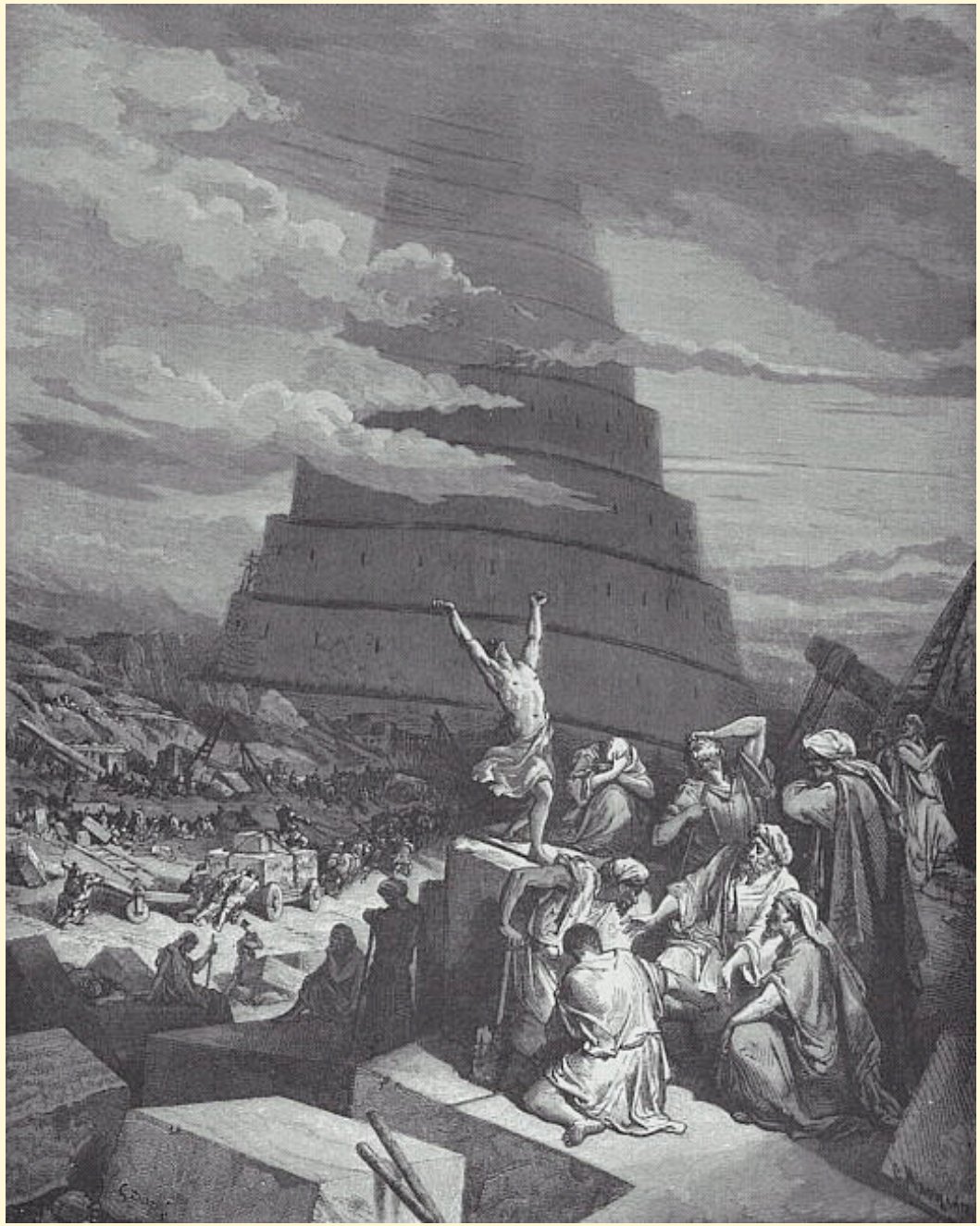
Lenguajes de programación

Abdiel E. Cáceres González
Instituto Tecnológico de Monterrey
Campus Ciudad de México
Verano 2004



Tower of Babel by Gustav Dore 1986

TECNOLÓGICO DE MONTERREY

Principios cubiertos en el capítulo anterior

- Principio de Ortogonalidad (de MacLennan):
  - Funciones independientes deben ser controladas por mecanismos independientes.

