```
/*
 * File:   dice.c
 * Program simulates rolling a traditional six faced dice as commonly used in board games
 * Target microcontroller: PIC 16f690
 * C compiler: XC8
 * Author: Phil
 *
 * Created on 09 October 2016, 19:23
 */



// PIC16F690 Configuration Bit Settings


// 'C' source line config statements


#include <xc.h>
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.


// CONFIG
#pragma config FOSC = INTRCIO   // Oscillator Selection bits (INTOSCIO oscillator: I/O function on RA4/OSC2/CLKOUT pin, I/O function on RA5/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select bit (MCLR pin function is MCLR)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
```

```
#pragma config BOREN = OFF      // Brown-out Reset Selection bits (BOR disabled)

#pragma config IESO = OFF       // Internal External Switchover bit (Internal External Switchover mode is disabled)

#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is disabled)

#define _XTAL_FREQ 4000000      // internal clock frequency 4MHz


char count;                  // global variable


void init(void)              // sets up inputs and outputs on PORTB and PORTC

{

   ANSEL = 0x00;             // ADC module off

   ANSELH = 0x00;

   CM1CON0 = 0x00;           // Comparator module off

   CM2CON0 = 0x00;

   TRISBbits.TRISB4 = 1;     // RB4 input

   TRISBbits.TRISB5 = 0;     // RB5 output

   PORTBbits.RB5 = 0;        // clear RB5

   TRISC = 0x00;             // RC0-RC7 outputs

   PORTC = 0x00;             // turn off all bits in PORTC

 }


void TMR0_setup(void)

{

   OPTION_REGbits.PS2 = 1;    // set prescaler to 1:128

   OPTION_REGbits.PS1 = 1;    // 4MHz internal clock yields

   OPTION_REGbits.PS0 = 0;    // 255 x 256us = 32.768ms between interrupts

   OPTION_REGbits.nRABPU = 1;  // PORTA/B pullups disabled
```

```
   OPTION_REGbits.INTEDG = 0;  // don't care - only applies to external edge triggered interrupts
   OPTION_REGbits.T0CS = 0;    // internal clock source for interrupt fosc/4
   OPTION_REGbits.T0SE = 0;    // don't care - only applies to external edge triggered interrupts
   OPTION_REGbits.PSA = 0;     // scaler assigned to TMR0 module NOT watchdog timer
   TMR0 = 0x00;              // TMR0 starts counting from 0 and overflows at 255
   INTCONbits.GIE = 1;        // global interrupts enabled
   INTCONbits.T0IE = 1;       // TMR0 interrupts enabled
   INTCONbits.T0IF = 0;       // TMR0 interrupt flag cleared

}

void interrupt isr()          // interrupt routine set to occur every 32.768ms to flash "rolling dice" leds
{
   if( INTCONbits.T0IF == 1)   // check that the interrupt occurred as a result of TMR0 overflowing
   {
      if( PORTBbits.RB4 == 0) // check that "rolling dice button pressed"
      {
         switch (count)     // take the current value of count (0 to 5) and match to a case statement
         {
               case 0: PORTCbits.RC1 = 1;  // if count = 0, light RC1 LED
                     count = count + 1;  // increment count by one
                     break;            // leave the switch case code block
               case 1: PORTCbits.RC4 = 1;  // if count = 1,  light RC4 LED
                     PORTCbits.RC1 = 0;  // turn off RC1 LED
                     count = count + 1;  // increment count by one
                     break;            // leave the switch case code block
               case 2: PORTCbits.RC7 = 1;  // if count = 2,  light RC7 LED
```

```
                PORTCbits.RC4 = 0;  // turn off RC4 LED
                count = count + 1;  // increment count by one
                break;              // leave the switch case code block
            case 3: PORTCbits.RC2 = 1;  // if count = 3,  light RC2 LED
                PORTCbits.RC7 = 0;  // turn off RC7 LED
                count = count + 1;  // increment count by one
                break;              // leave the switch case code block
            case 4: PORTCbits.RC5 = 1;  // if count = 4,  light RC5 LED
                PORTCbits.RC2 = 0;  // turn off RC2 LED
                count = count + 1;  // increment count by one
                break;              // leave the switch case code block
            case 5: PORTCbits.RC6 = 1;  // if count = 5,  light RC6 LED
                PORTCbits.RC5 = 0;  // turn off RC5 LED
                count = count + 1;  // increment count by one
                break;              // leave the switch case code block
          default:    PORTCbits.RC1 = 1;  // if count > 5, light RC1 LED
                PORTCbits.RC6 = 0;  // turn off RC6 LED
                count = 0;          // reset count to zero
        }
    }
    TMR0 = 0x00;                    // TMR0 starts counting from 0x00 again
    INTCONbits.T0IF = 0;            // clear TMRO interrupt flag to allow new interrupts
    PORTBbits.RB5 = ~PORTBbits.RB5;     // visual debug of interrupt routine via LED
                                // attached to RB5 (optional) flips RB5 on entry to ISR
  }
}
void main(void)
```

```
{   char i = 0;                         // i is index variable for array dice[i]
    char bit_pattern = 0x00;             // bit_pattern is variable copied to PORTC
    char dice[6] = {0x01, 0x06, 0x07, 0x36, 0x37, 0xf6};// LED patterns to represent 1 through 6
    init();                             // initialise microcontroller
    TMR0_setup();                       // setup TMR0 for interrupt routine
    PORTCbits.RC0 == 0;                  // clear RC0
    while(1==1)                         // do {code block forever}
    {
      while (RB4 == 0)                  // do {code block while button pressed}
      {
        if(i == 6)                      // variable i should be contained from 0 to 5
        {
          i = 0;                        // if it equals 6, clear it to zero
        }
        bit_pattern = dice[i];          // select dice pattern based on current i value
        i = i+1;                        // increment i by one
      }
      while(RB4 == 1)                   // do {code block while button released}
      {
      PORTC = 0x00;                     // turn off all LEDS
      __delay_ms(200);                  // wait 200ms
      PORTC = bit_pattern;              // display dice value
      __delay_ms(200);                  // wait 200ms
      }
    }
}
```