Lenguajes de Programación

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

Enumeraciones

- Los únicos operadores permisibles son:

:=, **succ**, **pred**, =, < >, <, >, <=, >=

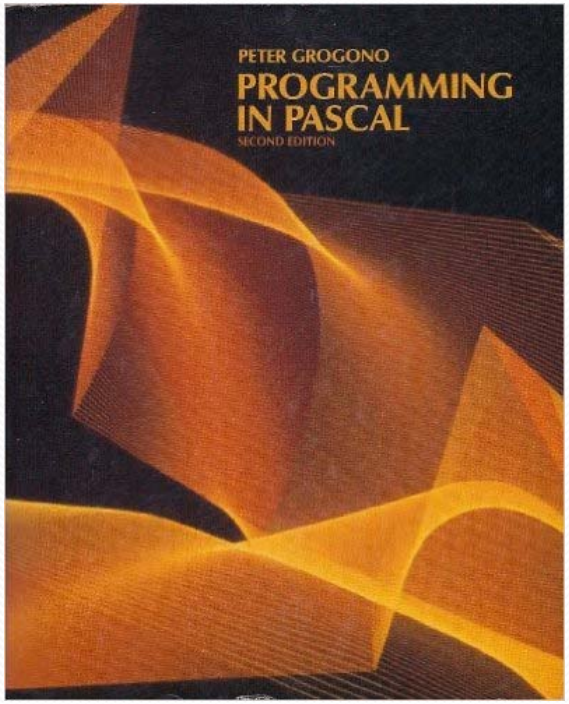
y operaciones tales como **succ**(Sab) y **pred**(Dom) producirán mensajes de error.

Enumeraciones

- Podemos resumir los beneficios de las enumeraciones de la manera siguiente:

1) Son de **alto nivel** y **orientadas a las aplicaciones**. Permiten a los programadores expresarse de una manera más directa.
2) Son **eficientes** porque permiten al compilador economizar almacenamiento y las operaciones asociadas a ellas se pueden realizar de manera eficiente.
3) Son **seguras** porque el compilador impide que el programador no realice operaciones sin sentido.

Definición de Subrangos



- Pascal permite al programador definir subrangos de cualquier tipo de datos discreto (enteros, caracteres y enumeraciones).

Definición de Subrangos

- Ejemplo:

var DiaMes : 1..31;

- Esto significa que *DiaMes* puede tomar valores en el rango 1 a 31 (inclusive).

Definición de Subrangos

- Si intentamos asignar a esta variable un valor fuera de este rango, obtendremos un mensaje de error.

- Este es un mecanismo muy **seguro** (es fácil interceptar errores al usar subrangos) y **eficiente** (podemos usar sólo unos cuantos bits para representar un rango grande en vez de usar enteros de longitud fija).

Definición de Subrangos

- Existe evidencia de que si un programador usa frecuentemente subrangos, eso propicia que muchos de sus errores de programación se manifiesten a través de violaciones de subrangos.

- Los subrangos permiten economizar espacio de almacenamiento. Por ejemplo, si queremos almacenar DiaMes (ejemplo que vimos anteriormente), se requieren sólo 5 bits, porque el subrango toma 31 valores y $2^5 = 32$.

Definición de Subrangos

- Pascal permite definir subrangos de cualquier tipo discreto (o sea, enteros, caracteres y enumeraciones).
- Sin embargo, no permite definir subrangos de los números reales, dado que este tipo de datos es continuo.

Conjuntos

- Pascal proporciona la capacidad de manipular pequeños conjuntos finitos usando operaciones estándar de teoría de conjuntos.

- Los conjuntos son un tipo de datos muy útil que pueden ser implementados muy eficientemente y proporcionan un nivel muy alto de abstracción.

Conjuntos

- La descripción del tipo conjunto tiene la forma siguiente:
  - set of *<tipo ordinal>*

donde *<tipo ordinal>* es una enumeración (incluyendo los tipos **char** y **Boolean**), un subrango o el nombre de uno de éstos. Estas restricciones en el tipo que se puede usar para los conjuntos, permite una implementación eficiente de los mismos.

