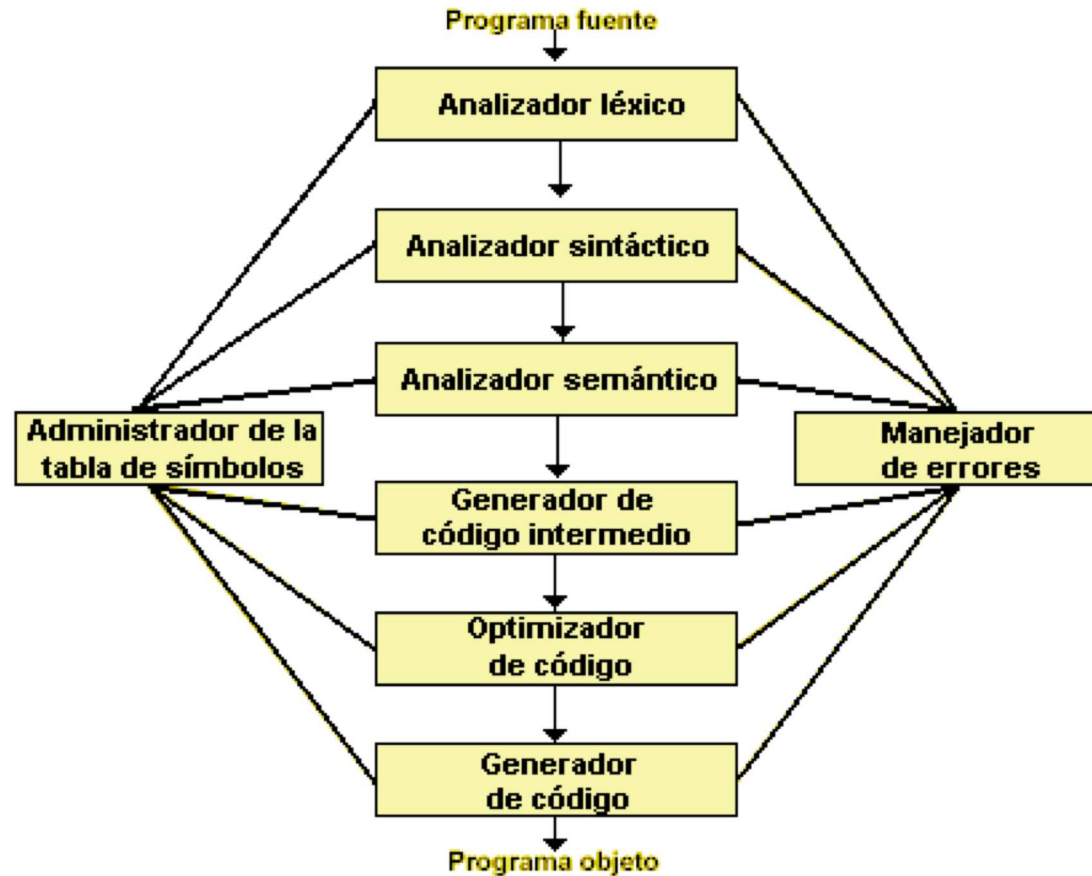


DE CERO A CIENCIA DE DATOS



COMPILADOR

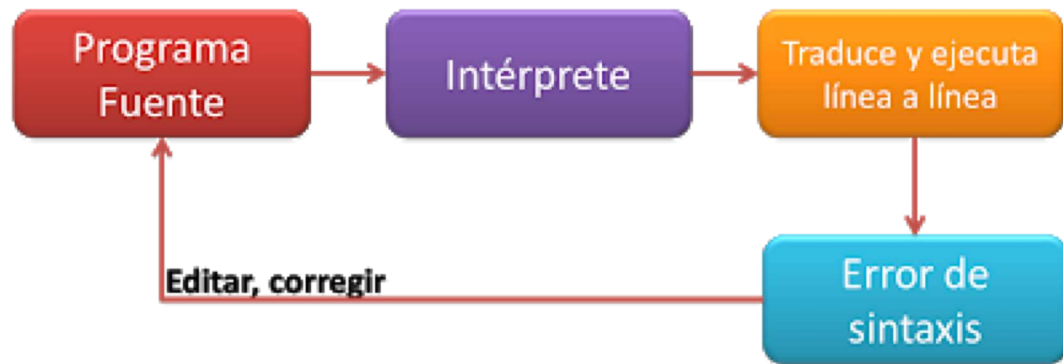


UN COMPILADOR ANALIZA EL PROGRAMA Y LO TRADUCE AL IDIOMA "MAQUINA". LA ACCIÓN FUNDAMENTAL LOS COMPILADORES ES EQUIVALENTE A LA DE UN TRADUCTOR HUMANO , QUE TOMA NOTA DE LO QUE ESTA ESCUCHANDO Y REPRODUCE POR ESCRITO EN OTRA LENGUA.

CARACTERISTICAS DE LOS COMPILADORES

- **PRODUCE UN CÓDIGO OPTIMIZADO.**
- **LA EJECUCIÓN DEL PROGRAMA OBJETO ES MUCHO MAS RÁPIDA QUE SI SE INTERPRETA EL PROGRAMA FUENTE.**
- **EL COMPILADOR TIENE UNA VISIÓN GLOBAL DEL PROGRAMA, POR LO QUE LA INFORMACIÓN DE MENSAJES DE ERROR ES MÁS DETALLADA.**
- **SE DEBE EJECUTAR MUCHAS VECES EL CÓDIGO FUENTE PARA VER LOS CAMBIOS EN EL RESULTADO.**
- **MAYOR CONSUMO DE MEMORIA.**

INTERPRETE



INTÉRPRETE O INTERPRETADOR ES UN PROGRAMA INFORMÁTICO CAPAZ DE ANALIZAR Y EJECUTAR OTROS PROGRAMAS, ESCRITOS EN UN LENGUAJE DE ALTO NIVEL. LOS INTÉRPRETES SE DIFERENCIAN DE LOS COMPILADORES EN QUE MIENTRAS ESTOS TRADUCEN UN PROGRAMA DESDE SU DESCRIPCIÓN EN UN LENGUAJE DE PROGRAMACIÓN