Principios cubiertos en el capítulo anterior

- Principio de Seguridad (de MacLennan):
  - Ningún programa que viole la definición del lenguaje, o su propia estructura original, debe escapar detección.

Principios cubiertos en el capítulo anterior

- Principio de Abstracción (de MacLennan):
  - Evite requerir que la misma cosa se tenga que definir más de una vez; factorice el patrón recurrente.

Principios cubiertos en el capítulo anterior

- Principio de Etiquetado (de MacLennan):
  - No requiera que el usuario se tenga que saber la posición absoluta de un elemento en una lista. En vez de eso, asocie etiquetas con cualquier posición que debe ser referenciada más adelante.

Historia de FORTRAN

- Estrictamente hablando, FORTRAN no fue el primer lenguaje de programación que existió (Laning & Zierler, del MIT, ya tenían un compilador de un lenguaje algebraico en 1952), pero sí fue el primero en atraer la atención de una parte importante de la reducida comunidad de usuarios de computadoras de la época.

Historia de FORTRAN

El desarrollo de FORTRAN comenzó en 1955 y el lenguaje se liberó finalmente en abril de 1957, después de 18 años-hombre de trabajo.

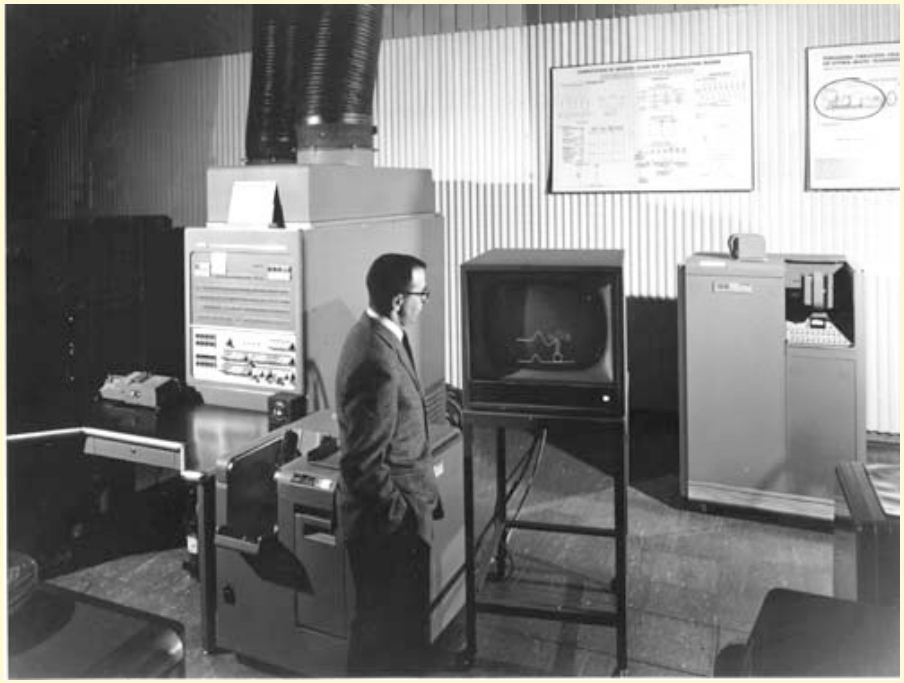
Historia de FORTRAN

FORTRAN (FORmula TRANslating system) fue desarrollado principalmente por John Backus en IBM. Backus recibió el "Turing Award" en 1977.

Historia de FORTRAN

El principal énfasis de FORTRAN fue la eficiencia. Su diseño se basó en un intérprete llamado “Speedcoding”, que fue desarrollado por Backus para la IBM 701.

Historia de FORTRAN



FORTRAN fue desarrollado inicialmente en una IBM 704, aprovechando sus rutinas de punto flotante (proporcionadas en hardware).

Historia de FORTRAN

La especificación preliminar de FORTRAN fue recibida con frialdad y escepticismo. Pocos creían que pudiera diseñarse un compilador que fuese equiparable (en eficiencia) a un humano.

Historia de FORTRAN

John Backus fue fuertemente criticado incluso por luminarias de la época como John von Neumann, quienes creían que el desarrollo de FORTRAN sólo era un desperdicio de dinero (el proyecto estuvo a punto de ser cancelado).

