```c
/* jdefo002_lab4_part1.c − April 22, 2013
 * Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * CS Login: jdefo002
 * Partner Name: Ariana DeJaco E−mail adeja001@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#5 Exercise#1
 * Exercise Description: Complete the following program by defining
 * the function "void Write7Seg(unsigned char x)" that sets the
 * 7−segment display to display x for values of 0 to 9. So if x is 0,
 * illuminate segments A, B, C, D, E, and F (but not G). If x is 1,
 * illuminate segments B and C. If x > 9, illuminate nothing.
 * Carefully examine the above photo, the ATmega32 pinout, and the
 * seven−segment pinout, to properly define the function.
 */

// Use for debouncing the switch (How many Hz)
#define F_CPU 1000000

#include <avr/io.h>
#include <util/delay.h>
#include <avr/sfr_defs.h>

// Bit−access function
unsigned char GetBit(unsigned char x, unsigned char k)
{
        return ((x & (0x01 << k)) != 0);
}

// Bit−set function
unsigned char SetBit(unsigned char x, unsigned char k, unsigned char b)
{
        return (b ? x | (0x01 << k) : x & ~(0x01 << k));
}

// Current Port Definitions
#define LED_DDR                         DDRC
#define LED_INPORT                      PINC
#define LED_OUTPORT                     PORTC
#define SENSOR_DDR                      DDRA
#define SENSOR_INPORT           PINA
#define SENSOR_OUTPORT          PORTA
#define UNUSEDB_DDR         DDRB
#define UNUSEDB_PIN         PINB
#define UNUSEDB_PORT        PORTB
#define UNUSEDD_DDR         DDRD
#define UNUSEDD_PIN         PIND
#define UNUSEDD_PORT        PORTD

 // Additional macros not defines in sfr_defs.h
 #define SET_PORT_BIT(OUTPORT, BIT)             OUTPORT |= (1 << BIT)
 #define CLEAR_PORT_BIT(OUTPORT, BIT)   OUTPORT &= ~(1 << BIT)

//DDRA: Configures each of port A’s physical pins to input (0) or output(1)
//PORTA: Writing to this register writes the port’s physical pins
// (Write only)
//PINA: Reading this register reads the values of the port’s physical pins
// (Read only)

// Creating a type "statetpe" and making 2 variables of this type and initializi
ng them to InitReset
typedef enum { InitReset, Increment, Decrement, WaitForButtonPress, WaitForButto
nRelease, ShiftLeft, ShiftRight, ErrorState } statetype;
statetype CurrentState  = InitReset;
statetype ActionState = InitReset;
#include <avr/io.h>


unsigned char Write7Seg(unsigned char x)
{
        unsigned char segvalue = 0;
        switch(x)
        {
                case 0:
```

```c
                        segvalue = 0x3F;
                        break;
                case 1:
                        segvalue = 0x0C;
                        break;
                case 2:
                        segvalue = 0x5B;
                        break;
                case 3:
                        segvalue = 0x5E;
                        break;
                case 4:
                        segvalue = 0x6C;
                        break;
                case 5:
                        segvalue = 0x76;
                        break;
                case 6:
                        segvalue = 0x77;
                        break;
                case 7:
                        segvalue = 0x1C;
                        break;
                case 8:
                        segvalue = 0x7F;
                        break;
                case 9:
                        segvalue = 0x7C;
                        break;
                deault:
                        segvalue = 0x73;
                        break;
        }

        return segvalue;
}

int rand(void);

int main(void)
{
        unsigned char cnt=0;
        DDRA = 0x00; PORTA = 0xFF; // Config port A as inputs, init 1s
        DDRC = 0xFF; PORTC = 0x00; // Config port C as outputs, init 0s
        while(1) {
                if (GetBit(PINA, 0)==0) { // Button pressed
                cnt = rand() % 10; // cnt is rand num from 0−9
                PORTC = Write7Seg(cnt);
        }
}
}
```

```c
/* jdefo002_lab5_part2.c − April 22, 2013
 * Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * CS Login: jdefo002
 * Partner Name: Ariana DeJaco Email adejaoo1@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#5 Exercise#2
 * Exercise Description: (From an earlier lab) Buttons are connected
 * to PA0 and PA1. Output PORTC drives the 7−segment display initially
 * displaying 0.  Pressing PA0 increments the display (stopping at 9).
