# FORTRAN: Historia

FORmula TRANSlator.

IBM: 1953-1956. Fue el primer lenguaje de programacion de alto nivel, alternativa a ensamblador. Versiones I, II y III.

Estandarización

- Fortran IV
- Fortran 66
- Fortran 77
- Fortran 90, 95, 2003, 2008, 2015.

Fortran Working Group (WG5) http://www.nag.co.uk/sc22wg5/

# FORTRAN:Fortalezas

Orientado a cálculo numérico. Características:

- Sintaxis estricta, con poca variabilidad → aprendizaje rápido.
- Tipo COMPLEX.
- Estandarizado por ANSI e ISO. Portabilidad. Interoperable con C.
- Disponibilidad de compiladores optimizados para cada procesador.
- Amplia disponibilidad de bibliotecas, incluyendo procesamiento paralelo.

# FORTRAN: desventajas

- Lenguaje arcaico, aunque modernizado a partir de Fortran 90.
- Muchas líneas de código y chequeos del compilador.
- No es de uso general, o al menos no es cómodo.

Usar solamente para cálculos realmente costosos que vayan a durar mucho más que el tiempo de programación. Para cálculos no intensos, use Mathematica, Matlab, python, etc.

# Humor FORTRAN

## Programadores Matando Un Dragon

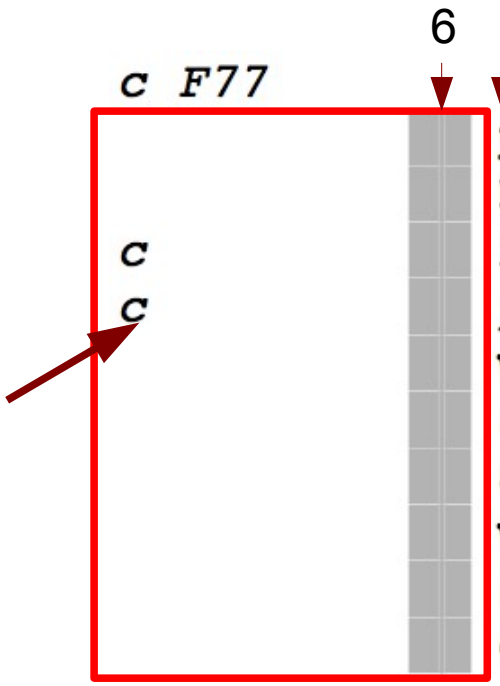
Ensamblador – Cree que está haciendo lo más correcto y eficiente… pero pone un A en lugar de un D y mata a la princesa que estaba secuestrada por el dragón.

Fortran – Llega y desarrolla una solución con 45 mil líneas de código, mata al dragón, va al encuentro de la princesa… pero ella le llama tirillas y se va corriendo detrás del programador de java que era elegante y además es rico.

Java – Llega, encuentra al dragón, desarrolla un framework para aniquilación de dragones en múltiples capas, escribe varios artículos sobre el framework… pero no mata al dragón.

# F77 vs F90

**No se distingue mayúsculas de minúsculas**




```
program circulo
Real r, area
Este programa lee un número real r
y muestra el área del círculo con radio r.
write (*,*) 'Escribe el radio r:'
read  (*,*) r
area = 3.14159*r*r
write (*,*) 'Area = ', area
stop
end
```

