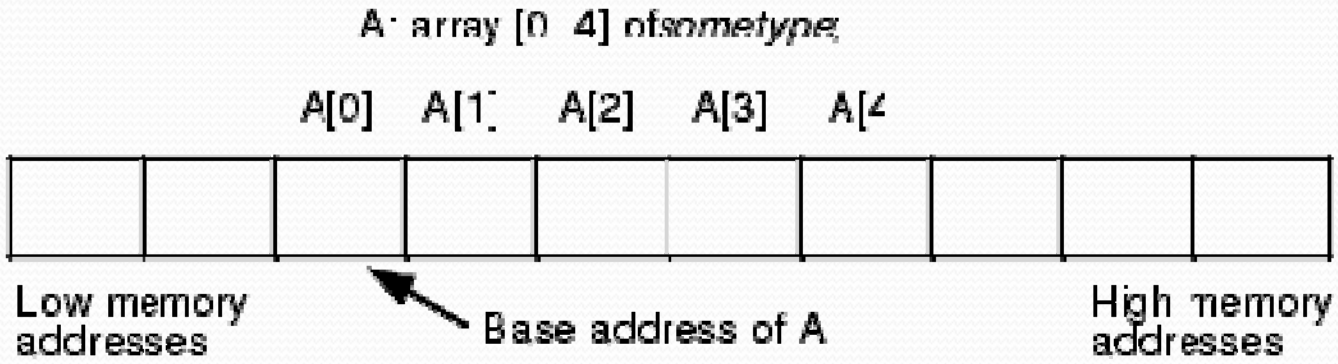
Lenguajes de Programación

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

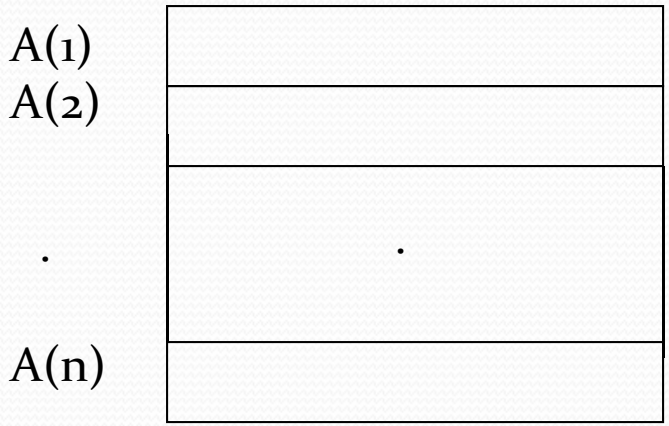
Implementación de Arreglos



- Los arreglos unidimensionales se implementan usando regiones contiguas de memoria. Esto permite un esquema de indización muy eficiente.

Implementación de Arreglos

- Por ejemplo, para el arreglo A(n), la memoria luciría así:



Implementación de Arreglos

- Si usamos $\alpha\{A(1)\}$ para expresar la ecuación de direccionamiento de A(1), entonces la expresión:

$$\alpha\{A(i)\}=\alpha\{A(1)\}-1+i$$

Para cualquier entero i>1. A esta se le llama la "ecuación de direccionamiento del arreglo A".

