87
88             state = ON_DUTY_STATIONARY_STATE;
89             break;
90          default:
91             cerr << "invalid event in state OFF_DUTY: " << event << endl;
92             return -1;
93           }
94          break;
95       case ON_DUTY_MOVING_STATE:
96    cout << "driver is moving" << endl;
```

```
 97          event = GetNextEvent();
 98          switch(event) {
 99          case STOP_EVENT:
100
101             state = ON_DUTY_STATIONARY_STATE;
102             break;
103          default:
104             cerr << "invalid event in state ON_DUTY_MOVING: " << event <<
endl;
105             return -1;
106           }
107          break;
108       case ON_DUTY_STATIONARY_STATE:
109  cout << "driver is stationary" << endl;
110          event = GetNextEvent();
111          switch(event) {
112          case OFF_DUTY_EVENT:
113  cout << "driver coming off duty " << now();
114             state = OFF_DUTY_STATE;
115             break;
116          case START_EVENT:
117
118             state = ON_DUTY_MOVING_STATE;
119             break;
120          default:
121             cerr << "invalid event in state ON_DUTY_STATIONARY: " << event
<< endl;
122             return -1;
123           }
124          break;
125       default:
126          cerr << "INVALID STATE " << state << endl;
127          return -1;
128       }
129    }
130}
131
132
133
134
135Event GetNextEvent() {
136   string event_string;
137   cin >> event_string;
138   return string_to_event(event_string);
```

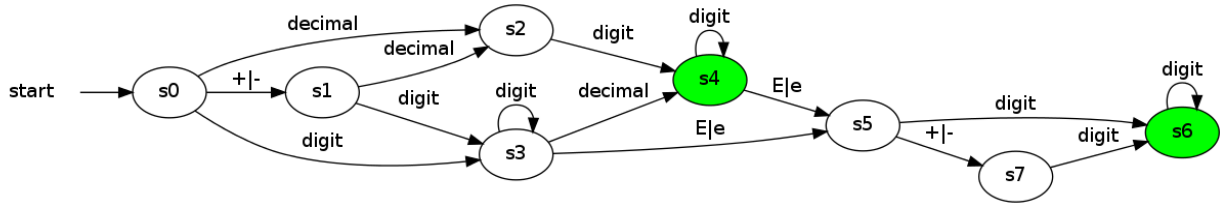
```
139}
140
141
142int main() {
143    int result = HOS(OFF_DUTY_STATE);
144    return result >= 0 ? EXIT_SUCCESS : EXIT_FAILURE;
145}
```

Example 2: Floating Point Number Validation

A floating-point number can be matched by a regular expression such as this one (whitespace ignored using **/x** flag):

```
(
  (
    ([0-9]+\.[0-9]*)
  |
    (\.[0-9]+)
  )
  (
    [Ee][-+]?[0-9]+
  )?
)
|
(
  [0-9]+[Ee][-+]?[0-9]+
)
```

Regular expressions are compiled into finite-state machines. You can also implement your own finite-state machine directly:



We can implement the the diagram with this description file:

```
1  #include <cctype>
2  #include <cstdlib>
3  #include <iostream>
4
5
6  %machine float_check
7  %state S0
```

```
 8 %event DIGIT S3
 9 %event SIGN S1
10 %event DECIMAL S2
11
12 %state S1
13 %event DECIMAL S2
14 %event DIGIT S3
15
16 %state S2
17 %event DIGIT S4
18
19 %state S3
20 %event DIGIT S3
21 %event DECIMAL S4
22 %event EXPONENT S5
23
24 %state S4
25 %event DIGIT S4
26 %event EXPONENT S5
27 %event END_OF_INPUT ACCEPT
28
29 %state S5
30 %event DIGIT S6
31 %event SIGN S7
32
33 %state S6
34 %event DIGIT S6
35 %event END_OF_INPUT ACCEPT
36
37 %state S7
38 %event DIGIT S6
39
40 %state ACCEPT
41 cout << "Input is valid float" << endl;
42 return 1;
43
44 %end_machine
45
46
47 Event GetNextEvent() {
48   int c;
49   while (isspace(c = cin.get())) {
50   }
51   if (c < 0) {
```

```
52      return END_OF_INPUT_EVENT;
53    }
54    if (isdigit(c)) {
55      return DIGIT_EVENT;
56    }
57    if (c == '+' || c == '-') {
58      return SIGN_EVENT;
59    }
60    if (c == '.') {
61      return DECIMAL_EVENT;
62    }
63    if ((c == 'e') || (c == 'E')) {
64      return EXPONENT_EVENT;
65    }
66    return INVALID_EVENT;
67  };
68
69
70  int main() {
71    int result = float_check(S0_STATE);
72    return result > 0 ? EXIT_SUCCESS : EXIT_FAILURE;
73  }
```

```
1   #include <cctype>
2   #include <cstdlib>
3   #include <iostream>
4
5
6
7
8   #include <iostream>
9   using namespace std;
10
11
12  enum State {
13      ACCEPT_STATE,
14      S0_STATE,
15      S1_STATE,
16      S2_STATE,
17      S3_STATE,
18      S4_STATE,
19      S5_STATE,
```

```
    S6_STATE,
    S7_STATE,
};


enum Event {
    INVALID_EVENT,
    DECIMAL_EVENT,
    DIGIT_EVENT,
    END_OF_INPUT_EVENT,
    EXPONENT_EVENT,
    SIGN_EVENT,
};


Event GetNextEvent();


