```
///////////////////////////////////////////////////////////////////////////
//
// This is the Program to control the Robot Head
// Program By: Arushi Raghuvanshi
// Date: Oct 2006
//
///////////////////////////////////////////////////////////////////////////

// Define the sensor constants to use later in the program for readability
const tSensors touchSensor          = (tSensors) S2;    //touch Sensor
const tSensors leftLightSensor      = (tSensors) S3;    //light sensor
const tSensors rightLightSensor     = (tSensors) S1;    //light sensor

// Define Global Variables for use in the program
int M[4][4];        // This is the 4x4 matrix for the gate
int iState[4];      // This is 4x1 matrix for the input state
int oState[4];      // This is 4x1 matrix for the output state

bool a, b;          // a and b are two inputs, mapped to some combination of ser
bool P, Q;          // P and Q are two outputs, mapped to some combination of mc

void setMatrixM()
{
  // Initialize the matrix for the gate
  // The program behavior changes just by changing this matrix

  // This matrix here is a simple identity matrix
  // implying that inputs are directly connected to outputs

  M[0][0] = 1; M[0][1] = 0; M[0][2] = 0; M[0][3] = 0;
  M[1][0] = 0; M[1][1] = 1; M[1][2] = 0; M[1][3] = 0;
  M[2][0] = 0; M[2][1] = 0; M[2][2] = 1; M[2][3] = 0;
  M[3][0] = 0; M[3][1] = 0; M[3][2] = 0; M[3][3] = 1;
}

void setInputState()
{
  // Initialize the input matrix, by looking at the input variables a and b
  iState[0] = !a & !b;  //00
  iState[1] = !a & b;   //01
  iState[2] = a & !b;   //10
  iState[3] = a & b;    //11
}

void multiplyMatrix()
{
  // do the regular matrix multiplation of M with iState to get matrix oState
  int i, j;

  for (i=0; i<4; i++)
  {
    oState[i] = 0;
    for (j=0; j<4; j++)
    {
      oState[i] = oState[i] + (M[i][j] * iState[j]);
    }

    // if the value is negative, set it to positive, since sign is lost in
    // measurement of quantum gate output
    if (oState[i] < 0)
      oState[i] = oState[i] * (-1);
  }
}

void interpretOutputState()
{
  // interpret oState matrix in terms of P and Q actions

  int total=0;
  int i=0;
  int oneCounter=0;
  int x=0;

  // First reset P & Q
  P = false;
  Q = false;

  // first count how many ones
  total = oState[0] + oState[1] + oState[2] + oState[3];

  // probablity of each '1's in the matrix is equally randomized
  // following routine returs a number from 0 to total-1
  // so if the total # of ones are 2, then the following routine
  // returns either 0 or 1, with equal probability
  if (total > 1 )
    x = random(total-1);
  else
    x = 0;

  // Now out of the oState entries that have a value of '1', we pick
  // an the entry based on the random value above
  for (i=0; i<4; i++)
  {
    if (oState[i] != 0)
    {
      if (x == oneCounter)
      {
         // set the action flags, use the value of i to calculate P and Q
          switch (i)
          {
            case 0 : P = false; Q = false; break;
            case 1 : P = false; Q = true; break;
            case 2 : P = true; Q = false; break;
            case 3 : P = true; Q = true; break;
          }

          // the above switch/case can also be replaced with following two state
          // P = (bool) i/2;
          // Q = (bool) i%2;
          break;  // got a match, break out of for loop
      }
      oneCounter++;
    }
  }
}

// Following two routines are the only routines that are specific to
// a particular robot design

void executeAction()
{
  // P and Q are either true or false based on output matrix
  // This routine translates P and Q into motor actions

  // Port A is attached to base motor, rotates the head, can move at normal spee
  // Port B has two motors attached, moves both eyebrows - needs to move at low
  // Port C has 3 motors attached - moves 2 eyes and mouth -
  //            This needs to move at low power, and also need to move back & fc
  //            so has to know previous state


  if (P)
  {
    // turn the head right, wait a while, and turn it back straight
    motor[motorA] = 100; wait1Msec(700);
    motor[motorA] = -100; wait1Msec(700); motor[motorA] = 0;
  }
  else
  {
    // do not rotate the head
    motor[motorA] = 0;
  }

  if (Q)
  {
    // Move eyebrows back and forth
    motor[motorB] = 15; wait1Msec(250);
    motor[motorB] = -15; wait1Msec(100); motor[motorB] = 0;

    // Move eyes and mouth, and come back to normal position
    motor[motorC] = 30; wait1Msec(500);
    motor[motorC] = -30; wait1Msec(500); motor[motorA] = 0;
  }
  else
  {
    // do not change any facial expressions
    motor[motorB] = 0;
    motor[motorC] = 0;
  }

}

void checkSensors()
{
  // This routine checks the value of sensors and sets the variables a and b

  // Check for a significant difference (10%) between left light sensor and righ
  // To test, shine a flashlight in front of left sensor
  if (SensorValue(leftLightSensor) > (SensorValue(rightLightSensor)+10))
```

```
    a = true;
  else
    a = false;

  // Check if either of the touch sensors are pressed
  // Both touch sensors are attached to same port and are OR'ed
  if (SensorValue(touchSensor) == 1 )
    b = true;
  else
    b = false;

  ClearSensorValue(leftLightSensor);
  ClearSensorValue(rightLightSensor);
  ClearSensorValue(touchSensor);
}

// This is the main program

task main()
{
  // Initialize sensor types, connected to three ports
  SetSensorType(leftLightSensor, sensorReflection);
  SetSensorType(rightLightSensor, sensorReflection);
  SetSensorType(touchSensor, sensorTouch);

  // Clear all sensors
  ClearSensorValue(leftLightSensor);
  ClearSensorValue(rightLightSensor);
  ClearSensorValue(touchSensor);


  // Initialize all motors to 0 - stopped
  motor[motorA] = 0;
  motor[motorB] = 0;
  motor[motorC] = 0;

  // Set the matrix for the gate
  setMatrixM();

  while (true)
  {
    PlayTone(220, 10);      //debugging tool - Plays a 220hz tone for 1 second
    wait1Msec(1000);        //wait 5 seconds before next cycle
    checkSensors();         //Check Sensors and set input variables
    setInputState();        //Create input matrix
    multiplyMatrix();       //Compute output matrix
    interpretOutputState(); //Determine what actions to do
    executeAction();        //Execute those actions
    PlayTone(990, 50);      //debugging tool - Plays a 330hz tone for 1 second

    wait1Msec(3000);         //wait 5 seconds before next cycle
  }
}
```