 * Pressing PA1 decrements the display (stopping at 0). If both buttons
 * are depressed (even if not initially simultaneously), the display
 * resets to 0. Use a state machine (not synchronous) captured in C.
 */

// Used for debouncing the button (How many Hz)
#define F_CPU 1000000

#include <avr/io.h>
#include <util/delay.h>
#include <avr/sfr_defs.h>

// Bit−access function
unsigned char GetBit(unsigned char x, unsigned char k)
{
        return ((x & (0x01 << k)) != 0);
}

// Current Port Definitions
#define LED_DDR                         DDRC
#define LED_INPORT                      PINC
#define LED_OUTPORT                     PORTC
#define SENSOR_DDR                      DDRA
#define SENSOR_INPORT           PINA
#define SENSOR_OUTPORT          PORTA
#define UNUSEDB_DDR         DDRB
#define UNUSEDB_PIN         PINB
#define UNUSEDB_PORT        PORTB
#define UNUSEDD_DDR         DDRD
#define UNUSEDD_PIN         PIND
#define UNUSEDD_PORT        PORTD

 // Additional macros not defines in sfr_defs.h
 #define SET_PORT_BIT(OUTPORT, BIT)             OUTPORT |= (1 << BIT)
 #define CLEAR_PORT_BIT(OUTPORT, BIT)   OUTPORT &= ~(1 << BIT)

//DDRA: Configures each of port A’s physical pins to input (0) or output(1)
//PORTA: Writing to this register writes the port’s physical pins
// (Write only)
//PINA: Reading this register reads the values of the port’s physical pins
// (Read only)

enum Counter_States { InitReset, Increment, Decrement, WaitForButtonPress, WaitF
orButtonRelease, ErrorState } CounterState;

unsigned char TckFct_Counter(unsigned char inputData, unsigned char LedValue)
{
        // Variable we are returning
        unsigned char tempLedValue = LedValue;

        switch(CounterState)
        {
                //Transitions
                case InitReset:  // Initial Transition
                        CounterState = WaitForButtonRelease;
                        break;
                case Increment:
                    CounterState = WaitForButtonRelease;
                        break;
                case Decrement:
                        CounterState = WaitForButtonRelease;
                        break;
                case WaitForButtonPress:
                    if(inputData == 0x00)
```

```c
                        {
                                CounterState = WaitForButtonPress;
                        }
                        else if(inputData == 0x01)
                        {
                                if(tempLedValue < 9)
                                {
                                        CounterState = Increment;
                                }
                                // Will stay in wait if not < 9
                        }
                        else if(inputData == 0x02)
                        {
                                if(tempLedValue > 0)
                                {
                                        CounterState = Decrement;
                                }
                                // Will stay in wait if not > 0
                        }
                        else if(inputData == 0x03)
                        {
                                CounterState = InitReset;
                        }
                        else
                        {
                                CounterState = ErrorState;
                        }
                        break;
                case WaitForButtonRelease:
                    if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        break;
                case ErrorState:
                        break;
                default:
                        CounterState = ErrorState;
                        break;
        }

        switch(CounterState)
        { // Actions
                case Increment:
                        tempLedValue++;
                        break;
                case Decrement:
                        tempLedValue−−;
                        break;
                case InitReset:
                        tempLedValue = 0x00;
                        break;
                case WaitForButtonPress:
                        break;
                case WaitForButtonRelease:
                        break;
                case ErrorState:
                        tempLedValue = 0xFF;
                        break;
                default:
                        break;
        }
        return tempLedValue;
}

unsigned char Write7Seg(unsigned char x)
{
        unsigned char segvalue = 0;

        switch(x)
        {
                case 0:
                        segvalue = 0x3F;
```

```
                                break;
                        case 1:
                                segvalue = 0x0C;
                                break;
                        case 2:
                                segvalue = 0x5B;
                                break;
                        case 3:
                                segvalue = 0x5E;
                                break;
                        case 4:
                                segvalue = 0x6C;
                                break;
                        case 5:
                                segvalue = 0x76;
                                break;
                        case 6:
                                segvalue = 0x77;
                                break;
                        case 7:
                                segvalue = 0x1C;
                                break;
                        case 8:
                                segvalue = 0x7F;
                                break;
                        case 9:
                                segvalue = 0x7C;
                                break;
                        deault:
                                segvalue = 0x73;
                                break;
        }

        return segvalue;
}

int main(void)
{
        SENSOR_DDR = 0x00;
        LED_DDR    = 0xFF;
        UNUSEDB_DDR= 0x00;
        UNUSEDD_DDR= 0x00;

        // Initialize LEDs to off
        LED_OUTPORT = 0x00;
        unsigned char ButtonValue = 0;
        unsigned char CurrentLEDValue = 0;
        unsigned char SevenSegValue = 0;

    while(1)
    {
                // Code to debounce the switch.
