LENGUAJES DE PROGRAMACIÓN

1.- Preliminares

2.- Paradigmas de Programación.

3.- Lenguajes de Programación

Preliminares

- **Programar** = Plantear solución a un problema mediante un Lenguaje de Programación.
- El Paradigma de Programación condiciona la forma en que se expresa la solución a un problema.

- El Lenguaje de Programación (que se encuadra en un determinado paradigma) es la herramienta que permite expresar nuestra solución.

Paradigmas de programación

- Colección de patrones conceptuales (estructuras o reglas) que juntos modelan el proceso de diseño y que determinan en última instancia la estructura de los programas realizados.

- Un Lenguaje de Programación se dice que pertenece a un determinado paradigma si recoge adecuadamente los patrones conceptuales definidos en el paradigma.

Paradigmas: Clasificación inicial

- En un primer nivel los paradigmas se pueden clasificar en función de la aproximación que adoptan para la solución del problema:

  - ✓ ***Paradigma Operacional***: La solución se describe paso a paso (ejemplo: una receta de cocina es una secuencia ordenada de pasos a realizar).

  - ✓ ***Paradigma Declarativo***: un programa está constituido por hechos, reglas, restricciones, ecuaciones, transformaciones, u otras propiedades que debe cumplir el conjunto solución.

Paradigma Operacional

- **Paradigma Operacional: La solución al problema se describe paso a paso.**
- Se puede subdividir en**: Paradigmas con efecto de lado y sin efecto de lado.**
- **Paradigmas con efecto de lado:** son aquellos que modifican continuamente las variables. Ejemplos de estos son: **Paradigma Imperativo** y **Paradigma Orientado a Objetos.**
- **Paradigmas sin efecto de lado:** son aquellos que crean nuevas variables (no posee la asignación) como por ejemplo el **Paradigma Funcional (Operacional).**

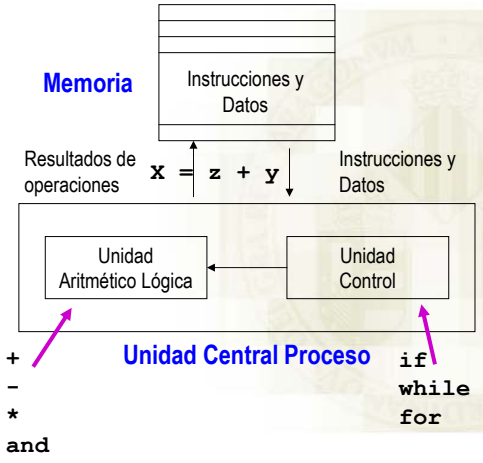
Paradigma Imperativo (I)

- Se basa en un modelo abstracto de computadora que consiste en un gran almacén. La máquina almacena una representación codificada del computo a realizar y ejecuta una secuencia de ordenes que modifican el almacén.
- En su forma pura únicamente soporta la **modificación del almacén, el salto condicional y el salto incondicional**
  - La arquitectura von Neumann favorece los lenguajes imperativos, pues se asemejan más a lo que "sabe" hacer la máquina → los lenguajes imperativos son más eficientes.

Paradigma Imperativo (II)

- **Reconstruye la máquina para hacerla adecuada a la programación.**
  - ✓ Variables: Localizaciones de memoria
  - ✓ Asignación: Cambia valores almacenados en la máquina
  - ✓ Enunciados de control de flujo: `if, while`, ...

Paradigma Imperativo (III)



Paradigma Imperativo (IV): Ejemplo *C*

- **Ordenación de un vector de enteros mediante el método de la burbuja.**

```
#include <stdio.h>

void main(){
   int N = 10;
   int datos[] = {1,4,5,6,19,12,20,3,7,10};
   int i,j;
   int aux;

   for (i=0;i<(N-2);i++)
      for (j=(N-1);j>i;j--)
         if (datos[j]<datos[j-1]){
            aux = datos[j-1];
            datos[j-1] = datos[j];
            datos[j] = aux;
         }
   for (i=0;i<=(N-1);i++)
      fprintf(stdout,"%d,",datos[i]);
}
```


Paradigma Orient. Objetos (I)

