NOMBRE: David Israel Leon FAMILIARES

(deffacts inicio

(hombre David)

(mujer Lili)

(hombre Jose)

(mujer Maria)

(hombre Luis)

(mujer Demi)

(hombre Mario)

(mujer Fanny)

(hombre Roberto)

(mujer Luisa)

(progenitor-de David Demi)

(progenitor-de Luisa David)

(marido-de David Lili)

(progenitor-de Luisa Jose)

)

```
(defrule progenitor
  (padre-de ?padre ?hijo)
  (marido-de ?padre ?madre)
  =>
  (assert (progenitor-de ?madre ?hijo)))

(defrule esposa
  (marido-de ?hombre ?mujer)
  =>
  (assert (esposa-de ?mujer ?hombre))
  (printout t ?mujer " es la esposa de " ?hombre crlf))

(defrule padre
  (progenitor-de ?padre ?hijo)
  (hombre ?padre)
  =>
  (assert (padre-de ?padre ?hijo))
  (printout t ?padre " es padre de " ?hijo crlf))

(defrule madre
  (progenitor-de ?madre ?hijo)
  (mujer ?madre)
```

```
  =>
  (assert (madre-de ?madre ?hijo))
  (printout t ?madre " es madre de " ?hijo crlf))


; Primos -----

(defrule prima
  (tios ?tia ?sobrina)
  (progenitor-de ?tia ?prima)
  (mujer ?prima)
  =>
  (assert (prima-de ?sobrina ?prima))
  (printout t ?prima " es prima de " ?sobrina crlf))

(defrule primo
  (tios ?tios ?sobrino)
  (progenitor-de ?tia ?primo)
  (hombre ?primo)
  =>
  (assert (primo-de ?sobrino ?primo))
  (printout t ?primo " es primo de " ?sobrino crlf))
```

```
(defrule prima
   (tios ?tia ?sobrina)
   (progenitor-de ?tia ?prima)
   (mujer ?prima)
   =>
   (assert (prima-de ?sobrina ?prima))
   (printout t ?prima " es prima de " ?sobrina crlf))

(defrule primo
   (tios ?tios ?sobrino)
   (progenitor-de ?tia ?primo)
   (hombre ?primo)
   =>
   (assert (primo-de ?sobrino ?primo))
   (printout t ?primo " es primo de " ?sobrino crlf))

 ; Abuelos -----
(defrule abuelos
   (progenitor-de ?padre ?hijo)
   (progenitor-de ?hijo ?nieto)
   =>
   (assert (abuelos-de ?padre ?nieto)))
```

```
(defrule abuelo
  (abuelos-de ?padre ?nieto)
  (hombre ?padre)
  =>
  (assert (abuelo-de ?padre ?nieto))
  (printout t ?padre " es el abuelo de " ?nieto crlf))

(defrule abuela
  (abuelos-de ?madre ?nieto)
  (mujer ?madre)
  =>
  (assert (abuelo-de ?madre ?nieto))
  (printout t ?madre " es el abuela de " ?nieto crlf))
; Hermanos  -----

(defrule hermanos-padre
  (padre-de ?padre ?hijo1)
  (padre-de ?padre ?hijo2)
  (test (neq ?hijo1 ?hijo2))
  =>
  (assert (hermanos ?hijo1 ?hijo2)))

(defrule hermanos-madre
```

```
   (madre-de ?madre ?hijo1)
   (madre-de ?madre ?hijo2)
   (test (neq ?hijo1 ?hijo2))
   =>
   (assert (hermanos ?hijo1 ?hijo2)))

(defrule hermano
   (hermanos ?hijo1 ?hijo2)
   (hombre ?hijo1)
   =>
   (assert (hermano-de ?hijo1 ?hijo2))
   (printout t ?hijo1 " es hermano de " ?hijo2 crlf))

(defrule hermana
   (hermanos ?hijo1 ?hijo2)
   (mujer ?hijo1)
   =>
   (assert (hermana-de ?hijo1 ?hijo2))
   (printout t ?hijo1 " es hermana de " ?hijo2 crlf))


; Tios -----
```

```
(defrule tios
  (progenitor-de ?padre ?hijo)
  (hermanos ?padre ?hermano)
  =>
  (assert (tios ?hermano ?hijo)))

(defrule tio
  (tios ?tio ?sobrino)
  (hombre ?tio)
  =>
  (assert (tio ?tio ?sobrino))
  (printout t ?tio " es tio de " ?sobrino crlf))

(defrule tia
  (tios ?tia ?sobrino)
  (mujer ?tia)
  =>
  (assert (tia-de ?tia ?sobrino))
  (printout t ?tia " es tia de " ?sobrino crlf))

(defrule sobrino
  (tios ?tios ?sobrino)
  (hombre ?sobrino)
```

```
  =>
  (assert (sobrino-de ?sobrino ?tios))
  (printout t ?sobrino " es sobrino de " ?tios crlf))

(defrule sobrina
  (tios ?tios ?sobrina)
  (mujer ?sobrina)
  =>
  (assert (sobrina-de ?sobrina ?tios))
  (printout t ?sobrina " es sobrina de " ?tios crlf))
```