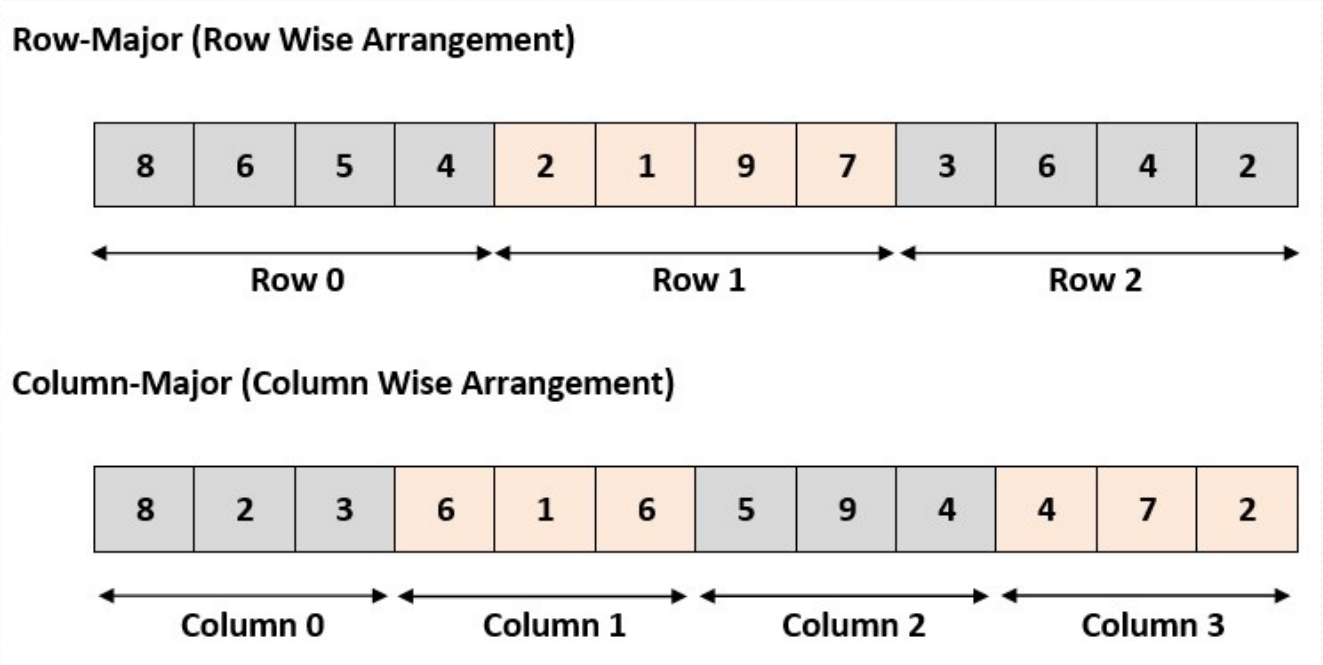
Implementación de Arreglos

- Extendamos ahora nuestro análisis a arreglos bidimensionales.

- Consideremos la siguiente declaración:

DIMENSION A(m,n)

Implementación de Arreglos



- FORTRAN organiza los arreglos en orden “columna principal” (*column-major*), lo que significa que las columnas ocupan posiciones de memoria adyacentes tal y como se muestra en la figura del acetato siguiente.

Columna Principal

A(1,1)	α
A(2,1)	$\alpha + 1$
. . .	
A(m,1)	$\alpha + m - 1$
A(1,2)	$\alpha + m$
. . .	
A(m,2)	$\alpha + m + m - 1$
A(1,3)	$\alpha + m + m = \alpha + 2m$
. . .	
A(m, n)	$\alpha + nm - 1$