Historia de FORTRAN

El secreto del éxito de FORTRAN se atribuye a su excelente documentación y a las sofisticadas técnicas de optimización que se usó en el diseño de su compilador. De acuerdo a John Backus, las técnicas utilizadas en FORTRAN no fueron superadas sino hasta los 1960s.

Cronología de FORTRAN

- En 1958 se liberó FORTRAN II.
- Otro dialecto, llamado FORTRAN III, fue liberado también en 1958, pero no resultó muy exitoso debido a su enorme dependencia de la IBM 704.
- En 1962 se liberó FORTRAN IV, que ha sido quizás la versión más popular del compilador.

Cronología de FORTRAN

- En 1958, el American National Standards Institute (ANSI) estandarizó FORTRAN (a esta versión se le llama ANS FORTRAN).
- Un nuevo estándar fue liberado en 1977 (se le conoce como FORTRAN 77).
- Las versiones más recientes de FORTRAN (FORTRAN 90 y FORTRAN 2000) difieren mucho del lenguaje original.

Estructura de los programas en FORTRAN

- Los programas en FORTRAN se dividen en 2 partes:
  - 1) Una parte declarativa, la cual describe las áreas de datos, sus longitudes (tipos) y sus valores iniciales. Las instrucciones declarativas existen en la mayor parte de los lenguajes estructurados, aunque se les llama de maneras distintas. En FORTRAN se les denomina "sentencias no ejecutables".

Estructura de los programas en FORTRAN

- 2) Una parte imperativa que contiene los comandos a ser ejecutados en el momento en que corre un programa. A las instrucciones imperativas de FORTRAN se les denomina “sentencias ejecutables”.

Parte declarativa de un programa

- La parte declarativa del programa realiza 3 funciones principales:
  - 1) Asignar un área de memoria de un tamaño especificado. Los tipos de datos disponibles en FORTRAN son: INTEGER, REAL, DOUBLE PRECISION y arreglos. El tipo de las variables numéricas suele considerarse REAL por omisión.

Parte declarativa de un programa

- 2) Asociar un nombre simbólico a un área específica de memoria (a esto se le llama “binding” o declaración de variables).
- 3) Inicializar el contenido de esa área de memoria. En el caso de FORTRAN, las variables no tienen que ser inicializadas. Esto suele ser una fuente común de errores en FORTRAN.

Parte declarativa de un programa

- Ejemplo de una declaración:

```
DIMENSION MIARREGLO(100)
```

- Esto inicializa el arreglo `MIARREGLO` a `100` posiciones de memoria (se presuponen números reales por omisión).

Parte declarativa de un programa

- Ejemplo de inicialización:

```
DATA MIARREGLO, MIVAR/100*0.0,5.0
```

- Esto inicializa las `100` posiciones del arreglo `MIARREGLO` a `0.0`. Adicionalmente, inicializamos la variable `MIVAR` a `5.0`.

Parte imperativa de un programa

- Las sentencias imperativas son de 3 tipos diferentes:
-
  - 1) Sentencias computacionales: Son análogas a las operaciones aritméticas y de movimiento de datos del pseudo-código que vimos anteriormente. La asignación es la sentencia computacional más importante de FORTRAN.

Parte imperativa de un programa

- 2) Sentencias de control de flujo: Comparaciones y ciclos. FORTRAN proporciona instrucciones condicionales (sentencias IF), ciclos (usando DO) y saltos incondicionales (usando GOTO). El lenguaje tiene sentencias muy primitivas para flujo de datos.

Parte imperativa de un programa

- Sentencias de entrada/salida: FORTRAN I tenía 12 sentencias de entrada/salida de un total de 26 del lenguaje. La razón era la enorme variedad de dispositivos de E/S que debía ser capaz de manejar el compilador. Las versiones más recientes de FORTRAN han reducido y/o simplificado la necesidad de usar tantas sentencias de E/S.

Parte imperativa de un programa

- Ejemplo de asignación:

```
RESULTADO = TOTAL/FLOAT(N)
```

- Note que en este caso, FLOAT se usa para convertir N (presumiblemente una variable entera) a un valor real, de manera que el resultado de la operación sea un número de punto flotante.

Parte imperativa de un programa

- ¿Qué otro lenguaje conoce donde deba hacerse una reducción de tipo (casting) similar a ésta?
-
- Es importante recordar que el uso de notación algebraica convencional fue una de las principales contribuciones de FORTRAN.

