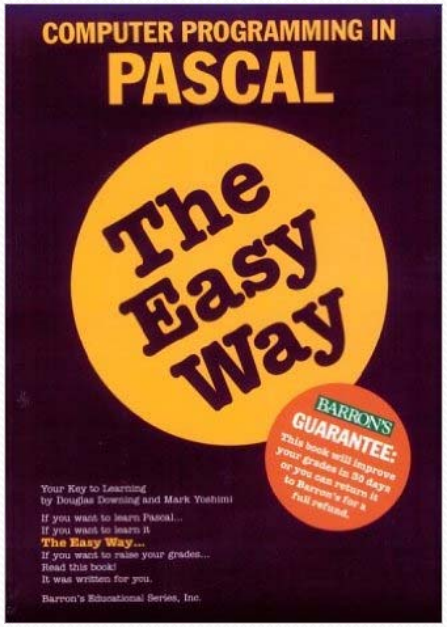
Lenguajes de Programación

Dr. Carlos A. Coello Coello
Departamento de Computación
CINVESTAV-IPN
ccoello@cs.cinvestav.mx

Registros Variantes



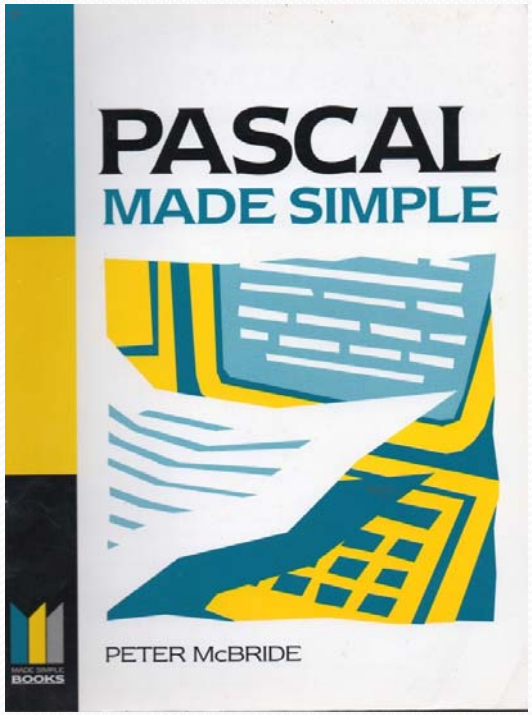
- Por tanto, los registros variantes son muy eficientes y seguros, porque sólo permiten al programador realizar operaciones que tengan algún significado dentro del contexto dado y que sean permisibles dentro del registro.

Registros Variantes

- Sin embargo, esta estructura de datos introduce un problema en el sistema de tipos de Pascal, porque no requiere ser inicializada después de cambiar el valor de un campo identificador.

- Esto significa que podríamos tener cualquier valor dentro de un registro variante después de modificar el valor de un campo identificador.

Registros Variantes



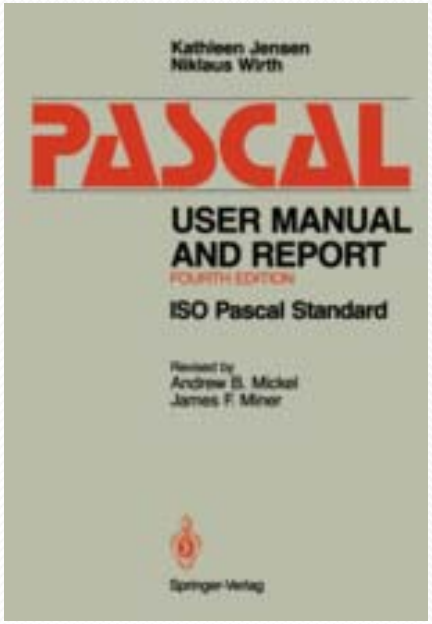
- Este problema se produce porque Pascal (al igual que la mayoría de los lenguajes estructurados) no requiere que se inicialicen las variables al entrar a su entorno.

Registros Variantes

- El acceso a variables no inicializadas puede dar lugar a que se usen valores que fueron dejados previamente en el bloque y que compartían la misma zona de memoria.

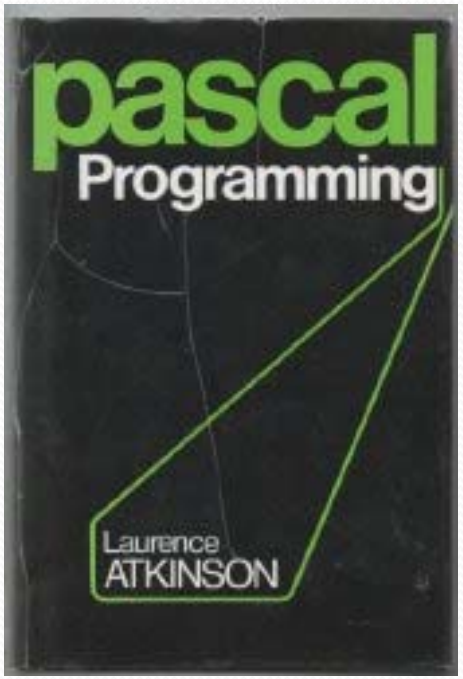
- En el caso de los registros variantes, este valor puede incluso ser de un tipo diferente al esperado.

