baby.clp

```
(defrule change-baby-if-wet
    "if bay is wet, change nappy"
    ?w <- (baby-is-wet)
    =>
    (change-baby)
    (retract ?w)
)

(deffunction change-baby ()
  (printout t "baby is now being changed" crlf))
```

---

rules.clp

```
; Rule r1 looks like it has no antecedent, but in fact it has an
implicit one which is initial-fact. r1 won't fire until
initial-fact is asserrted.

(defrule r1
     =>
     (assert (b))
)

(defrule r2
    ?f <- (b)
    =>
    (retract ?f)
    (assert (c))
)

(defrule r2a
     (b)
     =>
     (assert (c2))
)

(defrule r3
    (c)
    =>
    (assert (d))
    (do-something)
)
```

```
(deffunction do-something ()
    (printout t "I fired 3 rules" crlf))
```

---

faucet.clp

```
(defrule turn-water-on
    (faucet open)
    =>
    (assert (water flowing)))

; here not means the absence of a fact, which is
; different a logical not in an if-statement.

(defrule turn-water-off
    (initial-fact)
    (not (faucet open))
    ?water <- (water flowing)
    =>
    (retract ?water))

(reset)
```

---

traffic.clp

```
(reset)

(assert (car moving))

(defrule traffic-stop-rule
      (traffic-light red)
      ?c <- (car moving)
      =>
      (printout t "Stop car as light has changed to red" crlf)
      (retract ?c)
      (assert (car stopped))
)
```

---

avg.clp

```
; LAB 3 - mentioned | code which he gave and then got the soln of
```

```
(clear)

(bind ?l (create$ 1 2 3 4 5))
(bind ?pl (create$ milk 2.40 eggs 1.50 bread 1 specialK 4 sugar
2.30 tea 2 coffee 3))

(deffunction avg (?l)
    (bind ?n (length$ ?l))
    (bind ?i 0)
    (bind ?sum 0)
    (while (< ?i ?n) do
        (bind ?i (+ ?i 1))
        (bind ?sum (+ ?sum
                          (nth$ ?i ?l))))
    (return (/ ?sum ?n)))


(deffunction avg2 (?l)
    (bind ?n (length$ ?l))
    (bind ?sum 0)
    (foreach ?i ?l
        (bind ?sum (+ ?sum ?i)))
    (return (/ ?sum ?n)))
```

---

solution lab 2

```
; LAB 2

(clear)


(bind ?l (create$ 1 2 3 4 5))
(bind ?pl (create$ milk 2.40 eggs 1.50 bread 1 specialK 4 sugar
2.30 tea 2 coffee 3))

;(deffunction find-price (?item ?price-list)

; for a list of numbers
(deffunction avg (?l)
    (bind ?n (length$ ?l))
    (bind ?i 0)
    (bind ?sum 0)
    (while (< ?i ?n) do
        (bind ?i (+ ?i 1))
        (bind ?sum (+ ?sum
                      (nth$ ?i ?l))))
    (return (/ ?sum ?n)))

 ; for a list of numbers
(deffunction avg2 (?l)
    (bind ?n (length$ ?l))
    (bind ?sum 0)
    (foreach ?i ?l
        (bind ?sum (+ ?sum ?i)))
    (return (/ ?sum ?n)))

; for a list of numbers
(deffunction list-max (?l)
    (bind ?m (nth 1 ?l))
    (foreach ?i ?l
        (if (> ?i ?m) then
          (bind ?m ?i))
    (return ?m))
)

; for total cost of a grocery list
(deffunction  total-cost(?l)
    (bind ?n (length$ ?l))
    (bind ?i 0)
    (bind ?cost 0)
    (while (< ?i ?n) do
        (bind ?i (+ ?i 2))
        (bind ?cost (+ ?cost
                      (nth$ ?i ?l))))
    (return ?cost)
 )
```

---

person.clp

```
(clear)

(deftemplate person "People in actuarial database"
    (slot age)
    (slot name)
    (slot gender))
```

```
(assert (person (name "Bob Smith") (age 34) (gender Male)))
(assert (person (gender Male) (name "Tom Smith") (age 32) (gender
Male)))
(assert (person (name "Mary Smith") (age 34) (gender Female)))
(assert (person (gender Female)))


(defrule male-ages
    (person (name ?n) (age ?a) (gender Male) )
    =>
    (printout t ?n " is " ?a " years old " crlf))
```

---

person1.clp - soln

```
(clear)
(deftemplate person "People in actuarial database"
    (slot age)
    (slot name)
    (slot gender))

(assert (person (name "Bob Smith") (age 34) (gender Male)))
(assert (person (age 30) (name "Michael Smith") ))
(assert (person (gender Male) (name "Tom Smith") (age 32) ))
(assert (person (name "Mary Smith") (age 34) (gender Female)))
(assert (person (gender Female)))

(defglobal ?*count* = 0)

(defrule count-males-rule
    (person (gender Male) )
    =>
    (bind ?*count* (+ ?*count* 1)))

(deffunction show-male-count ()
    (printout t "The male count is " ?*count* crlf))

(run)
(show-male-count)
```