Notación algebraica

- Un ejemplo de notación algebraica en FORTRAN es el siguiente:

```
RESULTADO = A*B+C-9.0/D**3.4
```

Operadores aritméticos en FORTRAN

Operación	Operador (FORTRAN)
Suma	+
Resta	-
Multiplicación	*
División	/
Exponenciación	**

TECNOLÓGICO DE MONTERREY

Funciones matemáticas

- FORTRAN tiene también una cantidad considerablemente grande de rutinas matemáticas disponibles para el cálculo de logaritmos, funciones trigonométricas, números complejos, etc.

Jerarquías de los operadores

- Los operadores aritméticos también tienen prioridades:
  - 1) Exponenciación
  - 2) Multiplicación y división
  - 3) Suma y resta
- Estas mismas prioridades han sido adoptadas en la mayoría de los lenguajes de programación modernos.

Ciclos con DO

- La sentencia DO es la única estructura de control de alto nivel que proporciona FORTRAN, puesto que les permite a los programadores expresar lo que quieren que se haga y no cómo quieren que se haga.

Ciclos con DO

- La sentencia DO ilustra el principio de Automatización (de MacLennan):
-
  - Deben automatizarse las tareas mecánicas, tediosas o propensas a errores.

Ciclos con DO

- La sentencia DO permite realizar iteraciones definidas (llamadas en inglés “counted loops”):

```
DO 100I=1,N
haz lo que quieras
100
CONTINUE
```

Ciclos con DO

- En el ejemplo anterior, `100` es una etiqueta (sólo se permiten etiquetas numéricas en FORTRAN) que indica lo que está contenido dentro del ciclo. `I` es el contador del ciclo en este caso y toma un valor inicial de `1`, llegando hasta `N`. `CONTINUE` no tiene ningún efecto y sólo se usa para indicar la finalización del ciclo (como la sentencia NEXT en BASIC).

Ciclos con DO

Los ciclos con DO también pueden ser anidados, lo que proporciona mucha flexibilidad.

Ciclos con DO (ejemplo)

```
DO 100 I=1,M
   .
       DO 200 J=1,N
   .
   200
         CONTINUE

   .
   100
   CONTINUE
```

Ciclos con DO

FORTRAN tiene, en general, una estructura “lineal” y no “jerárquica”, dado que la mayoría de sus instrucciones no permiten ningún tipo de anidamiento. Por lo tanto, la sentencia DO resulta ser más bien una excepción que una función más del lenguaje.

Ciclos con DO

Adicionalmente, la seguridad del lenguaje se ve comprometida por el hecho de que FORTRAN permite saltos incondicionales (GOTOs) dentro y fuera de los ciclos DO bajo ciertas circunstancias. Esta es una de las razones por las cuales la sentencia GOTO se volvió obsoleta en la segunda generación de lenguajes de programación.

Ciclos con DO

Finalmente, el ciclo DO está altamente optimizado, lo que lo hace altamente útil y práctico desde la perspectiva de la meta principal del FORTRAN: la eficiencia.

Ciclos con DO

La capacidad de optimización del ciclo DO puede atribuirse parcialmente a su estructura y al hecho de que la variable de control y sus valores iniciales y finales se especifican explícitamente junto con la extensión del ciclo.

Ciclos con DO

Aunque parezca extraño, suele darse el caso de que las estructuras de más alto nivel (como el DO en este caso) resultan más fáciles de optimizar que las de bajo nivel.

Estructura básica de un programa en FORTRAN

- Declarations
-
- Main program
-
- Subprogram 1
- Subprogram 2
- .
- Subprogram n

TECNOLÓGICO DE MONTERREY

Subprogramas

- Los subprogramas son realmente una adición posterior al FORTRAN, ya que no aparecieron sino hasta en el FORTRAN II.
-
- Aunque FORTRAN I proporcionaba bibliotecas con funciones matemáticas y de E/S, no permitía que los programadores definieran sus propios subprogramas.

Subprogramas

FORTRAN II resolvió el asunto de los subprogramas al agregar las declaraciones SUBROUTINE y FUNCTION.

Eficiencia en la compilación

Puesto que una de las metas primordiales de FORTRAN durante su diseño fue la eficiencia, el enfoque tomado para generar un archivo ejecutable estaba altamente optimizado:

Eficiencia en la compilación

1) Compilación: Traducir subprogramas individuales en FORTRAN a código objeto relocalizable (código que usa direcciones relativas en vez de direcciones absolutas de memoria). Estas direcciones relativas se resolverán (o sea, se convertirán a direcciones absolutas) en tiempo de cargado.