                while (ButtonValue != (~SENSOR_INPORT & 0x03))
                {
                        _delay_ms(50);
                        ButtonValue = (~SENSOR_INPORT & 0x03);
                }

                CurrentLEDValue = TckFct_Counter(ButtonValue, CurrentLEDValue);
                SevenSegValue = Write7Seg(CurrentLEDValue);
                LED_OUTPORT = SevenSegValue;
    }
}
```

```c
/* jdefo002_lab5_part3.c − April 22, 2013
 * Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * CS Login: jdefo002
 * Partner Name: Ariana DeJaco Email adejaoo1@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#5 Exercise#3
 * Exercise Description: Using the seven−segment display, buttons,
 * and LEDs, create a very simple game. It can be any game, and it
 * can be very simple −− but each student pair must create and
 * implement their own game. Use a state machine (not synchronous)
 * as appropriate.
 */

// Used for debouncing the button (How many Hz)
#define F_CPU 1000000

#include <avr/io.h>
#include <util/delay.h>
#include <avr/sfr_defs.h>

// Bit−access function
unsigned char GetBit(unsigned char x, unsigned char k)
{
        return ((x & (0x01 << k)) != 0);
}

// Current Port Definitions
#define LED_DDR                         DDRC
#define LED_INPORT                      PINC
#define LED_OUTPORT                     PORTC
#define SENSOR_DDR                      DDRA
#define SENSOR_INPORT           PINA
#define SENSOR_OUTPORT          PORTA
#define UNUSEDB_DDR         DDRB
#define UNUSEDB_PIN         PINB
#define UNUSEDB_PORT        PORTB
#define UNUSEDD_DDR         DDRD
#define UNUSEDD_PIN         PIND
#define UNUSEDD_PORT        PORTD

 // Additional macros not defines in sfr_defs.h
 #define SET_PORT_BIT(OUTPORT, BIT)             OUTPORT |= (1 << BIT)
 #define CLEAR_PORT_BIT(OUTPORT, BIT)   OUTPORT &= ~(1 << BIT)

//DDRA: Configures each of port A’s physical pins to input (0) or output(1)
//PORTA: Writing to this register writes the port’s physical pins
// (Write only)
//PINA: Reading this register reads the values of the port’s physical pins
// (Read only)


enum Counter_States { InitReset, LEDLight, WaitForButtonPress, WaitForButtonRelease, ErrorState } CounterState;

unsigned char Write7SegNumber(unsigned char x)
{
        unsigned char segvalue = 0;

        switch(x)
        {
                case 0:
                        segvalue = 0x3F;
                        break;
                case 1:
                        segvalue = 0x0C;
                        break;
                case 2:
                        segvalue = 0x5B;
                        break;
                case 3:
                        segvalue = 0x5E;
                        break;
                case 4:
```

```c
                        segvalue = 0x6C;
                        break;
                case 5:
                        segvalue = 0x76;
                        break;
                case 6:
                        segvalue = 0x77;
                        break;
                case 7:
                        segvalue = 0x1C;
                        break;
                case 8:
                        segvalue = 0x7F;
                        break;
                case 9:
                        segvalue = 0x7C;
                        break;
                default:
                        segvalue = 0x73;
                        break;
        }

        return segvalue;
}

