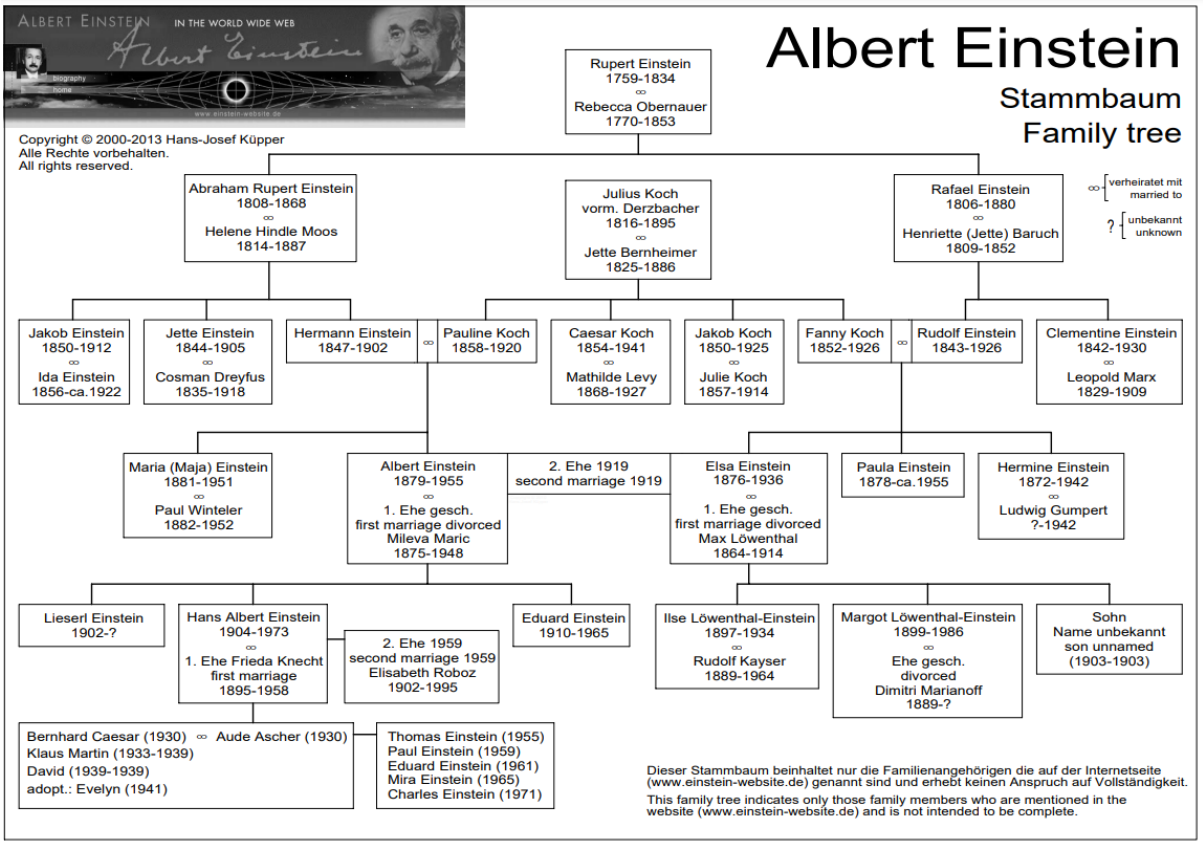
% En amarillo se lista la posible solución

Ejercicio 1

Exprese el siguiente árbol genealógico como una base de hechos a partir de los predicados:

a. persona(NombrePersona,AñoNacimiento,AñoFallecimiento).
b. padres(NombreProgenitor1,NombreProgenitor2,Nombre_Descendiente).

Donde los nombres (NombreX) se expresan con un String, mientras que los Años se expresan cómo números enteros positivos.



2. Implementar una regla que permita determinar si dos personas (a partir de su nombre) son hermano/as.
3. Implementar una regla que permite determinar si dos personas (a partir de su nombre) son hermanastro/as.
4. Implementar una regla que permite determinar si dos personas (a partir de su nombre) son primo/as.
5. Implementar el predicado menor(NombrePersona1,NombrePersona2) que permite determinar si NombrePersona1 es menor que NombrePersona2,
6. Documentar adecuadamente todo el programa.

