Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 3

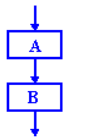
Composición de Algoritmos Secuencial y Condicional

Noticias

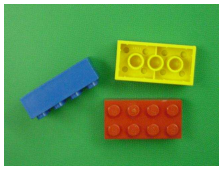
- **Hoja Aparte**
  - Publicación semanal de la UNRC con Noticias Universitarias en general. Sale todos los viernes y es gratuita. Se puede conseguir en el Comedor, Biblioteca, Facultad, Centro de Estudiantes, etc.

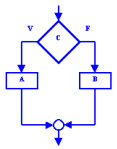
- **www.unrc.edu.ar**
  - Sitio institucional de la UNRC. Noticias diarias, información sobre aulas y horarios, becas, eventos, enlaces a sitios de interés, Biblioteca, Campus Virtual SIAT, publicaciones, Facultades, Carreras, etc.

Tipos de Composición de Algoritmos

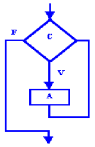


- Composición Secuencial





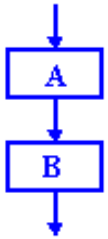
- Composición Condicional (Decisión, Selección)



- Composición Iterativa (Repetición)

Composición Secuencial

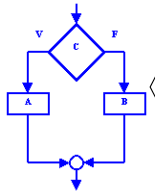
Es el tipo de composición más simple, está representada por una ***sucesión de acciones u operaciones*** (por ej. asignaciones), que se realizan una después de la otra, es decir, que el orden de ejecución coincide con el orden físico de aparición de las mismas, es decir, de arriba hacia abajo.



Las cajas A y B pueden ser definidas para ejecutar desde una simple acción hasta un módulo o algoritmo completo.

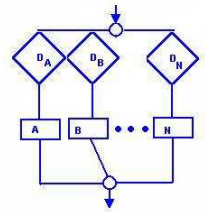
Composición Condicional

En un algoritmo representativo de un problema real es casi imposible que todo sea secuencial. Es necesario tomar ***decisiones*** en función de los datos del problema. La toma de decisión puede ser entre ***dos*** *o* ***más*** alternativas.



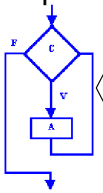
C es una condición que se evalúa. **A** es la acción que se ejecuta cuando la evaluación de este predicado resulta **verdadera** y **B** es la acción ejecutada cuando es **falsa**.

D_A, D_B,..., D_N son decisiones a tomar. **A** es la acción que se ejecuta cuando la **D_A** es verdadera. **B** es la acción ejecutada cuando **D_B** es verdadera. **N** es la acción ejecutada cuando **D_N** es verdadera.



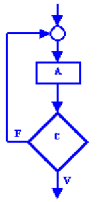
Composición Iterativa

En muchas ocasiones es necesario realizar una a varias acciones de manera repetida, ya sea una cantidad predeterminada de veces o no. La composición iterativa, en todas sus alternativas posibilita la ejecución repetida de un bloque de acciones. Veamos algunos ejemplos.



C es una condición que se evalúa. **A** es la acción o secuencia de acciones que se ejecuta cuando la evaluación de este predicado resulta **verdadera**. En caso de que la evaluación resulte **falsa**, **A** no se ejecuta más y **se continúa con la estructura siguiente**.

A es la acción o secuencia de acciones. **C** es una condición que se evalúa. Cuando la evaluación de este predicado resulta **falsa, A vuelve a ejecutarse**. Si la evaluación resulta **verdadera**, A no se ejecuta más y **se continúa con la estructura siguiente**.



Composición Secuencial

Debido a la sencillez de este tipo de composición, veremos su utilización a través de un ejemplo.

Más adelante veremos que es posible componer secuencialmente las demás estructuras, otros módulos, acciones y funciones.

"A partir del precio bruto de un producto, calcule el precio final del mismo, y luego informe el valor resultante. El precio bruto es aportado por el usuario. El precio final del producto se obtiene agregando el 21% (IVA) al precio bruto"

Composición Secuencial
Análisis del Problema

Preguntas concernientes a la entrada:
- ¿Cuáles y cuántos son los valores de entrada?
- ¿Cómo se llaman esos datos?
- ¿Cuáles son valores válidos de entrada?

