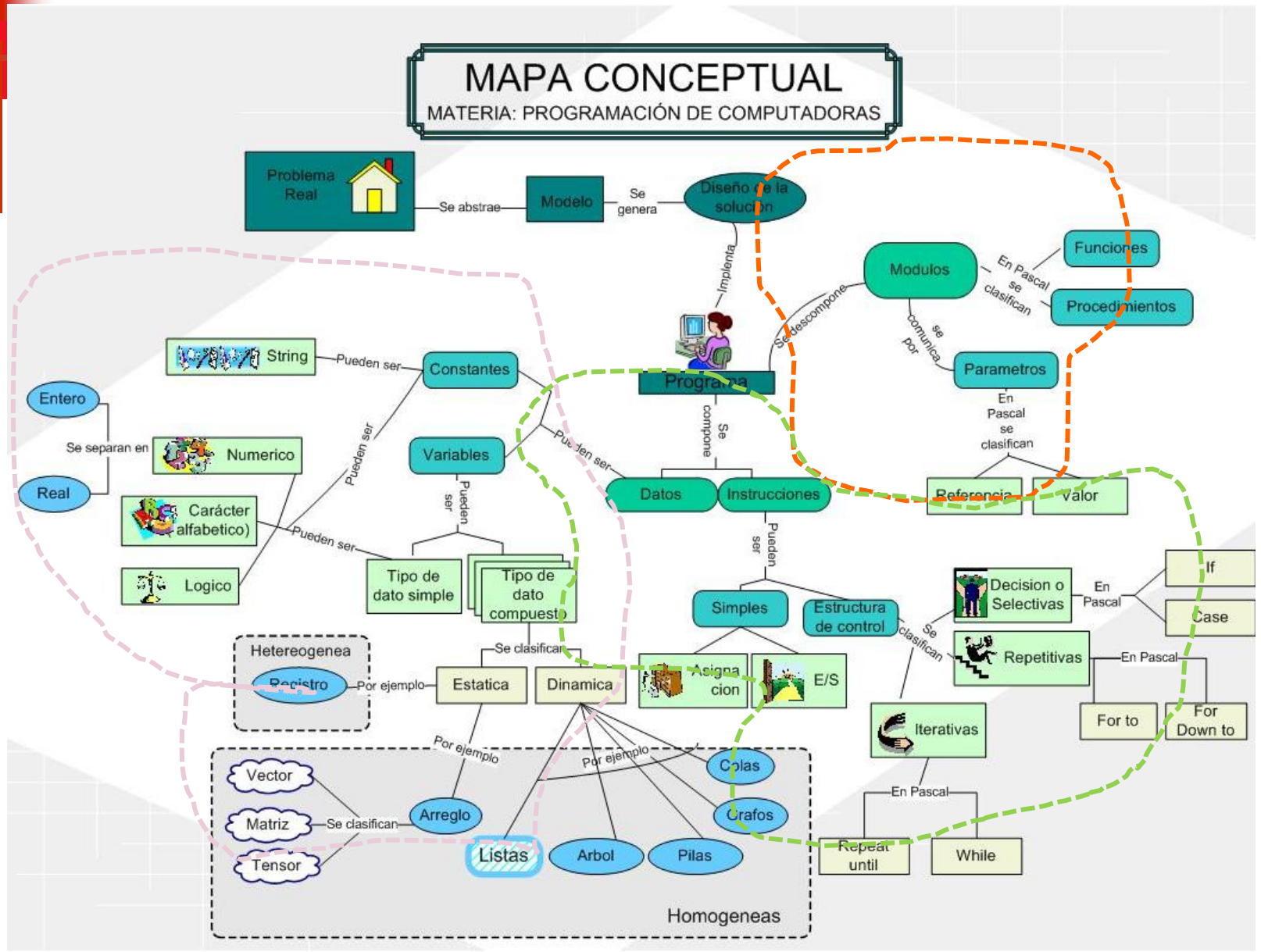
# Ejercitación con árboles


MAPA CONCEPTUAL
MATERIA: PROGRAMACIÓN DE COMPUTADORAS
Problema Real
Se abstrae
Modelo
Se genera
Diseño de la solución
Implenta
Programa
Modulos
Se descompone
En Pascal se clasifican
Funciones
Procedimientos
se comunica por
Parametros
En Pascal se clasifican
Referencia
Valor
Se compone
Datos
Instrucciones
Pueden ser
String
Pueden ser
Constantes
Entero
Se separan en
Numerico
Real
Carácter alfabetico)
Logico
Pueden ser
Variables
Pueden ser
Tipo de dato simple
Tipo de dato compuesto
Se clasifican
Hetereogenea
Registro
Por ejemplo
Estatica
Dinamica
Simples
Estructura de control
Asigna cion
E/S
Se clasifican
Decision o Selectivas
En Pascal
If
Case
Repetitivas
En Pascal
For to
For Down to
Iterativas
En Pascal
Repeat until
While
Por ejemplo
Por ejemplo
Vector
Matriz
Tensor
Se clasifican
Arreglo
Listas
Arbol
Pilas
Colas
Grafos
Homogeneas

