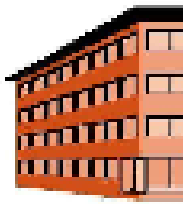
DESARROLLO DE APLICACIONES MULTIPLATAFORMA
PROGRAMACIÓN

UT2. Programación Estructurada y Modular.

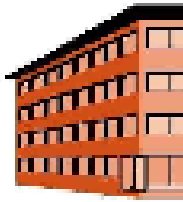
Abraham Pérez Barrera



UT2. Programación Estructurada y Modular.

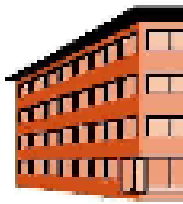
1. Nociones básicas sobre técnicas de programación.
   1.1. Programación estructurada.
   1.2. Programación modular.
2. Estructura general de un programa.
3. Datos.
4. Expresiones.
5. Operadores.
6. Instrucciones.

Abraham Pérez Barrera



1.1. Programación Estructurada.

- La **programación estructurada** está compuesta por un conjunto de técnicas que han ido evolucionando aumentando considerablemente la productividad del programa reduciendo el tiempo de depuración y mantenimiento del mismo.
- Se basa en el **Teorema de la Estructura** desarrollado por **Böhm y Jacopini** que demuestra que se pueden escribir algoritmos y programas usando únicamente tres **estructuras** básicas de **control**: **secuenciales**, **selectivas** y **repetitivas**.
- Utiliza un número limitado de **estructuras de control**, reduciendo así considerablemente los errores.



1.1. Programación Estructurada.

Las herramientas de la programación estructurada son:

- ✓Diseño descendente (top-down) (lo veremos en **Programación Modular**).
- ✓Recursos abstractos (simplicidad).
- ✓Estructuras básicas.

❑Recursos abstractos (simplicidad): consiste en descomponer las acciones complejas en otras más simples capaces de ser resueltas con mayor facilidad.

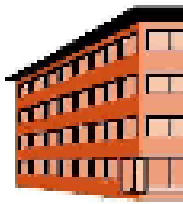
1.1. Programación Estructurada.

Herramientas de la programación estructurada:

- Estructuras básicas: existen tres tipos,
  - **Estructuras secuenciales**: cada acción sigue a otra acción secuencialmente (una detrás de otra). La salida de una acción es la entrada de otra.
  - **Estructuras alternativas**: en estas estructuras se evalúan las condiciones y en función del resultado de las mismas se realizan unas acciones u otras.

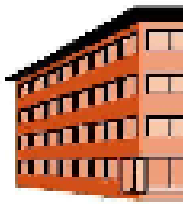
    Ejemplo: Si (se cumple una condición) ENTONCES
  - **Estructuras repetitivas**: son secuencias de instrucciones que se repiten un número determinado de veces.

    Ejemplo: MIENTRAS (se cumpla “algo”) HACER



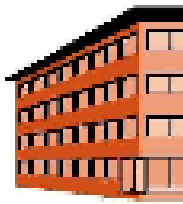
1.1. Programación Estructurada.

- Las principales ventajas de la **programación estructurada** son:
  - ✓Los programas son más fáciles de leer y de entender.
  - ✓Se reduce la complejidad de las pruebas.
  - ✓Aumenta la productividad del programador porque es más fácil de mantener.
  - ✓Los programas quedan mejor documentados internamente.
  - ✓Fácil de codificar en una amplia gama de lenguajes y en distintos sistemas.



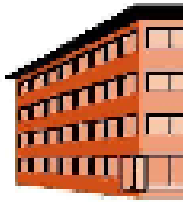
1.1. Programación Estructurada.

- Un programa está **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.
  - ✓ Si todas las instrucciones son ejecutables sin que aparezcan bucles infinitos.



1.1. Programación Estructurada.

- Inconveniente de la **programación estructurada**:
  - Se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático el manejo de su código fuente. Por eso, normalmente se emplea conjuntamente con la **Programación Modular**.
- Aunque las metodologías de **programación estructurada** y de **programación modular** son “antiguas”, son básicas para aprender a programar y aún en la actualidad son muy utilizadas, juntamente con la **programación orientada a objetos**.



1.1. Programación Estructurada.

Actividad 1: ¿cuáles de las siguientes son características de un programa bien **estructurado**?

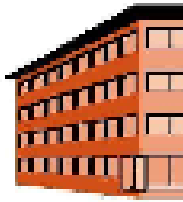
- Todas las sentencias son *alcanzables*, es decir, no contiene *código muerto*, que no se ejecutará nunca.
- No hay ambigüedades: cada sentencia tiene una única interpretación.
- Todos los posibles caminos llevan desde el punto de entrada al de salida.

Actividad 2: cuando escribimos el algoritmo para freír un huevo, ¿qué tipos de estructuras usamos?