Permitido en F90
Formato fijo
Extensión .f

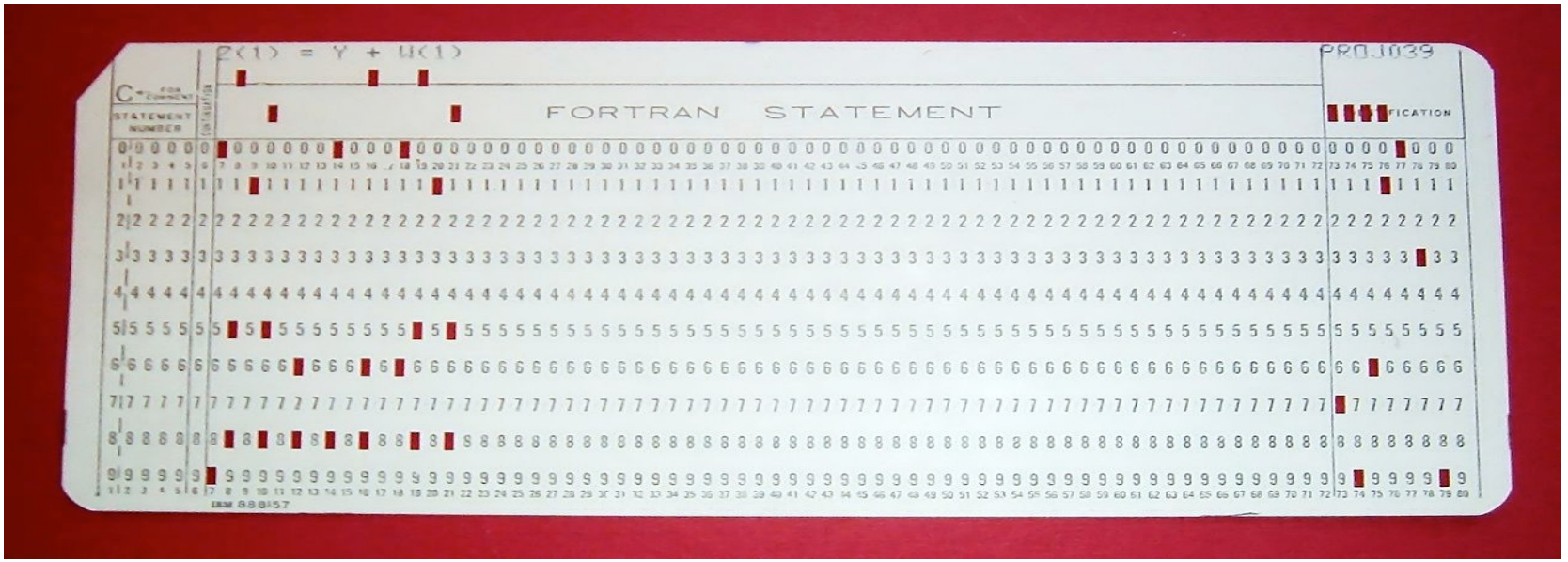
```
! F90 y siguientes
program circulo
    Real :: r, area
    ! Este programa lee un número real r y muestra
    ! el área del círculo con radio r.
    write (*,*) 'Escribe el radio r:'
    read  (*,*) r
    area = 3.14159*r*r
    write (*,*) 'Area = ', area
stop
end
```

Formato libre
Extensión .f90

# Formato fijo (F77)

- Col. 1 : Blanco (espacio), o un caracter "c" o "*" para comentarios
  Col. 2-5 : Etiqueta de sentencia (opcional)
- Col. 6 : Continuación de una línea previa: “&”, “+”, “.”, o un número.
- Col. 7-72 : Sentencias
- Col. 73-80: Número de secuencia (opcional)

  esto viene de la época de las tarjetas perforadas, lineas de 80 char



# Ejercicio: escribir programa

**Nombre del archivo test1.f**

```
      6   7
c F77
      program circulo
      Real r, area
c     Este programa lee un número real r
c     y muestra el área del círculo con radio r.
      write (*,*) 'Escribe el radio r:'
      read  (*,*) r
      area = 3.14159*r*r
      write (*,*) 'Area = ', area
      stop
      end
```

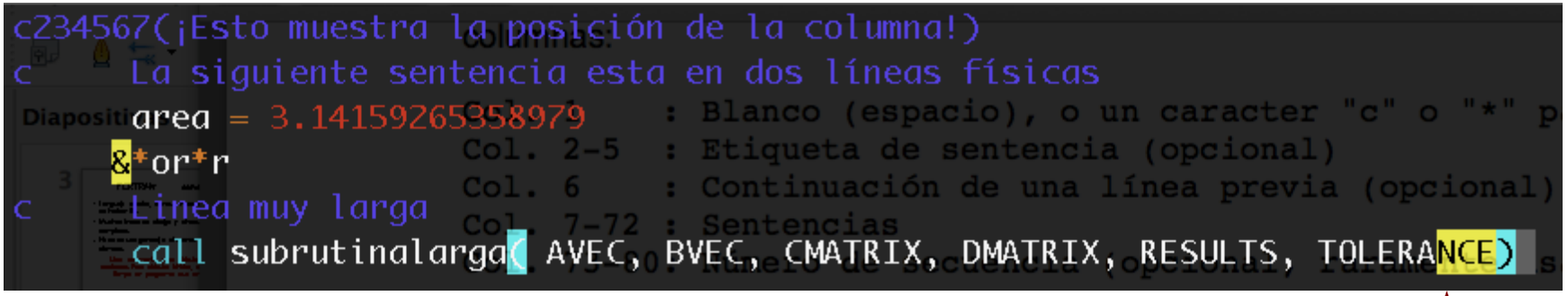
**Nombre del archivo test2.f90**

```
! F90 y siguientes
program circulo
    Real :: r, area
    ! Este programa lee un número real r y muestra
    ! el área del círculo con radio r.
    write (*,*) 'Escribe el radio r:'
    read  (*,*) r
    area = 3.14159*r*r
    write (*,*) 'Area = ', area
stop
end
```