Registros Variantes



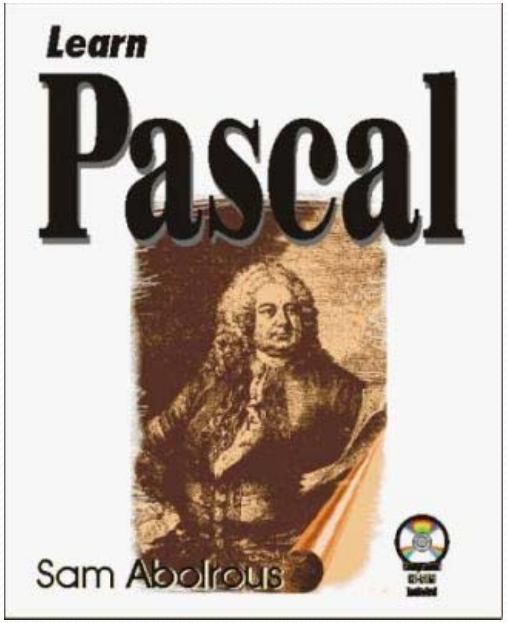
- De hecho, algunos programadores aprovechaban este subterfugio de Pascal para violar su sólido sistema de tipos y realizar algún tipo de programación de sistemas.

Registros Variantes



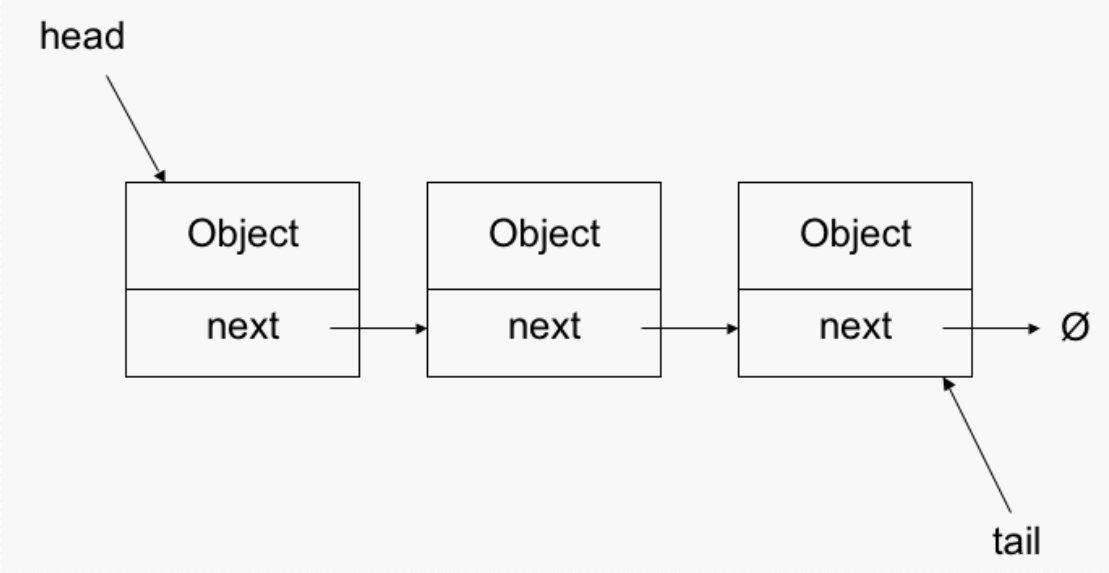
- La raíz del problema es que los registros variantes permiten una forma de “aliasing”, ya que los campos de las diferentes variantes son todos “alias” de las mismas posiciones de memoria.

Apuntadores



- FORTRAN y ALGOL no proporcionaban apuntadores porque estaban orientados a aplicaciones científicas y las listas ligadas se usaban más frecuentemente en la programación de sistemas.

Apuntadores



- Un apuntador nos permite acceder una posición de memoria directamente.

- Los apuntadores son muy útiles para implementar listas ligadas, árboles, grafos, etc.

Apuntadores

- Niklaus Wirth decidió, sin embargo, incorporar apuntadores en Pascal, con lo que hizo al lenguaje más poderoso.

- Además, hizo que este mecanismo fuera muy seguro al requerir que los apuntadores tuviesen un tipo asociado a ellos.

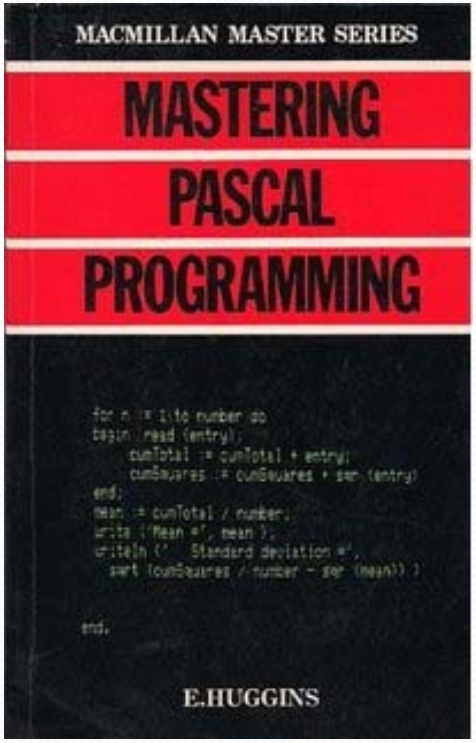
Apuntadores

- Ejemplo:

var p : ↑**real**;

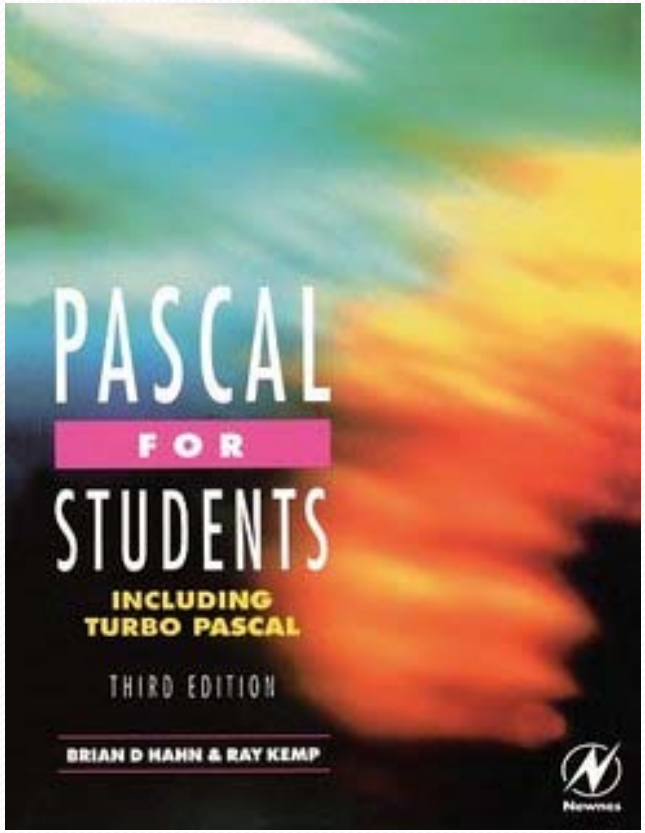
- Esto nos permite mantener las ventajas de un sistema fuerte de tipos al usar los apuntadores. De tal manera, un apuntador en Pascal es la dirección de un objeto de un tipo en particular.