%Por simplicidad sólo se listan algunos casos

persona(“Rupert Einstein”,1759,1834).

persona(“Rebeca Obernauer”,1770,1853).

……
persona(“Albert Einstein”,1879,1955).

padres(“Rupert Einstein”,“Rebeca Obernauer”,”Abraham Rupert Einstein”).
padres(“Rupert Einstein”,“Rebeca Obernauer”,”Rafael Einstein”).
padres(“Hermann Einstein”,“Pauline Koch”,”Albert Einstein”).
….

1. Implementar una regla que permita determinar si dos personas (a partir de su nombre) son hermano/as.

%Esta implementación asume que se respeta el orden en que figura listados los padres en el %árbol. Si no se asume esto, se pueden considerar los casos alternando los progenitores.

```
hermanos(P1,P2):-
                padres(Pg1,Pg2,P1),
                padres(Pg1,Pg2,P2).
```

2. Implementar una regla que permite determinar si dos personas (a partir de su nombre) son hermanastro/as.

%Esta implementación asume que se respeta el orden en que figura listados los padres en el %árbol. Si no se asume esto, se pueden considerar los casos alternando los progenitores.

```
hermanastros(P1,P2):-(padres(Pg1,Pg2,P1),
                padres(Pg1,Pg3,P2),
                Pg2\=Pg3);

                 (padres(Pg1,Pg2,P1),
                padres(Pg3,Pg2,P2),
                Pg2\=Pg3).
```

3. Implementar una regla que permite determinar si dos personas (a partir de su nombre) son primo/as.

%Esta implementación asume que se respeta el orden en que figura listados los padres en el %árbol. Si no se asume esto, se pueden considerar los casos alternando los progenitores.

```
primos(P1,P2):-padres(Pg1,Pg2,P1),
                padres(Pg3,Pg4,P2),
                Pg1\=Pg3,
                Pg2\=Pg4,
```

(hermanos(Pg1,Pg3);

hermanos(Pg1,Pg4);

hermanos(Pg2,Pg3);

hermanos(Pg2,Pg4))

4. Implementar el predicado menor(NombrePersona1,NombrePersona2) que permite determinar si NombrePersona1 es menor que NombrePersona2,

%Una manera a través de edad (considerando el tiempo total de vida de la persona)

```
menor(P1,P2):-persona(P1,FN1,FM1),
              persona(P2,FN2,FM2),P1\=P2,
              Edad1 is
              FM1-FN1,

              Edad2 is FM2 - FN2, Edad1<Edad2.
```

%otra manera, solo considerando fecha de nacimiento

```
menor(P1,P2):-persona(P1,FN1,_),persona(P2,FN2,_),P1\=P2,FN1>FN2
```

5. Implementar el predicado menor(NombrePersona1,NombrePersona2) que permite determinar si NombrePersona1 es menor que NombrePersona2,

%Una manera a través de edad (considerando el tiempo total de vida de la persona)

```
menor(P1,P2):-persona(P1,FN1,FM1),
              persona(P2,FN2,FM2),P1\=P2,
              Edad1 is
              FM1-FN1,

              Edad2 is FM2 - FN2, Edad1<Edad2.
```

%otra manera, solo considerando fecha de nacimiento

```
menor(P1,P2):-persona(P1,FN1,_),persona(P2,FN2,_),P1\=P2,FN1>FN2.
```

Documentar adecuadamente todo el programa.

%Dominios
%Nombre: String
%AñoNacimiento: Entero Positivo
%AñoMuerte: Entero Positivo

%Predicados
%hermanos(Nombre,Nombre)
%persona(Nombre,AñoNacimiento,AñoMuerte)
%padres(Nombre,Nombre)
%hermanastros(Nombre,Nombre)
%menor(Nombre,Nombre)
%primos(Nombre,Nombre)

%Metas
%primarias
%hermanastros
%menor
%primos

%secundarias
%hermanos
%persona
%padres
.