---

person2.clp - soln

```
(clear)
```

```
(deftemplate person "People in actuarial database"
    (slot age)
    (slot name)
    (slot gender))

(assert (person (name "Bob Smith") (age 34) (gender Male)))
(assert (person (gender Male) (name "Tom Smith") (age 32) ))
(assert (person (name "Mary Smith") (age 34) (gender Female)))
(assert (person (gender Female)))
(assert (person (name "Billy Bob") (gender Male)))
(assert (person (gender Male) (name "Jim") (age 22)))
(assert (person (gender Female) (name "Sue") (age 20)))

(defglobal ?*count* = 0 ?*sum* = 0)

(defrule males-avg-rule
    (person (gender Male) (age ?a))
    =>
   (bind ?*count* (+ ?*count* 1))
   (bind ?*sum* (+ ?*sum* ?a )))

(deffunction show-avg ()
    (bind ?avg  (/ ?*sum* ?*count*))
    (printout t "The average male age is " ?avg crlf))

(run)
(show-avg)
```

---

person3.clp - soln

```
(clear)

(deftemplate person "People in actuarial database"
    (slot age (default FALSE))
    (slot name)
    (slot gender))

(defglobal ?*count* = 0 ?*sum* = 0)

(assert (person (name "Bob Smith") (age 34) (gender Male)))
(assert (person (gender Male) (name "Tom Smith") (age 32) ))
(assert (person (name "Mary Smith") (age 34) (gender Female)))
(assert (person (gender Female)))
(assert (person (name "Billy Bob") (gender Male)))
```

```
(assert (person (gender Male) (name "Jim") (age 22)))
(assert (person (gender Female) (name "Sue") (age 20)))

(defrule male-ages-rule
   (person (name ?n) (age ?a) (gender Male))
    =>
    (if ?a  then
        (printout t ?n " is " ?a " years old " crlf)
    else
        (printout t ?n " is of unknown age"  crlf)))

(defrule males-avg-rule
    (person (gender Male) (age ?a))
    =>
    (if ?a then
        (bind ?*count* (+ ?*count* 1))
        (bind ?*sum* (+ ?*sum* ?a ))))

(deffunction show-avg ()
    (bind ?avg  (/ ?*sum* ?*count*))
    (printout t "The average male age is " ?avg crlf))

(run)
(show-avg )
```

---

person4.clp - soln

```
(clear)

(deftemplate person "People in actuarial database"
    (slot age)  (slot name)  ()slot gender))

(assert (person (name "sue") (gender Female) (age 7)))
(assert (person (name "Bob Smith") (age 34) (gender Male)))
(assert (person (name "Tom Smith") (age 32) (gender Male)))
(assert (person (name "Mary Smith") (age 36) (gender Female)))

(defglobal ?*min* = 200  ?*min-name* = "")

(defrule youngest-rule-1
    (person (name ?n) (age ?a ))
    =>
    (if (< ?a ?*min*) then
        (printout t "New youngest person " ?n " age " ?a " prev age
was " ?*m* crlf)
        (bind ?*min* ?a)
        (bind ?*min-name* ?n)))

(defrule youngest-rule-2
    (person (name ?n) (age ?a & :(< ?a ?*min*)))
    =>
    (printout t "New youngest person " ?n " age " ?a " prev age
was " ?*m* crlf)
    (bind ?*min* ?a)
    (bind ?*min-name* ?n))

(deftemplate youngest (slot name) (slot age))
(assert (youngest  (age 200))

(defrule youngest-rule-3
    ?y <- (youngest (age ?b))
    (person (name ?n) (age ?a & :(< ?a ?b)))
    =>
    (printout t "New youngest person " ?n " age " ?a " prev age
was " ?*m* crlf)
    (retract ?y)
    (assert (youngest (age ?a) (name ?n)))
)

(defrule youngest-rule-4
    ?y <- (youngest (age ?b))
    (person (name ?n) (age ?a & :(< ?a ?b)))
    =>
    (printout t "New youngest person " ?n " age " ?a " prev age
was " ?*m* crlf)

    (modify ?y (age ?a) (name ?n))
)
```

---

oldest1.clp - lab test

```
(clear)

(deftemplate person "People in actuarial database"
    (slot age (default -1))
    (slot name )
    (slot gender))
```

```
(deftemplate oldest-male (slot name) (slot age))

(assert (person (gender Male) (name "Mitt Romney") (age 61) ))
(assert (person (name "Bob Smith") (age 34) (gender Male)))
(assert (person (gender Male) (name "Tom Smith") (age 32) ))
(assert (person (name "Mary Smith") (age 34) (gender Female)))
(assert (person  (name "George Bush") (gender Male)))

(assert (person (gender Female)))

(defrule oldest-male-rule
    ;student to complete this rule
)

(defrule show-oldest-male
    ?f1 <- (done)
    ; complete this rule which fires when other rules has
finished.
)

(deffunction find-oldest-male ()
    ; student to complete this function
    ; involves asserts, runs and retracts
    (assert (oldest-male (age 0))

)
```