# Ejercitación Arboles

## Ejemplo:

Se tiene una lista de personas anotadas en una maestría ordenada por apellido.

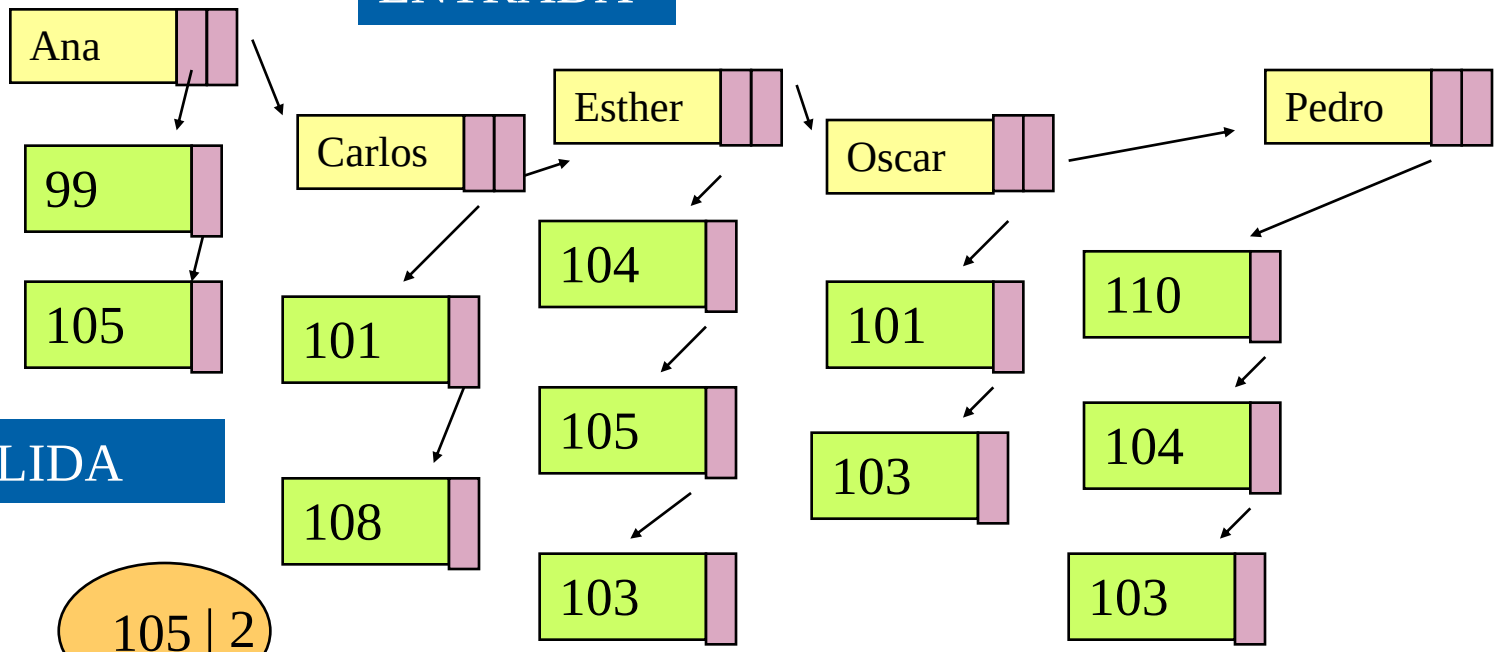
Para cada persona se tiene una lista de los cursos ya aprobados (los cursos aparecen identificados por su código).

Se pide generar un árbol binario ordenado de cursos donde para cada curso se tenga la cantidad de alumnos que lo aprobó.

