Presentación
Lenguajes de programación

Grado en Matemáticas

Temas

1. Fundamentos de Programación C1 y C2
2. Variables y tipos de datos C3 y C4
3. Asignaciones y expresiones C5 y C6
4. Control de flujo del programa C7 y C8
5. Subprogramas C9 y C10
6. Estructuras de datos C11 y C12
7. Algoritmos C13 y C14

- Página web de la asignatura

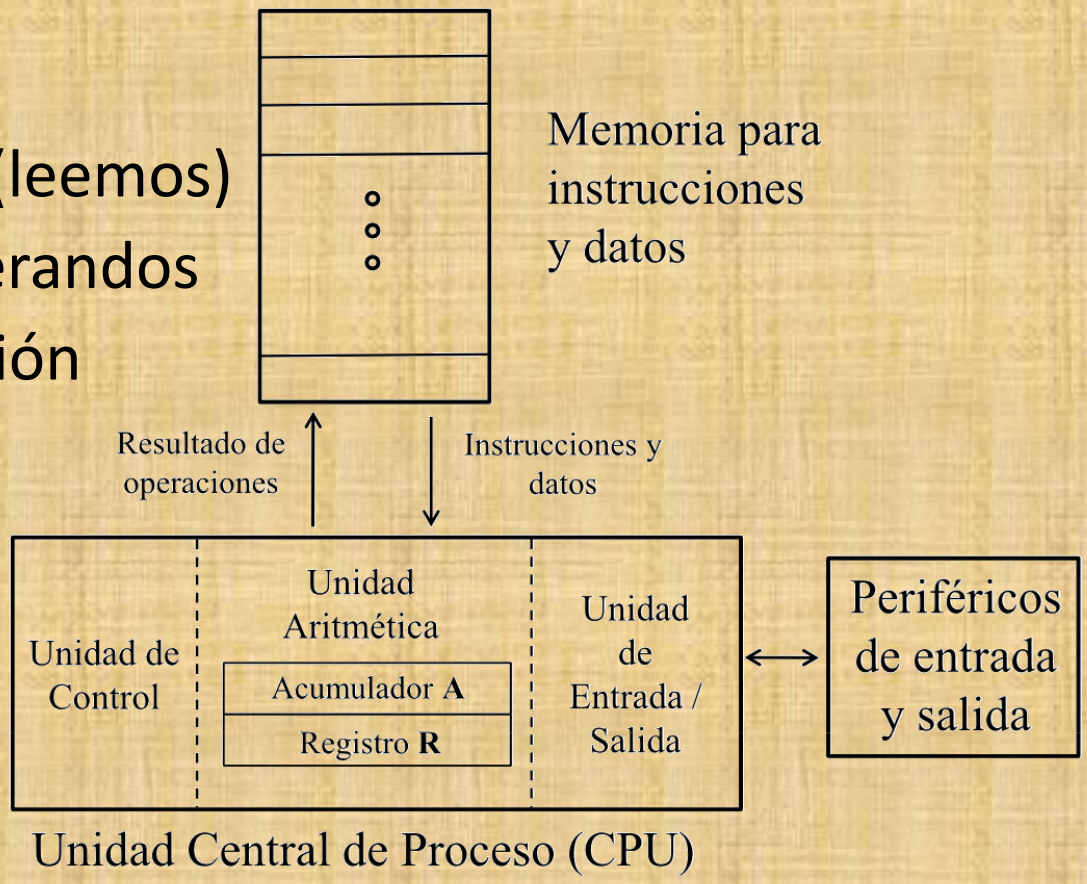
http://www.uned.es/6102210/

Tema 1
Fundamentos de programación

¿Cómo funciona un ordenador?

- máquina de Von Newman:
  - Memoria (Guarda programas y datos)
    - Se accede sabiendo la dirección
    - Se organiza en palabras 00000010101111001010
  - Unidad central de proceso CPU
    - Lee y escribe de la memoria
    - Tiene registros (memoria de gran velocidad y coste)
    - Posee una unidad aritmética (que hace cálculos)
    - Mantiene el contador de programa (PC) y el acumulador
  - Unidad de entrada/salida (Periféricos, ratón, teclado, pantalla, disco…)