1.2. Programación Modular.

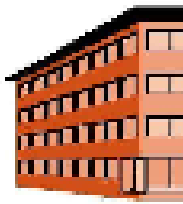
Consiste en dividir un programa en **módulos** o **subprogramas** con el fin de hacerlo más legible y manejable.

- Es una evolución de la programación **estructurada** para solucionar problemas de programación más grandes y complejos de lo que ésta puede resolver.



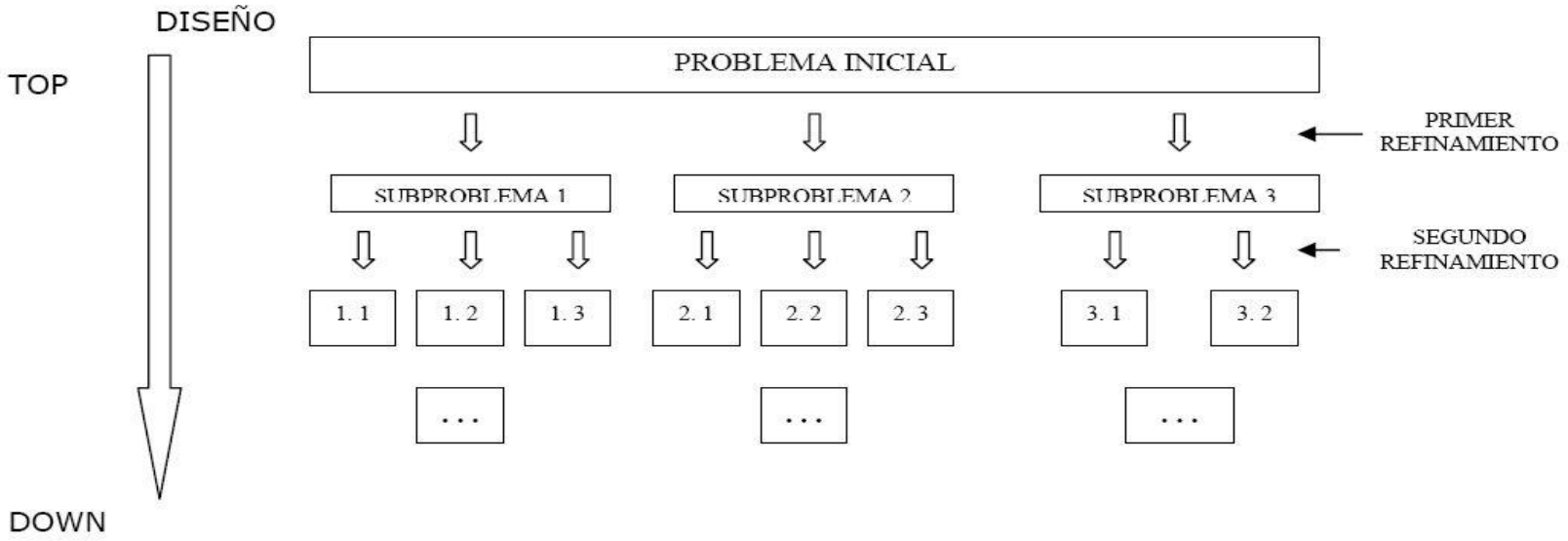
1.2. Programación Modular.

- 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. Ésta técnica se llama refinamiento sucesivo, DIVIDE Y VENCERÁS ó análisis descendente (**Top-Down**).