Eficiencia en la compilación

La mayor parte de los errores de sintaxis se interceptarán en esta etapa (p.ej., sentencias inexistentes, uso erróneo de variables u operadores, etc.)

Eficiencia en la compilación

2) Ligado (Linking): Incorporar bibliotecas de subprogramas que han sido previamente escritos, depurados y compilados. Los programas en FORTRAN contienen referencias externas que deben resolverse durante esta etapa.

Eficiencia en la compilación

Cualquier referencia errónea a funciones externas se resuelve en esta etapa.

3) Cargado: Es el proceso en el cual se coloca un programa en la memoria de la computadora. Las direcciones relativas (o direcciones relocalizables) se traducen a direcciones absolutas de memoria.

Eficiencia en la compilación

4) Ejecución: El control de la computadora se pasa al programa que se encuentra en memoria. Puesto que el programa se ejecuta directamente y no a través de un intérprete, tiene el potencial de correr significativamente más rápido.

Pasos de la compilación

- La compilación de un programa en FORTRAN se realiza normalmente en 3 pasos:
  - 1) Análisis sintáctico: El compilador debe clasificar las sentencias y constructores de FORTRAN y extraer sus partes.

Pasos de la compilación

- 2) Optimización: Los compiladores originales de FORTRAN incluían un optimizador muy sofisticado cuya meta era producir código tan bueno como el de un buen programador humano. Hasta la fecha, la mayor parte de los compiladores de FORTRAN que existen realizan algún tipo de optimización.

Pasos de la compilación

- 3) Síntesis de código: Juntar las porciones de las instrucciones en código objeto en un formato relocalizable.

Estructuras de control

En sus orígenes, FORTRAN fue un lenguaje muy dependiente de la máquina en la que se desarrolló (la IBM 704). Esa es la razón por la que algunas de sus instrucciones lucen un tanto extrañas, redudantes y hasta tontas.

Estructuras de control

Algunas de estas sentencias fueron removidas en versiones posteriores del lenguaje, pero la mayor parte de ellas han permanecido ahí durante más de 40 años.

Estructuras de control

- La sentencia IF aritmética:
  - `IF (e) n1, n2, n3`
- evalúa la expresión "`e`" y salta a `n1`,`n2` o `n3`, dependiendo de si el resultado de la evaluación es `<0`, `=0` o `>0`.

Estructuras de control

La sentencia IF aritmética corresponde de manera exacta a una instrucción en lenguaje máquina de la IBM 704 y es un buen ejemplo de cómo NO diseñar una sentencia IF.

Estructuras de control

Llevar el control de 3 etiquetas es algo difícil y el hecho de que en muchos casos dos de estas etiquetas pueden fusionarse (por ser idénticas) hacen de ésta una instrucción muy primitiva.

Estructuras de control

La sentencia IF lógica:

```
   IF (X .EQ. 5) M=A+3
.LT., .GT., .GE., .LE., .AND., .OR.,
.NOT., .EQV., .NEQV.
```

Esta sentencia reemplaza a la primitiva sentencia IF aritmética y la vuelve más fácil de usar y más regular.

Estructuras de control

Sentencias Condicionales

GOTO era el caballito de batalla del control de flujo en FORTRAN, ya que no existía estructura IF-THEN-ELSE como en la mayoría de los lenguajes de programación modernos.

Estructuras de control

```
IF (se cumple) GOTO 100

. . . caso para condición falsa . . .

GOTO 200
100
. . . caso para condición cierta . . .
200
. . .
```

Estructuras de control

El uso de GOTO puede llevarnos (cuando no se aplica adecuadamente) al desarrollo del denominado “código spaghetti”, lo cual genera programas difíciles de leer y modificar.

Estructuras de control

Sentencias para casos:

Como no había CASE como en los lenguajes de programación modernos, se utilizaba en su lugar el denominado “GOTO calculado” (computed GOTO).

Estructuras de control

```
GOTO (10, 20, 30, 40), I
10
. . . manejar caso 1 . . .

GOTO 100
20
. . . manejar caso 2 . . .

GOTO 100
30
. . . manejar caso 3 . . .
GOTO 100
40
. . . manejar caso 4 . . .
100
. . .
```