Conjuntos

- Podemos declarar conjuntos usando:

var S,T : **set of** 1..10;

Esto significa que S y T son variables que pueden conformarse por un conjunto de 10 elementos cada una.

Conjuntos

- Posteriormente, podemos asignar valores a un conjunto usando:

```
S:=[1,2,3,4,6];
T:=[1..4];
```

- Esta segunda asignación significa que el conjunto contiene los enteros del 1 al 4.

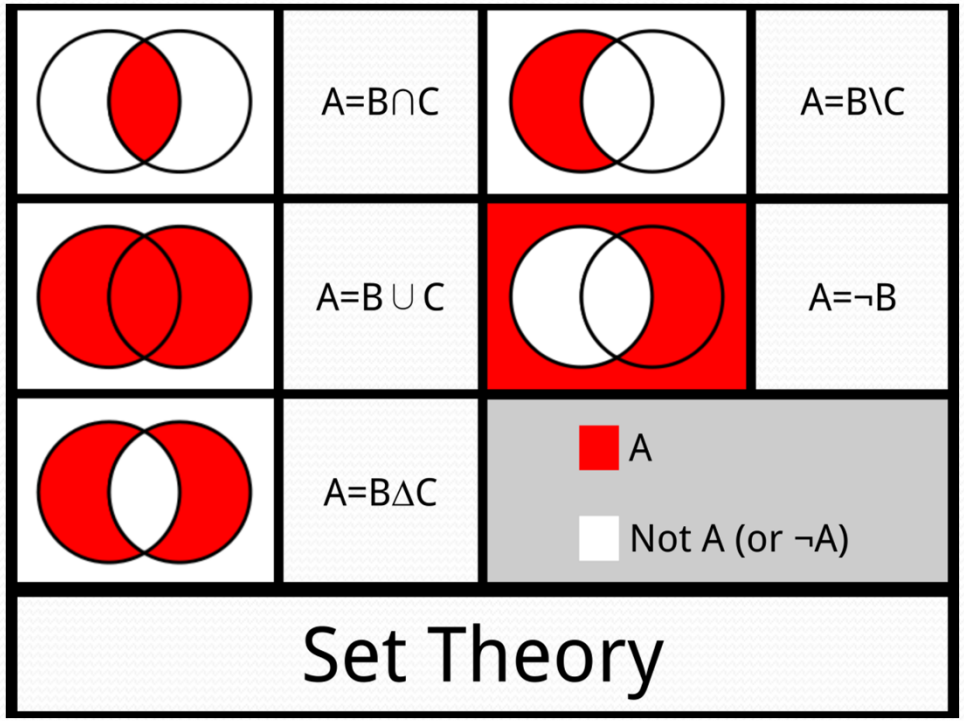
Conjuntos

- También se proporcionan operaciones regulares de conjuntos. Por ejemplo, la intersección (∩) se denota mediante el asterisco (*):

```
T:=S*T;
```

En este caso, T contendría [1, 2, 3, 4].

Conjuntos



- La unión se denota mediante el signo '+', la diferencia se indica mediante el signo '-' y la igualdad mediante el signo '='.

Conjuntos

- La relación de subconjunto se denota mediante <=.
- Por ejemplo, si:

S:=[1,2,3,5,7];
T:=[1,2,3,5];

entonces 'T<=S' devolverá CIERTO (**true**).

Conjuntos

- Adicionalmente, se proporcionan los operadores relacionales ‘<>’ (desigualdad) , ‘<=‘ y ‘>=‘.
- Aunque no se proporcionan ‘<‘ y ‘>’, pueden derivarse muy fácilmente.
- Por ejemplo, ‘S<T’ es equivalente a ‘S<=T **and** S<>T’.

Conjuntos

- El constructor de conjuntos es de muy alto nivel, puesto que permite a los programadores presentar algoritmos en una notación muy natural.

- Además, el constructor es muy eficiente porque está implementado usando operaciones sobre vectores de bits.