- **Elemento básico el Objeto.**
  - ✓ Oculta en su interior datos y operaciones (tipo abstracto de datos)
  - ✓ Los objetos colaboran, mediante el envío de mensajes, para resolver el problema.
- La solución a un problema consiste en crear una serie de objetos de unos determinados tipos y utilizarlos para definir secuencias operacionales paso a paso.
- Las características básicas comunes que presentan los lenguajes que siguen el paradigma orientado a objetos son:
  - ✓ **Encapsulación de la información**
  - ✓ **Herencia**
  - ✓ **Envío de mensajes**

Paradigma Orient. Objetos (II): Ejemplo *Java*

- **Obtención de la mediana de la nota de los alumnos de una determinada asignatura.**

```
class Alumno{
   private String nombre;
   private float nota;

   Alumno(String nomb){
      nombre = nomb;
   }
   public void setNota(float n){
      nota = n;
   }
   public float getNota(){
      return nota;
   }
}
```

```
class Asignatura{
   private String nombre;
   private Alumno[] alumnos;
   private boolean ordenado = false;
   private int posicion;
   private int nAlumnos;

   Asignatura(String nomb, int nAl){
      nombre = nomb;
      posicion=0;
      nAlumnos = nAl;
      alumnos = new Alumno[nAlumnos];
   }
```


Paradigma Orient. Objetos (II): Ejemplo *Java*

```
   public void addAlumno(Alumno al){
      if (posicion<nAlumnos)
         alumnos[posicion] = al;
      posicion++;
   }
   private void ordena(){
      //...
   }
   public float mediana(){
      ordena();
      return alumnos[Math.round(alumnos.length/2)].getNota();
   }
}
```

Paradigma Orient. Objetos (II): Ejemplo *Java*

```
public class Ejemplo{
   public static void main(String[] args){
      Asignatura as;
      as = new Asignatura("LP", 5);

      Alumno a = new Alumno("A");
      a.setNota(6.5f);
      as.addAlumno(a);

      a = new Alumno("B");
      a.setNota(6.2f);
      as.addAlumno(a);

      a = new Alumno("C");
      a.setNota(8.2f);
      as.addAlumno(a);

      a = new Alumno("D");
      a.setNota(5.2f);
      as.addAlumno(a);

      a = new Alumno("E");
      a.setNota(7.2f);
      as.addAlumno(a);

      System.out.println(as.mediana());

   }
}
```


Paradigma Declarativo

- En los paradigmas declarativos, un programa está constituido por hechos, reglas, restricciones, ecuaciones, transformaciones, u otras propiedades que debe cumplir el conjunto solución.
- A partir de esta información el sistema debe derivar un esquema que incluya una secuencia de evaluaciones para calcular la solución.
- El programador no proporciona una descripción paso a paso de cómo llegar a la solución.
- Ejemplos: **Funcional**, **Lógico**, basado en Formularios, basado en Flujo de Datos y basado en Restricciones.

Paradigma Lógico (I)

- Se asume que se comienza con un conjunto de hechos conocidos, del estilo *"Antonio es padre"*, y un conjunto de reglas que permiten la deducción de otros hechos.
- Por ejemplo, a partir del hecho anterior y de la regla: "*Para todo X, si X es padre entonces X es varón*" se puede deducir el hecho *"Antonio es varón"*.
- La programación lógica desde la perspectiva del programador consiste en establecer correctamente todos los hechos y reglas, el cálculo está implícito.

Paradigma Lógico (II)

- La mayoría de los lenguajes de programación lógicos se basan en claúsulas de Horn, que son un subconjunto de los predicados de primer orden.
- Por ejemplo la regla:

  `abuelo(X,Y)` ← `padre(X,Z)` ∧ `padre(Z,Y)`

  establece que si *X* es el padre de un *Z* y éste *Z* es a su vez padre de *Y*, entonces *X* es abuelo de *Y*
- Las claúsulas de Horn son una forma restringida de lógica de predicados con una única conclusión por claúsula.

Paradigma Lógico (III): Ejemplo *Prolog*

- **Ejemplo:**

```
padre(jose,juan).
padre(jose,maria).
padre(juan,carlos).
padre(juan,alberto).

abuelo(X,Y) :- padre(X,Z),padre(Z,Y).
```

Si se pregunta al sistema:

```
abuelo(jose,X).
```

la respuesta es:

```
X = carlos
X = alberto
```

Si se pregunta al sistema:

```
abuelo(X,carlos).
```

la respuesta es:

```
X = jose
```


Paradigma Funcional

- **Programación sin asignaciones(!), basado en el concepto matemático de función:**
  - ✓ Una expresión es una función que se puede calcular a partir del valor de sus subexpresiones (que también son funciones).
  - ✓ Los valores son funciones.
  - ✓ Almacenamiento implícito.