unsigned char Write7SegLetter(unsigned char randNum, unsigned char guess)
{
        unsigned char segvalue = 0;
        unsigned char ranvalue = randNum;

        /*if(randNum > guess)
        {
                segvalue = 0x6D;
        }
        else if(randNum < guess)
        {
                segvalue = 0x23;
        }
        else*/ if(randNum == guess)
        {
                segvalue = Write7SegNumber(ranvalue);
        }
        else
        {
                segvalue = 0x00;
        }

        return segvalue;
}

unsigned char TckFct_Counter(unsigned char inputData, unsigned char randNum, unsigned char sevSegVal)
{
        // Variable we are returning
        unsigned char tempLedValue = sevSegVal;

        switch(CounterState)
        {
                //Transitions
                case InitReset:  // Initial Transition
                        CounterState = WaitForButtonRelease;
                        break;
                case LEDLight:
                        CounterState = WaitForButtonRelease;
                        break;
                case WaitForButtonPress:
                        if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        else
                        {
                                CounterState = LEDLight;
                        }
                        break;
                case WaitForButtonRelease:
                        if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        break;
                /*case ErrorState:
                        break;*/
                default:
                        CounterState = InitReset;//ErrorState;
                        break;
        }

        switch(CounterState)
        { // Actions
                case LEDLight:
                        tempLedValue = Write7SegLetter(inputData, randNum);
                        break;
                case InitReset:
                        tempLedValue = 0x00;
                        break;
                case WaitForButtonPress:
                        break;
                case WaitForButtonRelease:
                        break;
                /*case ErrorState:
                        tempLedValue = 0xFF;
                        break;*/
                default:
                        break;
        }
        return tempLedValue;
}

int rand(void);

int main(void)
{
        SENSOR_DDR = 0x00;
        LED_DDR    = 0xFF;
        UNUSEDB_DDR= 0x00;
        UNUSEDD_DDR= 0x00;

        // Initialize LEDs to off
        LED_OUTPORT = 0x00;
        unsigned char ButtonValue = 0;
        unsigned char SevenSegValue = 0;

        unsigned char RandNumber = 4;

    while(1)
    {
                // Code to debounce the switch.
                while (ButtonValue != (~SENSOR_INPORT & 0x0F))
                {
                        _delay_ms(50);
                        ButtonValue = (~SENSOR_INPORT & 0x0F);
                }

                SevenSegValue = TckFct_Counter(ButtonValue, RandNumber, SevenSeg
Value);
                LED_OUTPORT = SevenSegValue;
    }
}
```

```c
/* jdefo002_lab5_partchallenge.c - April 22, 2013
 * Name: Joshua DeForest-Williams E-mail jdefo002@ucr.edu
 * CS Login: jdefo002
 * Partner Name: Ariana DeJaco Email adejaoo1@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#5 Exercise#challenge
 * Exercise Description: Create a more complex game -- make it fun!
 * However, still use basic (not synchronous) state machines -- so no time-interval behavior in your game.
