UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

Programación lógica y funcional

ING GINO BARROSO VIRUEZ

BENJAMIN ANTONIO ROMERO CORTEZ
REGISTRO: 215049543

Practico

Problemas de naturaleza iterativa (y recursiva).

Porcentaje terminado:

95%

Comentarios:

En estos ejercicios, vi una forma muy fácil de hacer los ejercicios recursivos, la cual consiste el llevar el resultado de la "Iteración" en el parámetro de recursividad. También, al menos para mi, se me hizo mas fácil empezar a hacer los ejercicios iterativos, luego los recursivos en java y por ultimo los recursivos en prolog. Creo que es cuestión de mucha practica para poder hacer cualquier cosa en prolog al igual que en los lenguajes convencionales.

1. mostrarDivisores(N) : Predicado que muestra los divisores del entero N. Muestra desde el 1 a N.

JAVA – ITERATIVO

```
public void mostrarDivisores(int n){
        int i = 1;
        while(i <= n){
            if(n % i == 0){
                System.out.println(i);
            }
            i++;
        }
    }
```

Llamada:

```
// iterativos.mostrarDivisores(12);
```

Java - Recursivo

```
public void mostrarDivisores(int n){
        mostrarDivisores(n, 1);
    }
    private void mostrarDivisores(int n, int i){
        if(i <= n){
            if(n % i == 0)
                System.out.println(i);
                mostrarDivisores(n, i+1);
        }
    }
```

Llamada:

```
// r.mostrarDivisores(51);
```

Prolog - Recursivo

```
Muestra desde el 1 a N.
mostrarDivisores(N) :- mostrarDivisores(N, 1) .
mostrarDivisores(N, I) :- I > N , ! .
mostrarDivisores(N, I) :-
    N mod I =:= 0 ,
    write(I), nl,
    I1 is I + 1,
    mostrarDivisores(N, I1) , ! .
mostrarDivisores(N, I) :-
    I1 is I + 1 ,
    mostrarDivisores(N, I1) .
```

2. primo(N) : Predicado que es True, si el entero N es un número primo.

Java – Iterativo

```
//Primos son los que solo se pueden dividir entre 1 y ellos mismos
    public boolean primo(int n){
        int i = 1;
        int c = 0;
        while (i <= n) {
            if(n % i == 0){
                c++;
            }
            i++;
        }
        if(c > 2) return false;
        return true;
    }
```

Llamada:

```
// System.out.println(iterativos.primo(9));
```

Java – Recursivo

```
public boolean primo(int n){
        return primo(n, 1, 0);
    }
```

```
    private boolean primo(int n, int i, int c){
        if(i <= n){
            if(n % i == 0){
                c++;
                // System.out.println(c);
            }
            return primo(n, i+1, c);
        }
        return c == 2;//
    }
```

Prolog – Recursivo

```
primo(N) :- primo(N, 1, 0) .

primo(N, I, C) :-
    I > N , C =:= 2 .

primo(N, I, C) :-
    I =< N ,
    (N mod I =:= 0 -> C1 is C + 1 ; C1 is C) ,
    I1 is I + 1 ,
    primo(N, I1, C1) .
```

3. mostrarDivisoresPrimos(N) : Predicado que muestra los divisores primos del entero N.

Java – Iterativo

```
public void mostrarDivisoresPrimos(int n){
        int i = 1;
        while(i <= n){
            if(n % i == 0){
                if(primo(i)){
                    System.out.println(i);
                }
            }
            i++;
        }
    }
```

Llamada:

```
// iterativos.mostrarDivisoresPrimos(5);
```

Java – Recursivo

```
public void mostrarDivisoresPrimos(int n){
        mostrarDivisoresPrimos(n, 1);
    }
    private void mostrarDivisoresPrimos(int n, int i){
        if(i <= n){
            if(n % i == 0){
                if(primo(i)){
                    System.out.println(i);
                }
            }
            mostrarDivisoresPrimos(n, i+1);
        }
    }
```

Prolog – Recursivo

```
    mostrarDivisoresPrimos(N) :- mostrarDivisoresPrimos(N, 1) .
    mostrarDivisoresPrimos(N, I) :- I > N , ! .
    mostrarDivisoresPrimos(N, I) :-
        I =< N ,
        ((N mod I =:= 0) , primo(I) ->
            write(I), nl,
            I1 is I + 1),
        mostrarDivisoresPrimos(N, I1) , ! .

    mostrarDivisoresPrimos(N, I) :-
```

```
        I1 is I + 1,
        mostrarDivisoresPrimos(N, I1) .
```

4. mostrarPrimos(A, B) : Predicado que muestra, los número primos entre A y B. inclusive.

