## Code & Program

### Program flow chart

A screenshot of a cell phone

Description automatically generated

Figure mode select

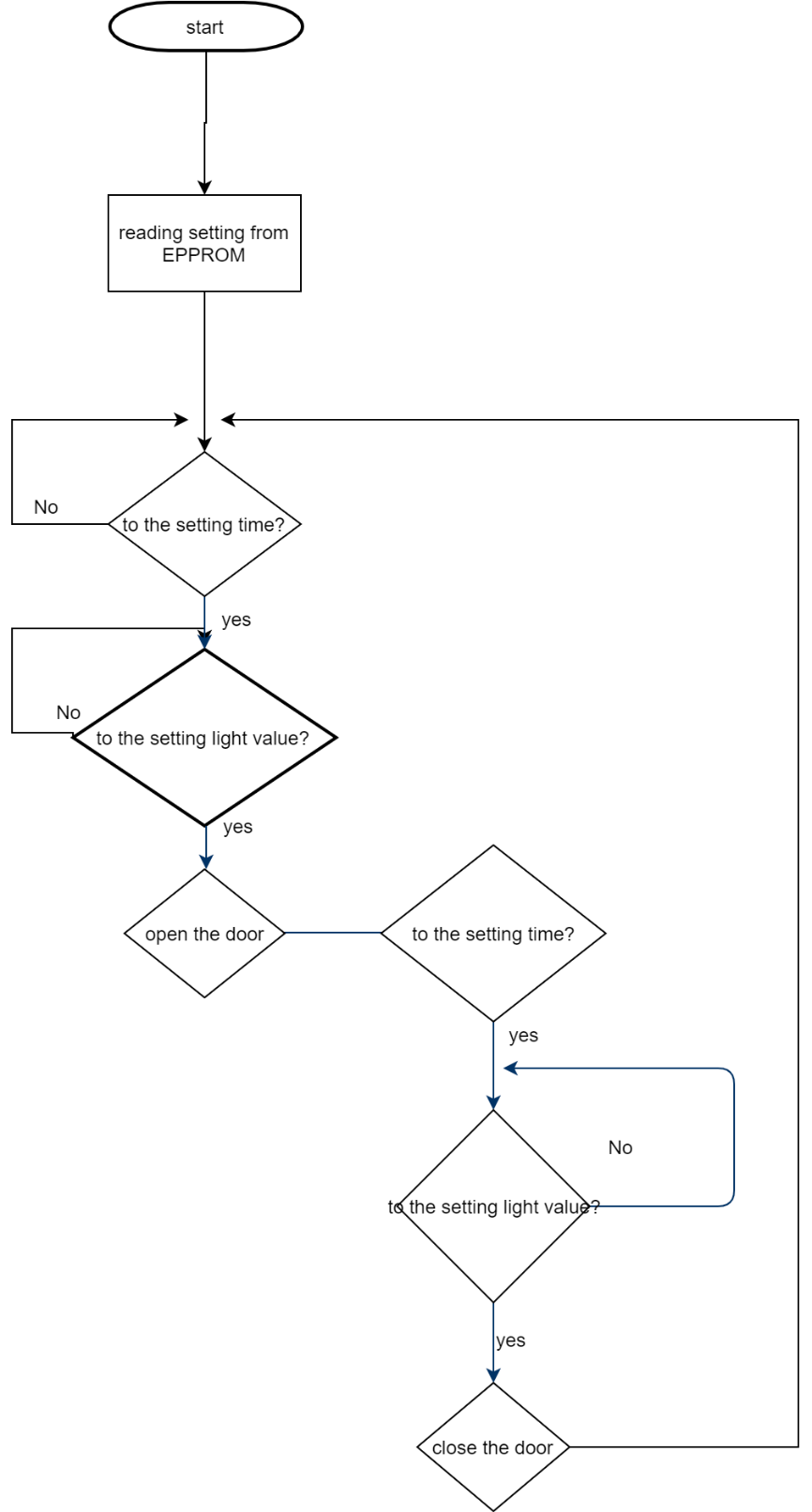


Figure operating for combine mode

Same principle for timer mode and light sensor mode, but the timer mode does not check the light and the light sensor mode does not care the time.