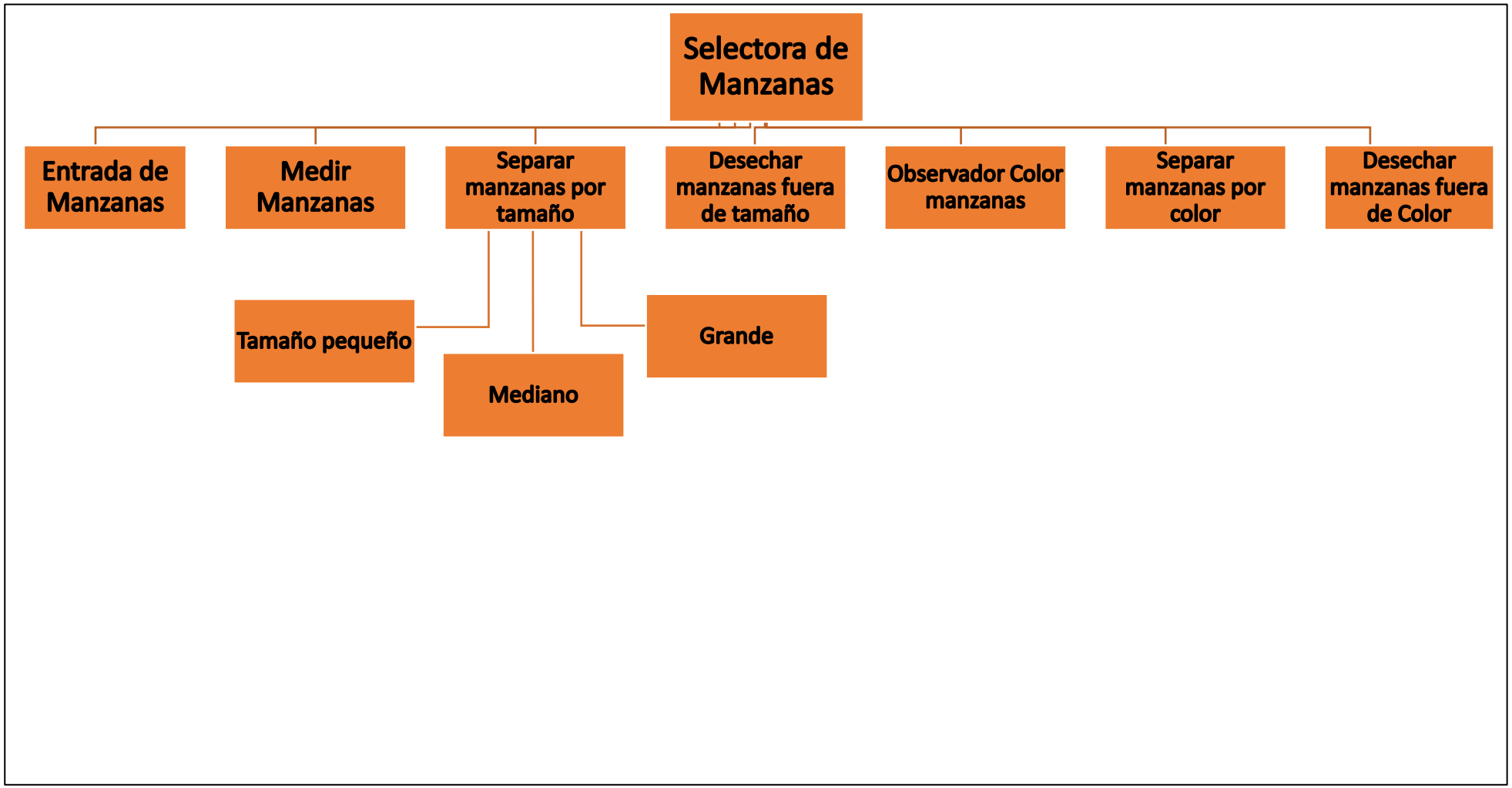
1.2. Programación Modular.

Los programas se diseñan de lo general a lo particular por medio de sucesivos refinamientos o descomposiciones que nos van acercando a las instrucciones finales del programa.



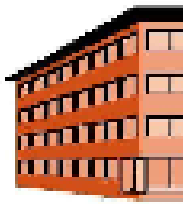
1.2. Programación Modular.

Ejemplo: Maquina selectora de Manzanas



1.2. Programación Modular.

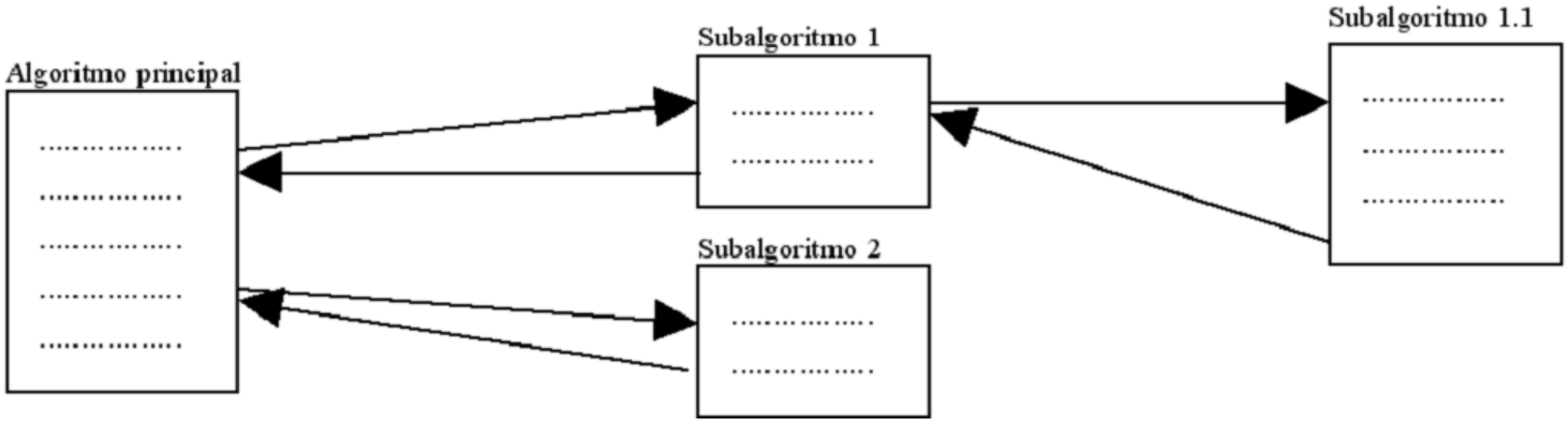
- **Módulo**:
  - Es cada una de las partes de un programa que resuelve uno de los subproblemas en que se divide el problema complejo original.
  - Cada uno de estos **módulos** tiene una tarea bien definida y algunos necesitan de otros para poder operar.
  - En caso de que un **módulo** necesite de otro, puede comunicarse con éste mediante una interfaz de comunicación que también debe estar bien definida.