Paradigma Concurrente

- **El elemento fundamental es el proceso:**

  ✓ Cómputo secuencial con su propio flujo de control.

  ✓ Los procesos interaccionan:
  - Comunicación: intercambio de información
  - Sincronización: relación temporal

Conclusiones

- Los paradigmas de programación influyen en los procesos de razonamiento a la hora de resolver los problemas.
- Proporcionan un marco y determinan la forma en la que se expresa la solución.
- Un mismo lenguaje puede soportar distintos paradigmas de programación.

Lenguajes de Programación (I)

- Como ya se comentó anteriormente, para obtener una solución utilizando un determinado paradigma se necesita un Lenguaje de Programación.

- A continuación se presentan algunos criterios de evaluación de los lenguajes de programación y se muestra la evolución de algunos de ellos.

LP (II): Criterios de evaluación

- **Facilidad de Lectura/Comprensión (Legibilidad) de los programas**
  - ✓ El software hay que mantenerlo (corregir errores, introducir cambios o mejoras,...), por lo tanto, el lenguaje debe facilitar la comprensión de los programas una vez escritos.
- **Facilidad de Escritura/Codificación de los programas**
  - ✓ Debe ser "fácil" utilizar el lenguaje para desarrollar programas que se ajusten al tipo de problemas para el que está orientado.
- **Fiabilidad de los programas**
  - ✓ Un programa es fiable si realiza sus especificaciones en cualquier condición.

LP (III): Criterios de evaluación >> Legibilidad

- **Simplicidad del lenguaje: número de componentes básicos y criterios de combinación para generar las estructuras de control y de datos del lenguaje.**
- **Estructuras de control: permiten seguir el flujo del programa.**
- **Tipos de datos y estructuras**
- **Consideraciones sintácticas: Identificadores, palabras reservadas, ...**

LP (IV): Criterios de evaluación >> Legibilidad

- **¿ es *C* simple?**
  - ✓ ¿Cómo se incrementa en 1 un contador?

```
cont = cont + 1
cont += 1
cont ++
++ cont
```

¿Demasiadas posibilidades?

  - ✓ Tipos estructurados, `array` y `struct`
    - Una función puede retornar `struct`, pero no `array`
    - Un `struct` no puede tener un campo de su mismo tipo, un `array` sí.
    - Los argumentos de las funciones siempre se pasan por valor, excepto los `array`, que siempre se pasan por referencia.

¿Demasiadas diferencias?

LP (V): Criterios de evaluación >> Codificación

- **Simplicidad del lenguaje:**
  - ✓ Pocas estructuras permiten codificar sistemas complejos.
- **Soporte para la abstracción de algoritmos y datos.**
  - ✓ Ejemplo: subrutinas (funciones)
- **Expresividad del lenguaje**
  - ✓ Ejemplo: Un bucle `for` se puede construir con un bucle `while`, pero la estructura `for` es más expresiva en muchos casos.

LP (VI): Criterios de evaluación >> Fiabilidad

- **Comprobación de tipos:**
  - ✓ Es una fuente importante de errores de ejecución.
  - ✓ Un comprobación estricta incrementa la robustez del programa.
- **Manejo de excepciones:**
  - ✓ Capacidad del programa para interceptar errores de ejecución y tomar medidas adecuadas.
- **Aliasing:**
  - ✓ Diversos nombres permiten acceder a las mismas posiciones de memoria (punteros, uniones,...)
  - ✓ Peligroso para la fiabilidad

Coste del Lenguaje de Programación

- **El "coste" de un lenguaje depende de sus características:**
  - ✓ Coste de aprendizaje
  - ✓ Coste de codificación
  - ✓ Coste de compilación
  - ✓ Coste de ejecución
  - ✓ Coste de uso del lenguaje (compiladores, entornos, ...)
  - ✓ Coste de fiabilidad (aplicaciones críticas)
  - ✓ Coste de mantenimiento de los programas

Evolución de los Lenguajes de Programación (1)