Conjuntos

- Ejemplo:

```
var S: set of 1..10;
S:=[1,2,3,5,7];
```

- En este caso, sólo se requieren 10 bits para representar a todo posible elemento de S.

Conjuntos

- Por ejemplo, para representar al conjunto S antes definido, haríamos lo siguiente:

S = 1 1 1 0 1 0 1 0 0 0

1 2 3 4 5 6 7 8 9 10

Conjuntos

- Las operaciones de conjuntos también se pueden implementar muy eficientemente.

- Por ejemplo:

S:=[1,2,3,5,7];
T:=[1,2,3,4,5,6];

Conjuntos

- **<u>Intersección</u>:** Se obtiene al efectuar un AND lógico de los bits que representan los elementos de los 2 conjuntos:

$$S = 1\ 1\ 1\ 0\ \ 1\ 0\ 1\ 0\ 0\ 0$$

$$T = 1\ 1\ 1\ 1\ \ 1\ 1\ \ 0\ 0\ 0\ 0$$

$$S \cap T = 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0$$

$$S*T = [1,2,3,5];$$

Conjuntos

- **<u>Unión</u>:** Se obtiene al efectuar un OR lógico de los bits que representan los datos de los 2 conjuntos:

S= 1 1 1 0 1 0 1 0 0 0

T= 1 1 1 1 1 1 0 0 0 0

$S \cup T$ = 1 1 1 1 1 1 1 0 0 0

S+T = [1,2,3,4,5,6,7];

Conjuntos

- **<u>Complemento</u>:** Se obtiene al efectuar un NOT lógico de los bits que representan los elementos del conjunto:

S= 1 1 1 0 1 0 1 0 0 0

NOT S= 0 0 0 1 0 1 0 1 1 1

COMPLEMENTO DE S = [4,6,8,9,10]

Conjuntos

- **<u>Diferencia</u>:** A-B=A AND (NOT B).

S = 1 1 1 0 1 0 1 0 0 0

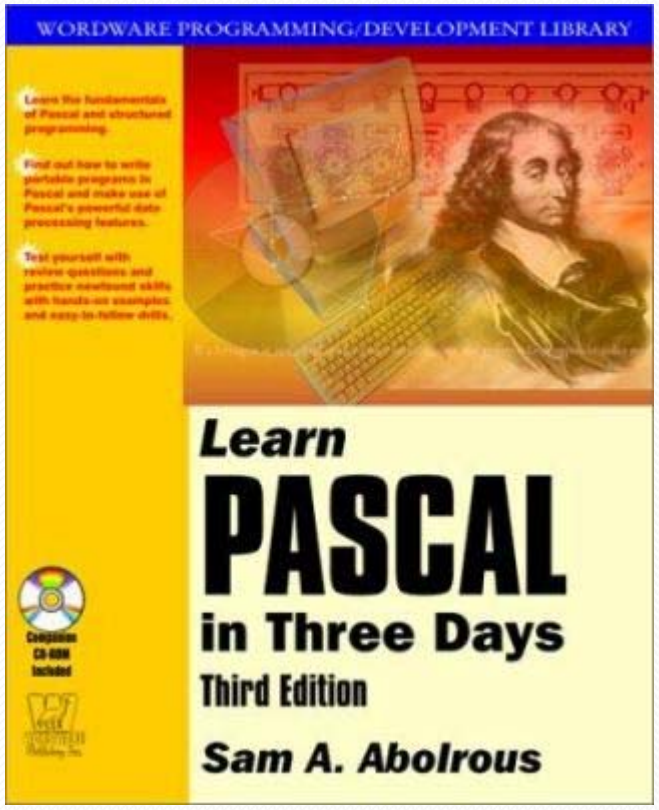
T= 1 1 1 1 1 1 0 0 0 0

NOT T= 0 0 0 0 0 0 1 1 1 1

S&(!T) = 0 0 0 0 0 0 1 0 0 0

S-T = [7]

Arreglos



- Los arreglos en Pascal pueden usar subíndices de tres tipos: enteros, enumeraciones o caracteres.