Formato libre
Extensión .f90

# Ejercicio: editar con vim

- Dar vim test1.f
- :syntax on
- Lo anterior genera una sensibilidad a la estructura del archivo. Las letras en las columnas 1-6 y >72 son coloreadas.



# Compilación

- El archivo en del programa (test1.f o test2.f90) se llama código fuente.
- Compilemos test1.f con el comando

  gfortran test1.f
- Si está instalado gfortran, debe aparecer un archivo a.out. Listar con “ls -l”.
- Ejecutar el comando

  ./a.out

# Compilación en detalle

- Especificando el nombre del ejecutable

  gfortran -o test1.x test1.f

  genera el ejecutable test1.x
- Compilación (pura) del archivo objeto

  gfortran -c test1.f

  ls -l
- Debe aparecer el archivo test1.o (archivo objeto). Es un archivo binario. El siguiente paso es “linkear” (enlazar).
- gfortran -o test1.x test1.o

  Los programas grandes se seccionan en varios (muchos) archivos fuente. Se compilan por separado y al final se enlazan.

# Estructura del programa

- Nombre del programa
- Declaraciones de variables
- Cuerpo del programa (instrucciones)
- END
- Subprogramas (subrutinas y funciones)
-

# Estructura de una línea en F90 (formato libre)

- Hasta 132 caracteres. (Hasta 72 en F77)
- Espacios en blanco al principio se ignoran.
- Un signo & al final de una linea indica que el comando continua en la linea siguiente. (en F77 es un caracter cualquiera en la columna 6 de la linea de continuación).
- Todo lo que siga de un signo ! se considera un comentario y es ignorado por el compilador.
- Es posible poner varios comandos en una linea separándolos con punto y coma.
- Importante: FORTRAN no distingue entre mayúsculas y minúculas en un programa, también ignora más de un espacio en blanco y líneas en blanco.

# 1) Nombre del programa

- Ejemplo

  program hola

  resto del programa

  …..

  end program hola

LOS PROGRAMAS ANTIGUOS SE ESCRIBIAN CON MAYUSCULAS. ACTUALMENTE ES OPCIONAL Y FORTRAN NO DISTINGUE MAYUSCULA Y MINUSCULAS.

# 2) Declaración de variables

- Sintaxis:

  Tipo[, atributos] :: var1, var2, …
- Ejemplos:

  real :: presion, temperatura

  real, allocatable :: matriz1(:,:),energias(:)

  integer :: npar, ntotal

  logical :: validez

# Declaración implícita

Si no se declara el tipo de una variable, el compilador asume el tipo en dependencia de la letra inicial

- i,j,k,l,m,n : integer
- Otra letra: real.
- Se puede desactivar con la sentencia

  implicit none

  Es recomendado, evita que por errores de digitación se autodeclaren variables no previstas.

# Tipos de datos

- **Real** : Valores reales guardados en 4 bytes y con 8 cifras significativas. Se indican con punto decimal, y de ser necesario el exponente de la potencia de 10 después de una E: 1., -3.1416, 6.25E-10, etc.

  CPU Intel: También se declaran real(4) o real(kind=4).

- **Double precision** : Valores reales de doble precisión guardados en 8 bytes y con 16 cifras significativas. Se indican con punto decimal y el exponente de la potencia de 10 después de una D: 1.D0, -3.1416D0, 6.25D-10, etc.

  CPU Intel: También se declaran como real(8) o real(kind=8).