Java – Iterativo

```
public void mostrarPrimos(int a, int b){
        if(a <= b){
            if(primo(a)){
                System.out.println(a);
            }
            mostrarPrimos(a+1, b);
        }
    }
```

Llamada:

```
// System.out.println(iterativos.mostrarPrimos(1, 20));
```

Java – Recursivo

```
    public void mostrarPrimos(int a, int b){
        if(a <= b){
            if(primo(a)){
                System.out.println(a);
            }
            mostrarPrimos(a+1, b);
        }
    }
```

Prolog – Recursivo

```
mostrarPrimos(A, B) :-
    A > B .

mostrarPrimos(A, B) :-
    A =< B ,
    primo(A) -> write(A), nl ,
    A1 is A + 1 ,
    mostrarPrimos(A1, B) , ! .

mostrarPrimos(A, B) :-
    A1 is A + 1 ,
    mostrarPrimos(A1, B) , ! .
```

5. mostrarDivisoresDesc(N): Predicado que muestra los divisores del entero N. Muestra desde el N a 1.

Java – Iterativo

```
public String mostrarDivisoresDesc(int n){
        int i = n;
        String s = "Divisores desc: ";
        while(i > 0){
            if(n % i == 0){
                s += i + " - ";
            }
            i--;
        }
        return s;
    }
//Funcion auxiliar:
    private boolean esDivisor(int n, int divisor){
        return n % divisor == 0;
    }
```

Llamada:

```
// System.out.println(iterativos.mostrarDivisoresDesc(20));
```

Java – Recursivo

```
public void mostrarDivisoresDesc(int n){
        mostrarDivisoresDesc(n, n);
    }
    private void mostrarDivisoresDesc(int n, int i){
        if(i >= 1){
            if(n % i == 0){
                System.out.println(i);
            }
            mostrarDivisoresDesc(n, i-1);
        }
    }
```

Prolog – Recursivo

```
mostrarDivisoresDesc(N) :-
    mostrarDivisoresDesc(N, N) .

mostrarDivisoresDesc(_, I) :-
    I < 1 , ! .

mostrarDivisoresDesc(N, I) :-
    (N mod I =:= 0 ->
        write(I), nl ,
        I1 is I - 1) ,
    mostrarDivisoresDesc(N, I1) , ! .

mostrarDivisoresDesc(N, I) :-
    I1 is I - 1,
    mostrarDivisoresDesc(N, I1) .
```

6. mostrarDivisoresComunes(N, M) : Predicado que muestra los divisores comunes de los entero N y M.

Java – Iterativo

```
public String mostrarDivisoresComunes(int n, int m){
        int i = 1;
        int j = 1;
        String s = "Divisores Comunes: ";
        while(i <= n || j <= m){
            if(esDivisor(n, i) && esDivisor(m, j))
            s+= i + " - ";
            i++;
            j++;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarDivisoresComunes(int n, int m){
        mostrarDivisoresComunes(n, m, 1);
    }
    private void mostrarDivisoresComunes(int n, int m, int i){
        if(i <= n && i <= m){
            if(n % i == 0 && m % i == 0)
                System.out.println(i);
            mostrarDivisoresComunes(n, m, i+1);
        }
    }
```

Prolog – Recursivo

```
mostrarDivisoresComunes(N, M) :-
    mostrarDivisoresComunes(N, M, 1) .

mostrarDivisoresComunes(N, M, I) :-
    (I > N ; I > M) , ! .

mostrarDivisoresComunes(N, M, I) :-
    (N mod I =:= 0) , (M mod I =:= 0) ,
```

```
    write(I) , nl ,
    I1 is I + 1 ,
    mostrarDivisoresComunes(N, M, I1) , ! .

mostrarDivisoresComunes(N, M, I) :-
    I1 is I + 1 ,
    mostrarDivisoresComunes(N, M, I1) , ! .
```

7. mostrarDivisoresPares(N) : Predicado que muestra los divisores pares de N.