A close up of a logo

Description automatically generated

Figure motor operation

### Code

/\*

\* \* Project: DC door controller

\*\* Author: xKang

\*\* Date: 15/11/2019

\*/

#include <**LiquidCrystal**.h>

#include <**L298N**.h>

#include <AnalogKeypad.h>

#include <Wire.h>

#include <TimeLib.h>

#include <Eeprom24C32\_64.h>

#include <DS1307RTC.h>

#define INTERPUT\_PIN 2 //over current interput

#define LDR\_PIN A1

#define REED\_TOP\_PIN 13

#define REED\_BOTTOM\_PIN A3

#define HES\_PIN 3

#define BUTTON\_PIN A0

#define EEPROM\_ADDRESS 0x50

static Eeprom24C32\_64 \_eeprom(EEPROM\_ADDRESS);

AnalogKeypad \_kp(A0);

//L298N \_motor(EN, IN1, IN2);

**L298N** \_motor(11, 12, 10);

//LiquidCrystal \_lcd(rs, en, d4, d5, d6, d7);

**LiquidCrystal** \_lcd(8, 9, 4, 5, 6, 7);

mode \_mode = combine;

settingStore \_conf;

tmElements\_t tm;

int \_turns = 0;

enum mode{

 lightSensor,

 timmer,

 combine,

 programming

};

static inline String modeToString(enum mode m)

{

   String str[] = { "lightSensor", "timmer", "combine", "programming"};

   return str[m];

}

class settingStore {

 public:

   settingStore();

   mode \_defaultMode = combine;

   int \_max\_HES\_turns;

   int \_lightOn;

   int \_lightOff;

   int \_timeOn;

   int \_timeOff;

   void write(mode defaultMode, int max\_HES\_turns,int lightOn,int lightOff,int timeOn,int timeOff,int baseAddress = 0);

   void read(int baseAddress = 0);

};

void setup() {

**Serial**.begin(9600);

 \_lcd.begin(16, 2);

 if(\_kp.getKey() == AnalogKeypad::select){

  \_mode = programming;

 }else{

   runningSetup();

 }

}

void loop() {

 switch(\_mode){

  case programming:{

     programmingMode();

  break;

  }

  case lightSensor:{

  lightOpenFun();

  openingFun();

  lightCloseFun();

  closingFun();

  break;

  }

  case timmer:{

  timeOpenFun();

  openingFun();

  timeCloseFun();

  closingFun();

  break;

  }

  case combine:

  default:{

  timeOpenFun();

  lightOpenFun();

  openingFun();

  timeCloseFun();

  lightCloseFun();

  closingFun();

  break;

  }

 }

}

/\*

\*\* other functions

\*/

void setText(String text, int row,int col=0){

 if (row == 0) {

   \_lcd.clear();

 }

 \_lcd.setCursor(col, row);

 \_lcd.print(text);

}

/\*

\*\* statuses

\*/

void timeOpenFun() {

 setText("waiting timer", 0);

 setText(("/"+String(\_conf.\_timeOn)), 1, 14);

 while (true) {

   if (RTC.read(tm)) {

     if (tm.Hour > \_conf.\_timeOn) {

       return;

     }

   }

   if (\_kp.getKey() == AnalogKeypad::up) {

   return;

   }

   setText(String(tm.Hour,DEC),1);

   delay(500);

 }

}

void lightOpenFun() {

 setText("waiting light", 0);

 setText("/"+String(\_conf.\_lightOn), 1, 16-4);

 int luxReading;

 while (true) {

   luxReading = analogRead(LDR\_PIN);

   if (luxReading > \_conf.\_lightOn){

     return;