- **Antes de 1950: Lenguajes interpretados:**
  - ✓ No hay hardware específico para cálculo con reales.
- **FORTRAN (1952): Primer lenguaje de alto nivel compilado**
  - ✓ Orientado a cálculo científico (FORmula TRANslator)
  - ✓ Imperativo
  - ✓ Última versión FORTRAN 90

Evolución de los LP (2)

- **LISP (1958): Primer lenguaje funcional**
  - ✓ Originalmente orientado a IA (LISt Processing)
  - ✓ Dialectos de LISP:
    - Scheme (1975)
    - COMMON LISP (1984)
  - ✓ Otros lenguajes funcionales:
    - ML
    - Miranda
    - Haskell

Evolución de los LP (3)

- **ALGOL (1960): Gran influencia posterior**
  - ✓ Surge a partir de un comité internacional para definir un lenguaje "universal".
  - ✓ Generaliza características de FORTRAN
  - ✓ Introduce:
    - Tipo de datos
    - Sentencia compuesta
    - Identificadores de cualquier longitud
    - Condiciones anidadas
    - Paso de parámetros
    - Recursión,....

Evolución de los LP (4)

- **COBOL (1959): Muy usado pero poca influencia posterior**
  - ✓ Origen Dpt. de Defensa Americano: Inicialmente CBL (Common Business Language)

Evolución de los LP (5)

- **Simula 67: Introduce los primeros conceptos de O.O.**
  - ✓ Interés en procesos de simulación.
  - ✓ Hereda las estructuras de bloques y de control de ALGOL
  - ✓ Introduce la idea de clase como el "empaquetamiento" de una estructura de datos y las operaciones que actúan sobre ella.
- **SmallTalk (1972): Primer lenguaje O.O.**
  - ✓ Es considerado el lenguaje O.O. "puro"
  - ✓ El objeto es la base del lenguaje
  - ✓ No es sólo un lenguaje, es un entorno de desarrollo completo.

Evolución de los LP (6)

- **Pascal (N. Wirth, 1971): Gran impacto educativo**
  - ✓ Descendiente de ALGOL
  - ✓ Combina simplicidad y expresividad → Seguridad
  - ✓ Su popularidad decrece a partir de mitad de los años 90.
    - Descendientes: Modula 2 y 3, Delphi
- **C (D. Ritchie, 1972): Ampliamente utilizado**
  - ✓ Desarrollo muy relacionado con UNIX
  - ✓ No aporta novedades destacables
  - ✓ *Ausencia de comprobaciones de tipo*

Evolución de los LP (7)

- **Prolog (1972)**
  - ✓ PROgramming LOGic
  - ✓ Un programa consiste en una colección de hechos y reglas
  - ✓ Incluye un motor de inferencia que deduce el resultado a partir de los hechos de entrada.
  - ✓ Interesante pero aplicación reducida.

Evolución de los LP (8)

- **Ada: Historia de un gran esfuerzo**
  - ✓ Nace en el DoD-USA para unificar los lenguajes empleados en sus sistemas "empotrados" (aprox. 500 lenguajes diferentes).
  - ✓ Documento de requisitos en 1975.
  - ✓ A partir de 1977 se desarrolla el lenguaje Ada:
    - Desarrollo por contrato.
    - Participan cuatro empresas, con desarrollos diferentes (todos basados en Pascal).
  - ✓ Estándar en 1983, primer compilador en 1985 (!).
  - ✓ Paquetes, Concurrencia, Excepciones, Genericidad, Comprobación (muy) estricta de tipos.

Evolución de los LP (9)

- ***C*++ (B. Stroustrup, 1980): Imperativo + O.O.**
  - ✓ Lenguaje nuevo aprovechando la sintaxis de *C*.
  - ✓ Objetivo:
    - combinar la P.O.O. con la eficiencia de *C*.
  - ✓ Incorpora biblioteca estándar de clases
  - ✓ Más seguro que *C*... pero no lo suficiente

Evolución de los LP (y 10)

- **Java: SmallTalk con sintaxis de C++ pensando en Internet.**
  - ✓ Origen: programación de pequeños electrodomésticos
  - ✓ Más seguro que C++:
    - Elimina la aritmética punteros
    - Elimina cambios automáticos de tipo
    - Incorpora comprobación de rango de índices en arrays
  - ✓ Incorpora librerías para aplicaciones gráficas y comunicación.
  - ✓ Soporta concurrencia.
  - ✓ Java es independiente de la plataforma.