Event string_to_event(string event_string) {
    if (event_string == "DECIMAL") {
        return DECIMAL_EVENT;
    }
    if (event_string == "DIGIT") {
        return DIGIT_EVENT;
    }
    if (event_string == "END_OF_INPUT") {
        return END_OF_INPUT_EVENT;
    }
    if (event_string == "EXPONENT") {
        return EXPONENT_EVENT;
    }
    if (event_string == "SIGN") {
        return SIGN_EVENT;
    }
    return INVALID_EVENT;
}


int float_check(State initial_state) {

    State state = initial_state;
    Event event;
    while (true) {
        switch(state) {
```

```
64        case ACCEPT_STATE:
65 cout << "Input is valid float" << endl;
66 return 1;
67
68           event = GetNextEvent();
69           switch(event) {
70           default:
71              cerr << "invalid event in state ACCEPT: " << event << endl;
72              return -1;
73            }
74           break;
75        case S0_STATE:
76           event = GetNextEvent();
77           switch(event) {
78           case DECIMAL_EVENT:
79
80              state = S2_STATE;
81              break;
82           case DIGIT_EVENT:
83              state = S3_STATE;
84              break;
85           case SIGN_EVENT:
86              state = S1_STATE;
87              break;
88           default:
89              cerr << "invalid event in state S0: " << event << endl;
90              return -1;
91            }
92           break;
93        case S1_STATE:
94           event = GetNextEvent();
95           switch(event) {
96           case DECIMAL_EVENT:
97              state = S2_STATE;
98              break;
99           case DIGIT_EVENT:
100
101             state = S3_STATE;
102             break;
103          default:
104             cerr << "invalid event in state S1: " << event << endl;
105             return -1;
106           }
107          break;
```

```
108       case S2_STATE:
109          event = GetNextEvent();
110          switch(event) {
111          case DIGIT_EVENT:
112
113             state = S4_STATE;
114             break;
115          default:
116             cerr << "invalid event in state S2: " << event << endl;
117             return -1;
118           }
119          break;
120       case S3_STATE:
121          event = GetNextEvent();
122          switch(event) {
123          case DECIMAL_EVENT:
124             state = S4_STATE;
125             break;
126          case DIGIT_EVENT:
127             state = S3_STATE;
128             break;
129          case EXPONENT_EVENT:
130
131             state = S5_STATE;
132             break;
133          default:
134             cerr << "invalid event in state S3: " << event << endl;
135             return -1;
136           }
137          break;
138       case S4_STATE:
139          event = GetNextEvent();
140          switch(event) {
141          case DIGIT_EVENT:
142             state = S4_STATE;
143             break;
144          case END_OF_INPUT_EVENT:
145
146             state = ACCEPT_STATE;
147             break;
148          case EXPONENT_EVENT:
149             state = S5_STATE;
150             break;
151          default:
```

```
152             cerr << "invalid event in state S4: " << event << endl;
153             return -1;
154           }
155         break;
156       case S5_STATE:
157         event = GetNextEvent();
158         switch(event) {
159         case DIGIT_EVENT:
160           state = S6_STATE;
161           break;
162         case SIGN_EVENT:
163
164           state = S7_STATE;
165           break;
166         default:
167           cerr << "invalid event in state S5: " << event << endl;
168           return -1;
169         }
170         break;
171       case S6_STATE:
172         event = GetNextEvent();
173         switch(event) {
174         case DIGIT_EVENT:
175           state = S6_STATE;
176           break;
177         case END_OF_INPUT_EVENT:
178
179           state = ACCEPT_STATE;
180           break;
181         default:
182           cerr << "invalid event in state S6: " << event << endl;
183           return -1;
184         }
185         break;
186       case S7_STATE:
187         event = GetNextEvent();
188         switch(event) {
189         case DIGIT_EVENT:
190
191           state = S6_STATE;
192           break;
193         default:
194           cerr << "invalid event in state S7: " << event << endl;
195           return -1;
```

```
196           }
197           break;
198       default:
199           cerr << "INVALID STATE " << state << endl;
200           return -1;
201       }
202   }
203}
204
205
206
207
208Event GetNextEvent() {
209  int c;
210  while (isspace(c = cin.get())) {
211  }
212  if (c < 0) {
213    return END_OF_INPUT_EVENT;
214  }
215  if (isdigit(c)) {
216    return DIGIT_EVENT;
217  }
218  if (c == '+' || c == '-') {
219    return SIGN_EVENT;
220  }
221  if (c == '.') {
222    return DECIMAL_EVENT;
223  }
224  if ((c == 'e') || (c == 'E')) {
225    return EXPONENT_EVENT;
226  }
227  return INVALID_EVENT;
228};
229
230
231int main() {
232  int result = float_check(S0_STATE);
233  return result > 0 ? EXIT_SUCCESS : EXIT_FAILURE;
234}
```

Notes

The hours-of-service machine description file is 47 lines and the generated C++ file is 145 lines. If we get the generator for free, it approximately triples your productivity and reduces errors. Your humble instructor's hastily-constructed solution is a couple of hundred lines of Python. If we factor that into the cost of development, it is not much of a win.
Being able to reuse the *same* generator for both HOS *and* the floating-point recognizer give us an approximate break-even point:

generator	215
mdf files	120
subtotal	335
generated code	379
net gain	24
direct gain	249 (116%)

The intangible gain is that the **.mdf** file is more readable.[6]
Please put the states and events into alphabetical order so your assignment can be graded.

Footnotes

1 A particularly amusing generator is the quine: a program that produces itself as output
2 *Standard* input.
3 *Standard* output.
4 Yes, it *is* a thing.
5 For some state machines, the new state might be the same as the old state, so it looks like nothing happened, but it's still considered a state change.
6 YMMV.