Java – Iterativo

```
public String mostrarDivisoresPares(int n){
        int i = 1;
        String s = "Divisores pares: ";
        while(i < n){
            if(esDivisor(n, i) && i % 2 == 0)
                s+= i + " - ";
            i++;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarDivisoresPares(int n){
        mostrarDivisoresPares(n, 1);
    }

    private void mostrarDivisoresPares(int n, int i){
        if(i <= n){
            if(i % 2 == 0 && n % i == 0)
                System.out.println(i);
            mostrarDivisoresPares(n, i+1);
        }
    }
```

Prolog – Recursivo

```
mostrarDivisoresPares(N) :-
    mostrarDivisoresPares(N, 1) .

mostrarDivisoresPares(N, I) :-
    I > N , ! .

mostrarDivisoresPares(N, I) :-
    (I mod 2 =:= 0) , (N mod I =:= 0) ,
    writeln(I) ,
    I1 is I + 1 ,
    mostrarDivisoresPares(N, I1) , ! .

mostrarDivisoresPares(N, I) :-
    I1 is I + 1,
    mostrarDivisoresPares(N, I1) , ! .
```

8. mostrarDivisoresImpares(N): Predicado que muestra los divisores impares de N.

Java – Iterativo

```
private boolean esImpar(int n){
        return n % 2 != 0;
    }
    public String mostrarDivisoresImpares(int n){
        int i = 1;
```

```
        String s = "Divisores impares: ";
        while(i <= 20){
            if(esImpar(i) && esDivisor(n, i))
                s+= i + " - ";
            i++;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarDivisoresImpares(int n){
        System.out.println(mostrarDivisoresImpares(n, 1, ""));
    }

    private String mostrarDivisoresImpares(int n, int i, String s){
        if(i <= n){
            if(i % 2 != 0 && n % i == 0)
                s+= i + "  ";
            return mostrarDivisoresImpares(n, i+1, s);
        }
        return s;
    }
```

Prolog – Recursivo

```
mostrarDivisoresImpares(N) :-
    mostrarDivisoresImpares(N, 1) .

mostrarDivisoresImpares(N, I) :-
    I > N , ! .

mostrarDivisoresImpares(N, I) :-
    (I mod 2 =\= 0) , (N mod I =:= 0) ,
    writeln(I) ,
    I1 is I + 1 ,
    mostrarDivisoresImpares(N, I1) , ! .

mostrarDivisoresImpares(N, I) :-
    I1 is I + 1,
    mostrarDivisoresImpares(N, I1) , ! .
```

5. mostrarDivisores(N, A, B) : Predicado que muestra los divisores de N, comprendidos entre A y B inclusive.

Java – Iterativo

```
public String mostrarDivisoresEntre(int n, int a, int b){
        int i = a;
        String s = "Divisores entre " + a + " y " + b + ": ";
        while(i <= b){
            if(esDivisor(n, i))
            s+= i + " - ";
            i++;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarDivisoresEntre(int n, int a, int b){
        if(a <= b){
            if(n % a == 0)
                System.out.println(a);
            mostrarDivisoresEntre(n, a+1, b);
        }
    }
```

Prolog – Recursivo

```
mostrarDivisores(N, A, B) :-
    (A > B) ; (A > N) , ! .

mostrarDivisores(N, A, B) :-
    (N mod A =:= 0) ,
    writeln(A),
    A1 is A + 1 ,
    mostrarDivisores(N, A1, B) , ! .

mostrarDivisores(N, A, B) :-
    A1 is A + 1 ,
    mostrarDivisores(N, A1, B) , ! .
```

9. proximoPrimo(N, P) : Predicado que encuentra en P, el primo después de N. Si N es primo, P toma el valor de N.

Java – Iterativo

```
public int proximoPrimo(int n){
        int p = 0;
        while (p == 0){
            if(primo(n)){
                p = n;
            }
            n++;
        }
        return p;
    }
```

Java – Recursivo

```
public int proximoPrimo(int n){
        if(primo(n))
            return n;
        else
            return proximoPrimo(n+1);
    }
```

Prolog – Recursivo

```
proximoPrimo(N, P) :-
    primo(N) ,
    P is N , ! .

proximoPrimo(N, P) :-
    N1 is N + 1 ,
    proximoPrimo(N1, P) , ! .
```

10. anteriorPrimo(N, A) : Predicado que encuentra en A, el anterior primo antes de N. Si N es primo, A toma el valor de N.