AL CÓDIGO DE MÁQUINA DEL SISTEMA, LOS PRIMEROS (LOS INTÉRPRETES) SÓLO REALIZAN LA TRADUCCIÓN A MEDIDA QUE SEA NECESARIA, TÍPICAMENTE, INSTRUCCIÓN POR INSTRUCCIÓN, Y NORMALMENTE NO GUARDAN EL RESULTADO DE DICHA TRADUCCIÓN.

CARACTERÍSTICAS DE LOS INTERPRETES

- SU PRINCIPAL VENTAJA ES QUE PERMITEN UNA FÁCIL DEPURACIÓN. PERMITEN UNA MAYOR INTERACTIVIDAD CON EL CÓDIGO EN TIEMPO DE DESARROLLO.
- EN ALGUNOS LENGUAJES (SMALLTALK, PROLOG, LISP) ESTÁ PERMITIDO Y ES FRECUENTE AÑADIR CÓDIGO SEGÚN SE EJECUTA OTRO CÓDIGO, Y ESTA CARACTERÍSTICA SOLAMENTE ES POSIBLE IMPLEMENTARLA EN UN INTÉRPRETE.
- PUEDE SER INTERRUPTIDO CON FACILIDAD.
- PUEDE SER RÁPIDAMENTE MODIFICADO Y EJECUTADO NUEVAMENTE.
- UN INTÉRPRETE NECESITA MENOS MEMORIA QUE UN COMPILADOR.
- FACILITA LA BÚSQUEDA DE ERRORES.
- EN ALGUNOS LENGUAJES ESTÁ PERMITIDO AÑADIR CÓDIGO SEGÚN SE EJECUTA OTRO CÓDIGO.
- MENOR CONSUMO DE MEMORIA.
- CADA INSTRUCCIÓN DEBE SER TRADUCIDA A CÓDIGO MÁQUINA TANTAS VECES COMO SEA EJECUTADA
- DURANTE LA EJECUCIÓN, EL INTÉRPRETE DEBE RESIDIR EN MEMORIA YA QUE NO GENERA CÓDIGO OBJETO.