Arreglos

- Ejemplo:

```
var Dias : array[Lun..Vie] of 0..24;
```

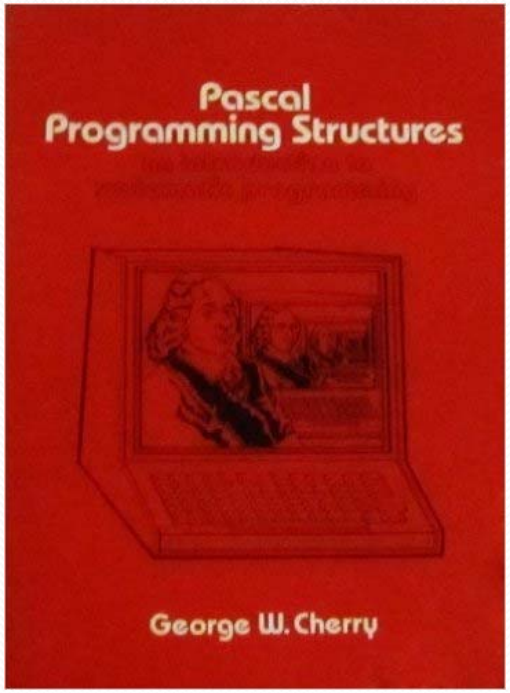
- Una de las ventajas principales de usar subrangos en este caso es que anulamos la posibilidad de tener subíndices ilegales.

Arreglos

- De esta manera, estamos dejando la tarea de chequeo para el tiempo de compilación, en vez de postergarla al tiempo de ejecución.

- De hecho, Pascal permite usar cualquier tipo de datos finito como subíndice de un arreglo.

Arreglos

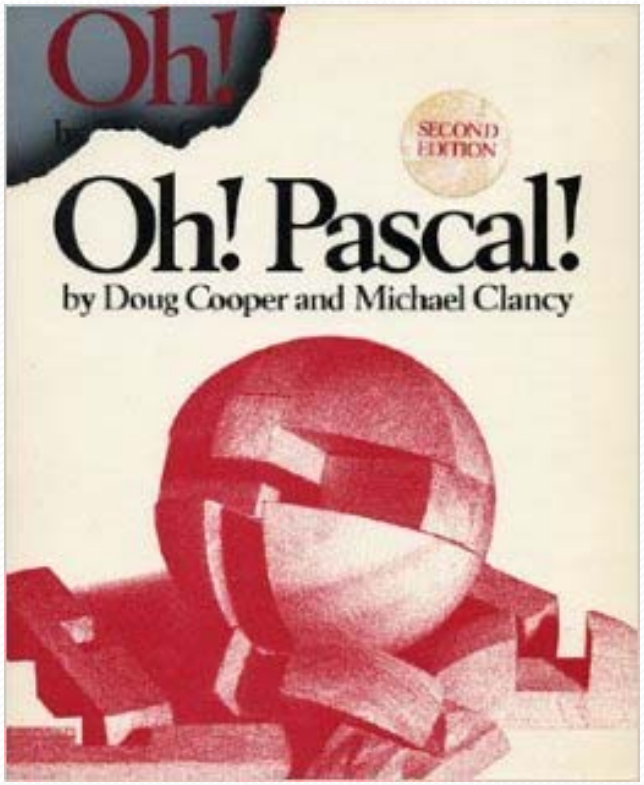


- Pascal permite arreglos de cualquier tipo (enteros, reales, caracteres, enumeraciones, subrangos, registros, apuntadores, etc.).

Arreglos

- Realmente Pascal no permite arreglos multidimensionales.
- Sin embargo, debido a que el tipo base de un arreglo puede ser cualquiera, se incluye también el caso de otro arreglo.
- De esa manera, podemos tener arreglos de arreglos (o sea, arreglos de cualquier número de dimensiones).

Arreglos



- Sin embargo, una limitante de los arreglos en Pascal es que son estáticos y no semi-dinámicos como en ALGOL.

Arreglos

- Esto se debe a una interacción de funciones:

  1) Todos los tipos deben poder determinarse en tiempo de compilación.