Preguntas concernientes a los resultados (salida):
- ¿Cuáles y cuántos son los valores del resultado?
- ¿Cómo se llaman esos datos?
- ¿Cuáles son valores válidos del resultado?

Relación entre datos de entrada y resultados:

- **Dato/s**: precio bruto es un número real+ (precioBruto) e IVA un número entero+
- **Resultado/s**: precio final es un número real+ (precioFinal)
- **Relación entre datos de entrada y resultados**:

precioFinal = precioBruto + (precioBruto*IVA)/100

Composición Secuencial
Diseño del Problema

Representaremos el precio bruto como una variable y el IVA como una constante.

Las acciones a ejecutar son: ingresar el precio bruto, calcular el precio final y mostrar el precio final. El algoritmo debe ejecutar estas acciones de manera secuencial y en este orden.

```
Algoritmo CalcularIva
Lexico
  Iva = 21              {constante para el Iva}
  precioBruto ∈ R       {variable dato}
  precioFinal ∈ R       {variable resultado}
Inicio
  Leer(precioBruto)
  precioFinal ← precioBruto + (precioBruto*Iva)/100
  Escribir('El precio final es: ', precioFinal)
Fin
```


Composición Secuencial
Implementación del Problema

```
PROGRAM CalcularIva;
CONST
  Iva = 21;                    {constante IVA}
VAR
  precioBruto: Real;     {variable dato}
  precioFinal: Real;     {variable resultado}
BEGIN
  READ(precioBruto);
  precioFinal := precioBruto + (precioBruto*Iva)/100;
  WRITELN('El precio final es: ', precioFinal)
END.
```


Composición Condicional

Existen diversas formas de composición condicional. **Una de las tareas en la etapa de diseño es la elección de la forma más adecuada** para el problema en cuestión.

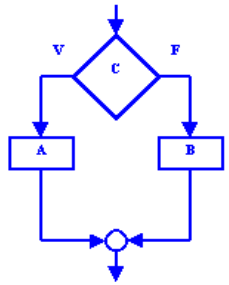
- Dos casos:
  - **si...entonces...sino**
- Múltiples casos :
  - **segun ...**
  - **segun ... otros**
- Un caso (condición excepcional):
  - **si...entonces...**

Veremos a continuación cada una de ellas...

Composición Condicional
si ... entonces ... sino

Sintaxis:

si <condicion>
entonces {comentario$_1$}
  <acciones$_1$>
sino {comentario$_2$}
  <acciones$_2$>
fsi



Semántica: Se evalúa la **condición**. Cuando la evaluación de la **condición** es **verdadera**, se ejecutan las **acciones** del **entonces** (acciones$_1$), únicamente. Cuando la evaluación de la **condición** es **falsa**, se ejecutan las **acciones** del **sino** (acciones$_2$), únicamente.

Composición Condicional
si ... entonces ... sino - Ejemplo

Detectar si un número introducido por el usuario es positivo o negativo.

Datos: numero es un número entero (num)
Resultados: num es positivo o negativo.
Relación entre e/s: num es positivo si el mayor o igual a cero. num es negativo si es menor a cero. El cero se considera positivo.

```
Algoritmo PositivoNegativo
Lexico
  num ∈ E {variable para almacenar el número a analizar}
Inicio
  Leer(num)
  si num < 0
  entonces {num es negativo}
    Escribir (num,' es negativo')
  sino {num es positivo}
    Escribir(num,' es positivo' )
  fsi
Fin
```

