```
/*

A Parallax robot that follows a black line in the shape of a maze.  It solves the maze in two stages.
On a first pass, it explores the maze following the left-hand rule, eventually reaching the end point.
Then, on a second pass, it travels to the end point via a direct route.

The hardware consists of:

Parallax Shield Kit (for Arduino)
stock 130-35000

Parallax QTI Line Follower AppKit for the Small Robot
Stock 28108 (2 of these; use 6 of the 8 sensors)

Arduino Uno

Breadboard button, press to make

The software is an adaptation of:

The source code for the Parallax
QTI Line Follower AppKit

and

Marcelo J. Rovai
https://create.arduino.cc/projecthub/mjrobot/maze-solver-robot-using-artificial-intelligence-4318cf

By studying the two software sources named above plus selected chapters of the excellent material written for the Parallax Shield Kit, you
can easily understand how this robot works.
```

```
Connections:
D7 – QTI.6 - Far left          D12 - Left servo
D6 – QTI.5 – Left              D11 - Right servo
D5 – QTI.4 - Mid left
D4 – QTI.3 - Mid right         D9 - buttonPin
D3 – QTI.2 - Right
D2 – QTI.1 - Far right
*/


#include <Servo.h>

#define STOPPED 0
#define FOLLOWING_LINE 1
#define NO_LINE 2
#define CONT_LINE 3            // a “T” junction, a “cross”, or the end of the maze
#define LEFT_TURN 4
#define RIGHT_TURN 5           // a “straight or right” junction or a “right only” junction

#define RIGHT 1
#define LEFT -1

const int farLeftPin = 7;
const int leftPin = 6;
const int midLeftPin = 5;
const int midRightPin = 4;
const int rightPin = 3;
const int farRightPin = 2;

const int buttonPin = 9;
```

```
int sensor[6] = {0, 0, 0, 0, 0, 0};
int mode = 0;
int error = 0;
int statos = 0;

char dir;
char path[100] = " ";
int pathLength = 0;
int pathIndex = 0;

Servo servoL;
Servo servoR;


void setup( )
{

servoL.attach(12);
servoR.attach(11);

pinMode(buttonPin, INPUT);

while(digitalRead(buttonPin)) {}

statos = 0;
mode = STOPPED;
}
```

```
void loop()
{
  readSensors();
  exploreMaze();
  while(digitalRead(buttonPin)) {}
  pathIndex = 0;
  statos = 0;
  directRoute();
  while(digitalRead(buttonPin)) {}
  mode = STOPPED;
  statos = 0;
  pathLength = 0;
  pathIndex =0;
  delay(10);
}

void readSensors()
{
  DDRD  |= B11111100;
  PORTD |= B11111100;
  delayMicroseconds(250);
  DDRD  &= B00000011;
  PORTD &= B00000011;
  delayMicroseconds(500);

  sensor[0] = digitalRead(farLeftPin);
  sensor[1] = digitalRead(leftPin);
  sensor[2] = digitalRead(midLeftPin);
  sensor[3] = digitalRead(midRightPin);
  sensor[4] = digitalRead(rightPin);
  sensor[5] = digitalRead(farRightPin);
  if(     (sensor[0]==1)&&(sensor[4]==0)) {mode = LEFT_TURN; error = 0;}
  else if((sensor[1]==0)&&(sensor[5]==1)) {mode = RIGHT_TURN; error = 0;}
  else if((sensor[1]==1)&&(sensor[2]==1)&&(sensor[3]==1)&&(sensor[4]==1)) {mode = CONT_LINE; error = 0;}
  else if((sensor[1]==0)&&(sensor[2]==0)&&(sensor[3]==0)&&(sensor[4]==0)) {mode = NO_LINE; error = 0;}
  else if((sensor[1]==0)&&(sensor[2]==0)&&(sensor[3]==0)&&(sensor[4]==1)) {mode = FOLLOWING_LINE; error = 3;}
  else if((sensor[1]==0)&&(sensor[2]==0)&&(sensor[3]==1)&&(sensor[4]==1)) {mode = FOLLOWING_LINE; error = 2;}
  else if((sensor[1]==0)&&(sensor[2]==0)&&(sensor[3]==1)&&(sensor[4]==0)) {mode = FOLLOWING_LINE; error = 1;}
  else if((sensor[1]==0)&&(sensor[2]==1)&&(sensor[3]==1)&&(sensor[4]==0)) {mode = FOLLOWING_LINE; error = 0;}
  else if((sensor[1]==0)&&(sensor[2]==1)&&(sensor[3]==0)&&(sensor[4]==0)) {mode = FOLLOWING_LINE; error = -1;}