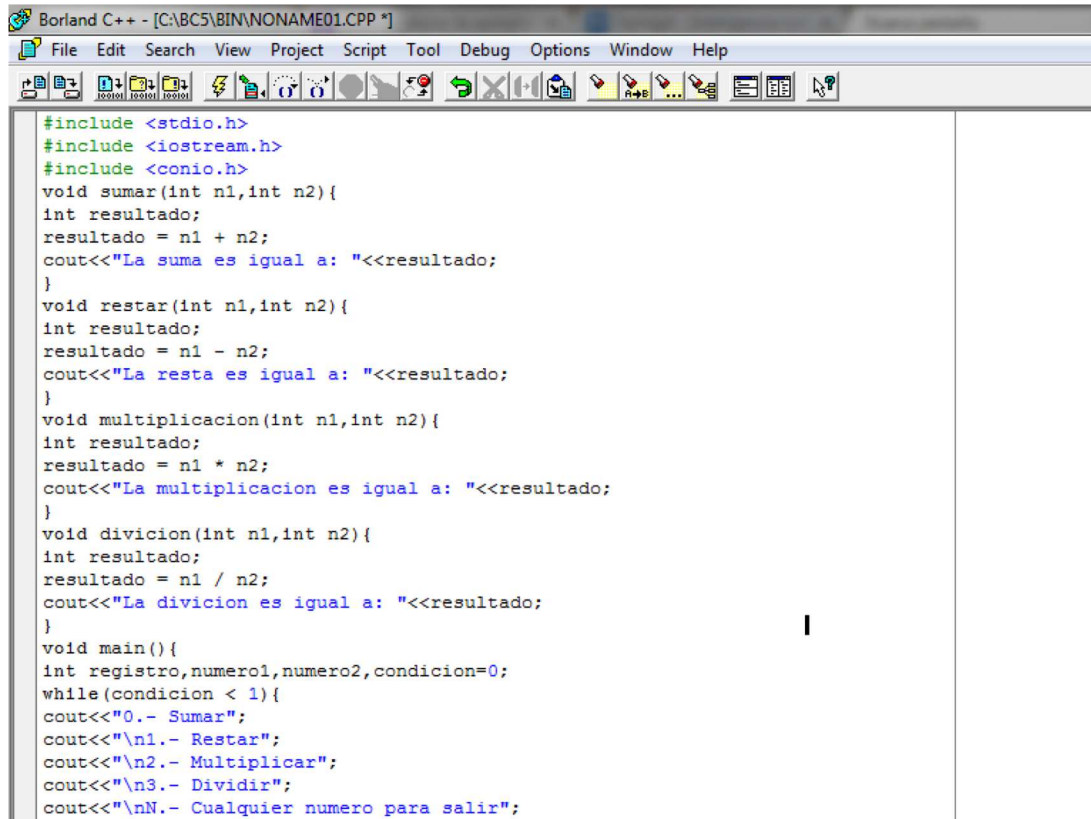
PROGRAMACIÓN ESTRUCTURADA

```
#!/usr/bin/python
#encoding=utf-8
import MySQLdb
def lee_nombres(x):
    mnombres=raw_input('Nombres y apellidos: ')
    mnombres=""+mnombres+""
    return mnombres

mci=""
mnombres=""
r='s'
b=MySQLdb.connect('localhost',user='root',passwd='1234567',db='melvin')
p=b.cursor()
while (r!='s'):
    mci=raw_input('Introduzca código a consultar: ')
    #p=b.cursor()
    p.execute('SELECT * FROM usuario');
    g=p.fetchone()
    existe=0
    while (g!=None and existe==0):
        if mci==g[0]:
            print 'Consulta Especifica: Registro existe...' + g[1]
            r2=raw_input('1. Modificar 2. Borrar 3. continuar')
            if r2=='1':
                mnombres=lee_nombres("")
                p.execute('update usuario set nombres="'+mnombres+'" where ci="'+mci+')
            if r2=='2':
                r3=raw_input('Esta seguro de eliminar s/n: ')
                if r3=='s':
                    p.execute('delete from usuario where ci="'+mci+')
                    print 'Registro borrado...'
                else:
                    print 'Eliminación abortada'
                existe=1
            else:
                g=p.fetchone() ###va al proximo registro
    if existe==0:
        print 'Registro nuevo...'
        r3=raw_input('Desea incluir s/n?')
        if r3=='s':
            mnombres=lee_nombres("")
            mci="" + mci + ""
            p.execute('INSERT INTO usuario values(' + mci + ', '+ mnombres + ')')
            print 'Inclusión realizada'
        else:
            print 'Inclusión no realizada...'
    r=raw_input('Desea procesar otro registro s/n: ')
p.close()
```