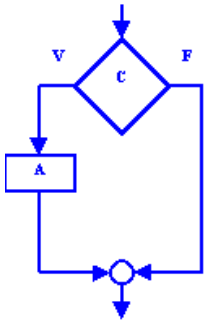

Composición Condicional
IF ... THEN ... ELSE - Ejemplo

```
PROGRAM PositivoNegativo;
VAR
  num: Integer; {variable para almacenar el numero a analizar}
BEGIN
  WRITELN('Ingrese un numero entero');
  READ(num);
  IF num<0
  THEN BEGIN {num es negativo}
    WRITELN(num,' es negativo')
  END
  ELSE BEGIN {num es positivo}
    WRITELN(num,' es positivo')
  END;
  WRITELN('Muchas gracias!')
END.
```


Composición Condicional
si ... entonces

Sintaxis:

```
si <condicion>
entonces {comentario}
  <acciones>
fsi
```



Semántica: Se evalúa la **condición**. Cuando la evaluación de la **condición** es **verdadera**, se ejecutan las **acciones** del **entonces**. Cuando la evaluación de la **condición** es **falsa**, no se hace **nada** y se continúa con la estructura siguiente.

Composición Condicional
si ... entonces - Ejemplo

Informar si un número introducido por el usuario es positivo.

Datos: número es número real (num)
Resultados: num es positivo o nada
Relación entre e/s: num es positivo si es mayor o igual a cero. El cero se considera positivo.

```
Algoritmo NumeroPositivo
Lexico
  num ∈ R {var para almacenar el número a analizar}
Inicio
  Escribir('Ingrese un número:')
  Leer(num)
  si num >= 0
  entonces
    Escribir(num,' es positivo')
  fsi
Fin
```

Hacer otra solución usando si … entonces … sino

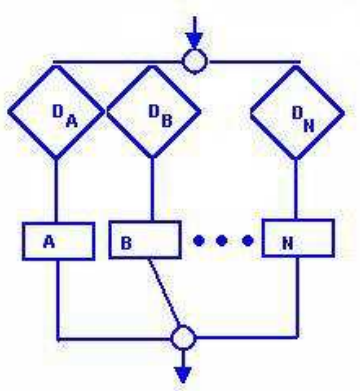
Composición Condicional
IF ... THEN - Ejemplo

```
PROGRAM NumeroPositivo;
VAR
  num: Integer; {variable para almacenar el numero a analizar}
BEGIN
  WRITELN('Ingrese un numero entero');
  READ(num);
  IF num>=0
  THEN BEGIN {num es positivo}
    WRITELN(num,' es positivo')
  END;
  WRITELN('Muchas gracias!')
END.
```

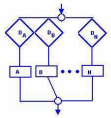

Composición Condicional
segun...

Sintaxis:

segun
<condicion$_1$>:<acciones$_1$>
<condicion$_2$>:<acciones$_2$>
⋮
<condicion$_n$>:<acciones$_n$>
fsegun



Composición Condicional
segun...



Semántica:

La toma de decisión involucra la evaluación de **una serie de condiciones**. Se evalúan cada una de las **condiciones**. Cuando la evaluación de una **condición** es **verdadera**, se ejecutan las **acciones** correspondientes a dicha **condición**.

Atención!!!

- Las condiciones deben cubrir todo el dominio.
- Las condiciones deben ser mutuamente excluyentes (solo una es verdadera).
- La evaluación de las condiciones es aleatoria.

Hay otras semánticas para el según

Composición Condicional
segun ... - Ejemplo

Determinar si un ciudadano es niño, adolescente, adulto, o adulto mayor, de acuerdo a la edad que posee.

Datos: edad es un número real (edad).
Resultados: niño, adolescente, adulto, o adulto mayor.
Relación entre e/s: edad entre 0 y 11 inclusive es niño; edad entre 12 y 17 inclusive es adolescente; edad entre 18 y 49 inclusive es adulto; edad de 50 en adelante es adulto mayor.