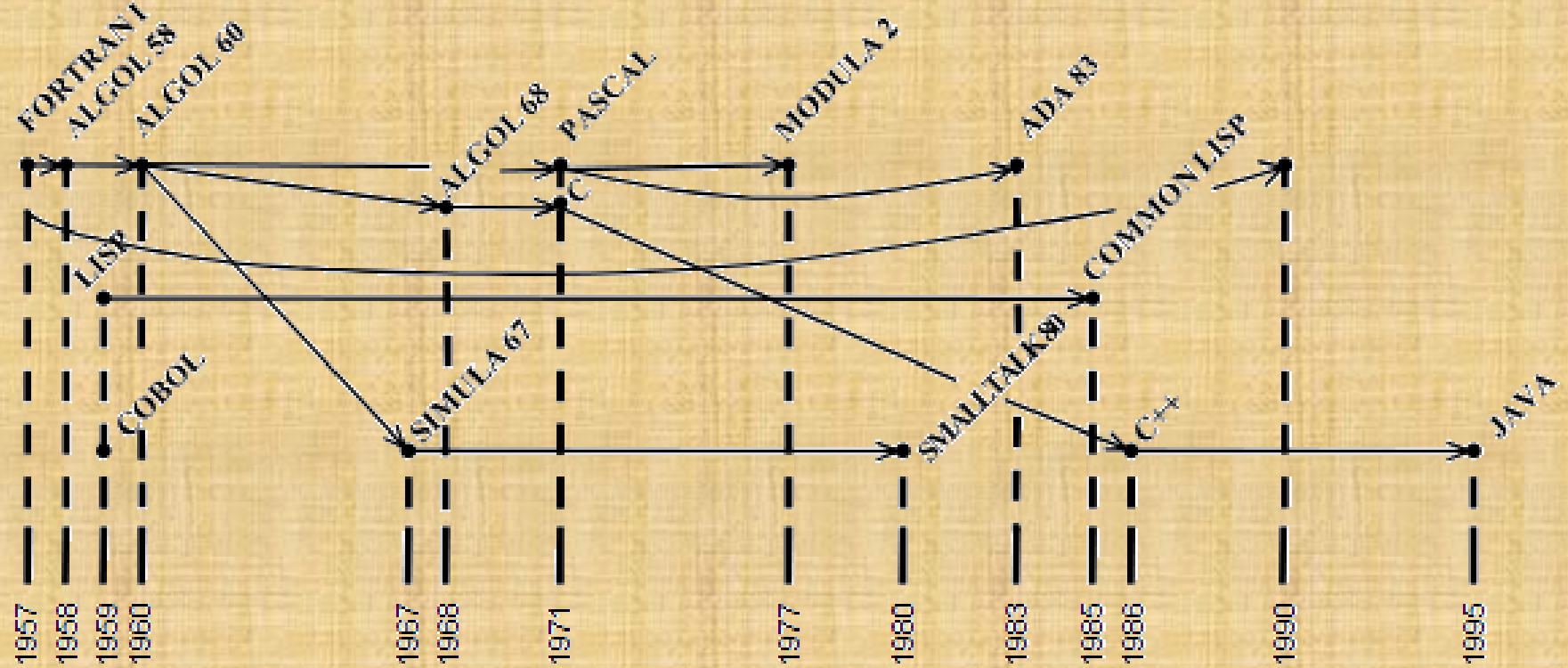
- Ciclo de una instrucción
  - Traemos una instrucción de memoria
  - Actualizamos el PC
  - La descodificamos (leemos)
  - Obtenemos los operandos
  - Hacemos la operación



¿Cómo funciona un ordenador?

- El ordenador ejecuta instrucciones
  - Mover datos (la más usada)
  - Operaciones aritméticas (+ - x : …)
  - Control de flujo (saltos)
    - Condicional (salto si se cumple una condición)
    - Incondicional (Salto siempre)
  - Cada instrucción especifica los datos sobre los que opera
- Las instrucciones están escritas en lenguaje máquina: 00000010101111001010
  - Difíciles de leer y escribir
  - Las posiciones son absolutas (leer posición 175) lo que hace difícil modificar el programa
  - Dependen totalmente de la máquina (no portables)

Evolución de los lenguajes



Lenguaje ensamblador (bajo nivel)

- Hace el lenguaje máquina inteligible
  - Permite asignar etiquetas a posiciones de memoria
  - Es directamente convertible a código máquina
    - Es dependiente del ordenador

```
00000010101111001010 ->  LOAD I
00000010111111001000 ->  ADD J
00000011001110101000 ->  STORE K
```