Java – Iterativo

```
//Nota: el 10 y 11 son similares
    public int anteriorPrimo(int n){
        while(!primo(n)){
            n--;
        }
```

```
        return n;
    }
```

Java – Recursivo

```
public int anteriorPrimo(int n){
        if(n > 1){
            if(primo(n)){
                return n;
            }
            return anteriorPrimo(n-1);
        }
        return 0;
    }
```

Prolog – Recursivo

```
anteriorPrimo(N, A) :-
    N < 1 , A is 0 , ! .

anteriorPrimo(N, A) :-
    primo(N) ,
    A is N , ! .

anteriorPrimo(N, A) :-
    N1 is N - 1 ,
    anteriorPrimo(N1, A) , ! .
```

11. mostrarDivisoresPrimosAsc(N) : Predicado que muestra los divisores primos de N. Muestra desde 1 hasta N.

Java – Iterativo

```
    public String mostrarDivisoresPrimosAsc(int n){
        String s = "Divisores primos asc : ";
        int i = 1;
        while(i <= n){
            if (primo(i) && esDivisor(n, i)) {
                s+= i + " - ";
            }
            i++;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarDivisoresPrimosAsc(int n){
        System.out.println(mostrarDivisoresPrimosAsc(n, 1, ""));
    }
    private String mostrarDivisoresPrimosAsc(int n, int i, String s){
        if (n >= i) {
            if(n % i == 0 && primo(i))
                s+= i + "  ";
            return mostrarDivisoresPrimosAsc(n, i+1, s);
        }
        return s;
    }
```

Prolog – Recursivo

```
mostrarDivisoresPrimosAsc(N) :-
    mostrarDivisoresPrimosAsc(N, 1) .

mostrarDivisoresPrimosAsc(N, I) :-
```

```
    I > N .

mostrarDivisoresPrimosAsc(N, I) :-
    (N mod I =:= 0) , (primo(I)) ,
    writeln(I) ,
    I1 is I + 1 ,
    mostrarDivisoresPrimosAsc(N, I1) , ! .

mostrarDivisoresPrimosAsc(N, I) :-
    I1 is I + 1 ,
    mostrarDivisoresPrimosAsc(N, I1) , ! .
```

12. mostrarDivisoresPrimosDesc(N) : Predicado que muestra los divisores primos de N. Muestra desde N hasta 1.

Java – Iterativo

```
    public String mostrarDivisoresPrimosDesc(int n){
        String s = "Divisores primos desc: ";
        int i = n;
        while (i >= 1) {
            if(primo(i) && esDivisor(n, i))
                s+= i + " - ";
            i--;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarDivisoresPrimosDesc(int n){
        System.out.println(mostrarDivisoresPrimosDesc(n, n, ""));
    }
    private String mostrarDivisoresPrimosDesc(int n, int i, String s){
        if(i > 1){
            if(n % i == 0 && primo(i)){
                s+= i + "  ";
            }
            return mostrarDivisoresPrimosDesc(n, i-1, s);
        }
        return s;
    }
```

Prolog – Recursivo

```
mostrarDivisoresPrimosDesc(N) :-
    mostrarDivisoresPrimosDesc(N, N) .

mostrarDivisoresPrimosDesc(_, I) :-
    I < 1 .

mostrarDivisoresPrimosDesc(N, I) :-
    (N mod I =:= 0) , (primo(I)) ,
    writeln(I) ,
    I1 is I - 1 ,
    mostrarDivisoresPrimosDesc(N, I1) .

mostrarDivisoresPrimosDesc(N, I) :-
    I1 is I - 1 ,
    mostrarDivisoresPrimosDesc(N, I1) .
```

13. mostrarFactoriales(N): Predicado que muestra los factoriales de 1 a N.
Si N = 5