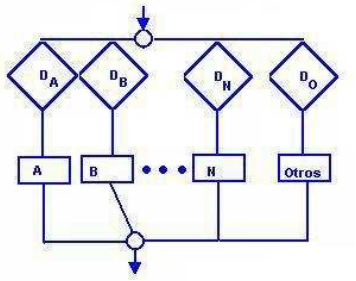
```
Algoritmo Edades
Lexico
  edad ∈ Z {var para almacenar la edad a analizar}
Inicio
  Escribir('Ingrese la edad del ciudadano:')
  Leer(edad)
  segun
    (0<=edad<=11):Escribir('el ciudadano es niño')
    (12<=edad<=17):Escribir('el ciudadano es adolescente')
    (18<=edad<=49):Escribir('el ciudadano es adulto')
    (edad>=50):Escribir('el ciudadano es adulto mayor')
  fsegun
Fin
```

Composición Condicional

segun... otros

Sintaxis:

segun
<condicion$_1$>: <acciones$_1$>
<condicion$_2$>: <acciones$_2$>
<condicion$_1$>: <acciones$_1$>
⋮
<condicion$_n$>: <acciones$_n$>
otros: <acciones$_m$>
fsegun



Semántica: En su estructura es similar al anterior. Se utiliza cuando uno de los casos corresponde al complemento de la unión de los restantes. Si **ninguna condición** es **verdadera** si o si se ejecutan las **acciones$_m$**. Digamos que **otros** es una tautología, siempre es verdadera.
La evaluación de las condiciones es similar al según anterior, donde ***otros*** entra en juego solo si ninguna de las **n** condiciones fue verdadera.

Composición Condicional

segun ... otros - Ejemplo

Determinar si un número introducido por el usuario es mayor o igual que 50, menor o igual que 10 o está entre ambos.

Datos: número es un número real (num)
Resultados: mayor o igual que 50; menor o igual que 10; o está entre 10 y 50.
Relación entre e/s: num>=50 es mayor o igual que 50; num<=10 es menor o igual que 10; 10<num<50 está entre 10 y 50.

```
Algoritmo EjemploOtros
Lexico
  num ∈ R {var para almacenar el número a analizar}
Inicio
  Escribir('Ingrese un número:')
  Leer(num)
  segun
    (num>=50):Escribir(num,' es mayor o igual que 50')
    (num<=10):Escribir(num,' es menor o igual que 10')
    otros: Escribir(num,' está entre 10 y 50')
  fsegun
Fin
```


Composición Condicional

segun

- Es una de las estructuras condicionales más poderosas y utilizadas en la construcción de algoritmos.
- Permite expresar más claramente una solución condicional.
- Ahorra código.
- Hace más legibles los algoritmos.
- Posee un alto grado de abstracción.

Composición Condicional

segun

No existe una estructura condicional similar en los lenguajes de programación más conocidos.

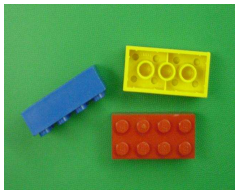
A la hora de implementar un algoritmo que utiliza el según debemos analizar como traducirlo a una estructura propia del lenguaje (generalmente similares al si ... entonces ... sino).

Atención!

El ***case*** de Pascal, y de la mayoría de los lenguajes, **no es equivalente** al ***segun***.

Composición Condicional

Equivalencias



Así como hay varias maneras de hacer una pared usando ladrillitos, también hay varias maneras de diseñar algoritmos. Entre los distintos tipos de **composiciones condicionales** hay ciertas **equivalencias** que posibilitan traducir unas en otras y viceversa.

Por ejemplo, es aconsejable usar el **segun** para construir la **primer versión** de un **algoritmo**, porque así es más legible, y luego ir acercándonos a las estructuras de los lenguajes de programación mediante el uso de equivalencias.

Composición Condicional

Equivalencias

si <condicion>
entonces
  <acciones$_1$>
sino
  <acciones$_2$>
fsi

=

segun
  <condicion>:<acciones$_1$>
  no <condicion>:<acciones$_2$>
fsegun

Composición Condicional

Equivalencias - Ejemplo

```
Algoritmo EjemploSi
Lexico
  num ∈ R
Inicio
  Leer(num)
  si num < 0
  entonces {num es negat.}
    Escribir('es negat.')
  sino {num es posit.}
    Escribir('es posit.' )
  fsi
Fin
```

```
Algoritmo EjemploSegun
Lexico
  num ∈ R
Inicio
  Leer(num)
  segun
   (num<0):Escribir('es negat.')
   no(num<0):Escribir('es posit.')
  fsegun
Fin
```


Composición Condicional

Equivalencias

según
  <condicion$_1$>:<acciones$_1$>
  <condicion$_2$>:<acciones$_2$>
  <condicion$_3$>:<acciones$_3$>
fsegún

Nota: solo si <condicion$_1$>, <condicion$_2$> y <condicion$_3$> son mutuamente excluyentes.