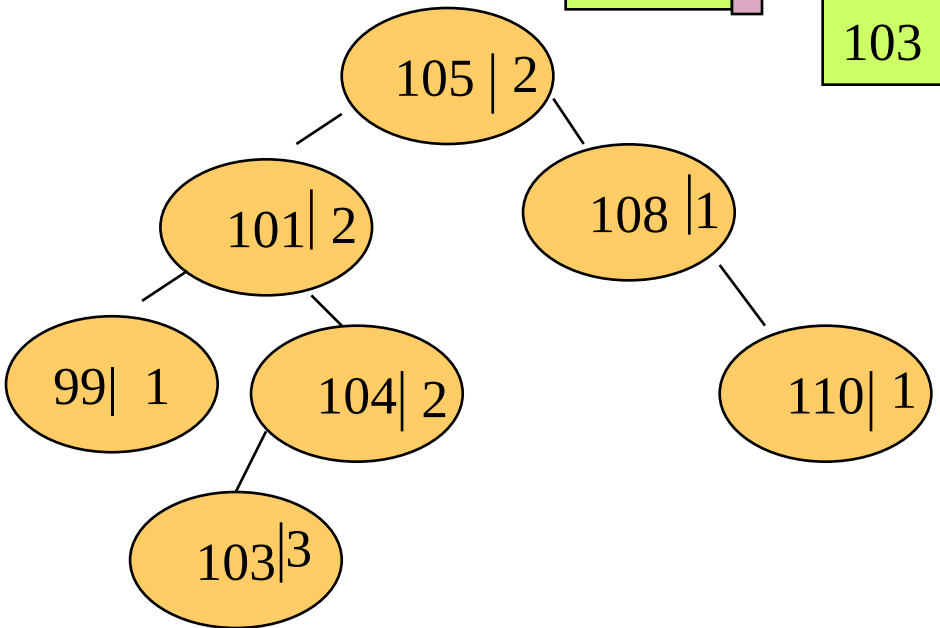
Nota: el orden del árbol es por código de curso.

# Ejercitación Arboles

ENTRADA




SALIDA




¿Qué pasa si el alumno no tiene ninguna materia aprobada?

# Ejercitación Arboles

**Una solución posible…**

*{Recorrer la lista de alumnos}*

*{Recorrer la lista de cursos del alumno}*

*{Buscar el curso en el árbol binario}*
*{Si está entonces aumentar en 1 el total de alumnos del curso}*
*{Sino Insertar en el árbol binario ordenado}*

# Ejercitación Arboles

```
Type
  ListaCursos = ^nodocurso;

  Nodocurso = record
                  Codigo:string [4];
                   Sig : ListaCursos;
                End;

 ListaAlumnos = ^Alumno;

 Alumno = record
              Apellido: string [20];
              Cursos: ListaCursos;
              Sig : ListaAlumnos;
            End;

ArbolCursos = ^NodoArbol;

NodoArbol =  record
                  Codigo : string[4];
                  Cant : integer;
                  Izq : ArbolCursos;
                  Der: ArbolCursos;
                End;
```

```
Var
  LC, listacur: ListaCursos;
  LA, listaalu: ListaAlumnos;
  AC, arbol : ArbolCursos;
```

# Ejercitación Arboles

```
Begin
 {crear la lista de alumnos con los cursos aprobados}
 {Recorrer la lista de alumnos}
    Listaalu := LA;
   AC := Nil;
   While (Listaalu <> Nil) do begin
{Recorrer la lista de cursos del alumno}
     Listacur := Listaalu^.cursos;
     While (Listacur<> Nil) do begin
         {buscar el curso en el árbol binario}
          arbol := Buscar (AC, listacur^.codigo);
            {Si está entonces aumentar en 1 el total de alumnos del curso}
          If arbol <> nil then arbol^.cant := arbol^.cant + 1
                        {Sino Insertar en el árbol binario ordenado}
                              Else Insertar (AC, listacur^.codigo);
          Listacur := listacur^.sig;
      End;
     Listaalu := listaalu^.sig;
   End;
End.
```

# Árboles Binarios Ordenados - Operaciones

- Operación Buscar: localiza un elemento X en el árbol

**Function Buscar (A: ArbolCursos;  x:integer) : ArbolCursos**

Esta operación retorna un puntero al nodo en el árbol A que tiene valor "x" o Nil si no existe.

- Operación Insertar:

**Procedure Insertar (Var A : ArbolCursos; dato : integer);**

Inserta el código en el árbol A. Es igual al insertar de la teoría solo que tendríamos que poner el campo cant en 1 (uno) dentro del código del caso base (cuando llegamos a A = nil).