UN PROGRAMA ESTA ESTRUCTURADO SI POSEE UN ÚNICO PUNTO DE ENTRADA Y SÓLO UNO DE SALIDA, EXISTEN DE "1 A N" CAMINOS DESDE EL PRINCIPIO HASTA EL FIN DEL PROGRAMA Y POR ÚLTIMO, QUE TODAS LAS INSTRUCCIONES SON EJECUTABLES SIN QUE APAREZCAN BUCLES INFINITOS.

PROGRAMACIÓN MODULAR



```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
void sumar(int n1,int n2){
int resultado;
resultado = n1 + n2;
cout<<"La suma es igual a: "<<resultado;
}
void restar(int n1,int n2){
int resultado;
resultado = n1 - n2;
cout<<"La resta es igual a: "<<resultado;
}
void multiplicacion(int n1,int n2){
int resultado;
resultado = n1 * n2;
cout<<"La multiplicacion es igual a: "<<resultado;
}
void division(int n1,int n2){
int resultado;
resultado = n1 / n2;
cout<<"La division es igual a: "<<resultado;
}
void main(){
int registro,numero1,numero2,condicion=0;
while(condicion < 1){
cout<<"0.- Sumar";
cout<<"\n1.- Restar";
cout<<"\n2.- Multiplicar";
cout<<"\n3.- Dividir";
cout<<"\nN.- Cualquier numero para salir";
```

AL APLICAR LA PROGRAMACIÓN MODULAR, UN PROBLEMA COMPLEJO DEBE SER DIVIDIDO EN VARIOS SUBPROBLEMAS MÁS SIMPLES, Y ESTOS A SU VEZ EN OTROS SUBPROBLEMAS MÁS SIMPLES. ESTO DEBE HACERSE HASTA OBTENER SUBPROBLEMAS LO SUFICIENTEMENTE SIMPLES COMO PARA PODER SER RESUELTOS FÁCILMENTE CON ALGÚN LENGUAJE DE PROGRAMACIÓN. ESTA TÉCNICA SE LLAMA REFINAMIENTO SUCESIVO, DIVIDE Y VENCERÁS O ANÁLISIS DESCENDENTE (TOP-DOWN).

PROGRAMACIÓN ORIENTA A OBJETOS

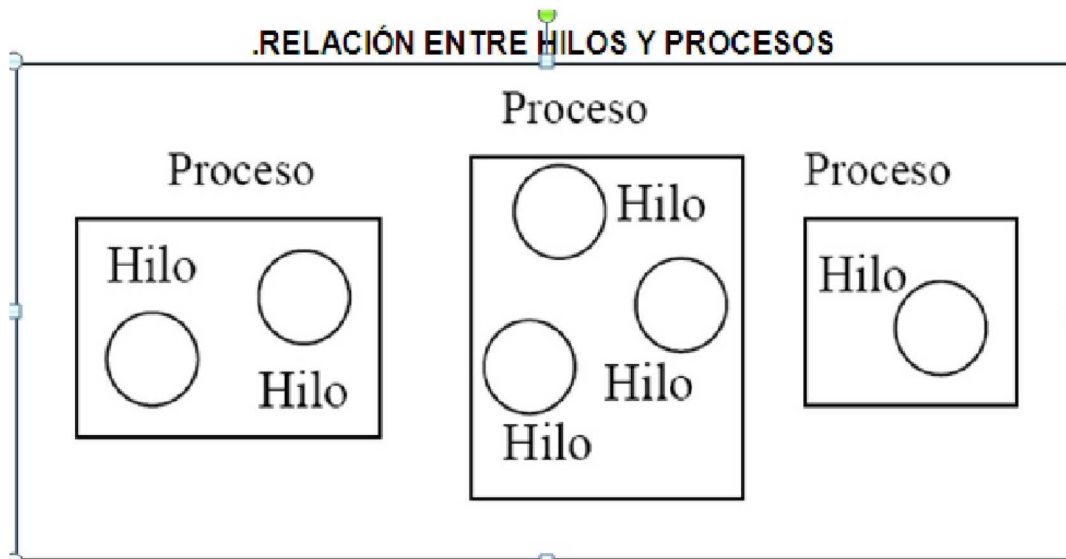
```
/**
 *Calcula los primos del 1 al 1000
 */
public class primos {
    /** Función principal */
    public static void main(String args[]){
        int nPrimos=10000;
        boolean primo[]=new boolean[nPrimos+1];
        short i;
        for (i=1;i<=nPrimos;i++) primo[i]=true;
        for (i=2;i<=nPrimos;i++){
            if (primo[i]){
                for (int j=2*i;j<=nPrimos;j+=i){
                    primo[j]=false;
                }
            }
        }
        for (i=1;i<=nPrimos;i++) {
            System.out.print(" "+i);
        }
    }
}
```