Esta equivalencia puede ser extendida a todos los casos que sean necesarios siguiendo la misma lógica.

=

si <condicion$_1$>
entonces
  <acciones$_1$>
sino
  si <condicion$_2$>
  entonces
    <acciones$_2$>
  sino
    <acciones$_3$>
  fsi
fsi

*Nota: A esta forma se la conoce como **si anidado***

Composición Condicional

Equivalencias - Ejemplo

```
Algoritmo EjemploSegun
Lexico
  num ∈ R {variable para almacenar el número a analizar}
Inicio
  Leer(num)
  segun
    (num>=50):Escribir(num,' es mayor o igual que 50')
    (num<=10):Escribir(num,' es menor o igual que 10')
    (10<num<50):Escribir(num,' está entre 10 y 50')
  fsegun
Fin
```

```
Algoritmo EjemploSiAnidado
Lexico
  num ∈ R {variable para almacenar el número a analizar}
Inicio
  Leer(num)
  si num>=50
  entonces {num es mayor o igual que 50}
    Escribir(num,' es mayor o igual que 50')
  sino {num es menor que 50}
    si num<=10
    entonces {num es menor que 50 y menor o igual que 10}
      Escribir(num,' es menor o igual que 10')
    sino {num es menor que 50 y mayor que 10}
      Escribir(num,' esta entre 10 y 50')
    fsi
  fsi
Fin
```


Composición Condicional

Equivalencias - Ejemplo

```
PROGRAM EjemploSiAnidado;
VAR
  num: Integer; {variable para almacenar el numero a analizar}
BEGIN
  WRITELN('Ingrese un numero entero');
  READ(num);
  IF num>=50
  THEN BEGIN {num es mayor o igual que 50}
    WRITELN(num,' es mayor o igual que 50 ')
  END
  ELSE BEGIN {num es menor que 50}
    IF num<=10
    THEN BEGIN {num es menor que 50 y menor o igual que 10}
      WRITELN(num,' es menor o igual que 10')
    END
    ELSE BEGIN {num es menor que 50 y mayor que 10}
      WRITELN(num,' esta entre 10 y 50 ')
    END
  END;
  WRITELN('Muchas gracias!')
END.
```


Datos y direcciones útiles

Compiladores Pascal

- **Free Pascal http://www.freepascal.org/**

Es un compilador estable y potente. Distribuido libremente bajo la licencia GPL. Posee entorno IDE (integrated development environment). Se puede mezclar código Turbo Pascal con código Delphi. Funciona bajo Linux, OS/2, FreeBSD, Windows. Es el que está instalado en la Sala 101 del Pab 2.

- **DevPascal http://www.bloodshed.net/devpascal.html**

Posee entorno IDE que permite crear programas usando el compilador FreePascal o el GNU Pascal. Distribuido libremente bajo la licencia GPL. Funciona bajo Windows.

- **GPC (GNU Pascal compiler) http://www.gnu-pascal.de/**

No posee entorno, pero puede usarse el editor **nedit** para la escritura de código (resalta la sintaxis). Funciona bajo SO Linux, OS/2, FreeBSD, Windows. Se distribuye bajo licencia GPL.

- **Turbo Pascal de Borland® http://www.borland.com**

Muy sencillo y con entorno integrado de programación, compliación y ejecución. Funciona bajo SO DOS. Gratuito hasta la vesión 5.5

- **Delphi http://www.borland.com/delphi/**

Es un producto tipo RAD (Rapid Application Development) de Borland. Utiliza Pascal + objetos para crear aplicaciones con interfaz gráfica. Funciona bajo SO Windows.