1! = 1
2! = 2
3! = 6
4! = 24
5! = 120

Java – Iterativo

```
private int factorial(int n){
        int i = 1;
        int factorial = 1;
        while(i <= n){
            factorial = factorial * i;
            i++;
        }
        return factorial;
    }
    public String mostrarFactoriales(int n){
        String s = "Factoriales: ";
        int i = 1;
        while(i <= n){
            s+= factorial(i) + " - ";
            i++;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarFactoriales(int n){
        mostrarFactoriales(n, 1, 1);
    }
    private void mostrarFactoriales(int n, int i, int factorial){
        if(i <= n){
            factorial = i * factorial;
            System.out.println(factorial);
            mostrarFactoriales(n, i+1, factorial);
        }
    }
```

Prolog – Recursivo

```
mostrarFactoriales(N) :-
    mostrarFactoriales(N, 1, 1) .

mostrarFactoriales(N, I, _) :-
    I > N .

mostrarFactoriales(N, I, F) :-
    F1 is I * F ,
    writeln(F1) ,
    I1 is I + 1 ,
    mostrarFactoriales(N, I1, F1) .
```

14. mostrarCoefBin(N) : Predicado que muestra los coeficientes binomiales de un binomio elevado a la N. Ej. Si N = 2, muestra 1, 2, 1. Si N = 4, muestra 1, 4, 6, 4, 1.

Java – Iterativo
Java – Recursivo
Prolog – Recursivo

15. mostrarFib(N): Predicado que muestra los primeros N términos de la secuencia de Fibonacci.

Java – Iterativo

```
public String mostrarFib(int n){
        String s = "";
        int i = 0;
        int ant2 = 0;
        int ant1 = 1;
        int fibo = 0;
        while(i <= n){
            fibo = ant1 + ant2;
            s+= fibo + " ";
            ant2 = ant1;
            ant1 = fibo;
            i++;
        }
        return s;
    }
```

Java – Recursivo

```
public void mostrarFib(int n){
        mostrarFib(n, 1, 0, 0, 1);//n, empieza en 1, resultado inicial 0
    }
    public void mostrarFib(int n, int i, int fibo, int ant2, int ant1){
        if(i <= n){
            fibo = ant1 + ant2;
            System.out.println(fibo);
            ant2 = ant1;
            ant1 = fibo;
            mostrarFib(n, i+1, fibo, ant2, ant1);
        }
    }
```

Prolog – Recursivo

```
mostrarFib(N) :-
    N > 0, % Aseguramos que N sea positivo
    writeln(1), % Mostrar el primer término
    mostrarFib(N, 1, 1, 1) .

mostrarFib(N, I, _, _) :-
    I >= N , ! .

mostrarFib(N, I, F1, F2) :-
    F is F1 + F2 ,
    writeln(F) ,
    % F2 is F1 ,
    % F1 is F ,
    I1 is I + 1 ,
    mostrarFib(N, I1, F2, F) .
```

16. Ejercicios de lógica. (no cíclicos)

dosPrimos(A, B),

dosPrimos(A, B, C),

dosPrimos(A, B, C, D) : Predicado que es True, si existe dos primos exactamente entre los argumentos.

Java – Iterativo

```
public boolean dosPrimos(int a, int b){
        return primo(a) && primo(b);
    }
    // dosPrimos(A, B, C),
    public boolean dosPrimos(int a, int b, int c){
        return
            primo(a) && primo(b) ||
            primo(a) && primo(c) ||
            primo(b) && primo(c);
    }

    // dosPrimos(A, B, C, D) : Predicado que es True, si existe dos primos
exactamente entre los argumentos.
    public boolean dosPrimos(int a, int b, int c, int d){
        return
            primo(a) && primo(b) && !primo(c) && !primo(d) ||
            primo(a) && primo(c) && !primo(b) && !primo(d) ||
            primo(a) && primo(d) && !primo(b) && !primo(c) ||
            primo(b) && primo(c) && !primo(a) && !primo(d) ||
            primo(b) && primo(d) && !primo(a) && !primo(c) ||
            primo(c) && primo(d) && !primo(a) && !primo(b);
    }
```

