Program: MultiTasking Digital Padlock

Description: This program controls a digital padlock. The padlock has the same user interface as a standard mechanical padlock: the inputs to the system are a rotary encoder to turn the dial and a switch to attempt to 'open' the lock. The outputs are a two-digit multiplexed LED display and a signal to unlock the lock. The combination that opens the lock is as follows: rotate to the right past zero (0) at least twice, then stop on the first number (19), rotate to the left past the first number (19) exactly once and stop on the second number (33), and, finally, rotate to the right and stop on the third number (5).

The program also makes use of the following available tasks:

DisplayLED(value:in, led:in)	The task takes two input parameters: value which is an integer value between 0 and 9, and led which an integer that is either 0 or 1. The value passed as the first parameter is displayed on the LED indicated by the second parameter with LED 0 being the rightmost LED.
Lock(state:in)	The task takes a single input parameter (state) which is a logical value. If the passed value is true the lock is opened and if it is false it is closed (locked).
OpenSwitch(state:out)	The task takes no input parameters and returns a single output parameter (state) which is a logical value. The returned value is the state of the open switch, true for pressed and false for not pressed.
RotarySwitch(state:out)	The task takes no input parameters and returns a single output parameter (state) which is an integer in the range 0 to 3. The returned value is the current state of the rotary encoder.

Revision History:			
	05 December 07	Glen George	Wrote initial descriptions and available tasks.
	10 December 07	Nadine Dabby	Wrote main and packages.

Events:

RotaryChangeEvent	The event is generated whenever there is a change on the rotary encoder inputs.
Timer1msEvent	The event is generated once per millisecond by the timer.
RightEvent	The event is generated when the user turns the dial right.
LeftEvent	The event is generated when the user turns the dial left.
PushEvent	The event is generated when the knob is pushed.

Package: Main
Description: Main package for digital padlock. It contains the main foreground task.

```
Task:         Padlock
Description:  This program controls a digital padlock. The padlock  has the same user
interface as a standard mechanical padlock: the inputs to the system are a rotary encoder to
turn the dial and a switch to attempt to ‘open’ the lock. The outputs are a two-digit
multiplexed LED display and a signal to unlock the lock. The combination that opens the lock
is as follows: rotate to the right past zero (0) at least twice, then stop on the first
number (19), rotate to the left past the first number (19) exactly once and stop on the
second number (33), and, finally, rotate to the right and stop on the third number (5).

Initial Step: Start
Final Step:   -----


Variables:
   CountDown    integer [0,  100]
                keeps track of how much we need to decrement


Steps:
   Start
      Description: start the lock and display tasks and start the dial by
                   initializing it to 0. waiting for input.
      Action:      PERFORM Lock(FALSE)
                   PERFORM SetLockValue(0)
                   lockNumber <-40
                   RUN  RotateRightHandler
                   RUN  RotateLeftHandler
                   RUN  PushHandler
                   RUN  Increment
                   RUN  Decrement
                   RUN  DisplayHandler

   Idle
     Description:  waiting to reach first 00. decrement countdown variable
     Action:    CountDown--o


   First00
        Description:    Passed Zero for the first time. Reset countdown variable.
        Action:         CountDown <-40


   Pass00
     Description:  waiting to reach second 00. decrement countdown variable
     Action: CountDown--o

   Second00
        Description:    Passed Zero again. Reset countdown variable.
        Action: CountDown <-20

   PassingSecond00
     Description:  waiting to reach firstdigit (19). decrement countdown variable
     Action: CountDown--o
```

FirstDigit
Description: got First digit. Reset countdown variable.
Action: CountDown <- 46

PassingFirstDigit
Description: waiting to reach second digit (13). decrement countdown variable
Action: CountDown--o

SecondDigit
Description: got second digit. Reset countdown variable.
Action: CountDown <- 22

AwaitingThird
Description: waiting to reach third digit (35). decrement countdown variable
Action: CountDown--o

ThirdDigit
Description: got third digit. waiting for push event
Action: --

Unlocked
Description: Code entered correctly , button pushed and lock is unlocked
Action: PERFORM Lock(TRUE)

Step Transitions:

	RightEvent	LeftEvent	PushEvent	Overflow
Start		Start	Start	
Idle	Idle	Start	Start	First00
First00	Pass00	Start	Start	
Pass00	Pass00	Start	Start	Second00
Second00	PassingSecond00	Start	Start	
PassingSecond00	PassingSecond00	Start	Start	FirstDigit
FirstDigit	First00	PassingFirstDigit	Start	
PassingFirstDigit	Start	PassingFirstDigit	Start	SecondDigit
SecondDigit	AwaitingThird	Start	Start	
AwaitingThird	AwaitingThird	Start	Start	ThirdDigit
ThirdDigit	Start	Start	Unlocked	
Unlocked	Start	Start	Start	