EL ELEMENTO PRINCIPAL DE LA PROGRAMACIÓN ORIENTADA A OBJETOS ES EL OBJETO.

EL OBJETO ES UN CONJUNTO COMPLEJO DE DATOS Y PROGRAMAS QUE POSEEN ESTRUCTURA Y FORMAN PARTE DE UNA ORGANIZACIÓN.


UN OBJETO CONTIENE VARIOS DATOS BIEN ESTRUCTURADOS Y PUEDEN SER VISIBLES O NO DEPENDIENDO DEL PROGRAMADOR Y LAS ACCIONES DEL PROGRAMA EN ESE MOMENTO.

PROGRAMACIÓN CONCURRENTE



**LA COMPUTACIÓN CONCURRENTE ES LA
SIMULTANEIDAD EN LA EJECUCIÓN DE MÚLTIPLES
TAREAS INTERACTIVAS**

PROGRAMACIÓN FUNCIONAL

function definition	trace of execution
<pre>function factI (n) local accumulator = 1 for i = 1,n do accum = accumulator*i end return accum end</pre>	 <pre>factI(4) accumulator = 1 i = 1 accumulator = 1 * 1 i = 2 accumulator = 1 * 2 i = 3 accumulator = 2 * 3 i = 4 accumulator = 6 * 4 return 24</pre>

IMPERATIVE

variable

function definition	trace of execution
<pre>function factR (n) if n == 1 then return 1 else return n*factR(n-1) end end</pre>	<pre>factR(4) = 4 * factR(3) = 3 * factR(2) = 2 * factR(1) = 1 2 6</pre>