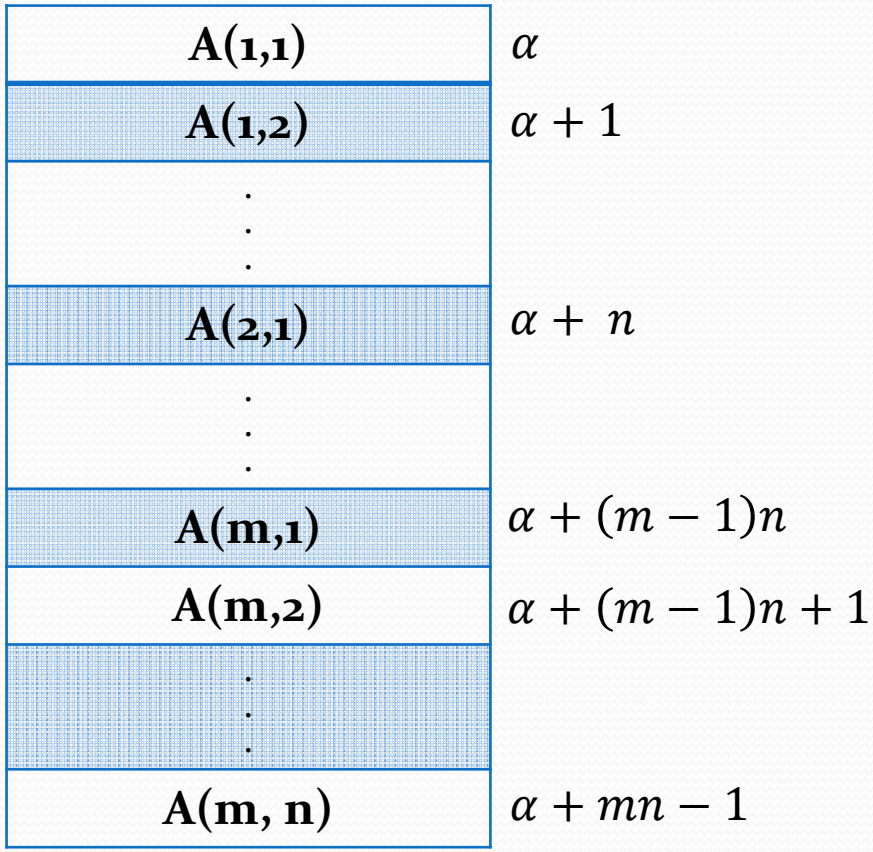
$$\alpha = \alpha\,\{A(1,1)\}$$

Implementación de Arreglos

- Donde: $\alpha=\alpha\{A(1,1)\}$. Por lo tanto, la ecuación de direccionamiento para arreglos bidimensionales con ordenamiento de columna principal es:

$$\alpha\{A(I,J)\}=\alpha\{A(1,1)\}+(J-1)m+I-1$$

Fila Principal

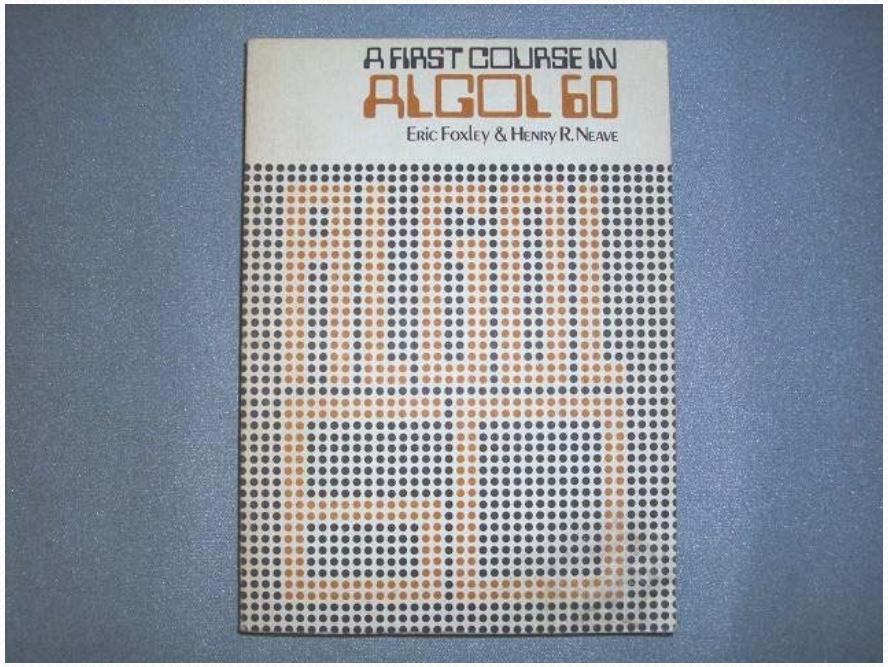


Implementación de Arreglos

- Por lo tanto, la ecuación de direccionamiento para arreglos bidimensionales con ordenamiento de fila principal es:

$$\alpha\{A(I,J)\}=\alpha\{A(1,1)\}+(I-1)n+J-1$$

Historia del ALGOL-60



- Un comité de 8 representantes de Europa y América diseñaron ALGOL (*Algorithmic Language*) en 1958, en una respuesta a la creciente necesidad de tener un solo lenguaje de programación que fuese universal e independiente de la arquitectura de las computadoras de la época.

