```cpp

#include "main.hpp"

Clock::Clock() {
    reset(0, 0);
}

void Clock::reset() {
    mStart = clk::now();
}

void Clock::reset(bool hour, unsigned val) {
    if (hour) mHour = val;
    else mMinute = val;
}

// https://stackoverflow.com/questions/15957805/
unsigned Clock::getHour() {
    auto elapsed = clk::now() - mStart;
    auto hour = std::chrono::duration_cast<std::chrono::hours>(elapsed).count();
    auto min = std::chrono::duration_cast<std::chrono::minutes>(elapsed).count();

    if ((min + mMinute) >= 60)
        ++hour;

    // std::cout << ".............................TIME: " << (hour + mHour) <<
    std::endl;

    return hour + mHour;
}

/// depreciated, using [[depreciated]] is giving me warning cause im
/// to lazy to turn on c++17
unsigned Clock::getMin() {
    time_t tt = clk::to_time_t(mStart);
    tm local_tm = *localtime(&tt);
    return local_tm.tm_min;
}
```

```
1
2   #include "main.hpp"
3
4   #include <thread>
5
6   Controller::Controller() {
7       mThread = std::thread{&start, this};
8   }
9
10  Controller::~Controller() {
11      mThread.join();
12  }
13
14
15  void Controller::start() {
16      while (!gStart)
17          ;
18
19      bool fire = false;
20
21      while (!gStop) {
22          if (!FireKey) fire = false;
23
24          auto floor = gMem.getFloor(); // get the next floor to go to
25
26          bool memEmpty = gMem.isEmpty();
27
28          // go in that direction
29          if (floor > gLift.mFloor)
30              gLift.mState = ES_UP;
31          if (floor < gLift.mFloor)
32              gLift.mState = ES_DOWN;
33
34          // Wait for the elevator to get to the floor unless FIRE!!
35          while (gLift.mFloor != floor) {
36              if (FireKey and !fire) {
37                  break;
38              }
39
40              if (!FireKey) fire = false;
41          }
42
43          if (FireKey and !fire) {
44              fire = true;
45              continue;
46          }
47
48          if (!FireKey) fire = false;
49
50          // set the state to wait
51          gLift.mStop = true;
52
53          // wait for the door to open
54          while (gLift.mDoor.mState == DOOR_CLOSED)
55              ;
56
57          gLift.mStop = false;
58          gLift.mState = ES_WAIT;
59
60          // because we aren't in the fire state we will close/open
61          // the door at instant
62          if (!FireKey) {
63              gLift.mDoor.mConDoorOpen = true;
64              gLift.mDoor.mConDoorClose = true;
65          }
66
67          // Wit for the door to close
68          while (gLift.mDoor.mState != DOOR_CLOSED)
69              ;
```

```
70
71          gLift.mDoor.mConDoorOpen = false;
72          gLift.mDoor.mConDoorClose = false;
73
74          // wait for the person to hit a button if the user doesnt
75          // press a button for 30 secs, we will assume there is no
76          // person in the elevator and we will reset to the default
77          // floor
78          if (memEmpty) {
79              auto end = clk::now() + 30s; // FIX THIS IT SHOULD BE 30 SECONDS
80              while (clk::now() < end) {
81                  // std::cout << "....................................." << std::endl;
82                  if (!gMem.isEmpty()) // user pushed a floor button
83                      break;
84              }
85          }
86          // the loop will continue for 30 seconds, if the user pushes abort
87          // button in that time then we break and go to that floor
88          // if the loops iterates for 30 seconds the next ask from memory
89          // will get the default floor, if the loop never iterates then
90          // the queue is not empty so we want to go to the next floor
91      }
92
93      std::cout << "Controller exiting" << std::endl;
94  }
```

```cpp

#include "main.hpp"

void Door::start() {
    // We are currently in DOOR_CLOSED state, we will change state and
    // Latch and send sound output
    mState = DOOR_LAT_SND;
    std::this_thread::sleep_for(2s); // wait for 2s to latch
    Sound = true;

    // Door is latched now we wait for the controller to tell us to
    // open the gate
    while (!mConDoorOpen) ;// { std::this_thread::yield(); }

    // Open Door (Turn off sound too)
    mState = DOOR_OPEN;
    std::this_thread::sleep_for(1s); // Door is opening

    // Door is open, now let people in and out
    mState = DOOR_IR; // Honestly this should be called Door Wait
    // but too lazy to change
    Sound = false;
    if (!FireKey) {
        mIRSen.start(); // This is a blocking function
        // this function will start the IRSensor and wait
        // for its finish to close the door
    }

    // Wait for the controller to send close door signal