Apuntadores



- Adicionalmente, esto elimina la posibilidad de que se produzcan muchos de los errores que suelen caracterizar a los programas en C o ensamblador.

Apuntadores



- El tipo base de un apuntador puede ser cualquiera, incluyendo registros y arreglos.

Apuntadores

- Veamos un ejemplo del uso de apuntadores:

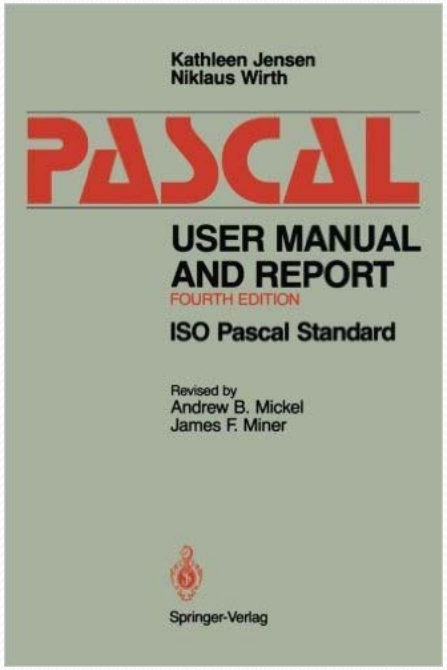
```
var   p: ↑real;
      x: real;
      c: char;
begin
  new(p);
  p ↑ := 3.14159;
  c := p ↑;    {¡Esto es ilegal!}
end
```

Apuntadores

- En el segmento de código mostrado en el acetato anterior, no podemos realizar la asigación marcada como “ilegal”, porque el tipo de p es **apuntador a un real**, por lo cual p ↑ es el nombre de una variable **real**.

- Permitir la asignación de una variable real a una variable definida como tipo “character”, violaría el sistema de tipos fuerte de Pascal y, por ende, esta asignación es ilegal.

Equivalencia Estructural vs. Equivalencia de Nombres



- El Reporte Revisado de Pascal dice que puede asignarse una variable a una expresión si ambas tienen un “tipo idéntico”.

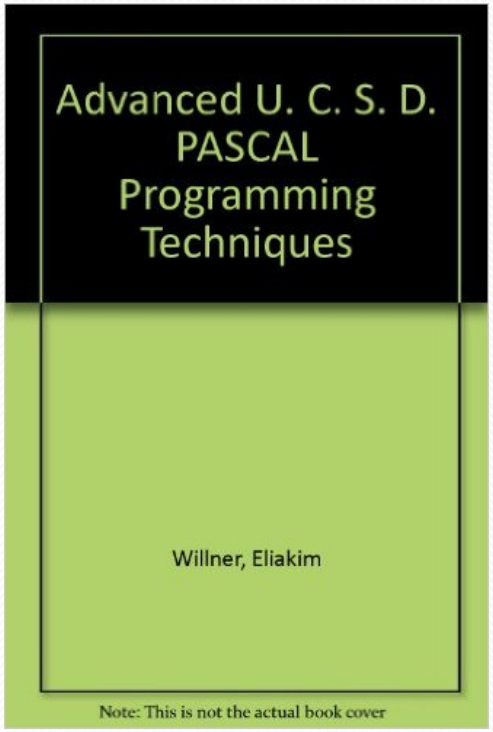
Equivalencia Estructural vs. Equivalencia de Nombres

- Desgraciadamente, el reporte no especifica lo que significa que dos tipos sean “idénticos”, y pueden dársele varias interpretaciones a este enunciado.

Equivalencia Estructural vs. Equivalencia de Nombres

- Esto dio origen a una famosa controversia en computación, produciéndose en consecuencia dos interpretaciones diferentes de la equivalencia de tipos:
  - Equivalencia Estructural
  - Equivalencia de Nombres

Equivalencia Estructural vs. Equivalencia de Nombres



- **<u>Equivalencia Estructural</u>:** Dos tipos se consideran equivalentes si tienen la misma estructura.

Equivalencia Estructural vs. Equivalencia de Nombres

Ejemplo:

```
type    persona = record id: integer; peso: real end;
        auto    =  record id: integer; peso: real end;

  var   x : persona;
        y : auto;
```

Equivalencia Estructural vs. Equivalencia de Nombres

- Bajo “equivalencia estructural”, **x:=y** es una sentencia válida, ya que los tipos de las 2 variables son exactamente los mismos (palabra por palabra).