1.2. Programación Modular.

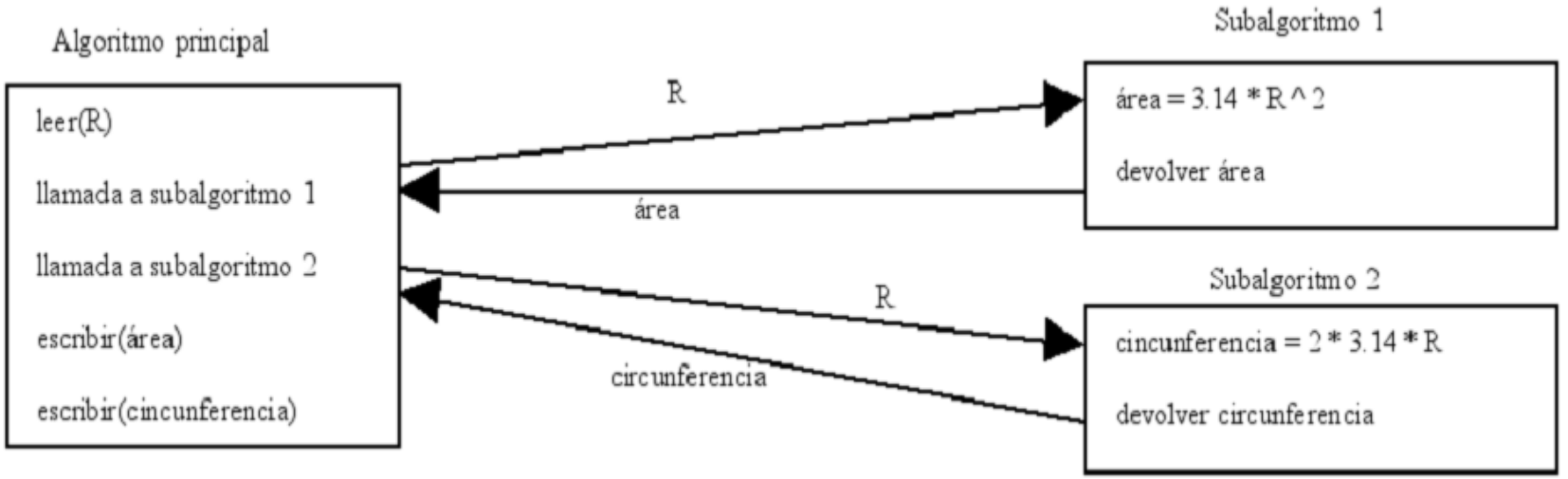
❑Relación jerárquica entre los módulos:

Un **módulo** invoca a los de nivel inferior, recogiendo los resultados que éstos le proporcionen y, así, genera sus propios resultados.



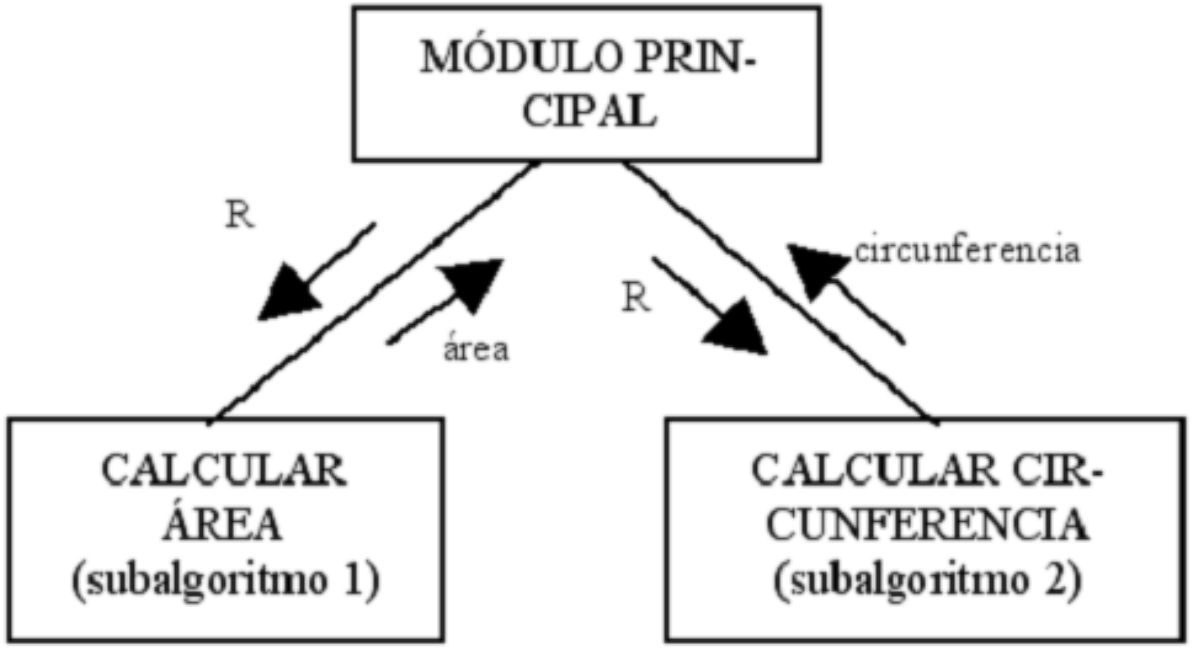
1.2. Programación Modular.