Historia del ALGOL-60

- Los 3 objetivos principales del ALGOL eran:

1) Debía ser tan cercano como fuese posible a la notación matemática estándar y debía ser legible dando pocas explicaciones adicionales.

2) Debía ser posible usarlo para la descripción de procesos de cómputo en las publicaciones científicas.

3) Debía ser traducible, de manera mecánica, a programas en lenguaje máquina.

Historia del ALGOL-60

- El diseño original de ALGOL tomó sólo 8 días (el lenguaje consistía de unos cuantos constructores de propósito general y no de una cantidad innumerable de funciones barrocas).
- En mayo de 1960, ALGOL-60 (la primera revisión de ALGOL-58) estaba listo.
- El reporte de ALGOL-60 constaba de sólo 15 páginas, lo que constituía un logro notable considerando las longitudes de los reportes de los lenguajes de programación de la época (y peor aún, de los actuales).

Historia del ALGOL-60

Lenguaje	Longitud del Reporte
FORTRAN 77	430
PL/1	420
BASIC	360
Ada	340
C	220

Historia del ALGOL-60

Lenguaje	Longitud del Reporte
Pascal	~30(120)
Scheme	~50
Common LISP	~1000
COBOL	810

Historia del ALGOL-60

- Una de las razones principales de la brevedad del reporte de ALGOL-60 fue el uso de la denominada notación BNF (*Backus-Naur Form*) para la sintaxis.

- Asimismo, la semántica del lenguaje se describió usando un inglés claro, preciso y carente de ambigüedades.

Diseño muy General

Reprinted from the COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY
Vol. 3, No. 5, May 1960
Printed in U.S.A.
With typographical corrections as of April 1, 1962

Report on the Algorithmic Language ALGOL 60

PETER NAUR (*Editor*)

J. W. BACKUS, F. L. BAUER, J. GREEN, C. KATZ, J. MCCARTHY, A. J. PERLIS, H. RUTISHAUSER, K. SAMELSON, B. VAUQUOIS, J. H. WEGSTEIN, A. VAN WIJNGAARDEN, M. WOODGER

Dedicated to the Memory of WILLIAM TURANSKI

- ALGOL-60 tiene reglas recursivas: una cuantas reglas son utilizadas para producir diferentes formas sintácticas, en vez de usar reglas especiales para casos especiales.

Diseño muy General

- Ejemplos:
- 1) Agregar un carácter a un identificador, usualmente nos produce un identificador diferente.
- ¿Por qué no hacer eso siempre?
- Como resultado, no deben imponerse límites a la longitud de los identificadores (**Principio Cero-Uno-Infinito**).

Diseño muy General

- 2) Un arreglo bidimensional puede verse como un arreglo de arreglos.
- ¿Por qué no permitir arreglos de tipo N (por ejemplo, un arreglo más pequeño) para *cualquier* tipo N?
- De esta manera, ya no existen restricciones en cuando a las dimensiones de los arreglos (**Principio Cero-Uno-Infinito**).

Diseño muy General

- 3) Un programa puede incluir subprogramas. ¿Por qué no permitir que un subprograma incluya subprogramas? Esto nos lleva a los entornos anidados.

- 4) Usualmente, un contexto que acepta una literal, también acepta una variable una expresión. ¿Por qué no hacer eso siempre? En consecuencia, podemos proporcionar arreglos cuyo tamaño se calcule a la entrada de un bloque.