# Tipos de datos

- **Real(16)** : Valores reales de cuádruple precisión guardados en 16 bytes y con 32 cifras significativas. Se indican con punto decimal y el exponente de la potencia de 10 después de una Q: 1.Q0, -3.1416Q0, 6.25Q-10.
- **Complex**: Dos valores reales formando un par y que en operaciones matemáticas se tratan como la parte real e imaginaria de un número complejo: (1.,-2.), (1.0E0,-2.0E0).
- **Double complex** o **complex(kind=8)**

  Avanzado: En algunas arquitecturas (e.g., Cray vector supercomputers), kind toma valores distintos de 4 y 8. F90 permite escribir programas totalmente portables mediante este mecanismo

  INTEGER, PARAMETER :: SP=KIND(1.0),DP = KIND(1.0D0)
  REAL(DP) :: VAR1, VAR2, …

# Avanzado: definiendo la precisión de forma independiente de la arquitectura del CPU

```
program real_kinds
integer,parameter :: p6 = selected_real_kind(6)
integer,parameter :: p10r100 = selected_real_kind(10,100)

real(kind=p6) :: x
real(kind=p10r100) :: y


  print *, precision(x), range(x)
  print *, precision(y), range(y)
end program real_kinds
```

# Avanzado: precisión independiente de la arquitectura

```
!-----------------------------------------------------------------------!
   MODULE kinds
!-----------------------------------------------------------------------!
      IMPLICIT NONE
      SAVE
! ... kind definitions
      INTEGER, PARAMETER :: DP = selected_real_kind(14,200)
      INTEGER, PARAMETER :: sgl = selected_real_kind(6,30)
      INTEGER, PARAMETER :: i4b = selected_int_kind(9)
      PRIVATE
      PUBLIC :: i4b, sgl, DP, print_kind_info
```

- 
- En otra unidad del programa se define
  use kinds, only : DP
  REAL(DP), PARAMETER :: eps6   = 1.0E-6_DP

# Tipos de datos

- Integer : Valores enteros guardados en 4 bytes y con hasta 8 cifras.

  Variantes: integer(kind=2), integer(kind=4), integer(kind=8).
- Logical  : valores .true. o .false. .
- Character:  Variables que corresponden a cadenas de caracteres. Al declarar una variable de este tipo se debe especificar cuantos caracteres puede tener. Estas variables deben estar contenidas en comillas, p. ej.: ’hola’

# Definición de constantes

real, parameter :: pi = 3.14159  ! formato F90

C  La misma definicion en F77 es

```
      real pi
      parameter (pi=3.14159)
```

```
Constantes complejas, ejemplo para F90
 complex(kind=8), parameter :: uno=(1.d0,0.d0) &
                               im1=(0.d0,1.d0)
```

Continuación de línea

# Ejercicio:

- Hacer programa que defina y que imprima:

  una constante real de doble precision

  una variable compleja real de simple precision

  una constante integer(kind=4)

  una variable logical.

  Una variable char con el valor 'FORTRAN'

# Arreglos

- Un arreglo es un conjunto de datos almacenados en una variable y enumerable con índices enteros.
- Ejemplos

  real, dimension(20) :: d

  real :: d(20)
- Ambas formas anteriores son equivalentes, pero tienen distintas utilidades. Ver proxima página

# Arreglos

```
real, dimension(20) :: d, e, f, g ! todos con la
                                  ! misma dimensión

real :: d(20), e(40), f(60), g(60)
```

- El  índice comienza en 1 por defecto, pero se puede cambiar en F90 y superior

```
real :: b(0:19), c(-162:237)
```

# Arreglos bidimensionales

Real :: A(3,5), B(0:2,1:5), C(-1:1,-2:2)  ! todos de dimension 3x5

real, dimension (3, 5) :: A

Es útil pensar que el primer índice describe la fila y el segundo la columna

A(1,1)  A(1,2)  A(1,3)  A(1,4)  A(1,5)

A(2,1)  A(2,2)  A(2,3)  A(2,4)  A(2,5)

A(3,1)  A(3,2)  A(3,3)  A(3,4)  A(3,5)

Avanzado: La memoria se barre por columnas, es decir, en la memoria se almacenan los datos en el orden

A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2), A(1,3), …

# Arreglos dinámicos

- Una de las limitaciones mas fastidiosas de F77 era la imposibilidad de definir la dimensión de los arreglos durante la ejecución. En F90 es fácil.

```
integer(kind=4), allocatable :: numeros(:)  ! ó

integer(4), dimension(:), allocatable :: numeros

! durante la ejecución

ndim=100

allocate(numeros(ndim))

numeros = …..

deallocate(numeros)
```

# Operaciones con arreglos

- Se pueden definir operaciones para arreglos completos. Ejemplo:

```
real(8), dimension (2,2) :: a,b,c

….

a=4.0d0 ! Hace todos los elementos = 4.0d0

b=0.0d0  ! hace todos los elementos = 0.0d0

b(2,2)=5.0d0 ! Cambia el elemento 2,2 a 5.0d0.

c=a+b  ! suma las dos matrices
```