Ejercicio 2

1. Documente de manera completa el siguiente programa en Prolog que expresa la relación de gustos entre una persona y un determinado elemento (comida, juego, actividad, etc.). Nótese que el siguiente listado solo ejemplifica algunos casos, por lo que podrían haber innumerables hechos más como los presentados a continuación:

```
%domains
%Nombre: atom-symbol
%Gusto: atom-symbol

%predicates
%like(Nombre,Gusto).

%goals
%Señalar los gustos de una persona (like)

%clauses
%hechos
like(pedro,pizza).
like(maria,pizza).
like(diego,deporte).
like(jose,bailar).
like(javiera,sushi).
like(pedro,sushi).
like(maria,bailar).
like(maria,poker).
like(pedro,astronomía).
like(jose,poker).
```

2. A partir del código anterior, define, documenta e implementa las cláusulas para el predicado *match(Persona1,Persona2)*, el cual se considera cierto si Persona1 y Persona2 (siendo estas diferentes) comparten un mismo gusto.

```
%domains
%Nombre1, Nombre2: atom-symbol

%predicates
%match(Nombre1,Nombre2)

%goals
%indicar match entre personas basado en gustos (match)

%clauses
%rules
match(Nombre1,Nombre2):- Nombre1 \= Nombre2,
like(Nombre1,G),like(Nombre2,G).
```

3. Asumiendo que los nombres de las personas que figuran en los hechos de la pregunta 1 actúan como identificador único (ej: los hechos relativos a maria se refieren a la misma persona), convierta el programa de la **Pregunta 1** expresando el predicado *like* como una relación binaria (aridad=2) que vincula un átomo (nombre de la persona) con una lista de todos los gustos de la persona (i.e., like(Persona,ListaGustos)).

```
like(pedro,[pizza,astronomia,sushi]).
like(maria,[pizza,bailar,poker]).
like(diego,[deporte]).
like(jose,[bailar,poker]).
like(javiera,[sushi]).
```

4. Implemente las cláusulas para el predicado *match* descrito en la **Pregunta 2** a partir de la nueva forma de expresar la relación *like* desarrollada en la **Pregunta 3**. Para esta pregunta **NO** puede utilizar los predicados de manipulación de listas de Prolog (ej.: member), por lo que sólo puede utilizar los recursos elementales del lenguaje (desarrollo de cláusulas, variables, listas).

```
%domains
%L,L1,L2: listas
%E: elemento de lista
%C: Integer

%predicates
%miembro(E,L)
%intersecta(L1,L2,C)

%goals
%primaria
%intersecta: determinar si dos listas intersectan y en cuantos elementos
(intersecta)
%este predicado sirve además para la pregunta 5

%secundaria
%miembro: determinar si un elemento E pertenece a una lista miembro

miembro(H,[H|_]).
miembro(H,[_|T]):-miembro(H,T).

intersecta([],_,0):-!.
intersecta([H|T],L2,Count):-miembro(H,L2),intersecta(T,L2,CountAux),Count is
CountAux + 1,!.
intersecta([_|T],L2,Count):-intersecta(T,L2,Count).

      match(Nombre1,Nombre2):- Nombre1 \= Nombre2,
                               like(Nombre1,G1),
                               like(Nombre2,G2),
                               intersecta(G1,G2,C),C>0.
```

```
match(Persona1,Persona2):-Persona1\=Persona2,
                          like(Persona1,ListaGustos1),
                          like(Persona2,ListaGustos2),
intersecta(ListaGustos1,ListasGusto2,Coincidencias),
                          Coincendecias>0.
```

5. (**2 pto**) A partir del desarrollo del predicado like de la **Pregunta 3** (listas de gustos), implementar el predicado *matchStrenght(Persona1,Persona2,Strenght)* que indica el grado de afinidad entre Persona1 y Persona2 (siendo estas diferentes) según la cantidad de gustos en común que tengan. Para el caso de este predicado, los *match* se van sumando unitariamente. A modo de ejemplo y tomando como referencia la base de conocimientos de la **Pregunta 1**, la Fuerza (Strength) del match entre pedro y maría es 1. Por otro lado, la Fuerza (Strenght) del match entre jose y maria es 2. Para esta pregunta **NO** puede utilizar los predicados de manipulación de listas de Prolog (ej.: member), por lo que sólo puede utilizar los recursos elementales de Prolog (desarrollo de cláusulas, variables, listas).

```
matchStrenght(Persona1,Persona2,Strenght):- Persona1 \= Persona2,
                                            like(Persona1,G1),
                                            like(Persona2,G2),
                                            intersecta(G1,G2,Strenght).
```