Diseño muy General

- Si en un arreglo se puede especificar el límite superior, ¿por qué no también especificamos el límite inferior? ALGOL-60 proporciona arreglos semi-dinámicos.

- Un subprograma usualmente llama a otros subprogramas. ¿Por qué no permitir que llame a *cualquier* subprograma, incluyéndose a sí mismo? Esto nos lleva a proporcionar recursividad.

Diseño muy General

- Una estructura de control se construye alrededor de otros comandos (p.ej., las asignaciones). Una estructura de control **es** un comando. ¿Por qué no permitir entonces el uso de cualquier estructura de control en lugares donde se espera un comando? Esto nos lleva a estructuras de control anidadas.

- Claro que este diseño tan general, tiene sus problemas también. Por ejemplo, las estructuras de control de ALGOL-60 son tan generales que los límites de un ciclo tienen que ser re-evaluados a cada iteración.

Organización Estructural

- ALGOL-60 tiene una estructura jerárquica, o anidada, a lo largo de todo su diseño, tal y como lo ejemplificamos previamente.

- Por ejemplo, los programas en ALGOL están compuestos de un número arbitrario de ambientes anidados. Esto decrementa significativamente el número de **goto**'s que se requieren en un programa.

Organización Estructural

- Los constructores en ALGOL-60 son de 2 tipos:

- 1) **Declarativos:** Se subdividen en 3 tipos: declaraciones de variables, declaraciones de procedimientos y declaraciones de tipo **switch**.

- Los únicos tipos permisibles son: **integer**, **real** y **boolean**. Los arreglos en ALGOL-60 pueden tener cualquier número de dimensiones y no tienen que empezar en 1 (como en FORTRAN).

Organización Estructural

- Asimismo, el lenguaje tiene arreglos semi-dinámicos , puesto que sus límites pueden ser calculados en tiempo de ejecución (es decir, el lenguaje permite el uso de variables para ambos límites del arreglo), pero no se les permite cambiar de tamaño en tiempo de ejecución, como en el ALGOL-68 (el cual sí contaba con arreglos dinámicos).

Organización Estructural

- ALGOL-60 distingue entre *typed procedures* (o funciones) y *untyped procedures* (subrutinas como las de FORTRAN). La palabra “typed” se usa para denotar si el procedimiento regresa o no un valor.

- ¿Qué otro lenguaje conoce que divida sus procedimiento de manera similar?

Organización Estructural

- 2) **<u>Imperativos</u>:** Se subdividen en 2 tipos: computaciones y de flujo de control. No existían constructores de entrada/salida en ALGOL-60, puesto que dichos procedimientos se proporcionaban en bibliotecas externas (como en C).

- La única sentencia computacional es la asignación (como en FORTRAN), la cual tiene el formato:

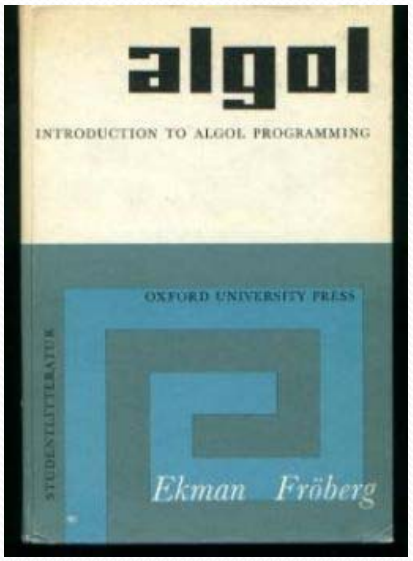
variable:=expresión

Organización Estructural

- Nótese el uso de ':=' para distinguir de la igualdad '=' (como en matemáticas). Esta decisión influenció varios lenguajes de programación modernos.

- ALGOL-60 tenía **goto**, **if-then-else** y ciclos con **for**. Asimismo, también tenía invocación a procedimientos.