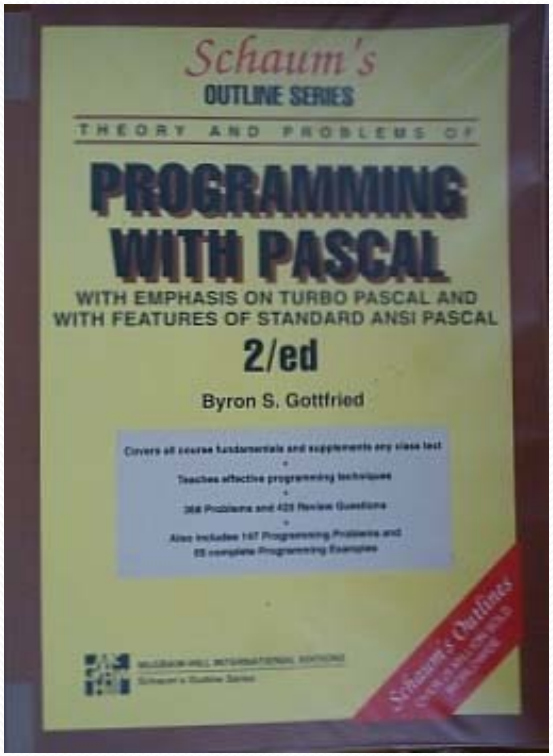
- Esto a pesar de que ambas variables fueron declaradas por separado.

Equivalencia Estructural vs. Equivalencia de Nombres

- De hecho, podría intuirse que el programador pretendía que estas dos variables tuviesen tipos diferentes y que fue una mera coincidencia que los nombres de sus tipos fueran iguales.

- Bajo este esquema, ambos tipos se consideran idénticos.

Equivalencia Estructural vs. Equivalencia de Nombres



- **<u>Equivalencia de Nombres</u>:** Dos objetos tienen el mismo tipo si los nombres de sus tipos son los mismos.

Equivalencia Estructural vs. Equivalencia de Nombres

```
type   persona = record id: integer; peso: real end;
       auto    =  record id: integer; peso: real end;

  var  x : persona;
       y : auto;
```

- Dado el ejemplo anterior, **x:=y** es ilegal porque el tipo de ‘x’ es ‘persona’ y el tipo de ‘y’ es ‘auto’.

Equivalencia Estructural vs. Equivalencia de Nombres

- Ejemplo:

```
var   x : record id: integer; peso: real end;
      y : record id: integer; peso: real end;
```

Equivalencia Estructural vs. Equivalencia de Nombres

- Bajo equivalencia de nombres “pura”, ‘x’ y ‘y’ son diferentes, porque el compilador asignará diferentes identificadores a cada una de estas declaraciones.
- Por lo tanto, los nombres de sus tipos no serán los mismos.

Equivalencia Estructural vs. Equivalencia de Nombres

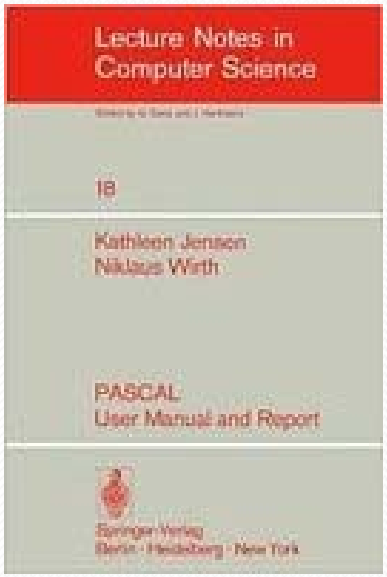
- La equivalencia de nombre tiene también algunos problemas adicionales. Por ejemplo:

```
type edad = 0 . . 150;
var   n : integer;
      a : edad;
```

Equivalencia Estructural vs. Equivalencia de Nombres

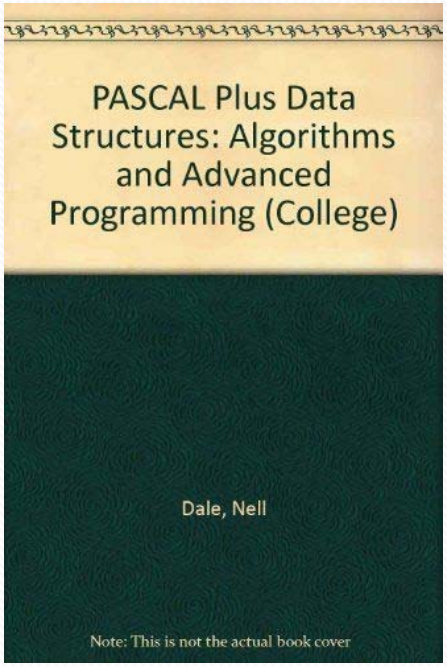
- ¿Es legal la asignación '**n:=a**'?
- Bajo equivalencia de nombres "pura", la respuesta es "no", puesto que 'n' y 'a' tienen tipos cuyos nombres son diferentes.

Equivalencia Estructural vs. Equivalencia de Nombres



- Sin embargo, el Reporte Revisado de Pascal explícitamente permite esta asignación, puesto que indica que dos objetos se consideran del mismo tipo si uno es un subrango del tipo del otro.

Equivalencia Estructural vs. Equivalencia de Nombres



- Esta no es una manera muy elegante de resolver el problema, porque se introduce una excepción al lenguaje que habrá de recordarse a la hora de programar.

Equivalencia Estructural vs. Equivalencia de Nombres

- En Algol-68, que se adhiere estrictamente a la equivalencia estructural, los 2 tipos de registros siguientes son equivalentes:

```
mode t1 = struct(int value; ref mode t1 link);
mode t2 = struct(int value; ref mode t2 link);
```

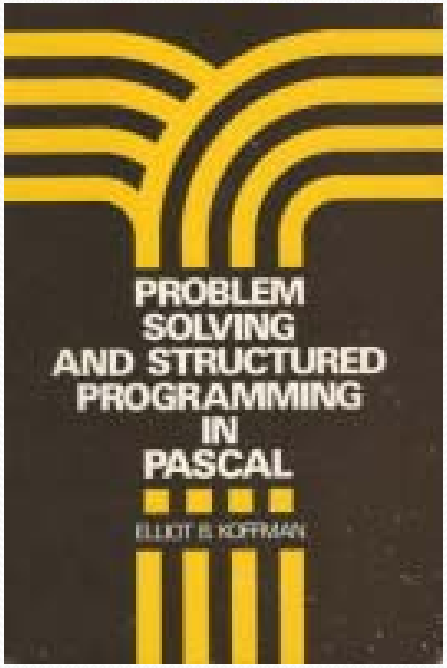
- Donde “mode” significa ‘tipo’ y “ref” introduce una declaración de un apuntador.