**Ver su implementación en la teoría**

# Ejercitación Arboles

a) Informar la cantidad de alumnos que tienen los cursos con código menor a 105.

```
Procedure cantidades (a: arbolcursos);
  begin
    If a <> Nil then
                    If (a^.codigo < 105) then begin
                                                write (a^.cant);
                                                Cantidades (a^.der);
                                                Cantidades (a^.izq);
                                              End
                                              Else cantidades (a^.izq);
 End;
```

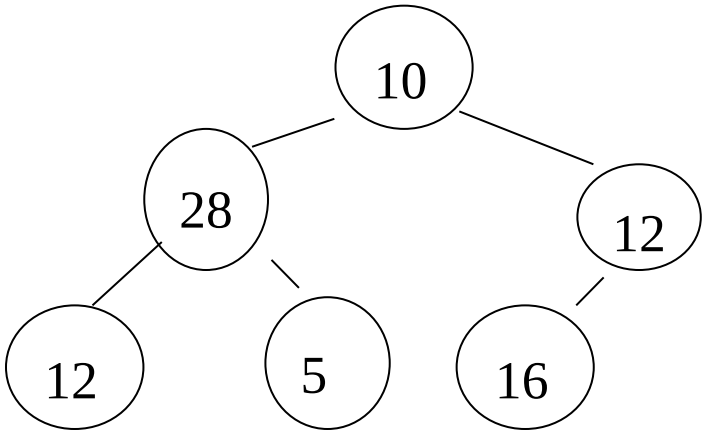
# Ejercitación Arboles

b) Informar cuales son las cantidades de alumnos de los cursos con código comprendidos entre 105 y 110.

```
Procedure intervaloCantidades (a: arbolcursos);
  begin
    If a <> Nil then
                  if (a^.codigo >= 105) then
                          if (a^.codigo <= 110) then begin
                                                      Write (a^.cant);
                                                      intervaloCantidades (a^.izq);
                                                      intervaloCantidades (a^.der);
                                                    End
                                                    Else intervaloCantidades (a^.izq);
                                                Else intervaloCantidades (a^.der);
 End;
```

# Ejercitación Arboles

2. Dado un árbol binario donde cada uno de sus nodos tiene como dato un número entero. Ejemplo:10 28 12 5 16 12




a) Imprimir el peso de cada rama del árbol en términos de la suma de los valores que contienen sus nodos.

# Ejercitación Arboles

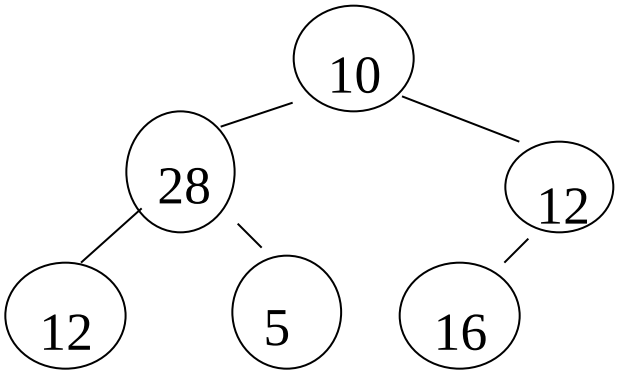
```
type
 Arbol = ^nodo;
  Nodo = record
                Dato: integer;
                Izq: arbol;
                Der: arbol;
              End;
Var
    A: arbol; cant: integer;
    max : integer {para parte (b)}
Procedure Imprimir (a: arbol; cant: integer);
  Begin
      Cant := cant + a^.dato;
      If (a^.izq <> Nil) then Imprimir (a^.izq, cant);
      If (a^.der <> Nil) then Imprimir (a^.der, cant);
      If (a^.izq = Nil) and  (a^.der = Nil) then
                  write ("La suma de la rama es:", cant);
  End;
```

```
{prog. ppal}
Begin
....
Cant := 0;
If (a <> nil) then
    Imprimir (a, cant);
...
End.
```

# Ejercitación Arboles

2.b)Supongamos que se desea calcular la rama del árbol que tiene más peso en términos de la suma de los valores que contienen sus nodos. Se pide informar la suma máxima.