Estructuras Denominativas



- El constructor principal en ALGOL-60 es el bloque. Una de las contribuciones más importantes de este lenguaje fue precisamente la idea de una *sentencia compuesta*. Esta estructura permite el uso de una secuencia de sentencias en donde antes se permitía sólo una.

Estructuras Denominativas

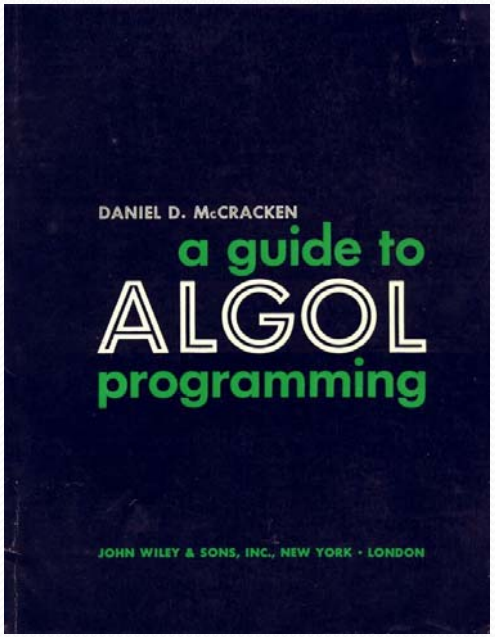
- Ejemplo:

```
for i:=1 step 1 until N do
  begin
        if Array[i]<450000 then Array[i]:=0;
        mult:=mult*Array[i];
        Print Real(mult)
  end
```

Estructuras Denominativas

- ALGOL-60 era tan regular que incluso el cuerpo de un procedimiento se consideraba como que estaba formado por una sola sentencia por omisión. Esto se contrapone a la práctica común en los lenguajes modernos (p.ej., Pascal y C) de usar **begin..end** para declarar un procedimiento.

Estructuras Denominativas



- Esto produjo eventualmente la denominada “estructura de bloques”, que originó toda una generación de lenguajes de programación. Esto fue también el origen del diseño modular y marcó el retiro de la sentencia **goto**.

Estructuras Denominativas

- ALGOL-60 resuelve el problema originado con el bloque **COMMON** de FORTRAN, permitiendo la definición de bloques anidados.

- Esto hace innecesario tener que repetir la declaración (o el bloque **COMMON**), y permite un alcance global sin introducir los problemas asociados con las variables globales.

Estructuras Denominativas

- Esto sigue el **Principio de Abstracción**, puesto que se factoriza un patrón común (en este caso una declaración de datos compartidos) y el lenguaje hace innecesario para el programador el tener que repetir el mismo proceso una y otra vez (como se hacía en FORTRAN).

Estructuras Denominativas

- Ejemplo:

```
begin
        integer array NAME, ADDRESS, PHONE[1:100];
        procedure FINDNAME(n);
        . . . FINDNAME procedure . . .
        procedure FINDPHONE(n, l, t);
        . . . FINDPHONE procedure . . .
        procedure SORTNAMES(n);
        . . . SORTNAMES procedure . . .
end
```

Estructuras Denominativas

- El problema con este esquema es que los programadores no pueden prevenir el acceso inadvertido a las variables.

- En el segmento de código anterior, por ejemplo, podría cambiarse accidentalmente (o intencionalmente) el contenido del arreglo NAME sin tener que pasar por ninguno de los procedimientos definidos dentro del bloque.

Estructuras Denominativas

- Esto crea un problema de mantenimiento, puesto que tenemos que dar seguimiento a cualquier referencia a las definiciones globales dentro del bloque para asegurarnos que cuando las cambiemos nada más resulte alterado.

Estructuras Denominativas

- A este problema se le denomina “acceso indiscriminado”, y no hay manera de resolverlo en ningún lenguaje de programación estructurado.

- Irónicamente, los bloques **COMMON** de FORTRAN no presentaban este problema (si bien, tenían otros peores).