Equivalencia Estructural vs. Equivalencia de Nombres

- Peor aún, estas 2 declaraciones mostradas anteriormente, son también equivalentes a:

```
mode t3=struct(int value; ref struct(int value;
                ref mode t3 link));
```

Equivalencia Estructural vs. Equivalencia de Nombres



- Por lo tanto, cualquier segmento de código que opere sobre un objeto de tipo “t1”, operará también sobre un objeto de tipo “t3” y viceversa.

Equivalencia Estructural vs. Equivalencia de Nombres

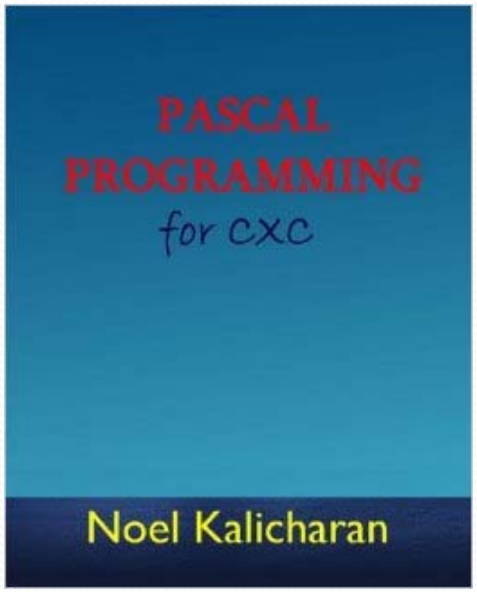
- Aunque la equivalencia estructural es intuitivamente muy atractiva, depara ciertas sorpresas a los programadores.
- Adicionalmente, es también difícil de implementar.

Equivalencia Estructural vs. Equivalencia de Nombres

- En Ada, que se adhiere estrictamente a la **equivalencia de nombres**, los dos arreglos siguientes son diferentes:

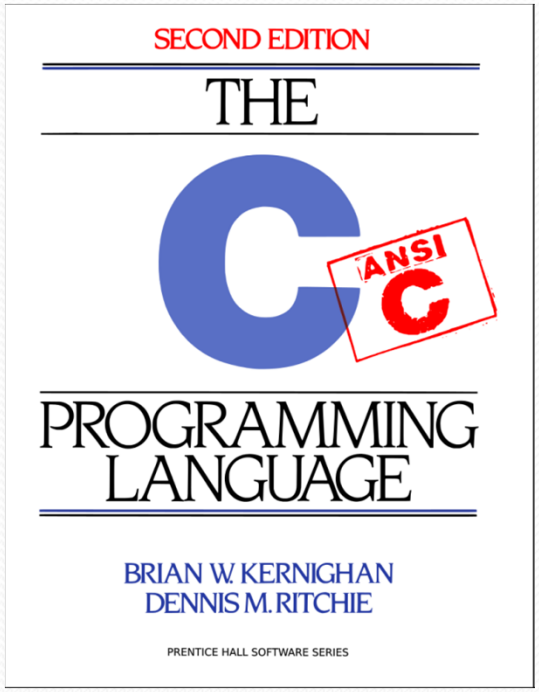
```
IA : array(Integer range 1..10) of Integer;
JA : array(Integer range 1..10) of Integer;
```

Equivalencia Estructural vs. Equivalencia de Nombres



- Estos dos arreglos son diferentes porque los nombres anónimos (los que les da el compilador) de sus tipos son diferentes.

Equivalencia Estructural vs. Equivalencia de Nombres



- En C se usa equivalencia de nombres para los registros y las uniones y se usa equivalencia estructural para lo demás.

Equivalencia Estructural vs. Equivalencia de Nombres

- ¿Cuál de los dos enfoques es mejor?
- Ha habido un largo debate en torno a este tema, aunque el consenso general parece ser que la **equivalencia de nombres** es mejor.

Equivalencia Estructural vs. Equivalencia de Nombres

- Las razones principales por las que suele preferirse la equivalencia de nombres son:

  1) La equivalencia de nombres suele ser más segura, puesto que es menos restrictiva. Se presume que si un programador declara dos veces el mismo tipo es porque tiene una razón para hacerlo.

Equivalencia Estructural vs. Equivalencia de Nombres

Por lo tanto, considerar estos tipos como diferentes protege la seguridad del programa (previene al programador de hacer algo sin sentido).

2) La equivalencia de nombres es más fácil de implementar. El compilador sólo tiene que comparar las cadenas de caracteres que representan los nombres de los tipos.

Equivalencia Estructural vs. Equivalencia de Nombres

- Para implementar la equivalencia estructural, es necesario escribir una función recursiva que compare las estructuras de datos que representan los tipos de los dos objetos bajo consideración.
- Esto también suele impactar la eficiencia del compilador (lo hace más lento).

Mecanismos de Asociación de Nombres

- Pascal proporciona seis mecanismos de asociación de nombres (o sea, métodos para declarar objetos en el lenguaje):

  1) Asociaciones constantes

  2) Asociaciones de tipos

  3) Asociaciones de variables

Mecanismos de Asociación de Nombres

4) Asociaciones de procedimientos y funciones.

5) Asociaciones de enumeraciones implícitas.

6) Asociaciones de etiquetas.

Mecanismos de Asociación de Nombres

- Pascal permite la declaración de constantes de tipos discretos (enteros, enumeraciones y caracteres), usando la palabra reservada **const**:

const <nombre> = <constante>;

Mecanismos de Asociación de Nombres

- El uso de constantes en un programa sigue el **Principio de Abstracción**, ya que permite a un programador hacer cambios más fácilmente en el código.