  2) Las dimensiones son parte del tipo de un arreglo.

Arreglos

- Por sí solas, ninguna de estas dos decisiones parecen peligrosas y de hecho, tienen mucho sentido si se analizan por separado.
- Sin embargo, cuando se juntan, tenemos un problema.
- Dado que la dimensión de un arreglo es parte del tipo y debido a que los tipos deben poderse determinar en tiempo de compilación, entonces en consecuencia las dimensiones del arreglo deben ser estáticas (o sea, conocerse en tiempo de compilación).

Arreglos

- Hay otro problema de interacción de funciones más serio todavía:

  1) Las dimensiones son parte del tipo de un arreglo.

  2) Pascal usa chequeo fuerte de tipos; por tanto, el tipo de un parámetro pasado a un procedimiento invocado debe coincidir con el del procedimiento invocador.

Arreglos

- Esto significa que si tenemos un arreglo cuyo subíndice está definido como una enumeración 1..100, y le queremos pasar a otro arreglo del mismo tamaño, pero cuyo subíndice está definido como una enumeración 0..99, los 2 arreglos se considerarán de tipos distintos y obtendremos un error.

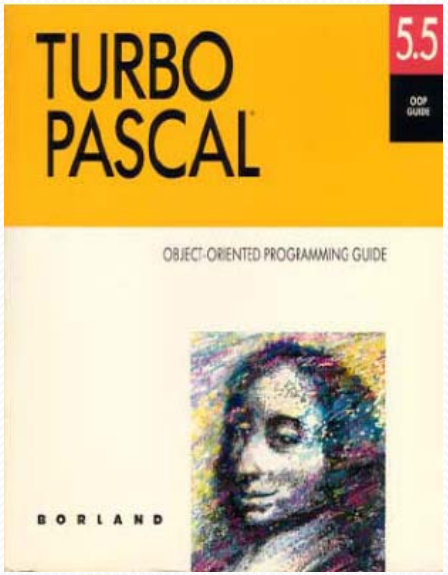
Arreglos

- Esto viola el **Principio de Abstracción**, porque tenemos que escribir un procedimiento separado para cada subrango distinto de nuestro programa.

Principio de Abstracción

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

Arreglos



- El comité que estandarizó Pascal resolvió este problema introduciendo el denominado “esquema de arreglo conformante” para especificar el parámetro de un procedimiento a invocarse.

Arreglos

- Ejemplo:

procedure suma(x:**array**[linf. .lsup:**integer**] **of real**):**real**;

- De esta manera, cualquier arreglo indizado con enteros puede pasarse al procedimiento “suma”.

Arreglos

- Con este esquema, cuando pasamos un arreglo al procedimiento “suma”, los identificadores “linf” y “lsup” se asocian a los límites inferior y superior del tipo de índice que tenga “x”.

- Un uso típico de estos identificadores sería en los límites de un ciclo for:

```
for i:= linf to lsup do
        total := total + x[i];
```

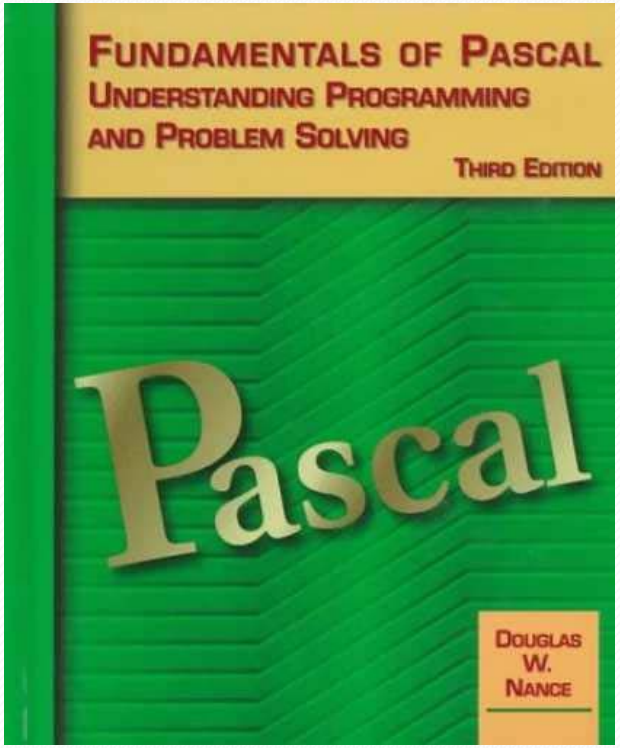
Arreglos

- La sintaxis de un arreglo conformante es la siguiente:

array [<identificador> .. <identificador>: <tipo del identificador>] **of** <tipo del identificador>

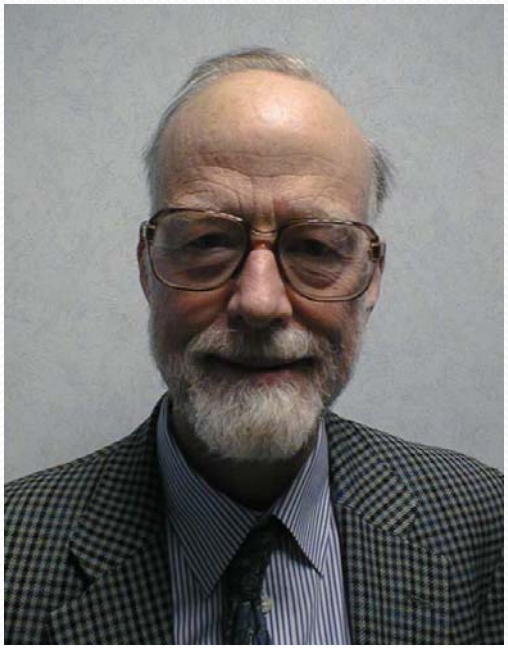
- Aunque esto resuelve el problema del paso de arreglos en Pascal, hace al lenguaje menos regular y le introduce una mayor complejidad.

Registros



- Una de las estructuras de datos más importantes de Pascal son los registros.
- Esta estructura de datos permite agrupar datos de diferente tipo.

Registros



- La idea de los registros se tomó de COBOL, y Anthony Hoare sugirió incorporarlos en los lenguajes orientados a aplicaciones científicas.

Registros

- Al igual que un arreglo, los registros tienen componentes.
- Sin embargo, a diferencia de los arreglos, los registros permiten componentes de tipos diferentes.

Registros

```
type persona=
   record
         nombre : cadena;
         edad : 18..90;
         genero : (masculino, femenino);
   end;

cadena=packed array[1..30] of char;
```

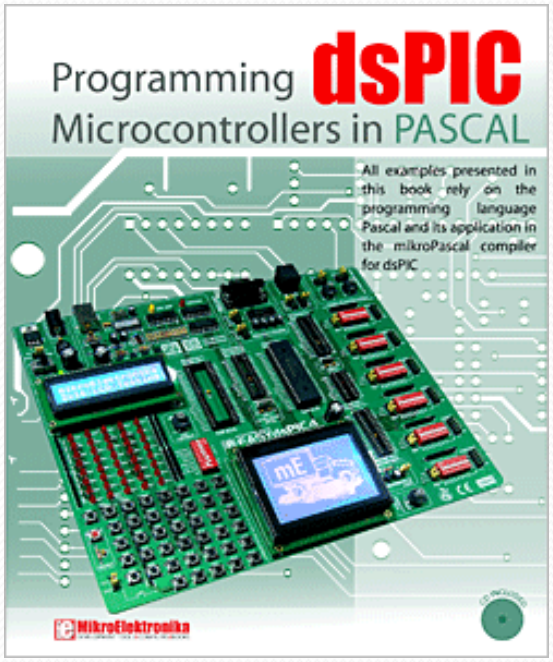
Registros

- Puede seleccionarse un componente de un arreglo usando un punto entre el nombre del registro y el nombre del componente.

- Ejemplo (suponiendo que nuevo es un registro de tipo “persona”):

  - nuevo.edad = 20;

Registros



- Los registros son estructuras definidoras de entornos, ya que agrupan nombres de campos que no son visibles fuera de la declaración del registro, a menos que se use el operador punto para “abrirlo”.

