#define CHANGE\_HEAP\_SIZE(size) \_\_asm\_\_ volatile ("\t.globl \_min\_heap\_size\n\t.equ \_min\_heap\_size, " #size "\n")

#include <mrbconf.h>

#include <mruby.h>

#include <mruby/irep.h>

#include <mruby/string.h>

#include <mruby/value.h>

extern const uint8\_t code[];

//init mruby

mrb\_state \*mrb = mrb\_open();

//print function

mrb\_value cputs(mrb\_state \*mrb, mrb\_value self){

mrb\_value val;

mrb\_get\_args(mrb, "S", &val);

Serial.println(RSTRING\_PTR(val));

return mrb\_nil\_value();

}

//original print function

mrb\_value cprint(mrb\_state \*mrb, mrb\_value self){

mrb\_int val;

mrb\_get\_args(mrb, "i", &val);

Serial.println(val);

return mrb\_nil\_value();

}

//digitalWrite function

mrb\_value cdigitalWrite(mrb\_state \*mrb, mrb\_value self){

mrb\_int pin;

mrb\_int val;

mrb\_get\_args(mrb, "ii", &pin, &val);

digitalWrite(pin, val);

return mrb\_nil\_value();

}

//delay function

mrb\_value cdelay(mrb\_state \*mrb, mrb\_value self){

mrb\_int val;

mrb\_get\_args(mrb,"i", &val);

delay(val);

return mrb\_nil\_value();

}

//digitalRead function

mrb\_value cdigitalRead(mrb\_state \*mrb, mrb\_value self){

mrb\_int pin;

int n = mrb\_get\_args(mrb, "i", &pin);

int val = digitalRead(pin);

return mrb\_fixnum\_value(val);

}

//analogWrite function

mrb\_value canalogWrite(mrb\_state \*mrb, mrb\_value self){

mrb\_int pin, val;

int n = mrb\_get\_args(mrb, "ii", &pin, &val);

analogWrite(pin, val);

return mrb\_nil\_value();

}

//analogRead function

mrb\_value canalogRead(mrb\_state \*mrb, mrb\_value self){

mrb\_int pin;

int n = mrb\_get\_args(mrb, "i", &pin);

int val = analogRead(pin);

return mrb\_fixnum\_value(val);

}

//startsetup function

mrb\_value start(mrb\_state \*mrb, mrb\_value self){

Serial.print("start");

while(1){

analogWrite(10,HIGH);

analogWrite(9,HIGH);

analogWrite(6,HIGH);

analogWrite(5,HIGH);

int val1 = analogRead(1);

int val2 = analogRead(2);

int val3 = analogRead(3);

int black = 50;

Serial.println(val1);

Serial.println(val2);

Serial.println(val3);

if(val1<black && val2>black &&val3<black){

break;

}

delay(50);

}

return mrb\_nil\_value();

}

//trace function

mrb\_value trace(mrb\_state \*mrb, mrb\_value self){

Serial.print("trace");

while(1){

int val1 = analogRead(1);

int val2 = analogRead(2);

int val3 = analogRead(3);

int black = 50;

Serial.println(val1);

Serial.println(val2);

Serial.println(val3);

digitalWrite(13,HIGH);

digitalWrite(7,HIGH);

analogWrite(10,250);

analogWrite(9,150);

analogWrite(6,250);

analogWrite(5,150);

if(val1>black){

analogWrite(10,HIGH);

analogWrite(9,HIGH);

}

if(val3>black){

analogWrite(6,HIGH);

analogWrite(5,HIGH);

}

if(val1>black && val2>black && val3>black){

break;

}

delay(50);

// if(val1<black && val2<black && val3<black){

// delay(1000);

// val1 = analogRead(1);

// val2 = analogRead(2);

// val3 = analogRead(3);

// if(val1>black || val2>black || val3>black){

// break;

// }

// if(val1<black && val2<black && val3<black){

// analogWrite(10,150);

// analogWrite(9,250);

// analogWrite(6,150);

// analogWrite(5,250);

// delay(2000);

// }

// }

}

return mrb\_nil\_value();