  - Seria deseable tener un lenguaje más rico donde se pudiera escribir K=I+J que hace lo mismo que antes

Lenguajes de Alto Nivel: FORTRAM

- FORmula TRANslation 1950
  - Parecido al leguaje matemático
  - Lenguaje conciso
  - Fácil de traducir a ensamblador
    - Premia la eficiencia
    - Realiza cálculos en coma flotante
    - Control de flujo inspirado en las instrucciones de IBM704
  - Portable
  - Mejoras II Subrrutinas compilables por separado
    - Más rápido
    - Mejor estructurado

```
!Programa en Fortran 90 almacenado
program CalculoSuma
        implicit none
        real:: x, y, result, suma
        write(*,*)'Introduce el primer sumando'
        read(*,*)x
        write(*,*) 'Introduce el segundo sumando'
        read(*,*)y
         result = suma(x,y)
        write(*,*) 'El resultado es: ', result
end program CalculoSuma  !Fin del programa principal

function suma(x,y) !Subrutina que se puede compilar por separado
        implicit none
        real:: suma, x, y
        suma = x+y
        return
end function suma
```

Lenguajes de alto nivel: ALGOL

- Intenta acercarse a la notación matemática
  - ALGOritmic Language
- Permite identificadores largos (FORTRAM solo 6 caracteres)
- Matrices multidimensionales y dinámicas
- Paso de parámetros a funciones
  - Por referencia
  - Por valor
- Bloques de código y variables locales (ámbito)
- Procedimientos recursivos (Se llaman a si mismos)
- Inspiración para leguajes posteriores
- No definía la E/S

Lenguajes de alto nivel: LISP

- Orientado a la programación funcional
- Especializado en el manejo de listas
  - Buscar
  - Extraer
  - Modificar
  - Borrar
  - Aplicar funciones
- Precursor de Scheme

Lenguajes de alto nivel: COBOL

- Lenguaje para aplicaciones de negocios y defensa 1959
- Sintaxis similar al lenguaje ingles
- Estructuras de datos jerárquicas
- Introduce por primera vez el preprocesador
  - DEFINE

Lenguajes de alto nivel: Prolog

- PROgraming LOGic 1970
- Basado en reglas

  P if Q1 and Q2 and… Qn
- Hechos

  P
- No importa el orden de las reglas o hechos
- El lenguaje implementa un motor de inferencia (deductor) que resuelve preguntas de forma automática

Lenguajes de alto nivel: Simula 67

- Simulación de eventos discretos
- Implementa la orientación a objetos Oo
  - Introduce el concepto de herencia
- Introduce los punteros y la gestión de memoria dinámica
  - La memoria se reserva sobre la marcha
  - La memoria reservada se accede mediante los punteros

Lenguajes de alto nivel: Pascal

- Mejora el control de flujo de FORTRAM eliminando los saltos GOTO (ir hacia) y substituyéndolo por la programación estructurada
  - Los programas son fácilmente legibles de arriba abajo sin pegar saltos (programación espagueti)
  - Es fácil reutilizar el código ya que las subrutinas son fáciles de localizar.

Estructuración de un programa en Pascal

{Programa en Pascal almacenado en un unico fichero}

```
Program CalculoSuma
var
        x, y, result: real;
        Function suma (x: real, y:real):real
        begin
        suma := x + y;
 end;
begin
        writeln('Introduce el primer sumando');
        readln(x);
        writeln('Introduce el segundo sumando');
        readln(y);
        result = suma(x,y);
        writeln('El resultado es: ', result);
end.
```

{Fichero CalculoSuma.pas}

```
#include externs.h
Program CalculoSuma
var
        x, y, result:real;
begin
        writeln('Introduce el primer sumando');
        readln(x);
        writeln('Introduce el segundo sumando');
        readln(y);
        result = suma(x,y);
        writeln('El resultado es: ', result);
end.
```

{Fichero externs.h}

```
function suma(x:real, y:real):real;external;
```

{Fichero funcSuma.pas}

```
#include extern.h
function suma;
        suma := x + y;
        return(suma);
end;
```

Lenguajes de alto nivel: C /C++

- Se usa para crear el Sistema Operativo UNIX en 1972
- Uno de los lenguajes mas utilizados

- Evoluciona en C++ en 1976 incorporando la programación orientada a objetos
  - C++ es compatible con código en C

