```
                                                    Justin Shuck
                                                CS333 - Fall 2014
                                                  Proj 3 - Main.c
######################################################################

code Main

  -- OS Class: Project 3
  --
  -- Justin Shuck
  --
  -- Due: 10/21/2014 2:00 PM

-----------------------------  Main  ---------------------------------

  function main ()
      InitializeScheduler()
      -- testSleepingBarberPart1()   -- Tests part 1 of Proj 3
      testGameParlorPart2()       -- Tests part 2 of Proj 3
    endFunction



const
  CHAIRS = 5
  CUST_COUNT = 20
  BARB_COUNT = 1

var
  customers: Semaphore = new Semaphore
  barbers:   Semaphore = new Semaphore
  mutexLock:     Semaphore = new Semaphore
  waitCounter:   int = 0
  threads: array[50] of Thread = new array of Thread {50 of new
Thread}

----------------------------------------------------------------------
---------
-------------------------  PART1: Sleeping Barber  -------------------
---------
----------------------------------------------------------------------
---------

function testSleepingBarberPart1()
    var
      index: int
      total: int
```

```
customers.Init(0)
barbers.Init(0)
mutexLock.Init(1)
total = BARB_COUNT+CUST_COUNT

--------------------------------------------------------------
-- COMMENTED CODE: Useful for testing large numbers of
-- Barbers/Customers. However I couldn't implement a
-- concat of a string "Barber" with the index. Adding a static
-- test below to demenstrate meaningful output usage.
--------------------------------------------------------------

-- for index = 0 to BARB_COUNT
--    thread[index].Init("Barber ")
--  endFor

-- for index = BARB_COUNT to CUST_COUNT
--    thread[index].Init("Customer ")
--  endFor

-- for index = 0 to BARB_COUNT
--    thread[index].Fork(barber,50)
--  endFor

-- for index = BARB_COUNT to CUST_COUNT
--  thread[index].Fork(customer, index * 50)
--endFor

print("-- PART 1: BEGIN TESTING -- \n")
threads[0].Init("Barber #1")
threads[1].Init("Customer #1")
threads[2].Init("Customer #2")
threads[3].Init("Customer #3")
threads[4].Init("Customer #4")
threads[5].Init("Customer #5")
threads[6].Init("Customer #6")
threads[7].Init("Customer #7")
threads[8].Init("Customer #8")
threads[9].Init("Customer #9")
threads[10].Init("Customer #10")
threads[11].Init("Customer #11")
threads[12].Init("Customer #12")
threads[13].Init("Customer #13")
threads[14].Init("Customer #14")
threads[15].Init("Customer #15")
threads[16].Init("Customer #16")
threads[17].Init("Customer #17")
threads[18].Init("Customer #18")
```

```
    threads[19].Init("Customer #19")
    threads[20].Init("Customer #20")

    threads[0].Fork(barber, 50)

    -- Iterate over the customers
    total = CUST_COUNT + BARB_COUNT - 1
    for index = BARB_COUNT to 20
      threads[index].Fork(customer, index * 50)
    endFor
    ThreadFinish()
    print("-- PART 1: END TESTING -- \n")
endFunction



---------------------------------------------------------------------
-- BARBER

---------------------------------------------------------------------
function barber(timeToWait: int)
    print("New Barber: ")
    print(currentThread.name)
    print("\n\n")

    wait(timeToWait)

    while (true)
        customers.Down()
        mutexLock.Down()
        waitCounter = waitCounter - 1
        barbers.Up()
        mutexLock.Up()
        cut_hair()
    endWhile
  endFunction


---------------------------------------------------------------------
-- CUSTOMER
---------------------------------------------------------------------
function customer(timeToWait: int)
    wait(timeToWait)    -- Wait a specific amount of time before a
'new' customer arrives
    print("New Customer Has Arrived: ")
    print(currentThread.name)
    print("\n")
    mutexLock.Down()
    -- If there is no one waiting, wake up the barber and get
haircut/take a seat
    if (waitCounter < CHAIRS)
        waitCounter = waitCounter + 1
        customers.Up()
        mutexLock.Up()
        barbers.Down()
        get_haircut()
    -- The shop is full (NO seats)
    else
        mutexLock.Up()
        print("--> SHOP FULL: ")
        print(currentThread.name)
        print(" will now leave the store.\n\n")
    endIf
  endFunction

---------------------------------------------------------------------
-- BUSY LOOP: Dummy function that just waits x-time
---------------------------------------------------------------------
function wait(timeToWait: int)
    var index: int
    for index = 1 to timeToWait
      endFor
  endFunction

---------------------------------------------------------------------
-- Print Helper Function that shows that
-- someone is getting their haircut
---------------------------------------------------------------------
function get_haircut()
    print("----> ")
