```cpp


#include <iostream>
#include <atomic>
#include <thread>
#include <vector>
#include <fstream>
#include <sstream>
// #include <string_view>

#include "main.hpp"

/// Things to work on if we get time
/**
    - Memory barriers (for atomics and synchronization)
    - Graphics (doubt we will have time) (procrastinators unite tmr)

**/

std::atomic_bool FireKey; // This global variable starts
// and stops the Fire state
std::atomic_bool IRon; // This global variable turns on
// when the a person is walking into an elevator
std::atomic_bool Sound; // Output sound after latching

std::atomic_bool gStop; // start the simulation
std::atomic_bool gStart{ false };

Controller gControl; // the controller
Elevator gLift; // Elevator object
Memory gMem;
Clock gClk;
FloorLights gFL;

void output();
void printer();

struct Token;
// std::vector<Token> parser(std::string_view file); // not using c++ 17
std::vector<Token> parser(std::string file);
void runner(std::vector<Token>& dat);

int main(int argc, char* argv[]) {
    if (argc <= 1) exit (-1);

    using std::clog;
    using std::endl;
    using std::cout;

    FireKey = false;
    IRon = false;
    Sound = false;
    gStop = false;

    std::thread ptr{ printer };

    // The main reason I decided to do this in two steps is because disk read
    // is very slow compared to ram reads. I want to first load the entire
    // file into memory and then run that rather than reading directly from
    // the file and executing live.
    auto pdat = parser(argv[1]);
    runner(pdat);

    while (!gStop)
        ;

}

void printer() {
```

```
70      using std::cout;
71      using std::endl;
72
73      while (!gStart)
74          ;
75
76      for (int i = 0; true; ++i) {
77          cout << "At time = " << i << endl;
78          output();
79          std::this_thread::sleep_for(1s);
80      }
81  }
82
83
84  void output() {
85      using std::cout;
86      using std::endl;
87
88      cout << "Elevator Floor:  " << std::to_string(static_cast<int>(gLift.mFloor)) <<
        endl;
89      cout << "Elevator State:  " << pretty(gLift.mState) << endl;
90      cout << "IR Sensor State: " << pretty(gLift.mDoor.mIRSen.mState) << endl;
91      cout << "Door State:      " << pretty(gLift.mDoor.mState) << endl;
92      cout << "Elevator Stop:   " << gLift.mStop << endl;
93      cout << "Door Open Sig:   " << gLift.mDoor.mConDoorOpen << endl;
94      cout << "Door Close Sig:  " << gLift.mDoor.mConDoorClose << endl;
95      cout << "Fire State:      " << FireKey << endl;
96      cout << "X20 register:    " << std::to_string(gFL.getLights()) << endl;
97      cout << "Sound:           " << Sound << endl;
98
99      cout << endl;
100
101
102 }
103
104 // Ya I know I know, not proper, but I'm procrastinating, Im
105 // just going to shove this part here
106 enum class TKN {
107     HOUR,
108     MIN,
109     ERST,
110     MEM,
111     START,
112     WAIT,
113     WFLR,
114     WDOOR,
115     IRON,
116     FIRE,
117     DOORO,
118     DOORC,
119 };
120
121 struct Token {
122     TKN mToken;
123     int mData;
124 };
125
126 std::vector<Token> parser(std::string file) {
127     // The instruction file is orgaized by one
128     // opcode and a data attached to it (one int)
129     std::vector<Token> ret;
130
131     std::ifstream readFile(file.c_str());
132     std::string line; // temp store for each command
133
134     while(std::getline(readFile,line)) {
135         std::stringstream iss(line);
136
137         // split command and data
```

```
138             std::string tknstr;
139             std::string dat;
140             std::getline(iss, tknstr, ' ');
141             std::getline(iss, dat);
142 
143             Token t;
144             t.mData = stoi(dat); // convert string number
145 
146             // convert token string to token for jump table during run
147             if        (tknstr == "hour") {
148                 t.mToken = TKN::HOUR;
149             } else if (tknstr == "min") {
150                 t.mToken = TKN::MIN;
151             } else if (tknstr == "erst") {
152                 t.mToken = TKN::ERST;
153             } else if (tknstr == "mem") {
154                 t.mToken = TKN::MEM;
155             } else if (tknstr == "start") {
156                 t.mToken = TKN::START;
157             } else if (tknstr == "wait") {
158                 t.mToken = TKN::WAIT;
159             } else if (tknstr == "waitff") {
160                 t.mToken = TKN::WFLR;
161             } else if (tknstr == "waitfd") {
162                 t.mToken = TKN::WDOOR;
163             } else if (tknstr == "iron") {
164                 t.mToken = TKN::IRON;
165             } else if (tknstr == "fire") {
166                 t.mToken = TKN::FIRE;
167             } else if (tknstr == "dooro") {
168                 t.mToken = TKN::DOORO;
169             } else if (tknstr == "doorc") {
170                 t.mToken = TKN::DOORC;
171             }
172             ret.push_back(t);
173         }
174 
175         return ret;
176     }
177 
178 
179     void runner(std::vector<Token>& dat) {
180         // take the parsed tokens and run it
181         for (auto& t : dat) {
182             switch (t.mToken) { // this is internally impl by a jump table
183 
184                 // For each token and data pair, do what its supposed to
185                 case TKN::HOUR:
186                     gClk.reset(true, t.mData);
187                 break;
188 
189                 case TKN::MIN:
190                     gClk.reset(false, t.mData);
191                 break;
192 
193                 case TKN::ERST:
194                     gLift.reset(static_cast<FloorNum>(t.mData));
195                 break;
196 
197                 case TKN::MEM:
198                     gMem.setFloor(static_cast<FloorNum>(t.mData));
199                 break;
200 
201                 case TKN::START:
202                     gClk.reset();
203                     gStart = true;
204                 break;
205 
206                 case TKN::WAIT:
```

```
207                 std::this_thread::sleep_for(std::chrono::seconds{ t.mData });
208             break;
209
210             case TKN::WFLR:
211                 while (gLift.mFloor != static_cast<FloorNum>(t.mData));
212             break;
213
214             case TKN::WDOOR:
215                 while (gLift.mDoor.mState != static_cast<DoorState>(t.mData));
216             break;
217
218             case TKN::IRON:
219                 IRon = static_cast<bool>(t.mData);
220             break;
221
222             case TKN::FIRE:
223                 FireKey = static_cast<bool>(t.mData);
224             break;
225
226             case TKN::DOORO:
227                 gLift.mDoor.mConDoorOpen = true;
228             break;
229
230             case TKN::DOORC:
231                 gLift.mDoor.mConDoorClose = true;
232             break;
233         }
234     }
235 }
```