Lenguajes de alto nivel: Modula 2

- Lenguaje académico de poco uso práctico
- Basado en Pascal y modula
- Desarrollado en 1976

Lenguajes de alto nivel: Ada

- Un lenguaje que supuso un gran esfuerzo de desarrollo
- Creado por defensa EEUU en 1970
- Piensa en la seguridad
  - Permite declarar paquetes
  - Maneja excepciones (ejemplo 1/0)
  - Permite definir unidades de programa genéricas (plantillas a rellenar)
  - Ejecución de tareas (*tasks*)

Lenguajes de alto nivel: Smalltalk

- Primero en introducir la orientación a objetos
  - Clase
  - Objeto
  - Herencia
- Primero en gestionar las interfaces gráficas de usuario (ventanas)

Lenguajes de alto nivel: Java

- Lenguaje moderno y Oo 1990
- Muy utilizado en la programación WEB
- Multiplataforma
- Elimina muchos errores de programación de C++ (oculta los punteros)

Paradigmas de programación

- Paradigmas
  - Imperativos (C, Fortram…)
    - Estructurados
    - No estructurados
  - Funcionales (LISP, Scheme)
  - Lógicos (Prolog)
  - Orientados a objetos (C++, Java)
- Formas de ejecución
  - Interpretado
  - Compilado
  - Híbrido

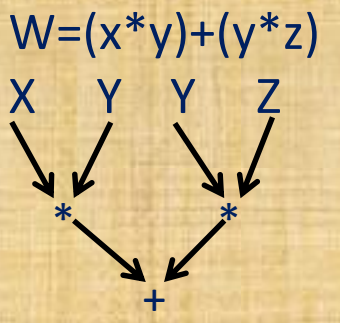
Programación imperativa

- Estructura
  - Instrucción 1
  - Instrucción 2
  - Variable=expresión;
  - If condición GOTO instrucción 1
  - Instrucción n
- Inspirado en la estructura Von Newmann subyacente
  - Es muy eficiente de implementar
- El orden importa
  - Hay un estado global
  - Cada instrucción modifica el estado
  - Las secuencias de control de flujo saltan en la cadena de instrucciones

W=(x*y)+(y*z)
V1=x*y
V2=y*z
W=V1+V2

Programación Funcional

- Basada en la aplicación de funciones a objetos
- Elementos
  - Funciones primitivas
  - Formas funcionales (composición de funciones)
  - Operación que aplica la función
  - Estructuras (típicamente listas) donde almacenar datos
- Se usan en IA
  - Más alto nivel
  - Menos eficientes
- Concurrencia implícita (grafos de evaluación)



Programación lógica

- Sistemas de demostración automática Prolog
- Se usa en sistemas expertos
- Permite codificar reglas y responder problemas por inferencia lógica:
  - Si tos y fiebre entonces gripe
  - Si fiebre y no tos entonces catarro
  - Fiebre, tos
  - ¿gripe?-->si
  - ¿catarro?-->no

Orientación a Objetos Oo

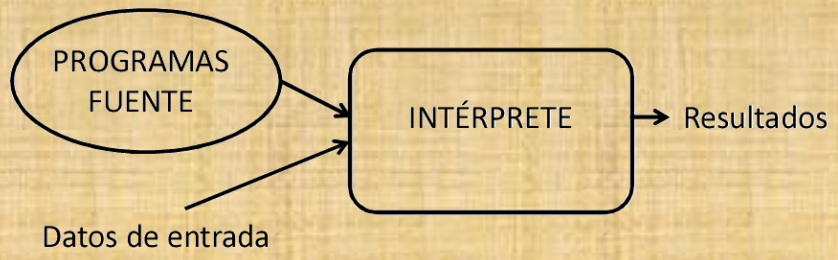
- Encapsula los datos y programas en objetos
  - Abstracción (se puede usar un objeto sin conocer su contenido)
  - Ocultación de información (las variables internas de un objeto no son visibles)
  - Modularidad (los objetos son fácilmente reciclables)
- Conceptos clave
  - Clase: Definición de un objeto:
    - Datos
    - Métodos
  - Herencia: Cuando se crea un objeto a partir de otro este hereda sus métodos
    - Los métodos pueden redefinirse → sobrecarga
  - Ligadura dinámica: cuando un método (sobrecargado) se invoca sobre un objeto el sistema remplaza la llamada por el método adecuado a la clase. (si en tiempo de compilación se conociera el método se podría hacer estáticamente)