Package: Rotary
Description: This package handles the rotary encoder input, watching out for noise (bouncing) and rotation direction. The rotary encoder goes through the values 0, 1, 3, 2, 0, 1, 3, 2, ... when rotating to the right. When rotating to the left it goes through these values in the reverse order. Only the values 1 and 2 have detents and thus the padlock output and direction of rotation change only when the encoder is on those positions.

```
Variables:
   currentState integer [0, 3]
                The current detent state of the rotary encoder
   prevState    integer [0, 3]
                The previous detent state of the encoder
   intState     integer [0, 3]
                The intermediate state of the encoder


Task:        RotateRightHandler
Description:  This task handles the rotate right input.  It periodically
              reads the rotary and debounces it, generating an event when the
              switch is being rotated to the right.  No event is generated when the
              switch is not being rotated to the right.

Steps:

  Start
      Description: Initialize the rotate right variable to false.
      Action:      PERFORM  RotarySwitch(lastState)
                   PERFORM  RotarySwitch(prevState)
                   PERFORM  RotarySwitch(currentState)



  TurnRight
      Description: The dial is being turned right.  Set the flags indicating that we
                   are being turned right and generate an event.
      Action:      prevState <-currentState
                   generate RightEvent


   CheckSwitch
      Description: Check if the switch setting.  Just read the
                   current state of the switch.
      Action:   intState <- currentState
                PERFORM RotarySwitch(currentState)


Step Transitions:

                  RotaryChangeEvent

Start               CheckSwitch
TurnRight           CheckSwitch



                 (currentState = (1 | 2)) &             (currentState = (1 | 2)) &
                  (prevState != currentState) &           (prevState != currentState) &
                    (prevState = 1) & (intState = 3)        (prevState = 2) & (intState = 0)
```

```
CheckSwitch             TurnRight                                TurnRight
```

```
Task:        RotateLeftHandler
Description:  This task handles the rotate left input.  It periodically
              reads the rotary and debounces it, generating an event when the
              switch is being rotated to the left.  No event is generated when the
              switch is not being rotated to the left.

Steps:

   Start
      Description: Initialize the rotate left variable to false.
      Action:      PERFORM RotarySwitch(lastState)
                   PERFORM RotarySwitch(prevState)
                   PERFORM RotarySwitch(currentState)

  TurnLeft
      Description: The dial is being turned left.  Set the flags indicating that we
                   are being turned left and generate an event.
      Action:      prevState <-current
                   generate LeftEvent


   CheckSwitch
      Description: Check if the switch setting.  Just read the
                   current state of the switch.
      Action:      intState <- currentState
                   PERFORM RotarySwitch(currentState)



Step Transitions:

                  RotaryChangeEvent

Start               CheckSwitch
TurnLeft            CheckSwitch



                 (currentState = (1 | 2)) &             (currentState = (1 | 2)) &
                  (prevState != currentState) &           (prevState != currentState) &
                    (prevState = 1) & (intState = 0)        (prevState = 2) & (intState = 3)


CheckSwitch             TurnLeft                                 TurnLeft
```

```
Task: PushHandler
```

Description: This task monitors the pushing of the knob on the padlock.

Variables:
   Knob   logical
                True for pressed, false for not pressed

Initial Step: Start
Final Step: --

Steps:

   Start
        Description: Initialize knob to FALSE
        Action: Knob <- FALSE

   Pressed
        Description: The knob is pushed so generate event
        Action: generate PushEvent

   CheckKnob
        Description: Checks value of switch
        Action: OpenSwitch(Knob)

Step Transitions:

            Timer1msEvent

Start   CheckKnob
Pressed   CheckKnob

            (Knob = true)

CheckKnob   Pressed

Package: LED
Description: This package handles the LED display by implementing a multiplexor.

Variables:
   LockValue   integer [0, 39]
   tensValue   integer [0, 9]
   onesValue   integer

Task: DisplayHandler
Description: This task oscillates between displaying the tens digit and the ones digit on every millisecond event. This way the user doesn't realize each is only being updated every other millisecond.

```
Initial Step: Start
Final Step:   --

Steps:

   Start
        Description:    set digit values to div and mod of lockValue. Display tens digit
        Action:    tensValue <- LockValue / 10
                   onesValue <- LockValue % 10
                   PERFORM  DisplayLED(tensValue, 1)

   DisplayTens
        Description:    reset to new value,  display tens digit.
        Action:    tensValue <- LockValue / 10
                   PERFORM  DisplayLED(tensValue, 1)

   DisplayOnes
        Description:    reset to new value,  Display ones digit.
        Action:    onesValue <- LockValue % 10
                   PERFORM  DisplayLED(onesValue, 0)

Step Transitions:

                Timer1msEvent

Start           DisplayOnes
DisplayTens     DisplayOnes
DisplayOnes     DisplayTens


---


Task: increment
Description:  This task increments the lock value when a LeftEvent occurs

Initial Step: Start
Final Step:   --

Steps:

   Start
        Description:  Wait for RightEvent
        Action: --

   GoUp
        Description: Increment LockValue
        Action: LockValue++o

   Past39
        Description: Set Lock Value to 0 (because we passed 39).
        Action: LockValue <- 0

Step Transitions:
```

	LeftEvent	Overflow
Start	GoUp	
GoUp	Start	Past39
Past39	Start	

Task: decrement
Description: This task decrements the lock value when a RightEvent occurs

Initial Step: Start
Final Step: --

Steps:

Start
Description: Wait for LeftEvent
Action: --

GoDown
Description: Decrement LockValue
Action: LockValue--o

Past0
Description: Set Lock Value to 39 (because we passed 0).
Action: LockValue <- 39

Step Transitions:

	RightEvent	Overflow
Start	GoDown	
GoDown	Start	Past0
Past0	Start	

Task: setLockValue
Description: This is a mutator task for the lock value. It sets the value of the dial to the passed integer parameter.
Parameters: var:in integer [0, 39]
the value on the dial

Initial Step: Start
Final Step: Start

Variables:
None

```
Steps:

   Start
      Description: Set the value of the LockValue package variable to the
                   input parameter.
      Action:      LockValue <- var



---

Task:         getLockValue
Description:  This is an accessor task for the lock value in the LED package.  It sets the
              passed parameter to the integer value of the dial.
Parameters:   flag:out  integer [0, 39]
                        the value on the dial

Initial Step: Start
Final Step:   Start

Variables:
   None

Steps:

   Start
      Description: Get the value of the LockValue package variable into the
                   parameter.
      Action:      flag <- LockValue
```