    print(currentThread.name)
    print(" is getting_haircut! \n")
  endFunction

---------------------------------------------------------------------
-- Print Helper Function that shows that a
-- barber is cutting hair
---------------------------------------------------------------------
function cut_hair()
    print("-------------> ")
    print(currentThread.name)
    print(" is cutting_hair! \n")
    wait(100)
    print("------------->")
    print(currentThread.name)
    print(" finished cutting_hair! \n")
  endFunction

-----------------------------------------------------------------------
---------
-------------------------  PART2: Game Parlor  -----------------------
---------
-----------------------------------------------------------------------
---------
const
  GROUPS = 8        -- Total available groups
  DICE = 5          -- Total available dice
  GAMES_PLAYED = 5  -- Total games played
  WAIT_COUNTER = 50 -- Mock time for waiting

var
  gameParlor: GameParlor
  thread: array[GROUPS] of Thread = new array of Thread {GROUPS of new
Thread}

function testGameParlorPart2()
    gameParlor = new GameParlor
    gameParlor.Init()

    print("-- PART 1: BEGIN TESTING -- \n")
    thread[0].Init("A - Backgammon")
    thread[0].Fork(mockGame, 4)
    thread[1].Init("B - Backgammon")
    thread[1].Fork(mockGame, 4)
    thread[2].Init("C - Risk")
    thread[2].Fork(mockGame, 5)
    thread[3].Init("D - Risk")
    thread[3].Fork(mockGame, 5)
    thread[4].Init("E - Monopoly")
    thread[4].Fork(mockGame, 2)
    thread[5].Init("F - Monopoly")
    thread[5].Fork(mockGame, 2)
    thread[6].Init("G - Pictionary")
    thread[6].Fork(mockGame, 1)
    thread[7].Init("H - Pictionary")
    thread[7].Fork(mockGame, 1)
    ThreadFinish()
    print("-- PART 2: END TESTING -- \n")

  endFunction

---------------------------------------
-- Iterates over the total GAMES_PLAYED
-- and uses a method similar to Part 1's
-- 'wait' method where the currentThread
-- yields until WAIT_COUNTER is complete
----------------------------------------
function mockGame(dice: int)
    var
      index1: int
      index2: int

    for index1 = 1 to GAMES_PLAYED
        gameParlor.getDice(dice)
        for index2 = 1 to WAIT_COUNTER
            currentThread.Yield()
          endFor
        gameParlor.releaseDice(dice)
      endFor
  endFunction

behavior GameParlor
    -----------------------------------
    -- Init method, Initializes the variables
    -- that we're going to use by either
    -- calling an Init or by setting its
    -- value
    -----------------------------------
    method Init()
        numDiceLeft = DICE        -- Set Dice
        numWaitingGroups = 0      -- Set the counter for the groups
waiting

        monitoringLock = new Mutex
        monitoringLock.Init()

        firstInLine = new Condition
        firstInLine.Init()

        restOfLine = new Condition
        restOfLine.Init()
      endMethod

    -----------------------------------
    -- Print method: Generic use, passes
    -- in a string and the number of dice
    -- remaining for the particular action
    -----------------------------------
    method print(printString: String, num: int)
        print("THREAD[")
        print(currentThread.name)
        print("] ")
        print(printString)
        print(" using ")
        printInt(num)
        print(" dice! \n ---Now there are ")
        printInt(numDiceLeft)
        print(" dice left...\n\n")
      endMethod

    -----------------------------------
    -- Get Dice method
    -----------------------------------
    method getDice(diceNeeded: int)
        monitoringLock.Lock()
        self.print(" NEEDS ", diceNeeded)
        numWaitingGroups = numWaitingGroups + 1

        -- if there are more than one person in line,
        -- then have the rest of the line wait
        if (numWaitingGroups > 1)
            restOfLine.Wait(&monitoringLock)
          endIf

        -- Wait until the appropriate number of dice
        -- are available
        while (numDiceLeft < diceNeeded)
            firstInLine.Wait(&monitoringLock)
          endWhile

        -- At this point they can get dice. We need
        -- to decrement the dice counter and the number
        -- of groups waiting.
        numDiceLeft = numDiceLeft - diceNeeded
        numWaitingGroups = numWaitingGroups - 1
        restOfLine.Signal(&monitoringLock)
        self.print("PROCEEDS", diceNeeded)
        monitoringLock.Unlock()
      endMethod

    -----------------------------------
    -- Release Dice method
    -----------------------------------
    method releaseDice(diceReturned: int)
        monitoringLock.Lock()
        numDiceLeft = numDiceLeft + diceReturned

        self.print("DICE ADDED BACK", diceReturned)

        firstInLine.Signal(&monitoringLock)
```

```
        monitoringLock.Unlock()
      endMethod
  endBehavior
endCode
```