- **Kylix http://www.borland.com**

Es descendiente de Delphi, con soporte para el sistema operativo **Linux**. Actualmente este proyecto está descontinuado.

Datos y direcciones útiles

- **Otros Compiladores Pascal**

http://www.ucm.es/info/dsip/clavel/courses/ip0203/node5.html

http://www.thefreecountry.com/compilers/pascal.shtml

- **Más sobre Pascal**

http://www.itlp.edu.mx/publica/tutoriales/pascal/organiza.html (tutorial)

http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Pascal

- **Convenciones para escribir algoritmos y programas**

En fotocopiadora del CECEx y en el sitio de la materia: ConvencionesPseudocódigoHastaCiclos.rtf y ConvencionesPascalHastaCiclos.rtf

- **Introducción a la Algorítmica y Programación**

http://www.ucm.es/info/dsip/clavel/courses/ip0203/ip0203.html

http://www.algoritmica.com.ar/

Bibliografía

– Scholl, P. y Peyrin, J.-P. “Esquemas Algorítmicos Fundamentales: Secuencias e iteración”:
- Composición secuencial (pags. 35 - 55)
- Composición condicional (pags. 57 - 69)

– Biondi, J. y Clavel, G. “Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes”:
- Composición condicional (35 - 53)

– Quetglás, Toledo, Cerverón. "Fundamentos de Informática y Programación“
- http://robotica.uv.es/Libro/Indice.html
- Capítulo 3 (91 - 98, 125 - 126)

- Diagramas de Flujo – Pseudocódigo
- www.programacion.com/tutorial/jap_data_alg/
- es.wikipedia.org/wiki/Pseudocódigo
- www.desarrolloweb.com/articulos/2198.php
- www.itver.edu.mx/comunidad/material/algoritmos/U2-22.htm
- fcqi.tij.uabc.mx/docentes/mgarduno/Program1/Unidad1/u1_1.htm

- C.Böhm, G.Jacopini, Comm. ACM vol.9, nº5, 366-371,1966

Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 3 (continuación)

Composición de Algoritmos Secuencial y Condicional

Pascal

Comentarios en Pascal:

Un programa puede contener comentarios en cualquier lugar. Los comentarios se delimitan encerrándolos entre llaves { }. También es posible comenzar un comentario con (* y terminarlo con *).

```
PROGRAM AreaCuadrado;
VAR
  lado: Integer;        {variable dato}
  areaCuad: Integer;    (*variable resultado*)
BEGIN
  READ(lado);
  areaCuad := lado * lado;
  WRITELN('El area es: ', areaCuad)
END.
```


Pascal

Cuando va el BEGIN y END:

El BEGIN END determina un bloque de sentencias. Cuando el bloque tiene una sola sentencia no hace falta ponerlos. Así se ahorran unas cuantas líneas de código. Pero si el bloque tiene perspectivas de crecer en cantidad de sentencias es mejor escribirlos igual para tener el bloque bien armadito de entrada.

Las palabras BEGIN y END son delimitadores, no son sentencias. Se usan para separar las partes funcionales de un programa. Con ellas se indica el principio y el final de la sección ejecutable. También sirven para delimitar una sentencia compuesta. Cada BEGIN debe estar asociado con un END, excepto en dos casos: la sentencia CASE y la declaración de RECORD.

Pascal
Cuando va el punto y coma

El **punto y coma significa en Pascal una sentencia vacía, que no nada hace,** no tiene efecto, no cambia el estado de nada. El punto y como se usa para indicar una secuencia de sentencias, es decir cuando a una sentencia le sigue otra.

Cuando el bloque tiene una sola sentencia no hace falta colocarlo

En general, en la última sentencia de cada bloque no hace falta colocarlo.

```
BEGIN
  suma:=1000;
  incr:=20;
  total:=suma+incr {observar la ausencia del ; al final de esta instrucción}
END.
```

No es necesario escribir el punto y coma antes de END ya que el *punto y coma* se usa para separar instrucciones, no para terminarlas.