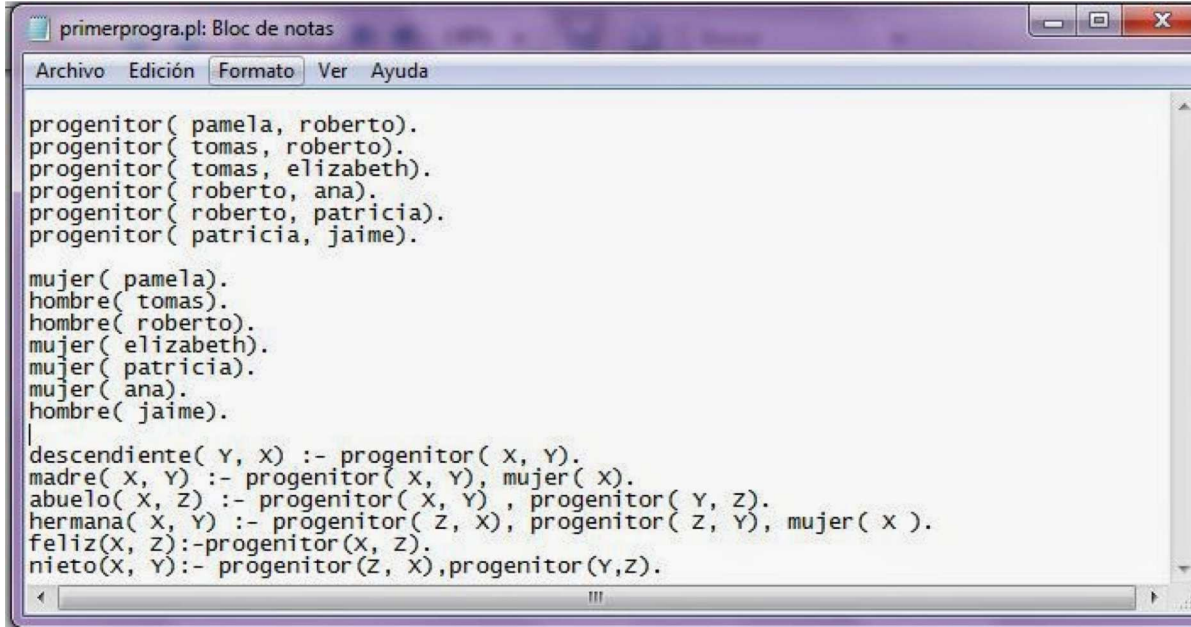
FUNCTIONAL

No variables
just param...
use recursion & loops

LOS PROGRAMAS ESCRITOS EN UN LENGUAJE FUNCIONAL ESTÁN CONSTITUIDOS ÚNICAMENTE POR DEFINICIONES DE FUNCIONES, ENTENDIENDO ÉSTAS NO COMO SUBPROGRAMAS CLÁSICOS DE UN LENGUAJE IMPERATIVO, SINO COMO FUNCIONES PURAMENTE MATEMÁTICAS, EN LAS QUE SE VERIFICAN CIERTAS PROPIEDADES COMO LA TRANSPARENCIA REFERENCIAL (EL SIGNIFICADO DE UNA EXPRESIÓN DEPENDE ÚNICAMENTE DEL SIGNIFICADO DE SUS SUBEXPRESIONES), Y POR TANTO, LA CARENCIA TOTAL DE EFECTOS COLATERALES.

OTRAS CARACTERÍSTICAS PROPIAS DE ESTOS LENGUAJES SON LA NO EXISTENCIA DE ASIGNACIONES DE VARIABLES Y LA FALTA DE CONSTRUCCIONES ESTRUCTURADAS COMO LA SECUENCIA O LA ITERACIÓN

PROGRAMACIÓN LÓGICA

A screenshot of a Notepad window titled "primerprogra.pl: Bloc de notas". The window has a menu bar with "Archivo", "Edición", "Formato", "Ver", and "Ayuda". The text area contains Prolog code defining a family tree. The code includes facts for "progenitor", "mujer", and "hombre", and rules for "descendiente", "madre", "abuelo", "hermana", "feliz", and "nieto".

```
primerprogra.pl: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda

progenitor( pamela, roberto).
progenitor( tomas, roberto).
progenitor( tomas, elizabeth).
progenitor( roberto, ana).
progenitor( roberto, patricia).
progenitor( patricia, jaime).

mujer( pamela).
hombre( tomas).
hombre( roberto).
mujer( elizabeth).
mujer( patricia).
mujer( ana).
hombre( jaime).

descendiente( Y, X) :- progenitor( X, Y).
madre( X, Y) :- progenitor( X, Y), mujer( X).
abuelo( X, Z) :- progenitor( X, Y), progenitor( Y, Z).
hermana( X, Y) :- progenitor( Z, X), progenitor( Z, Y), mujer( X ).
feliz(X, Z):-progenitor(X, Z).
nieto(X, Y):- progenitor(Z, X),progenitor(Y,Z).
```

LA PROGRAMACIÓN LÓGICA ENCUENTRA SU HÁBITAT NATURAL EN APLICACIONES DE INTELIGENCIA ARTIFICIAL O RELACIONADAS: SISTEMAS EXPERTOS, DONDE UN SISTEMA DE INFORMACIÓN IMITA LAS RECOMENDACIONES DE UN EXPERTO SOBRE ALGÚN DOMINIO DE CONOCIMIENTO. DEMOSTRACIÓN AUTOMÁTICA DE TEOREMAS, DONDE UN PROGRAMA GENERA NUEVOS TEOREMAS SOBRE UNA TEORÍA EXISTENTE.

RECONOCIMIENTO DE LENGUAJE NATURAL, DONDE UN PROGRAMA ES CAPAZ DE COMPRENDER (CON LIMITACIONES) LA INFORMACIÓN CONTENIDA EN UNA EXPRESIÓN LINGÜÍSTICA HUMANA.