    while (!mConDoorClose) ;// { std::this_thread::yield(); }
    mState = DOOR_CLOSING;
    std::this_thread::sleep_for(1s); // Door is closing

    mState = DOOR_CLOSED; // Door is closed

}
```

```cpp

#include "main.hpp"

Elevator::Elevator()
: mState{ ES_WAIT }, mStop{ false } {
    mThread = std::thread{&start, this};
}

Elevator::~Elevator() {
    mThread.join();
}

void Elevator::start() {
    while (!gStart) // Wait for all go signal
        ;

    while (!gStop) {
        if (mStop) { // Controller wants us to stop at this foor
            mDoor.start();
        }

        if (mState == ES_UP) {
            inc(mFloor); // to lazy to change it back to ++/--

            std::this_thread::sleep_for(5s);
            continue;
        }

        if (mState == ES_DOWN) {
            dec(mFloor);

            std::this_thread::sleep_for(5s);
            continue;
        }
    }

    std::cout << "Elevator exiting" << std::endl;
}

void Elevator::reset(FloorNum flr) {
    mState = ES_WAIT;
    mFloor = flr;
}
```

```cpp

#include <thread>
#include "main.hpp"

FloorLights::FloorLights()
: X20{ 0 } {
    mThread = std::thread{&start, this}; // invokes the callable
}

FloorLights::~FloorLights() {
    mThread.join();
}

void FloorLights::start() {
    while (!gStop) { // So while the global variable hasnt told us to stop
        unsigned char newValue = 0;

        /// get the state from the elevator
        ElevState eleState = gLift.mState;

        /// get current floor from the elevator
        FloorNum eleFloor = gLift.mFloor;

        switch (eleState) {
            case ES_WAIT: continue; // if we are waiting then dont shine any lights

            case ES_UP: {
                int shift = static_cast<int>(eleFloor);
                newValue |= (0x80 >> shift);
            } break;

            case ES_DOWN: {
                int shift = static_cast<int>(eleFloor);
                newValue |= (0x01 << (3 - shift));
            } break;

            default: break;
        }

        X20 = newValue;

        std::this_thread::sleep_for(5s); // SOOO
        // My testing computer has 2 threads -- hyperthreaded
        // thos thread is the least significant so Im adding this
        // in a higher core machine, remove this
    }
}

unsigned char FloorLights::getLights() {
    return X20;
}
```

```cpp

#include "ir_sensor.hpp"
#include "main.hpp"

void IR_Sensor::start() {
    /// Start the IR Sensor loop
    while (true) {
        mState = IR_BUSY;
        if (IRon) continue;

        /// THis first section just loops endlesly until the IRon sensor
        /// gives us a 'no one is here signal'

        bool startOver = false;
        auto end = clk::now() + 2s;
        while (clk::now() < end) {
            if (IRon) {
                startOver = true;
                break;
            }

            /// This here will loop for 2 seconds or until someone walks through the door
            /// if we get a person through the door in that 2sec then we want to mark
            startOver
            /// to true and start this process over again
        }

        if (startOver) continue; // This repeats the loop if a person
        // walks through if not then IR should be done and we send

        /// The return of this function will be the signal that tells
        // the Door FSM that IR Sesor is dont
        mState = IR_OFF;
        return; /// We dont need to start over, the IR sensor is done
    }
}
```

```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 }
```

```cpp

#include <mutex>

#include "main.hpp"

// deprecated cant use [[depreciated]]
void Memory::clearMem() {
    std::lock_guard<std::mutex> { mGuard };

    // https://stackoverflow.com/questions/709146
    decltype(mFloors) empty; // swap with an empty one
    std::swap(mFloors, empty);
}

bool Memory::isEmpty() {
    return mFloors.empty();
}

void Memory::setFloor(FloorNum flr) {
    if (FireKey) {
        mFireFloor = flr;
        return;
    }

    std::lock_guard<std::mutex> { mGuard }; // Protection
    mFloors.push(flr);
}

FloorNum Memory::getFloor() {
    if (FireKey) // no need for protection
        return mFireFloor;

    std::lock_guard<std::mutex> { mGuard };

    if (mFloors.empty())
        return getDefaultFloor();

    auto flr = mFloors.front();
    mFloors.pop();
    return flr;
}


FloorNum Memory::getDefaultFloor() {
    // return F1; /// FOR TESTING PURPOSES ONLY
    auto hour = gClk.getHour();

    // std::cout << "....................................." << std::to_string(hour)
    << std::endl;

    if (hour > 8 and hour < 14)
        return F1;

    if (hour >= 14 and hour < 18)
        return F2;

    return F1;
}
```