Pascal
Cuando va el punto y coma

```
PROGRAM puntoYComa;
VAR
  x,b,c,d: Integer;
BEGIN
  x:=7;
  b:=1;
  IF x>0
  THEN BEGIN
    IF b<>1
    THEN BEGIN
      c:=1 {no debe escribirse el ; precediendo al ELSE,
            porque se terminará allí la sentencia IF y
            se roducirá un error de compilación}
    END
    ELSE BEGIN {este ELSE pertenece el IF anidado}
      d:=1  {no debe escribirse el punto y coma porque
             No sigue otra sentencia sino que termina el
             bloque}
    END
  END; {pertenece el THEN del IF principal. Lleva ; porque
        termina el bloque y sigue otra instruccion}
  WRITELN('Muchas gracias!')
END.
```


Pascal
Cuando va el punto y coma

```
PROGRAM puntoYComa;
VAR
  x,b,c,d: Integer;
BEGIN
  x:=7;
  b:=1;
  IF x>0
  THEN BEGIN
    IF b<>1
    THEN
      c:=1 {no debe escribirse el ; precediendo al ELSE,
            porque se terminará allí la sentencia IF y
            se roducirá un error de compilación}
    ELSE  {este ELSE pertenece el IF anidado}
      d:=1  {no debe escribirse el punto y coma porque
             No sigue otra sentencia sino que termina el
             bloque}
  END; {pertenece el THEN del IF principal. Lleva ; porque
        termina el bloque y sigue otra instruccion}
  WRITELN('Muchas gracias!')
END.
```


Pascal
Cuando va el punto y coma

Cuando se anidan varias sentencias IF, hay que tener en cuenta que la cláusula ELSE afecta a la IF-THEN más próxima.

Ejemplo:

```
IF a=1
THEN
  IF b<>1
  THEN
    c:=1
  ELSE
    c:=0;
```

En este caso, independientemente de la forma de escribir las sentencias, la cláusula ELSE quedará asociada con el "IF b<>1 THEN c:=1", de forma que si la condición a=1 es falsa, no se tomará ninguna acción.

Pascal
Cuando va el punto y coma

Regla de la Sentencia IF

IF ExpresionLogica
THEN Sentencia1
ELSE Sentencia2;

Tanto Sentencia1 como Sentencia2 pueden ser sentencias simples o compuestas; en el caso de ser compuestas deberán ir agrupadas por un bloque BEGIN...END
Notar que justo antes del ELSE no puede haber un punto y coma. Si Sentencia1 es compuesta, su correspondiente END no podrá ir terminado en punto y coma en el caso de que detrás haya un ELSE. Si Sentencia2 es compuesta, su correspondiente END sí que debe de ir terminado en punto y coma. Como Sentencia1 o Sentencia2 puede ir, por supuesto, otro condicional.

```
IF Condicion1
THEN
  IF Condicion2
  THEN Sentencia1
  ELSE Sentencia2;
```

¿A qué IF pertenece el ELSE? Las reglas del Pascal dicen que el ELSE pertenece siempre al IF más próximo que se encuentra para el que no exista una cláusula ELSE. Esto quiere decir, en el ejemplo, que el ELSE pertenece al segundo IF.

Composición Condicional
SEGUN vs. CASE

La estructura SEGUN de la notación algorítmica, pseudocódigo, es mucho más general y poderosa que el CASE de Pascal. La ventaja del SEGUN es que permite encontrar soluciones simples, cortas y fáciles de leer.

El CASE de Pascal determina cual bloque de acciones va a ejecutar mediante una **única expresión denominada selector**. No permite colocar una expresión por bloque. Cada bloque se etiqueta con una constante.

```
CASE <selector> OF
  constante.1: BEGIN
                 <instrucciones1>
               END;
  constante.2: BEGIN
                 <instrucciones2>
               END;
  ......
  constante.N: BEGIN
                 <instruccionesN>
               END
  ELSE
    BEGIN
      <instruccionesELSE>;
    END;
END; {fin de CASE}
```


Composición Condicional
SEGUN vs. CASE

Reglas del CASE:

1. La expresión <selector> se evalúa y se compara con las constantes; las constantes son listas de uno o más posibles valores de <selector> separados por comas. Ejecutadas la(s) <instrucciones>, el control

se pasa a la primera instrucción a continuación de END (fin de CASE).