Principio de Abstracción

Evite requerir que algo se plantee más de una vez; factorice el patrón recurrente.

Mecanismos de Asociación de Nombres

- Consideremos un programa que manipula arreglos. En dicho programa, normalmente habrán muchas dependencias entre las dimensiones de estos arreglos y otras partes del programa.

- Por ejemplo, si los arreglos tienen dimensiones [1..100] entonces, normalmente habrán ciclos **for** definidos de 1 a 100 (1 **to** 100) o de 100 a 1 (100 **downto** 1).

- Además, pueden haber otros arreglos con dimensiones [1..100] o [0..99], que sean compatibles con el primer arreglo antes indicado. Todas estas interdependencias son implícitas, aunque el uso de documentación adecuada puede ayudar a hacerlas explícitas.

Mecanismos de Asociación de Nombres

- Ahora supongamos que se decir cambiar el tamaño de los arreglos a [1..150]. Esto implica, por ejemplo, que deben cambiarse los ciclos **for**. Tradicionalmente, esto implicará bastante esfuerzo de parte del programador, pues deberá buscar todas las partes del código asociadas a las dimensiones del arreglo.

- La definición de constantes permite aliviar este problema, pues que permiten que las dependencias implícitas se vuelvan explícitas.

Mecanismos de Asociación de Nombres

- Ejemplo:

const Maximo = 100;

De esta forma, podemos declarar un arreglo como:

var B : **array** [1..Maximo] **of real**;

Y podemos usar:

for i := 1 **to** Maximo **do**

Mecanismos de Asociación de Nombres

- Esto hace más fácil cualquier cambio al tamaño del arreglo, pues sólo se requerirá modificar Maximo en el código.

- Nótese, sin embargo, que hay una limitación importante relacionada con el uso de constantes en Pascal: no se permiten expresiones para definir una constante. Por ejemplo, si ahora tuviésemos un arreglo de dimensiones [0..99], no se permite usar:

var A : **array** [0..Maximo-1] **of real**;

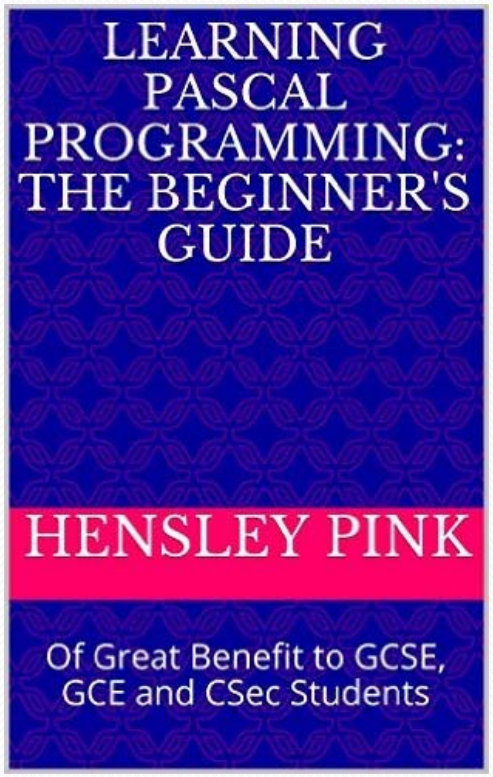
Mecanismos de Asociación de Nombres

- Lo mejor que puede hacerse en este caso es declarar dos constantes y documentar su dependencia:

```
const  Maximo = 100;
       Maximomenosuno = 99;

var    A : array [0..Maximomenosuno] of real;
```

Mecanismos de Asociación de Nombres



- Versiones más recientes de Pascal y otros lenguajes de programación (p.ej. Ada) han tendido a eliminar esta restricción.

Mecanismos de Asociación de Nombres

- En Pascal, existen dos constructores principales de estructuras de nombres.
- Una es el registro (**record**) que ya vimos anteriormente.
- El otro constructor es el procedimiento (o función), que es el principal constructor que define ambientes.

Mecanismos de Asociación de Nombres

- Una declaración de procedimiento en Pascal tiene la forma siguiente:

procedure <nombre> (<parámetros>)
    <declaraciones>
begin
    <sentencias>
end;

Mecanismos de Asociación de Nombres

- Este formato es muy similar al utilizado en Algol:

procedure <nombre> (<parámetros>);
begin
  <declaraciones>
  <sentencias>
end

Mecanismos de Asociación de Nombres

- Las reglas de entorno de las asociaciones son esencialmente iguales en Algol y en Pascal.
- El alcance de las <declaraciones> es el bloque completo, incluyendo todas las <declaraciones> y las <sentencias>.
- El alcance de los <parámetros> incluye tanto a las <declaraciones> locales como a las <sentencias> en el cuerpo del procedimiento.

Mecanismos de Asociación de Nombres

- El estándar ISO de Pascal agrega una restricción adicional al orden de las declaraciones, con la finalidad de permitir compilación en una sola pasada.

- El alcance de cada una de las <declaraciones> incluye a esa declaración, a las que le siguen y a las <sentencias>.

Mecanismos de Asociación de Nombres

- Esto significa que los nombres deben estar asociados antes de que (textualmente) se usen.
- En contraste, en Algol y en el Pascal original, este no es el caso.
- Esta restricción simplifica el compilador, porque pueden procesarse todas las declaraciones tras recorrer una sola vez el código fuente.

Mecanismos de Asociación de Nombres

- Sin embargo, esta restricción introduce dos problemas:

1) Los métodos de programación estructurada motivan a los programadores a organizar sus programas de "arriba hacia abajo" (top-down).

   Es decir, los procedimientos superiores se definen primero y luego los inferiores, que son a los que éstos invocan, se definen después.