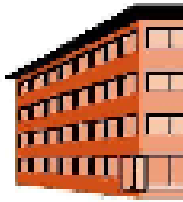
- Relación jerárquica entre los módulos:
  - Los **módulos** deben tener un nombre.
  - Existirá alguna forma de invocarlos.



1.2. Programación Modular.

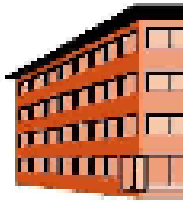
- Relación jerárquica entre los módulos:





1.2. Programación Modular.

- Ventajas de la programación **modular**:
  - Facilita la comprensión del problema.
  - Aumenta la claridad y legibilidad de los programas.
  - Permite que varias personas trabajen simultáneamente.
  - Reduce el tiempo de desarrollo al permitir la REUTILIZACIÓN del **módulo**.
  - Más sencillo el diseño.
  - Más sencillas las pruebas.
  - Más sencillo el mantenimiento.
  - Independiente del lenguaje de programación.



1.2. Programación Modular.

Actividad 3: dado el siguiente diseño **top-down**, ¿qué hace el programa? Valora la descomposición modular realizada.

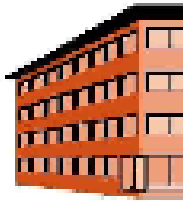
2. Estructura General de un Programa.

- Un **programa** es una secuencia de acciones (**instrucciones**) que manipulan un conjunto de objetos (**datos**).
- Un **programa** consta de tres partes principales:
  - **Entrada:** lectura de **datos**.
  - **Proceso o algoritmo:** donde se tratan los **datos** de entrada.
  - **Salida de resultados:** mostrar los **datos** al usuario.

