Assignment 2 tips

- data structures in Gate.mod

```
GATE = POINTER TO RECORD
                     Head: GateDesc ;
                     name: ARRAY [0..MaxCharName]
                           OF CHAR ;
                   END ;
```

- the `Head` field contains a list of all processes performing `SleepOn` on this `GATE`

- the `name` field is used for debugging

InitGate

- to initialize a `GATE` we need to

- ```
  create a new GATE, g
  initialize the Head to NIL
  and initialize the name field
  finally need to return the new GATE, g
  ```
  ```

GateDesc

```
GateDesc = POINTER TO RECORD
                         Q    : Queue ;
                         d    : DESCRIPTOR ;
                         e    : EVENT ;
                         time: CARDINAL ;
                       END ;
```

- each item on the `GATE` list is a `GateDesc`
- `Q:` `Queue` used to contain further `GateDesc` items
- `d:` `DESCRIPTOR` the processes actually performing `SleepOn`
- `e:` `EVENT` the `TimerHandler.mod` being used

GateDesc

- `time: CARDINAL` the time the process, `d`, must `SleepOn`

SleepOn

- examine function `SleepOn` to aid understanding of data structures
-

```
PROCEDURE SleepOn (g: GATE;
                   millisecs: CARDINAL)
                   : BOOLEAN ;
VAR
   ToOldState: OnOrOff ;
   WasWoken  : BOOLEAN ;
BEGIN
   ToOldState := TurnInterrupts(Off) ;
   IF g=NIL
   THEN
      WasWoken := FALSE
   ELSE
      WasWoken := PerformSleepOn(g, millisecs)
   END ;
   ToOldState := TurnInterrupts(ToOldState) ;
   RETURN( WasWoken )
END SleepOn ;
```

PerformSleepOn

```
PROCEDURE PerformSleepOn (g: GATE;
                          millisecs: CARDINAL)
                          : BOOLEAN ;
VAR
   gd        : GateDesc ;
   Cancelled: BOOLEAN ;
BEGIN
   Wait(Mutex) ;      (* ensure only one process *)
   NEW(gd) ;          (* can add entities to Q   *)
   WITH gd^ DO
      AddTo(g^.Head, gd) ;   (* do the add to Q *)
      Signal(Mutex) ; (* now allow someone else *)

      (* need to fill in fields of gd *)

      Wait(Mutex) ; (* ensure only one process *)
      e := NIL ;    (* can manipulate the Q    *)
      SubFrom(g^.Head, gd)   (* perform sub Q  *)
   END ;
   DISPOSE(gd) ;    (* tidy up the GateDesc    *)
   Signal(Mutex) ;  (* and allow other procs   *)
   RETURN( Cancelled )   (* were we WokenUp?   *)
END PerformSleepOn ;
```

Middle of `PerformSleepOn`

- ```
  need to fill in the fields of gd,
  d needs to be set to the current process
  e needs to be set to the EVENT used to
    cause the sleep
  time needs to be set to amount of millisecs

  local variable Cancelled must be set if WakeUp
  was issued to the Gate
  ```

- examine `Executive.def` to find out how to obtain the current process

- examine `TimerHandler.def` to find out how `EVENT`s are to be created and used
  ```

TimerHandler and `EVENTs`

-
  TimerHandler provides the with the ability to make processes sleep for a predetermined time
  - processes can also be woken up

- to create an event use: `e := ArmEvent(` *time in ticks* `)`

- after *all* fields in `gd` have been filled in perform: `Cancelled := WaitOn(e)`

- causes calling process (CurrentProcess) to sleep for a time or until it is `Cancelled`

- finally note that *time in ticks* is not the same as millisecs, thus you will

need to convert

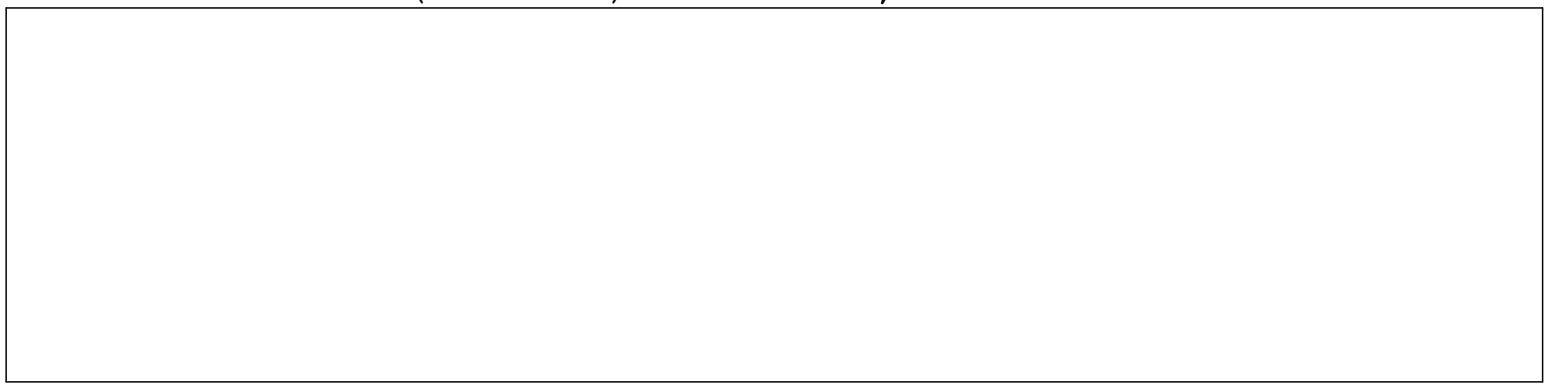
TimerHandler.def

```
(*
   ArmEvent - initializes an event, e,
              to occur at time, t.
              The time, t, is measured in ticks.
              The event is NOT placed onto
              the event queue.
*)

PROCEDURE ArmEvent (t: CARDINAL) : EVENT ;


(*
   WaitOn - places event, e, onto the event
            queue and then the calling
            process suspends. It is resumed
            up by either the event
            expiring or the event, e, being
            cancelled.
            TRUE is returned if the event
                 was cancelled
            FALSE is returned if the event
                  expires.
*)
```

```
PROCEDURE WaitOn (e: EVENT) : BOOLEAN ;
```



Cancel

```
(*
   Cancel - cancels the event, e, on the
            event queue and makes
            the appropriate process
            runnable again.
            TRUE  is returned if the
                  event was cancelled and
            FALSE is returned is the event
                  was not found or
                  no process was waiting
                  on this event.
*)

PROCEDURE Cancel (e: EVENT) : BOOLEAN ;
```

Testing your Gate.mod or Gate.c

- at present the directory ass5 is set to compile Gate.mod, csn.mod and speed.mod
  - speed.mod is a final message passing speed test

- it might not be appropriate for initial testing of your assignment
  - there are other test programs in this directory
    - ex1.mod and ex2.mod

- to use these test programs, edit makefile
  - alter `MAIN = speed # MAIN = ex1 # MAIN = ex2`
  - to

```
# MAIN = speed
MAIN = ex1
# MAIN = ex2
```

  - to use ex1.mod etc

Data structure of Gate.mod

- draw a data structure diagram showing 2 processes performing SleepOn on a single GATE