Registros

- Si necesitamos colocar datos en varios componentes de un registro de forma sucesiva, podemos usar la sentencia "**with**":

```
with nuevo do
        begin
                edad:=20;
                genero:=masculino;
        end;
```

Esta habilidad de poder entrar a otro ambiente y hacer visibles sus variables es un concepto que se volvió muy importante en los lenguajes de programación modernos.

Registros vs. Arreglos

	Tipo de Elementos	Selectores
Arreglo	Homogéneos (Menos generales)	Dinámicos (Más generales)
Registro	Heterogéneos (Más generales)	Estáticos (Menos generales)

Registros vs. Arreglos

- La razón por la que Pascal incluye tanto arreglos como registros es porque son estructuras de datos de naturaleza distinta.

- Los registros agrupan datos heterogéneos, mientras los arreglos sólo agrupan datos homogéneos.

Registros vs. Arreglos

- Los arreglos pueden usar subíndices cuyo valor se calcule en tiempo de ejecución. Por ejemplo:

A[E]

- donde “E” es una expresión cuyo valor se determina en tiempo de ejecución.

Registros vs. Arreglos

- Esto es muy útil porque permite, por ejemplo, la manipulación de los elementos de un arreglo (a través de un ciclo) en tiempo de ejecución.

Registros vs. Arreglos

- No puede hacerse lo mismo con los registros. Esto se debe a que los tipos deben ser checados en tiempo de compilación.

- De tal forma, no pueden hacerse cosas como:

registro.E

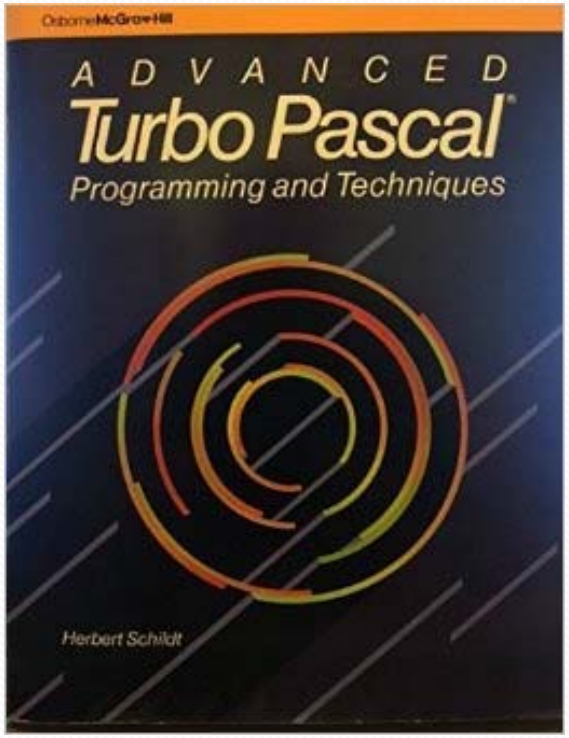
- donde “E” es una expresión cuyo valor se determina en tiempo de ejecución.

Registros Variantes

- Pascal también proporciona otro tipo de estructura de datos a la que se denomina "**registros variantes**".

- Estos registros permiten agrupar diferentes campos de acuerdo al estatus con el que estén asociados.

Registros Variantes



- Los registros variantes son una especie de sentencia “case” dentro de un registro y son muy útiles cuando tenemos ciertos atributos que sabemos de antemano que sólo usaremos en ciertas situaciones.

Registros Variantes

- Los registros variantes ahorran memoria, porque como sólo una de las estructuras definidas dentro de ellos puede usarse a la vez, el compilador sólo necesita reservar memoria para la mayor de ellas (en vez de tener que hacerlo para todos los componentes del registro).

Registros Variantes

- Esta propiedad de compartición de espacio en memoria es similar a la de los bloques disjuntos en ALGOL.

- De hecho, también en el caso de los registros variantes se permite el anidamiento.