3. Datos.

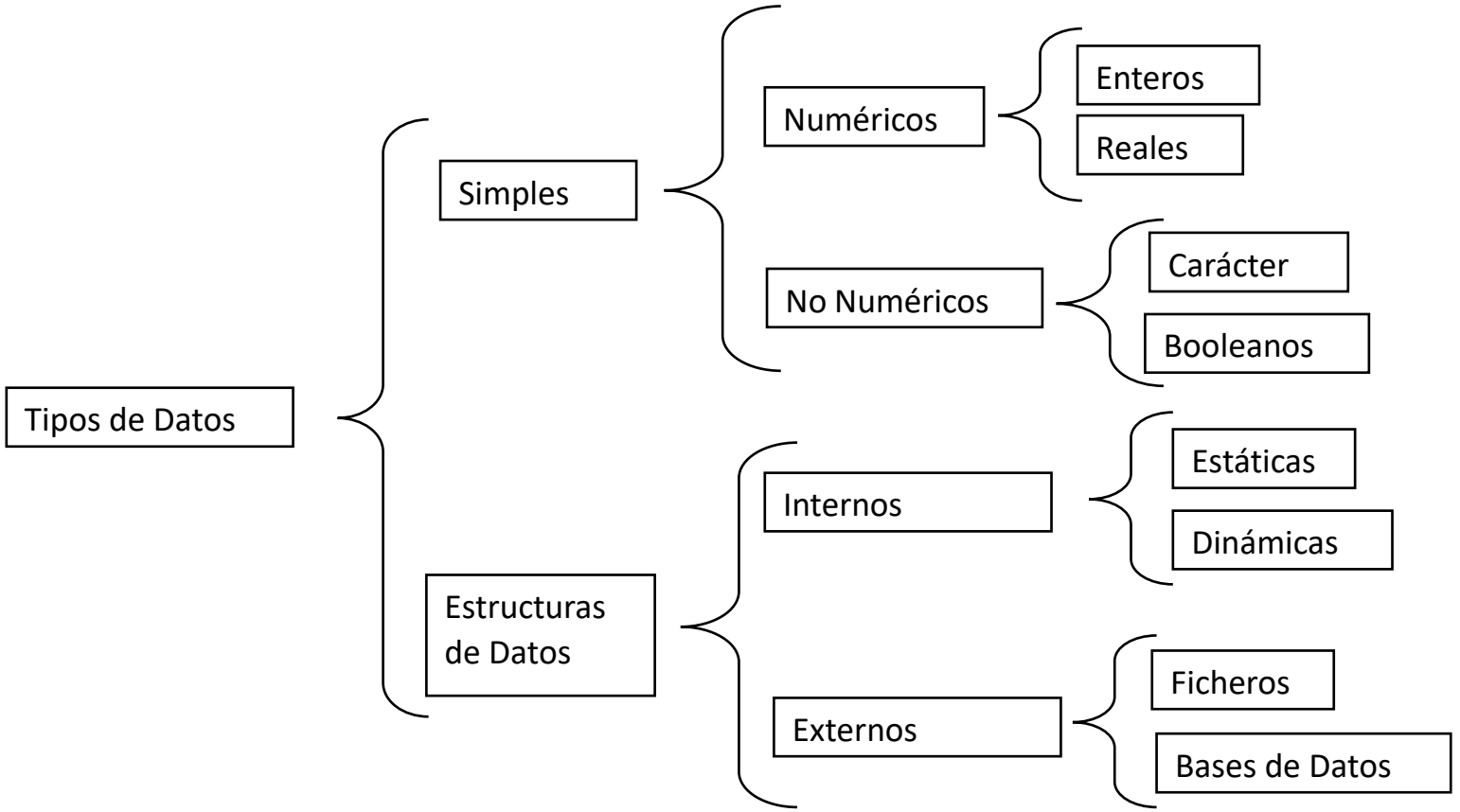
- El **tipo** de un dato es el conjunto de valores que puede tomar durante el programa.

  Si se le intenta dar un valor fuera del conjunto se producirá un error.



3. Datos.

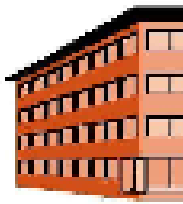
- Los datos que maneja un programa pueden clasificarse como:



3.Datos.

❑Los tipos de datos **simples** son los que no se derivan de otros tipos de datos. Pueden ser:

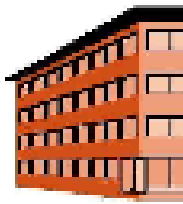
- **Numéricos**: son números reales o enteros.
  - **Enteros**: son los números enteros, se expresan mediante una cadena de dígitos que puede ir precedida del signo (+ o -), el rango lo determina el lenguaje de programación y el ordenador.
  - **Reales**: son los números decimales, pueden ir precedidos del signo (+ o -) y llevan un punto que separa la parte entera de la decimal.
- **No numéricos**: carácter y booleanos o lógicos.
  - **Booleano o lógico**: lo componen los valores verdadero o falso.
  - **Carácter**: son todos los caracteres en general (letras, números, caracteres especiales, cadenas de caracteres,…).



3.Datos.

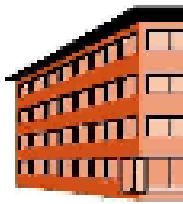
- Los tipos de datos **compuestos** (estructuras de datos) son los que se derivan de los tipos de datos **simples**. Pueden ser:

- **Internas**: los que se almacenan en memoria principal.
  - **Estáticas**: no varían de tamaño en tiempo de ejecución.
  - **Dinámicas**: varían de tamaño en tiempo de ejecución.
- **Externas**: los que se guardan en un medio de almacenamiento externo.
  - **Ficheros**: conjunto de bits almacenados en un dispositivo.
  - **Bases de datos**: conjunto de datos pertenecientes a un mismo contexto y almacenados siguiendo un criterio para su posterior uso.



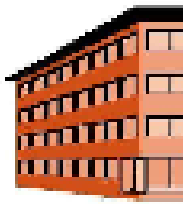
4. Expresiones.

- Una **expresión** se suele representar por la unión de varios operandos, variables, constantes y/o literales, unidos entre sí por operadores que realizan una acción sobre ellos, ya sea de relación, comparación o aritmética.
- Se llama **operandos** a los datos que intervienen en las expresiones, estos datos pueden aparecer de forma explícita, como **constantes** o hacer referencia a ellos por medio de **variables**.



4. Expresiones.

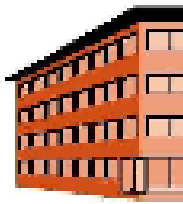
- **CONSTANTES**:
  - Es un valor que no puede ser alterado durante la ejecución de un programa. Ejemplos: pi (3,1416…), velocidad de la luz (300.000 km/s).
  - Corresponde a una longitud fija de un área reservada en la memoria del ordenador.
- **VARIABLES:**
  - El valor puede ser modificado a lo largo de la ejecución del programa.
  - Es un área reservada en la memoria principal del ordenador.