   }

   if (\_kp.getKey() == AnalogKeypad::up) {

     return;

   }

   setText(String(luxReading,DEC), 1);

   delay(500);

 }

}

void openingFun() {

 setText("opening", 0);

 setText("/"+String(\_conf.\_max\_HES\_turns), 1, 16-3);

 \_turns = 0;

 \_motor.setSpeed(225);

 \_motor.forward();

 while (digitalRead(REED\_TOP\_PIN) == HIGH && \_turns < \_conf.\_max\_HES\_turns) {

   setText(String(\_turns,DEC), 1);

   delay(500);

  }

 \_motor.fastStop();

 delay(500);

 \_motor.disable();

}

void timeCloseFun() {

 setText("waiting light", 0);

 setText("/"+String(\_conf.\_lightOff), 1, 16-4);

 int luxReading;

 while (true) {

   luxReading = analogRead(LDR\_PIN);

   if (luxReading < \_conf.\_lightOff){

     return;

   }

   if (\_kp.getKey() == AnalogKeypad::down) {

     return;

   }

   setText(String(luxReading,DEC), 1);

   delay(500);

 }

}

void lightCloseFun() {

 setText("waiting light", 0);

 setText("/"+String(\_conf.\_lightOff), 1, 16-4);

 int luxReading;

 while (true) {

   luxReading = analogRead(LDR\_PIN);

   if (luxReading < \_conf.\_lightOff){

     return;

   }

   if (\_kp.getKey() == AnalogKeypad::down) {

     return;

   }

   setText(String(luxReading,DEC), 1);

   delay(500);

 }

}

void closingFun() {

 setText("closing", 0);

 setText("/"+String(\_conf.\_max\_HES\_turns), 1, 16-3);

 \_turns = 0;

 \_motor.setSpeed(225);

 \_motor.backward();

 while (digitalRead(REED\_BOTTOM\_PIN) == HIGH && \_turns < \_conf.\_max\_HES\_turns) {

   setText(String(\_turns,DEC), 1);

   delay(500);

  }

 \_motor.fastStop();

 delay(500);

 \_motor.disable();

}

/\*

\* initialize function

\*/

void runningSetup(){

 pinMode(INTERPUT\_PIN, INPUT);

 //attachInterrupt(digitalPinToInterrupt(INTERPUT\_PIN), overHeatAction, LOW);

 pinMode(HES\_PIN, INPUT);

 attachInterrupt(digitalPinToInterrupt(HES\_PIN), HESAction, RISING);

 pinMode(LDR\_PIN, INPUT);

 pinMode(REED\_TOP\_PIN, INPUT);

 pinMode(REED\_BOTTOM\_PIN, INPUT);

 \_conf.read();

}

void programmingMode(){

mode defaultMode = combine;

 int max\_HES\_turns = 1,lightOn = 150,lightOff = 150,timeOn = 8, timeOff = 16;

 setText("programming mode", 0);

 setText("", 1);

 delay(1000);

//section

 setText("default mode", 0);

 while(\_kp.getKey() != AnalogKeypad::select){

   setText(modeToString(defaultMode) + " mode       ", 1);

   if(\_kp.getKey() == AnalogKeypad::up && defaultMode != 0){

     defaultMode = defaultMode - 1;

   }

   if(\_kp.getKey() == AnalogKeypad::down && defaultMode != 2){

     defaultMode = defaultMode + 1;

   }

   delay(500);

 }

 while(\_kp.getKey() == AnalogKeypad::select){}

 //section

 setText("door hight", 0);

 while(\_kp.getKey() != AnalogKeypad::select){

   setText(String(max\_HES\_turns) + "     ", 1);

   if(\_kp.getKey() == AnalogKeypad::up && max\_HES\_turns != 99){

     max\_HES\_turns += 1;

   }

   if(\_kp.getKey() == AnalogKeypad::down && max\_HES\_turns != 1){

     max\_HES\_turns -= 1;