Estructuras de Datos

- Puesto que ALGOL también era un lenguaje científico (como FORTRAN), sus tipos de datos primitivos son escalares matemáticos: **integer**, **real** y **boolean**.

Estructuras de Datos

- ALGOL-60 sólo permitía precisión simple porque sus diseñadores no pudieron encontrar la manera de hacer portátil la doble precisión.

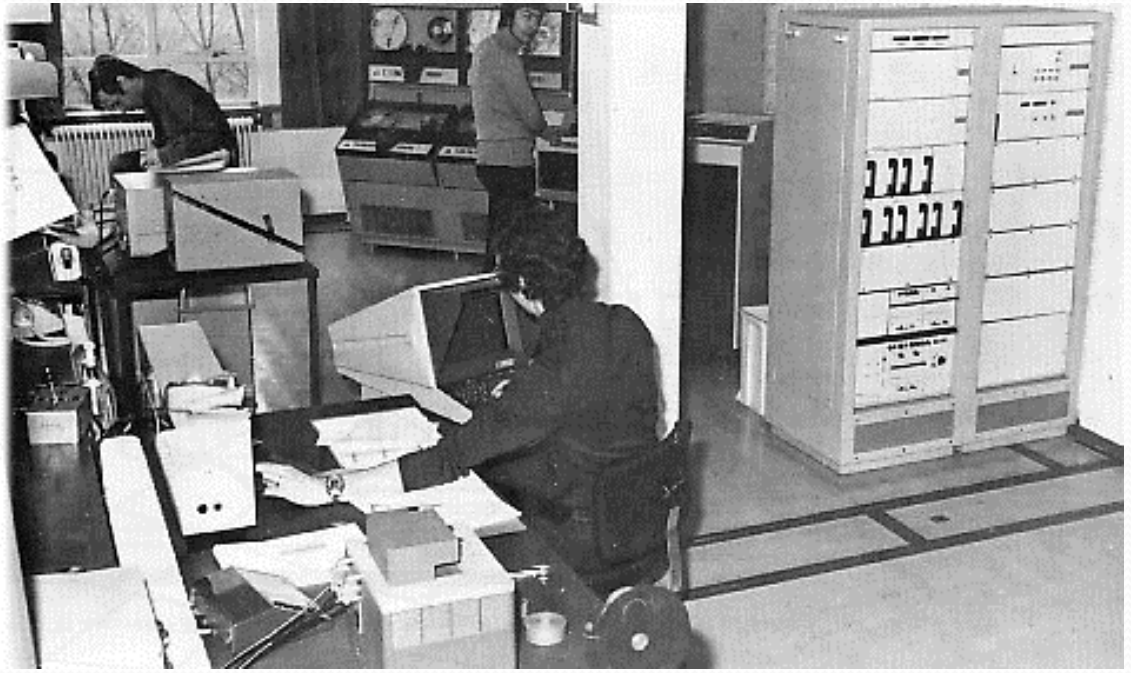
- Por ejemplo, si en una computadora la doble precisión involucra 32 bits, pero en otra involucra sólo 16, entonces todas las declaraciones de variables del programa debían modificarse para hacerlo funcionar adecuadamente.

Estructuras de Datos

- La solución propuesta por los diseñadores de ALGOL-60 fue proporcionar un solo tipo real que permitiría sólo precisión simple.

- Sin embargo, si la computadora tenía capacidad para mayor precisión entonces se usarían comandos especializados para acceder a ella.

Estructuras de Datos



- El problema es que la precisión simple también varía de computadora a computadora. ¿Por qué no usar mejor modificadores de tipo para permitir precisiones mayores?

Estructuras de Datos

- **<u>Solución en Ada</u>:** El programador especifica la precisión y el compilador escoge la representación.

  1) Confiabilidad

  2) Eficiencia para aquellos que conocen su computadora y su compilador

Estructuras de Datos

- **<u>Solución en C</u>:** Especificar modificadores de tipo denominados “short” y “long”.