}

//forward function

mrb\_value forward(mrb\_state \*mrb, mrb\_value self){

Serial.print("forward");

analogWrite(10,250);

analogWrite(9,150);

analogWrite(6,250);

analogWrite(5,150);

delay(2000);

return mrb\_nil\_value();

}

//left function

mrb\_value left(mrb\_state \*mrb, mrb\_value self){

Serial.print("left");

analogWrite(6,250);

analogWrite(5,150);

digitalWrite(9,HIGH);

digitalWrite(10,HIGH);

delay(1000);

int val2 = analogRead(2);

int black = 50;

while(1){

analogWrite(6,250);

analogWrite(5,150);

digitalWrite(9,HIGH);

digitalWrite(10,HIGH);

int val2 = analogRead(2);

if(val2 > black){

break;

}

delay(50);

}

return mrb\_nil\_value();

}

//right function

mrb\_value right(mrb\_state \*mrb, mrb\_value self){

Serial.print("right");

analogWrite(10,250);

analogWrite(9,150);

digitalWrite(6,HIGH);

digitalWrite(5,HIGH);

delay(1000);

int val2 = analogRead(2);

int black = 50;

while(1){

analogWrite(10,250);

analogWrite(9,150);

digitalWrite(6,HIGH);

digitalWrite(5,HIGH);

int val2 = analogRead(2);

if(val2 > black){

break;

}

delay(50);

}

return mrb\_nil\_value();

}

//back function

mrb\_value back(mrb\_state \*mrb, mrb\_value self){

Serial.print("back");

analogWrite(10,150);

analogWrite(9,250);

analogWrite(6,150);

analogWrite(5,250);

delay(2000);

return mrb\_nil\_value();

}

//wait function

mrb\_value wait(mrb\_state \*mrb, mrb\_value self){

Serial.print("wait");

analogWrite(10,HIGH);

analogWrite(9,HIGH);

analogWrite(6,HIGH);

analogWrite(5,HIGH);

delay(2000);

return mrb\_nil\_value();

}

void setup(){

CHANGE\_HEAP\_SIZE(102400); //100kb

Serial.begin(9600);

pinMode(1, INPUT); //set pin7 as OUTPUT

pinMode(2, INPUT);

pinMode(3, INPUT);

pinMode(5, OUTPUT);

pinMode(6, OUTPUT);

pinMode(7, OUTPUT);

pinMode(9, OUTPUT);

pinMode(10, OUTPUT);

pinMode(13, OUTPUT);

//register c functions

mrb\_define\_method(mrb, mrb->object\_class, "cputs", cputs, ARGS\_REQ(1));

mrb\_define\_method(mrb, mrb->object\_class, "cprint", cprint, ARGS\_REQ(1));

mrb\_define\_method(mrb, mrb->object\_class, "cdigitalWrite", cdigitalWrite, ARGS\_REQ(2));

mrb\_define\_method(mrb, mrb->object\_class, "cdelay", cdelay, ARGS\_REQ(1));

mrb\_define\_method(mrb, mrb->object\_class, "cdigitalRead", cdigitalRead, ARGS\_REQ(1));

mrb\_define\_method(mrb, mrb->object\_class, "canalogWrite", canalogWrite, ARGS\_REQ(2));

mrb\_define\_method(mrb, mrb->object\_class, "canalogRead", canalogRead, ARGS\_REQ(1));

mrb\_define\_method(mrb, mrb->object\_class, "start", start, ARGS\_REQ(0));

mrb\_define\_method(mrb, mrb->object\_class, "trace", trace, ARGS\_REQ(0));

mrb\_define\_method(mrb, mrb->object\_class, "forward", forward, ARGS\_REQ(0));

mrb\_define\_method(mrb, mrb->object\_class, "left", left, ARGS\_REQ(0));

mrb\_define\_method(mrb, mrb->object\_class, "right", right, ARGS\_REQ(0));

mrb\_define\_method(mrb, mrb->object\_class, "back", back, ARGS\_REQ(0));

mrb\_define\_method(mrb, mrb->object\_class, "wait", wait, ARGS\_REQ(0));

}

void loop(){

Serial.println("loop() called");

//load bytecode and execute

mrb\_load\_irep( mrb, code);

}