Mecanismos de Asociación de Nombres

Debe resultar por tanto muy evidente que este esquema de programación es exactamente lo contrario de lo que requiere el estándar ISO de Pascal.

En contraste, el lenguaje C sí tiene una estructura de "arriba hacia abajo".

Mecanismos de Asociación de Nombres

2) El segundo problema es que los procedimientos mutuamente recursivos no pueden definirse antes de ser invocados.

Veamos un ejemplo de esto. Supongamos que P invoca a Q y que Q invoca a P. Las declaraciones que pondré a continuación son incorrectas en Pascal, dado que Q se usa antes de ser declarado.

Mecanismos de Asociación de Nombres

```
procedure P ( ... );
begin
    ⋮
   Q ( ... );
    ⋮
end;

procedure Q ( ... );
begin
    ⋮
    P ( ... );
    ⋮
end;
```

Mecanismos de Asociación de Nombres

- Este es un problema más serio y requirió que el comité de estandarización de Pascal optara por agregar una declaración llamada "**forward**" para resolverlo.

- Veamos el código anterior, ya corregido con el uso de **forward**.

Mecanismos de Asociación de Nombres

```
procedure Q ( ... ); forward;

procedure P ( ... );
begin
    ⋮
   Q ( ... );
    ⋮
end;

procedure Q ( ... );
begin
    ⋮
    P ( ... );
    ⋮
end;
```

Mecanismos de Asociación de Nombres

- La mayor parte de los dialectos de Pascal (incluyendo el estándar ISO) requieren que las declaraciones en un procedimiento o en un programa se encuentren en un cierto orden: etiquetas, constantes, tipos, variables y subprogramas.

- Esto se ilustra en el acetato siguiente.

Mecanismos de Asociación de Nombres

procedure <nombre> (<parámetros>);

<declaraciones de etiquetas>

<declaraciones de constantes>

<declaraciones de tipos>

<declaraciones de variables>

<declaraciones de procedimientos y funciones>

begin

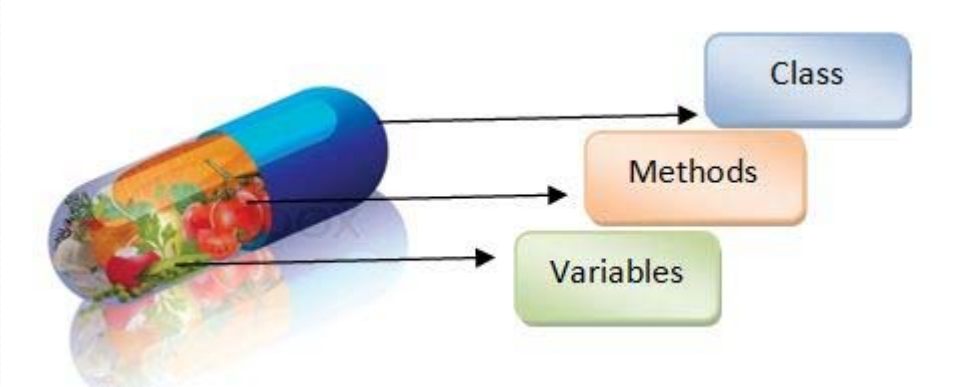
<sentencias>

end

Mecanismos de Asociación de Nombres

- Esto suena razonable pero podría resultar inconveniente en programas más grandes, puesto que impide al programador agrupar las declaraciones relacionadas de constantes, tipos, variables y subprogramas.

Mecanismos de Asociación de Nombres



- Este problema ha sido resuelto mediante el mecanismo de encapsulamiento que proporcionan los lenguajes de cuarta generación.

Mecanismos de Asociación de Nombres

- Pascal elimina el bloque usado en Algol, y lo mantiene sólo para las sentencias compuestas.

- Es importante recordar que la diferencia es que un bloque puede contener declaraciones locales y sentencias, mientras que una sentencia compuesta contiene sólo sentencias.

Mecanismos de Asociación de Nombres

- Los bloques facilitan el poder compartir almacenamiento, puesto que los arreglos declarados en bloques disjuntos pueden ocupar la misma región de memoria.

- Esto no es posible en Pascal. El almacenamiento en Pascal puede compartirse sólo entre procedimientos disjuntos, complicando el uso eficiente de memoria.

Estructuras de Control

- Pascal tiene más estructuras de control que Algol, pero éstas son más simples.
- Como vimos anteriormente, las estructuras de control anidadas de Algol originaron la programación estructurada.

Estructuras de Control

- Pascal continúa en esta misma dirección, proporcionando estructuras de control más estructuradas.

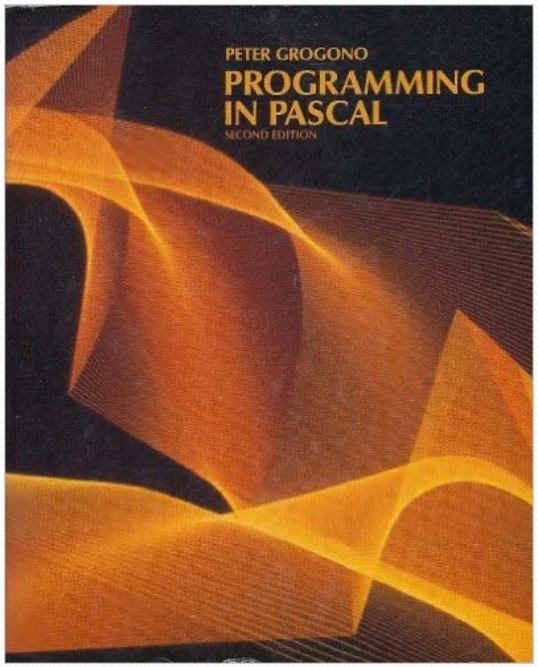
- Todas las estructuras de control de Pascal se caracterizan por tener un punto de entrada desde la sentencia previa y un punto de salida hacia la sentencia siguiente.