Implementación

- Los programas no están solos
  - El Sistema Operativo permite realizar operaciones de E/S y gestionar la memoria, ej:
    - Leer la entrada del teclado
    - Escribir en la consola
    - Abrir una ventana
    - Leer/escribir un archivo
  - El SO provee instrucciones especiales→ Máquina virtual
- La implementación de un lenguaje de programación es una tarea complicada

Interpretación Pura

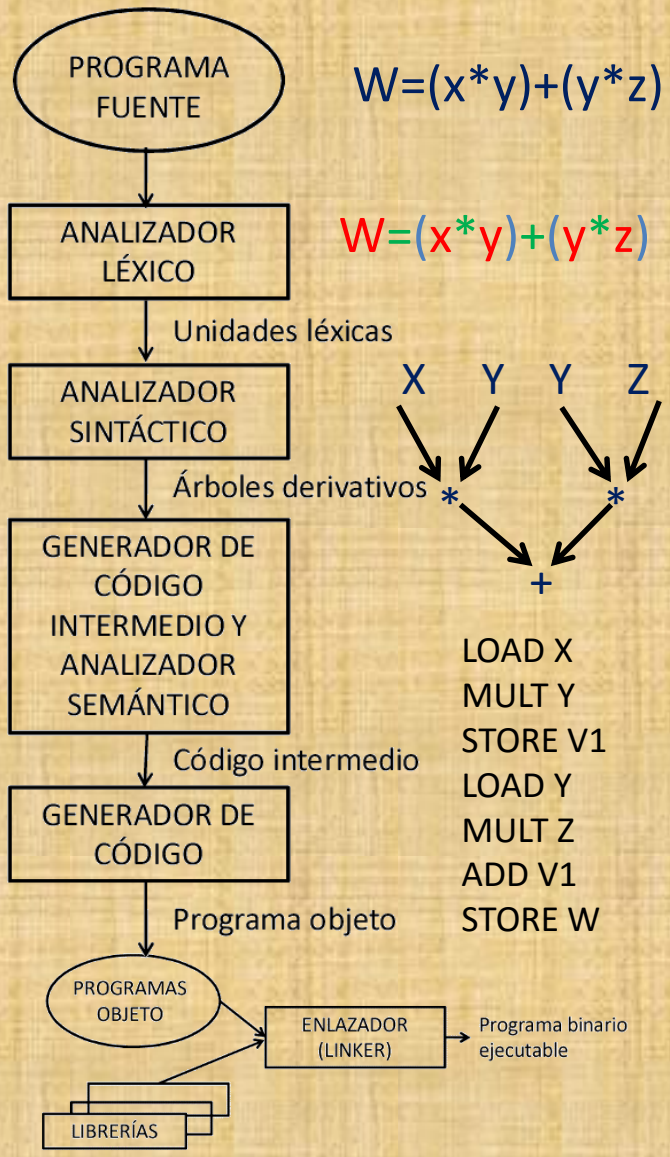
- El programa es un texto (código fuente) escrito en un lenguaje ej: Javascript, PHP…
- El Interprete es un programa que lee línea a línea el código fuente
  - Analiza la sintaxis
  - Realiza la ejecución



- Ventajas
  - Muy fácil de depurar (si falla podemos ver en que línea ha sido)
  - Independiente de la plataforma
- Inconvenientes
  - La interpretación lleva tiempo→ lento

Compilación

- El código fuente se compila a código objeto
- El programa que hace esto se llama compilador ej: C++
- Fases
  - El analizador léxico detecta *tokens*
  - El analizador sintáctico obtiene los árboles derivativos
  - Se genera código intermedio (típicamente ensamblador)
  - El generador de código crea el código máquina (ejecutable) y referencias a otro código (SO)
  - Posteriormente el enlazador enlaza el código con el SO y genera el binario ejecutable
- El proceso es lento pero
  - Se hace una sola vez
  - El ejecutable es muy rápido (hasta x100 interpretado)
  - Si se cambia el SO o la plataforma hay que volver a compilar



Sistema Híbrido

- Igual que la compilación hasta la generación de código intermedio.
- El código intermedio no se pasa a lenguaje máquina sino que se ejecuta en un interprete.
- El interprete es una máquina virtual
  - El programa así compilado es independiente del SO
  - Aproximación usada por Java (*bytecodes*)