  else if((sensor[1]==1)&&(sensor[2]==1)&&(sensor[3]==0)&&(sensor[4]==0)) {mode = FOLLOWING_LINE; error = -2;}
  else if((sensor[1]==1)&&(sensor[2]==0)&&(sensor[3]==0)&&(sensor[4]==0)) {mode = FOLLOWING_LINE; error = -3;}

}

void exploreMaze()
{
  while(!statos)
  {
    readSensors();
    switch(mode)
    {
      case NO_LINE:
      motorStop();
      motorTurn(LEFT, 180);
      recIntersection('B');
      break;
      case CONT_LINE:
      motorNudge();
      readSensors();
      if(mode !=CONT_LINE) {motorTurn(LEFT, 90); recIntersection('L');}
      else {mazeEnd();}
      break;

      case RIGHT_TURN:
      motorNudge();
      readSensors();
      if(mode == NO_LINE) {motorTurn(RIGHT, 90); recIntersection('R');}
      else {recIntersection('S');}
      break;

      case LEFT_TURN:
      motorTurn(LEFT, 90);
      recIntersection('L');
      break;

      case FOLLOWING_LINE:
      motorFollow();
      break;
    }
  }
}
```

```
void motorFollow()
{
  servoL.writeMicroseconds(1600 + 33*error);
  servoR.writeMicroseconds(1400 + 33*error);
  delay(50);
}

void motorStop()
{
  servoL.writeMicroseconds(1500);
  servoR.writeMicroseconds(1500);
  delay(200);
}

void motorNudge()
{
  servoL.writeMicroseconds(1600);
  servoR.writeMicroseconds(1400);
  delay(100);
  servoL.writeMicroseconds(1500);
  servoR.writeMicroseconds(1500);
  delay(200);
}
```

```
void motorTurn( int sense, int angle)
{
  servoL.writeMicroseconds(1600);
  servoR.writeMicroseconds(1400);
  delay(300);
  servoL.writeMicroseconds(1500 + 100*sense);
  servoR.writeMicroseconds(1500 + 100*sense);
  delay(round(6*angle + 20));
  servoL.writeMicroseconds(1500);
  servoR.writeMicroseconds(1500);
  delay(200);
}

void mazeEnd()
{
  motorStop();
  statos = 1;
  mode = STOPPED;
}

void recIntersection(char dir)
{
  path[pathLength] = dir;
  pathLength++;
  simplifyPath();
}
```

```
void simplifyPath()
{
  if(pathLength < 3 || path[pathLength - 2] != 'B') {return;}

  int totalAngle = 0;

  for(int i = 1; i <= 3; i++)
  {
    switch(path[pathLength - i])
    {
      case 'R':
      totalAngle = totalAngle + 90;
      break;

      case 'L':
      totalAngle = totalAngle + 270;
      break;

      case 'B':
      totalAngle = totalAngle + 180;
      break;

      case 'S':
      totalAngle = totalAngle + 0;
      break;
    }
  }

  totalAngle = totalAngle % 360;  // reminder upon division by 360
  switch(totalAngle)
  {
    case 0:
    path[pathLength - 3] = 'S';
    break;

    case 90:
    path[pathLength - 3] = 'R';
    break;

    case 180:
    path[pathLength - 3] = 'B';
    break;

    case 270:
    path[pathLength - 3] = 'L';
    break;
  }
  pathLength = pathLength - 2;
}
```

```
void directRoute()
{
  while(!statos)
  {
    readSensors();
    switch(mode)
    {
      case FOLLOWING_LINE:
      motorFollow();
      break;

      case CONT_LINE:
      if(pathIndex >= pathLength) {mazeEnd();}
      else {mazeTurn(path[pathIndex]); pathIndex++;}
      break;

      case LEFT_TURN:
      if(pathIndex >= pathLength) {mazeEnd();}
      else {mazeTurn(path[pathIndex]); pathIndex++;}
      break;

      case RIGHT_TURN:
      if(pathIndex >= pathLength) {mazeEnd();}
      else {mazeTurn(path[pathIndex]); pathIndex++;}
      break;
    }
  }
}
```

```
void mazeTurn(char dir)
{
 switch(dir)
 {
  case 'L':
  motorTurn(LEFT, 90);
  break;

  case 'R':
  motorTurn(RIGHT, 90);
  break;

  case 'B':
  motorTurn(RIGHT, 800);  // should never happen
  break;

  case 'S':
  motorNudge();
  break;
 }
}
```