Estructuras de Control

- El ejemplo clásico es la sentencia **if-then-else**, la cual es exactamente igual a la de ALGOL.

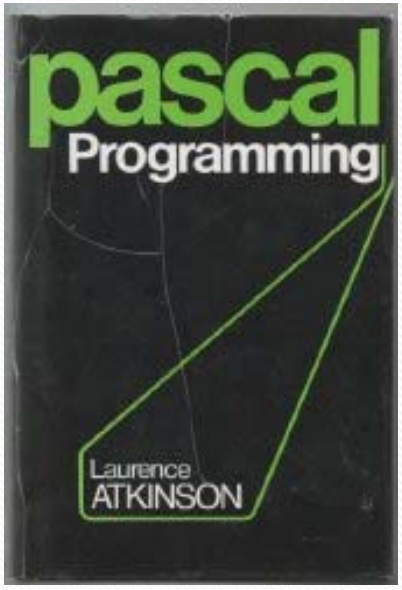
- Esta propiedad de contar con un solo punto de entrada y un solo punto de salida simplifica los programas al satisfacer el **Principio Estructural** (la estructura estática del programa corresponde en una manera simple con su estructura dinámica)

Estructuras de Control



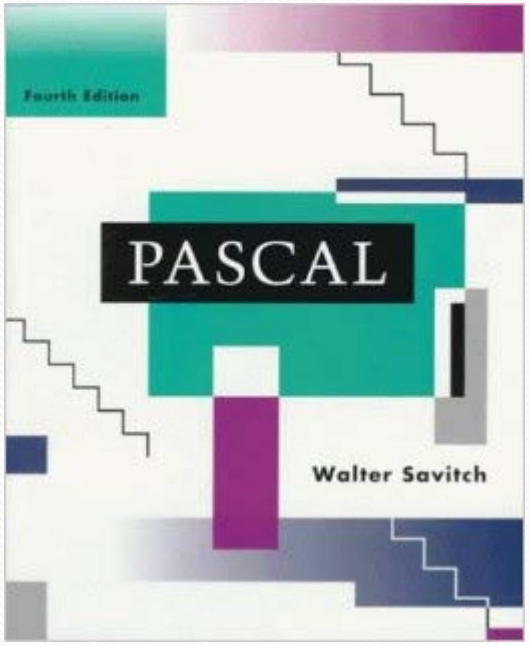
- Pascal cuenta con una sentencia GOTO, aunque rara vez se necesita debido a la riqueza de sus estructuras de control.

Estructuras de Control



- Una observación importante acerca del GOTO en Pascal es que, al igual que en ALGOL, se permiten las transferencias no locales, con todas las ineficiencias asociadas a ellas en lo que a la implementación se refiere.

Estructuras de Control



- Al igual que la mayoría de los descendientes de ALGOL, Pascal tiene procedimientos totalmente recursivos.

Estructuras de Control

- Recordemos la complejidad de los ciclos **for** de Algol. Esta estructura de control incluía elementos para definir el incremento en cada ciclo, condiciones **while** y listas de valores arbitrarios.

- Además, las expresiones en los extremos del ciclo se evaluaban a cada iteración, lo cual permitía que éstos se modificaran de una iteración a otra.

- Todo esto se intentó evitar en Pascal, que proporciona un ciclo **for** incluso más simple que el ciclo **DO** de FORTRAN.

Estructuras de Control

- La sintaxis para un ciclo **for** en Pascal es la siguiente:

for < nombre > **: =** < expresión > { **to** / **downto** }
<expresión > **do** < sentencias >

Estructuras de Control

- El uso de **to** o **downto** permite incrementos o decrementos de uno dentro del circlo.
- Algo curioso del ciclo **for** de Pascal es que no se permiten incrementos o decrementos diferentes de uno.
- Esto ha sido muy criticado y se considera como un mecanismo que se sobresimplificó en Pascal.

Estructuras de Control

- Este constructor debería permitir el uso de incrementos no unitarios (por ejemplo, fracciones decimales).

- El diseño del lenguaje no se complicaría mucho más con esta modificación, y el constructor se haría mucho más poderoso para el programador.

Estructuras de Control

- Pascal también especifica que los límites del ciclo de calculan una sola vez, al entrar al mismo y no en cada iteración, como se hacía en Algol.

- La consecuencia de esta decisión de diseño es que el ciclo **for** se ejecuta siempre durante un número definido de veces (a menos, por supuesto, que el programador decida usar un **goto** dentro del ciclo).

Estructuras de Control

- Debido a su diseño, el ciclo **for** de Pascal es más eficiente y más fácil de entender que la estructura equivalente proporcionada por Algol.

- El ciclo **for** es un *iterador definido*, es decir una estructura que itera durante un número definido (predecible) de veces.

Estructuras de Control

- Pascal proporciona dos *iteradores indefinidos*, es decir, para aquellos casos en que no sabemos cuántas iteraciones se realizarán al momento de entrar al ciclo:

while < condición > **do** < sentencia >

repeat < sentencia > **until** < condición >

Estructuras de Control

- El ciclo **while** es un ejemplo de un iterador con la decisión al inicio (*leading decision*).
- El ciclo **repeat** es un ejemplo de un iterador con la decisión al final (*trailing decision*).

Estructuras de Control

- Pascal no proporciona un iterador para tomar decisiones en medio (*mid decision*), así que tiene que usarse **goto** para este tipo de ciclos:

```
while true do
        begin
                . . . primera mitad del ciclo . . .
                if <terminamos> then goto 99;
                . . . segunda mitad del ciclo . . .
                end;
        99:
```

Estructuras de Control

- ¿Puede eliminarse el **goto** si se incluye un ciclo con la decisión en medio?

- Algunos lenguajes modernos como Ada proporcionan un iterador indefinido con la decisión en medio, haciendo innecesario el uso del **goto**.