Java – Recursivo

Prolog – Recursivo

```
% dosPrimos(A, B),
dosPrimos(A, B) :-
    primo(A) , primo(B) .

% dosPrimos(A, B, C),
dosPrimos(A, B, C) :-
    (primo(A) , primo(B) , not(primo(C))) ;
    (primo(A) , primo(C) , not(primo(B))) ;
    (primo(B) , primo(C) , not(primo(A))) .

% dosPrimos(A, B, C, D) : Predicado que es True, si existe dos primos exactamente
entre los argumentos.
dosPrimos(A, B, C, D) :-
    dosPrimos(A, B) , not(primo(C)) , not(primo(D)) ;
    dosPrimos(A, C) , not(primo(B)) , not(primo(D)) ;
    dosPrimos(A, D) , not(primo(B)) , not(primo(C)) ;
    dosPrimos(B, C) , not(primo(A)) , not(primo(D)) ;
    dosPrimos(B, D) , not(primo(A)) , not(primo(C)) ;
    dosPrimos(C, D) , not(primo(A)) , not(primo(B)) .
```

17.

existePrimoNP(A, B),

existePrimoNP(A, B, C),

existePrimoNP(A, B, C, D) : Predicado que es True, si existe al menos un primo y un no primo entre los argumentos.

Java – Iterativo

```
    public boolean existePrimoNP(int a, int b){
        return
            primo(a) && !primo(b) ||
            primo(b) && !primo(a);
    }
```

```
    //existePrimoNP(A, B, C),
    public boolean existePrimoNP(int a, int b, int c){
        return
        primo(a) && !primo(b) ||
        primo(b) && !primo(a) ||
        primo(a) && !primo(c) ||
        primo(c) && !primo(a) ||
        primo(b) && !primo(c) ||
        primo(c) && !primo(b);
    }

    // existePrimoNP(A, B, C, D) : Predicado que es True, si existe al menos un primo
y un no primo entre los argumentos.
    public boolean existePrimoNP(int a, int b, int c, int d){
        return
            existePrimoNP(a, b, c) || existePrimoNP(c, d);
    }
```

Java – Recursivo

Prolog – Recursivo

```
existePrimoNP(A, B) :-
    (primo(A), not(primo(B))) ; primo(B), not(primo(A)) .

% existePrimoNP(A, B, C),
existePrimoNP(A, B, C) :-
    existePrimoNP(A, B) ; existePrimoNP(A, C) ; existePrimoNP(B, C) .

% existePrimoNP(A, B, C, D) : Predicado que es True, si existe al menos un primo y un
no primo entre los argumentos.
existePrimoNP(A, B, C, D) :-
    existePrimoNP(A, B, C) ; existePrimoNP(A, B, D) ; existePrimoNP(A, C, D) ;
existePrimoNP(B, C, D) .
```

18.

primosAlternos(A, B),

primosAlternos(A, B, C),

primosAlternos(A, B, C, D) : Predicado que es True, si la secuencia de argumentos están alternados entre primos y no primos.

Java – Iterativo

```
    // primosAlternos(A, B),
    public boolean primosAlternos(int a, int b){
        return primo(a) && !primo(b);
    }

    // primosAlternos(A, B, C),
    public boolean primosAlternos(int a, int b, int c){
        return
            primo(a) && !primo(b) && primo(c) ||
            !primo(a) && primo(b) && !primo(c);
    }

    // primosAlternos(A, B, C, D) : Predicado que es True, si la secuencia de
argumentos están alternados entre primos y no primos.
    public boolean primosAlternos(int a, int b, int c, int d){
        return
            primo(a) && !primo(b) && primo(c) && !primo(d) ||
            !primo(a) && primo(b) && !primo(c) && primo(d);
    }
```

Java – Recursivo

Prolog – Recursivo

```
% primosAlternos(A, B),
primosAlternos(A, B) :-
    (primo(A) , not(primo(B))) ; (primo(B) , not(primo(A))) .

% primosAlternos(A, B, C),
primosAlternos(A, B, C) :-
    (primo(A) , not(primo(B)) , primo(C)) ; (not(primo(A)) , primo(B) ,
not(primo(C))) .

% primosAlternos(A, B, C, D) : Predicado que es True, si la secuencia de argumentos
están alternados entre primos y no primos.
primosAlternos(A, B, C, D) :-
    (primo(A) , not(primo(B)) , primo(C) , not(primo(D))) ;
    (primo(B) , not(primo(A)) , primo(D) , not(primo(C))) .
```