Resultado: La rama que tiene máxima suma es la que totaliza 50

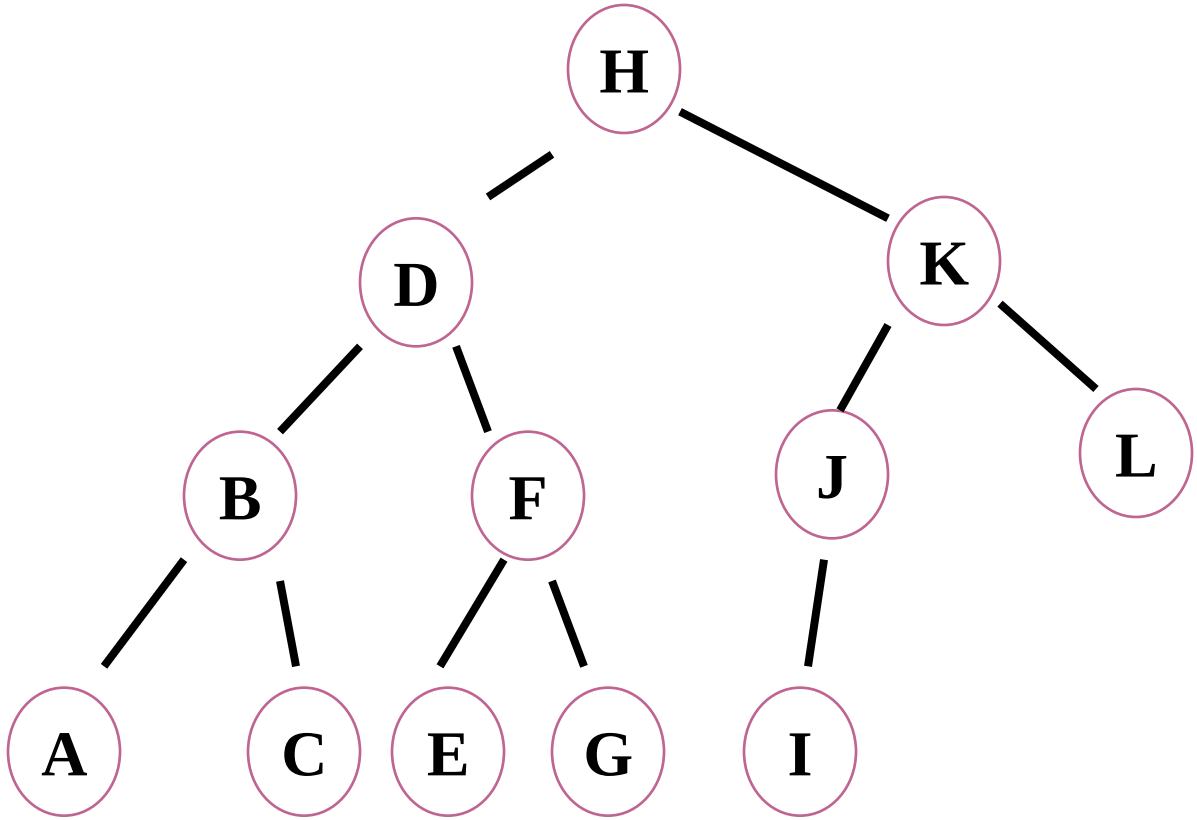


# Ejercitación Arboles

```
Procedure ImprimirModificado (a: arbol; cant: integer;
                                        var  max:integer);
  Begin
     Cant := cant + a^.dato;
     If (a^.izq <> Nil) then ImprimirModificado (a^.izq, cant, max);
     If (a^.der <> Nil) then ImprimirModificado (a^.der, cant, max);
     If (a^.izq = Nil) and  (a^.der = Nil) then
                                          If cant > max then max := cant;
   End;


{Prog Ppal}
 Begin
     ....
     max:= -1; Cant := 0;
     if (a <> nil) then ImprimirModificado (a, cant, max);
     Write (max);
     ...
 End.
```

# ABB – Otra representación



# ABB – Otra representación

**Una posible representación:**
**Se numeran los nodos de arriba abajo y de izquierda a derecha (nivel por nivel).**

**Se crea un arreglo conteniendo la información de los nodos del árbol binario.**

**Se pueden establecer algunas relaciones importantes:**

# ABB – Otra representación

- **El hijo a izquierda de A[i] --> A[2i] si 2i<=n.**
- **El hijo a derecha de A[i] --> A[2i+1] si 2i+1<=n.**
- **El padre de A[i] --> A[i div 2] si i >1.**
- **Se puede asegurar que A[i] es una hoja si y solo si 2i >n. En nuestro ejemplo 2i >12, es decir i >6-->los nodos A[7..12] son hojas.**
- **La raíz es A[1] --> si A no está vacío.**