```
//  Arduino Binary Counter
//  Computer Science 100

//  Dr. Christopher Vickery
//  Queens College of CUNY

/**************************************************************************
 *  This code is provided under a Creative Commons Attribution-ShareAlike *
 *  license.                                                              *
 *  That means you are free to copy it and pass it on to others provided  *
 *  (a) you cite me (Dr. Christopher Vickery, Queens College of CUNY) as  *
 *  the original author in comments in the code and (b) you require       *
 *  those you give it to to do the same.                                  *
 **************************************************************************/

/*  Implements a five-bit binary counter as a finite state machine (FSM)
 *  with three external inputs:
 *    UpDn_SW  A switch that controls whether to count up or down i.e.,
 *             in increasing or decreasing numerical sequence.
 *    Timed_SW  A switch that controls whether the counter changes state
 *              after a time delay (once per second) or in response to
 *              a button press.
 *    Count_PB  The button that causes the count to increase or decrease
 *              when it changes state. Used only when not doing timed counting.
 */

//  Define pins for different types of Arduino boards.
/*  Use UNO for "normal" boards, like UNO, Leonardo, and compatibles.
 *  Otherwise, assume the board is a Teagueduino.
 */
#define UNO

//  Normal Pin Numbers
#ifdef UNO
#define Out_0 12
#define Out_1 11
#define Out_2 10
#define Out_3 9
#define Out_4 8
#define Count_PB 2
#define Timed_SW 3
#define UpDn_SW  4

//  Teaguedino Pin Numbers
#else
#define Out_0 13
#define Out_1 14
#define Out_2 15
#define Out_3 16
#define Out_4 17
#define Count_PB 45
```

```
#define Timed_SW 44
#define UpDn_SW  43
#endif

// The five state bits
/*  The bits are combined to represent 32 different states.
 *  When viewed in order, with bit 0 on the right, the bits
 *  represent a binary number. When counting up, state transitions
 *  cause the binary number to increase by one. When counting down,
 *  state transitions cause the binary number to decrease by one.
 */
int bit_0 = 0;
int bit_1 = 0;
int bit_2 = 0;
int bit_3 = 0;
int bit_4 = 0;

//  Which direction to count
int up_dn  = 0;// 0 => up; 1 => dn

//  setup()
//  -------------------------------------------------------------------
/*  Initialize the I/O modes for the various pins.
 *  Note that the Count pushbutton is assumed to be zero when not
 *  pressed, which is automatic on Teagueduino, but requires a pulldown
 *  resistor on other Arduinos.
 */
void setup()
{
  //  Pin modes
  pinMode(Out_0, OUTPUT);
  pinMode(Out_1, OUTPUT);
  pinMode(Out_2, OUTPUT);
  pinMode(Out_3, OUTPUT);
  pinMode(Out_4, OUTPUT);
  pinMode(Count_PB, INPUT);  // Change states manually
  pinMode(Timed_SW, INPUT);  // 0 => manual; 1 => timed
  pinMode(UpDn_SW,  INPUT);  // 0 => count up; 1 => count down
}

//  loop()
//  --------------------------------------------------------------------
/*  Implements a Finite State Machine with five bits used to represent
 *  32 states.
 */
void loop()
{
  //  Display current state
  //  --------------------------------------------------------
  digitalWrite(Out_0, bit_0);
  digitalWrite(Out_1, bit_1);
  digitalWrite(Out_2, bit_2);
  digitalWrite(Out_3, bit_3);
  digitalWrite(Out_4, bit_4);

  //  Wait for next state transition
  //  --------------------------------------------------------
  //  Determine whether to use time or button presses
  if  (digitalRead(Timed_SW) )
  {
    //  Timed
    //  ------------------------------------------------------
    delay(1000);
  }
  else
  {
    //  Button press
    //  ------------------------------------------------------
    while ( 1 ==digitalRead(Count_PB) )
    {
      //  Be sure button is not already pressed
      delay(10);  //  Allow time for contact bounce
    }
    while ( 0 ==digitalRead(Count_PB) )
    {
      //  Wait for it to be pressed
      delay(10);  //  Bounce time
    }
  }

  //  Calculate next state
  //  --------------------------------------------------------
  /*  The algorithm is efficient but requires explanation.
   *
   *     First, note that the rightmost bit (bit_0) always
   *     toggles, whether the binary number is increasing or
   *    decreasing.
   *
   *    The next bit, bit_1 toggles if the rightmost bit was
   *    a 1 and the count direction is "up." It also toggles
   *    if the rightmost bit was a 0 and the count direction
   *    is "down." Note the word "was" in the previous sentences;
   *    just before testing whether or not to toggle bit_1, bit_0
   *    was toggled, so the testing has to be reversed. But this
   *    reversal is handled by defining up_dn to be 0 when
   *    counting up, and 1 when counting down.
   *
   *    Two more things:
   *      1.  This program never assigns values other than 0 or 1
   *          to the bit_x variables, so 1 - bit_x "toggles" the
   *          value (inverts the bit). 1 - 0 => 1 and 1 - 1 => 0.
   *      2.  The nested if statements mean that all the bits to
   *          the right of a particular bit_x have to have been
   *          toggled in order for bit_x to toggle.
```

```
   */
   up_dn  =digitalRead(UpDn_SW);  // Up or down counting?
      bit_0   =   1   -   bit_0;// Always toggle right bit
  if ( bit_0 == up_dn )
  {
    bit_1 = 1 - bit_1;
    if ( bit_1 == up_dn)
    {
      bit_2 = 1 - bit_2;
      if ( bit_2 == up_dn)
      {
        bit_3 = 1 - bit_3;
        if (bit_3 == up_dn)
        {
          bit_4 = 1 - bit_4;
        }
      }
    }
  }
  //  Here is an alternative piece of code that would do the same thing
  //  as the nested if statements above. It's less efficient because
  //  the same value is tested multiple times instead of just once. But
  //  the logic might be somewhat clearer. The && operator means "and".
//  if (bit_0 == up_dn)
//  {
//    bit_1 = 1 - bit_1;
//  }
//  if (bit_0 == up_dn && bit_1 == up_dn)
//  {
//    bit_2 = 1 - bit_2;
//  }
//  if (bit_0 == up_dn && bit_1 == up_dn && bit_2 == up_dn)
//  {
//    bit_3 = 1 - bit_3;
//  }
//  if (bit_0 == up_dn && bit_1 == up_dn && bit_2 == up_dn && bit_3 == up_dn)
//  {
//    bit_4 = 1 - bit_4;
//  }
}
```