 */

// Used for debouncing the button (How many Hz)
#define F_CPU 1000000

#include <avr/io.h>
#include <util/delay.h>
#include <avr/sfr_defs.h>

// Bit-access function
unsigned char GetBit(unsigned char x, unsigned char k)
{
        return ((x & (0x01 << k)) != 0);
}

// Current Port Definitions
#define LED_DDR                         DDRC
#define LED_INPORT                      PINC
#define LED_OUTPORT                     PORTC
#define SENSOR_DDR                      DDRA
#define SENSOR_INPORT           PINA
#define SENSOR_OUTPORT          PORTA
#define UNUSEDB_DDR         DDRB
#define UNUSEDB_PIN         PINB
#define UNUSEDB_PORT        PORTB
#define UNUSEDD_DDR         DDRD
#define UNUSEDD_PIN         PIND
#define UNUSEDD_PORT        PORTD

 // Additional macros not defines in sfr_defs.h
 #define SET_PORT_BIT(OUTPORT, BIT)             OUTPORT |= (1 << BIT)
 #define CLEAR_PORT_BIT(OUTPORT, BIT)   OUTPORT &= ~(1 << BIT)

//DDRA: Configures each of port A’s physical pins to input (0) or output(1)
//PORTA: Writing to this register writes the port’s physical pins
// (Write only)
//PINA: Reading this register reads the values of the port’s physical pins
// (Read only)


enum Counter_States { InitReset, LEDLight, WaitForButtonPress, WaitForButtonRelease } CounterState;

unsigned char Write7SegNumber(unsigned char x)
{
        unsigned char segvalue = 0;

        switch(x)
        {
                case 0:
                        segvalue = 0x3F;
                        break;
                case 1:
                        segvalue = 0x0C;
                        break;
                case 2:
                        segvalue = 0x5B;
                        break;
                case 3:
                        segvalue = 0x5E;
                        break;
                case 4:
                        segvalue = 0x6C;
                        break;
```

```c
                case 5:
                        segvalue = 0x76;
                        break;
                case 6:
                        segvalue = 0x77;
                        break;
                case 7:
                        segvalue = 0x1C;
                        break;
                case 8:
                        segvalue = 0x7F;
                        break;
                case 9:
                        segvalue = 0x7C;
                        break;
                default:
                        segvalue = 0x73;
                        break;
        }

        return segvalue;
}

unsigned char Write7SegLetter(unsigned char randNum, unsigned char guess)
{
        unsigned char segvalue = 0;
        unsigned char ranvalue = randNum;

        if(randNum < guess)
        {
                segvalue = 0x6D;
        }
        else if(randNum > guess)
        {
                segvalue = 0x23;
        }
        else if(randNum == guess)
        {
                segvalue = Write7SegNumber(ranvalue);
        }
        else
        {
                segvalue = 0x00;
        }

        return segvalue;
}

unsigned char TckFct_Counter(unsigned char inputData, unsigned char randNum, unsigned char sevSegVal)
{
        // Variable we are returning
        unsigned char tempLedValue = sevSegVal;

        switch(CounterState)
        {
                //Transitions
                case InitReset:  // Initial Transition
                        CounterState = WaitForButtonRelease;
                        break;
                case LEDLight:
                        CounterState = WaitForButtonRelease;
                        break;
                case WaitForButtonPress:
                        if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        else
                        {
                                CounterState = LEDLight;
                        }
                        break;
                case WaitForButtonRelease:
                        if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        break;
                default:
                        CounterState = InitReset;//ErrorState;
                        break;
        }

        switch(CounterState)
        { // Actions
                case LEDLight:
                        tempLedValue = Write7SegLetter(inputData, randNum);
                        break;
                case InitReset:
                        tempLedValue = 0x00;
                        break;
                case WaitForButtonPress:
                        break;
                case WaitForButtonRelease:
                        break;
                default:
                        break;
        }
        return tempLedValue;
}

int rand(void);

int main(void)
{
        SENSOR_DDR = 0x00;
        LED_DDR    = 0xFF;
        UNUSEDB_DDR= 0x00;
        UNUSEDD_DDR= 0x00;

        // Initialize LEDs to off
        LED_OUTPORT = 0x00;
        unsigned char ButtonValue = 0;
        unsigned char SevenSegValue = 0;

        unsigned char RandNumber = 4;

    while(1)
    {
                // Code to debounce the switch.
                while (ButtonValue != (~SENSOR_INPORT & 0x0F))
                {
                        _delay_ms(50);
                        ButtonValue = (~SENSOR_INPORT & 0x0F);
                }

                SevenSegValue = TckFct_Counter(ButtonValue, RandNumber, SevenSeg
Value);
                LED_OUTPORT = SevenSegValue;
    }
}
```