  1) Portabilidad (a veces)

  2) Eficiencia (a veces)

Estructuras de Datos

- ALGOL-60 no proporcionaba números complejos porque éste no es un tipo de datos primitivo. Por lo tanto, los diseñadores de ALGOL decidieron que no era demasiado difícil o ineficiente implementarlos directamente en el lenguaje.

- Esta decisión, sin embargo, tiene varias implicaciones. El programador ahora es responsable de escribir procedimientos para realizar todo tipo de operaciones con números complejos.

Estructuras de Datos

- El programador debe también resolver las coerciones con los otros tipos datos, así como hallar la representación adecuada para mantenerlos como ciudadanos de primera clase, etc.

- Los beneficios de esta decisión son que tendremos una definición más corta del lenguaje y un traductor a lenguaje máquina más corto y simple.

Estructuras de Datos

- Los costos asociados con esta decisión son los inconvenientes causados a los programadores que requieran números complejos.

- Asimismo, la implementación de este tipo será necesariamente lenta comparada con la que pudo haberse proporcionado como parte del lenguaje.

Estructuras de Datos

- ¿Qué piensa acerca de esta decisión de los diseñadores de ALGOL-60 de no incluir los números complejos como un tipo de datos más del lenguaje?

Estructuras de Datos

- Las cadenas en ALGOL-60 son ciudadanos de segunda clase, porque sólo pueden pasarse como parámetros a los procedimientos. No hay operadores, relaciones, coerciones o declaraciones de variables para las cadenas.

- Las cadenas, sin embargo, no comprometen la integridad del ALGOL-60 (como en el caso de FORTRAN), mas sin embargo son prácticamente inútiles, puesto que el programador no puede manipularlas.

Estructuras de Datos

- ¿Para qué incluir entonces a las cadenas? Hay que recordar que el argumento principal con respecto al por qué las cadenas son ciudadanos de segunda clase gira en torno al hecho de que lenguajes como ALGOL-60 y FORTRAN estaban orientados a aplicaciones científicas.

Estructuras de Datos

- En contraste, los lenguajes orientados a los negocios (como por ejemplo, COBOL), permitían una amplia gama de operaciones con cadenas.

- Sin embargo, ALGOL-60 fue el último lenguaje de programación importante en adoptar esta posición, ya que prácticamente todos los lenguajes modernos permiten manipular cadenas.

Estructuras de Datos

- ALGOL-60 también permite arreglos semi-dinámicos y el uso de variables para definir tanto el límite inferior como el superior de un arreglo.

integer array flujos [-20:100]

Estructuras de Datos

- En ALGOL-60 los identificadores podían tener cualquier longitud.
- En contraste, los identificadores en FORTRAN estaban limitados a 6 caracteres.
- En este y varios otros aspectos, ALGOL-60 es un buen ejemplo de un lenguaje que sigue el **Principio Cero-Uno-Infinito**.

Sistema de Tipos

- ALGOL-60 tiene un sistema de tipos “fuerte”.
- Esto significa que se checan estrictamente las abstracciones de tipos del lenguaje.
- Es importante tener en cuenta que un sistema de tipos fuerte impide al programador efectuar operaciones sin sentido sobre los datos.

Sistema de Tipos

- Claro que en algunas ocasiones es necesario violar el sistema de tipos (p.ej., en programación de sistemas) para poder efectuar una cierta operación que, de otra manera, sería imposible.

- Sin embargo, este tipo de excepción haría que nuestros programas fueran incompatibles con otras implementaciones (es decir, se sacrifica la portabilidad del lenguaje).

Sistema de Tipos

- Obviamente, las violaciones al sistema de tipos también hace que nuestros programas sean irregulares y difíciles de entender para aquellos que no estén familiarizados con la arquitectura de nuestra computadora.

- De hecho, suele ser el caso que en la mayoría de las situaciones en las que se efectúa una violación al sistema de tipos, existe una manera de efectuar la misma tarea sin tener que recurrir a dicha violación.