2. El selector debe ser un tipo ordinal (integer, char, boolean o enumerado). Los números reales no pueden ser utilizados ya que no son ordinales. Los valores ordinales de los límites inferiores y superiores deben de estar dentro del rango -32768 a 32767. Por consiguiente, los tipos string, longint y word no son válidos.

3. La cláusula ELSE es opcional.

Composición Condicional
SEGUN vs. CASE

Reglas del CASE (cont.):

4. Si el valor de <selector> no está comprendido en ninguna lista de constantes y no existe la cláusula ELSE, no sucede nada y sigue el flujo del programa; si existe la cláusula ELSE se ejecutan la(s) <instrucciones> a continuación de la cláusula ELSE.

5. Todas las constantes CASE deben ser únicas y de un tipo ordinal compatible con el tipo del selector.

6. Cada sentencia, excepto la última, deben ir seguidas del punto y coma.

Conviene tener presente que no debe escribirse *punto y coma* antes de la palabra ELSE.

Composición Condicional
SEGUN vs. CASE

```
PROGRAM Tecla;
VAR
  caracter: Char;
BEGIN
  WRITE('Escriba un caracter : ');
  READLN(caracter);
  CASE caracter OF
    '0': BEGIN WRITELN('Es un número') END;
    '1': BEGIN WRITELN('Es un número') END;
    '2': BEGIN WRITELN('Es un número') END;
    '3': BEGIN WRITELN('Es un número') END;
    '4': BEGIN WRITELN('Es un número') END;
    '5'..'9': BEGIN WRITELN('Es un número') END;
    'a'..'z': BEGIN WRITELN('Es una letra minusc') END;
    'A'..'Z': BEGIN WRITELN('Es una letra mayus') END
    ELSE
      BEGIN WRITELN('Es un caracter especial')  END
  END;
END.
```


Implementación en Lenguaje C
Algoritmo CalcularIva

```
#include <stdio.h>

/* Define una constante */
#define Iva 21

/* Variables */
float precioBruto;       //variable dato
float precioFinal;      //variable resultado

void main(){
  //Se debe utilizar como separador de decimales el punto (no la coma)
  scanf("%f",&precioBruto);
  precioFinal = precioBruto + (precioBruto*Iva)/100;
  printf("El precio final es: %f \n", precioFinal);
}
```


Implementación en Lenguaje C
Algoritmo PositivoNegativo

```
#include <stdio.h>
/* Variables */
int  num; //variable para almacenar el numero a analizar
void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num<0) {
    //num es negativo
    printf("%d es negativo \n", num);
  }
  else {
    //num es positivo
    printf("%d es positivo \n", num);
  }
  printf("Muchas gracias! \n");
}
```


Implementación en Lenguaje C
Algoritmo NumeroPositivo

```
#include <stdio.h>

/* Variables */
int  num; //variable para almacenar el numero a analizar

void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num>=0) {
    //num es positivo
    printf("%d es positivo \n", num);
  }
  printf("Muchas gracias! \n");
}
```

Implementación en Lenguaje C

Algoritmo EjemploSiAnidado

```
#include <stdio.h>
/* Variables */
int  num; //variable para almacenar el numero a analizar
void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num>=50) {
    //num es mayor o igual que 50
    printf("%d es mayor o igual que 50 \n", num);
  }
  else { //num es menor que 50
    if (num<=10) {
      //num es menor que 50 y menor o igual que 10
      printf("%d es menor o igual que 10 \n", num);
    }
    else { //num es menor que 50 y mayor que 10
      printf("%d esta entre 10 y 50 \n", num);
    }
  }
  printf("Muchas gracias! \n");
}
```