5. Operadores.

❑ Para la construcción de expresiones, se pueden utilizar los siguientes operadores:

- **Aritméticos**:
  - ^ Potencia. 2^3, a^b
  - * Producto. 2*3
  - / División.
  - +, -
  - %: resto de la división.
- **Alfanuméricos**: concatenación (+). Ejemplo: (“El valor de una variable es” + variable)
- **Relacionales**: se utilizan para realizar comparaciones. Son: == (igual a), <, <=, >, >=, != (distinto).
- **Lógicos**: NOT, AND, OR.
- **Paréntesis**. ( ). Anidan expresiones.

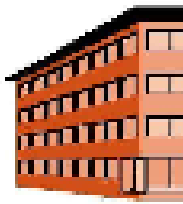


5. Operadores.

- El orden en que se realizan las operaciones es fundamental para determinar el resultado de una expresión. Los operadores de una expresión se evalúan según el siguiente orden (de mayor a menor precedencia):

1. Paréntesis. De más interno a más externo.
2. Potencias.
3. Productos y divisiones.
4. Sumas y restas.
5. Concatenaciones.
6. Relacionales.
7. Negación.
8. Conjunción(AND).
9. Disyunción (OR).

En igualdad de precedencia, de izquierda a derecha.

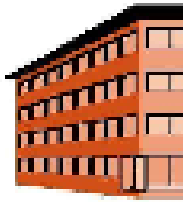


5. Operadores.

Ejemplos: evaluar las siguientes expresiones:

((3+2)^2-15)/2*5
(5^2-15)/2*5
(25-15)/2*5
10/2*5
5*5
25

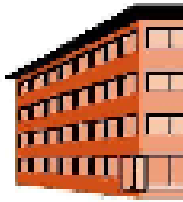
5-2>4 AND NOT 0,5 == 1/2
5-2>4 AND NOT 0,5 == 0,5
3>4 AND NOT 0,5 == 0,5
FALSO AND NOT 0,5 == 0,5
FALSO AND NOT CIERTO
FALSO AND FALSO
FALSO



5. Operadores.

- Las expresiones, según el resultado que obtengan, se clasifican en:

- **Numéricas**: obtienen resultados de tipo numérico. Se construyen mediante los operadores aritméticos. Ejemplos: PI*2*r^2, x+y.
- **Alfanuméricas**: usan operadores alfanuméricos. Se construyen mediante el operador + (concatena). Ejemplo: “Don” + N + AP.
- **Booleanas o lógicas**: usan los operadores lógicos y relacionales. Producen resultados CIERTO o FALSO. Ejemplos: A>B AND B>=C, A>0.



5. Operadores.

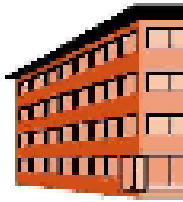
Actividad 7: ¿a qué tipo de datos pertenece cada uno de los siguientes datos? -5, 7.02, «a», falso.

Actividad 8: pon un ejemplo de dato constante y otro de dato variable.

Actividad 9: en qué orden se evaluarán las siguientes expresiones:

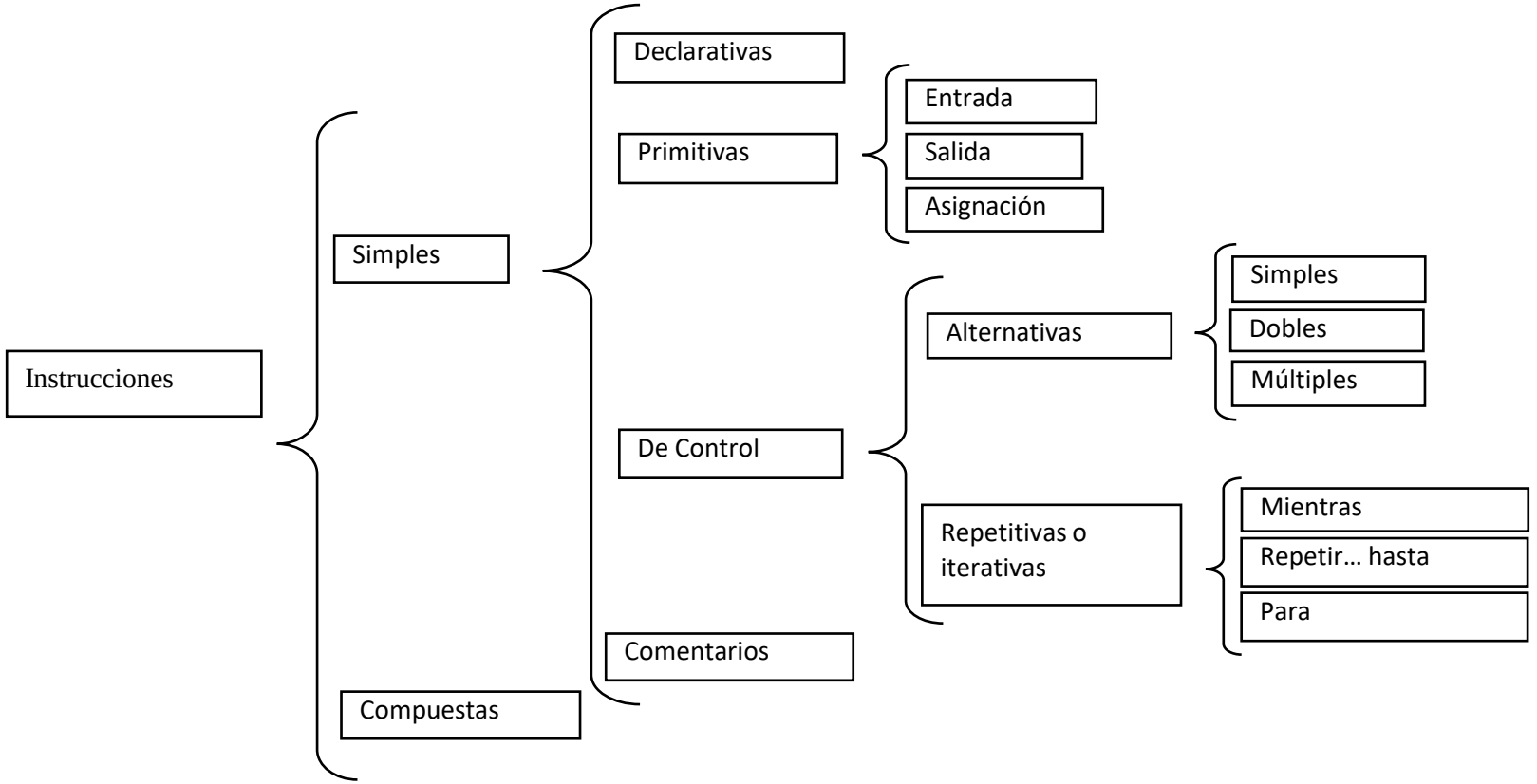
a) -b + sqrt(b^2 - 4*a*c)

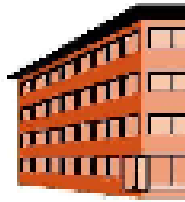
b) (a >= 0) and ((b+5) > 10)



6. Instrucciones.

- Según la función que desempeñan las **instrucciones** pueden ser:





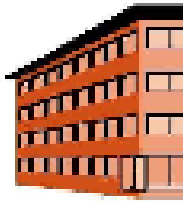
6. Instrucciones.

❑**De DECLARACIÓN**

➢Antes de poder usar una variable o una constante hay que declararla, es decir, hay que darle un nombre y asignarle un tipo.

➢Ejemplos:

- ❖Cociente: real
- ❖PI=3,1416
- ❖Contador: entero
- ❖Activo: booleano



6. Instrucciones.

❑**PRIMITIVAS**.

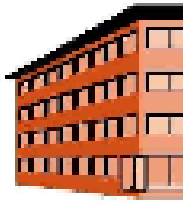
➢Aquellas que ejecuta el procesador de modo inmediato. Las principales son: asignación, de entrada y de salida.

- ❖**ASIGNACIÓN**: se utilizan para valores a una variable dentro del programa. Ejemplos: edad=16, casado=verdadero, nombre="Juan".
- ❖**De ENTRADA**: se utilizan para tomar datos del exterior y guardarlos en variables . Ejemplos: leer edad, leer numero.
- ❖**De SALIDA**. se utilizan para presentar en pantalla o impresora mensajes, contenidos de las variables,… Ejemplos: escribir edad, imprimir numero.

6. Instrucciones.

- **De CONTROL**: sirven para modificar el rumbo de ejecución de un programa según una cierta condición.
  - De SELECCIÓN: permiten ejecutar unas instrucciones u otras según se cumpla una condición o no.
  - De REPETICIÓN: repite las instrucciones un cierto número de veces.
  - De SALTO: permiten salir a un determinado punto del programa saltándose instrucciones.

  (Las veremos extensamente más adelante).
- **COMENTARIOS**: se usan para favorecer la comprensión del programa.
- **COMPUESTAS**: formadas por la unión de instrucciones simples. No se ejecutan de modo inmediato. Ejemplo: 2*PI*R^2



6. Instrucciones.

Actividad 10: ¿de qué tipo son las siguientes instrucciones?

a) Radio=6

b) Leer nombre

c) Escribir nombre

d) Si (edad<18) entonces

escribir «Es menor de edad»

a) Radio : real