   }

   delay(500);

 }

 while(\_kp.getKey() == AnalogKeypad::select){}

 //section

 setText("light to open", 0);

 while(\_kp.getKey() != AnalogKeypad::select){

   setText(String(lightOn) + "     ", 1);

   if(\_kp.getKey() == AnalogKeypad::up && lightOn != 1000){

     lightOn += 25;

   }

   if(\_kp.getKey() == AnalogKeypad::down && lightOn != 0){

     lightOn -= 25;

   }

   delay(500);

 }

 while(\_kp.getKey() == AnalogKeypad::select){}

 //section

 setText("light to close", 0);

 while(\_kp.getKey() != AnalogKeypad::select){

   setText(String(lightOff) + "     ", 1);

   if(\_kp.getKey() == AnalogKeypad::up && lightOff != 1000){

     lightOff += 25;

   }

   if(\_kp.getKey() == AnalogKeypad::down && lightOff != 0){

     lightOff -= 25;

   }

   delay(500);

 }

 while(\_kp.getKey() == AnalogKeypad::select){}

 //section

 setText("time to open", 0);

 setText("        (24 hrs)", 1);

 while(\_kp.getKey() != AnalogKeypad::select){

   setText(String(timeOn) + ":00  ", 1);

   if(\_kp.getKey() == AnalogKeypad::up && timeOn != 9){

     timeOn += 1;

   }

   if(\_kp.getKey() == AnalogKeypad::down && timeOn != 0){

     timeOn -= 1;

   }

   delay(500);

 }

 while(\_kp.getKey() == AnalogKeypad::select){}

 //section

 setText("time to close", 0);

 setText("        (24 hrs)", 1);

 while(\_kp.getKey() != AnalogKeypad::select){

   setText(String(timeOff) + ":00", 1);

   if(\_kp.getKey() == AnalogKeypad::up && timeOff != 23){

     timeOff += 1;

   }

   if(\_kp.getKey() == AnalogKeypad::down && timeOff != 13){

     timeOff -= 1;

   }

   delay(500);

 }

 while(\_kp.getKey() == AnalogKeypad::select){}

 \_conf.write(defaultMode,max\_HES\_turns,lightOn,lightOff,timeOn,timeOff,0);

 setText("setting finish", 0);

 setText("please reboot", 1);

 while(true){delay(10000);}

}

/\*

\* for interput

\*/

void overHeatAction() {

 \_motor.disable();

 setText("over heat", 0);

 setText("pls clear jam and reset", 1);

 //waiting for reset

 while(true){

   delay(10000);

 }

}

void HESAction(){

**Serial**.println("!!");

 \_turns++;

}

/\*

\* class settingStore

\* I really should put it to another file

\*/

settingStore::settingStore(){

 \_eeprom.initialize();

}

void settingStore::read(int baseAddress = 0){

 int count = 6;

 char buffer[count] = { 0 };

 \_eeprom.readBytes(baseAddress, count, buffer);

 this->\_defaultMode = (mode)buffer[0];

 this->\_max\_HES\_turns = buffer[1];

 this->\_lightOn = buffer[2] \* 25;

 this->\_lightOff = buffer[3] \* 25;

 this->\_timeOn = buffer[4];

 this->\_timeOff = buffer[5];

}

void settingStore::write(

 mode defaultMode,

 int max\_HES\_turns,

 int lightOn,

 int lightOff,

 int timeOn,

 int timeOff,

 int baseAddress = 0

){

 int count = 6;

 char buffer[count]  = { 0 };

 buffer[0] = defaultMode;

 buffer[1] = max\_HES\_turns;

 buffer[2] = lightOn / 25;

 buffer[3] = lightOff / 25;

 buffer[4] = timeOn;

 buffer[5] = timeOff;

 \_eeprom.writeBytes(baseAddress, count, buffer);

}