



Diseño Computarizado 15274

Programación: Fortran

Claudio García Herrera

Cristian Catrilef Pozo

Sergio González Pesce

Universidad de Santiago de Chile (USACH)

Laboratorio de Biomecánica y Biomateriales

Facultad de Ingeniería - Departamento de Ingeniería Mecánica

Correo: claudio.garcia@usach.cl

cristian.catrilef@usach.cl

Índice

- 1 Introducción
- 2 Estructura
- 3 Sentencias y variables
- 4 Otros
- 5 Ejemplos y Ejercicios



Introduccion

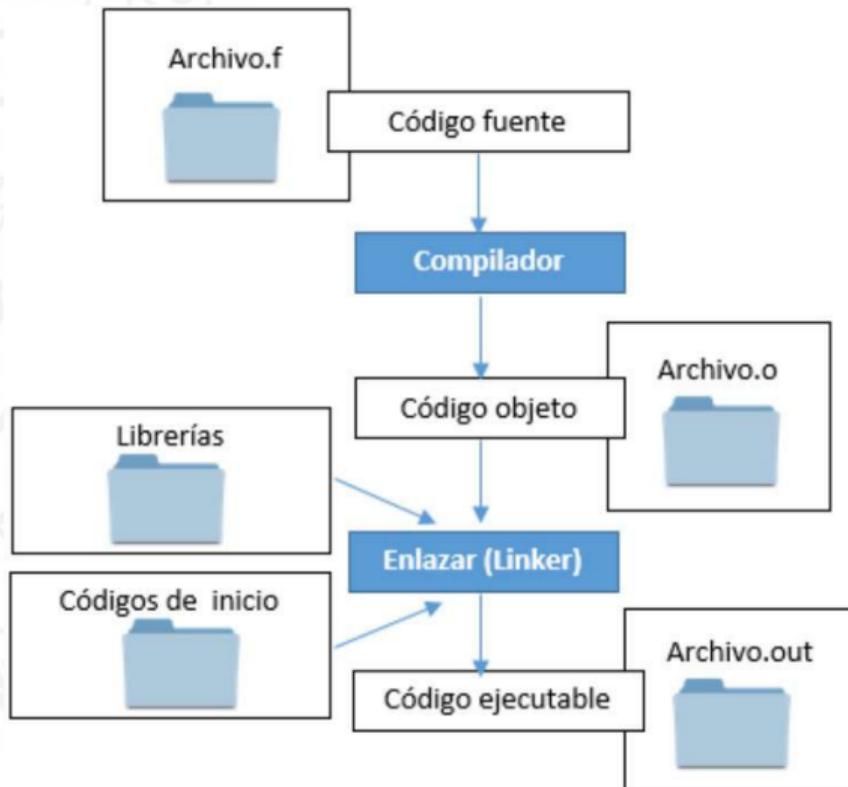


1. Fortran es un lenguaje de programación dominante usado en muchas aplicaciones de ingeniería y matemáticas, por lo que es importante que se tengan bases para poder leer y modificar un código de Fortran.
2. Un programa de Fortran es una secuencia de líneas de texto. El texto debe seguir una determinada sintaxis para ser un programa válido de Fortran.

Ejemplo de un programa en Fortran 77. Programa para calcular el área de un círculo.

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

Compilación





Estructura del programa

```
Program Nombre
      Inclusion de librerias y modulos externas
      Declaraciones de varianles y tipos
      Instrucciones
stop
end
```

Fortran 77 no es un lenguaje de formato libre. Tiene un conjunto de reglas estrictas de cómo el código fuente debe ser escrito en el editor (vi, vim, gedit, etc.). Las reglas más importantes son las reglas para la posición en columnas:

```
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 (opcional)
Col. 7-72   : Sentencias
```

Sentencias y variables



Sentencias de entrada y salida

- ▶ **Read** permite al ordenador leer la información que queremos suministrar al programa.
- ▶ **Write** permite al ordenador escribir la información que ha procesado

```
read (num_unidad , num_formato) listado_de_variables  
write(num_unidad , num_formato) listado_de_variables
```

```
read (*,*) listado_de_variables  
write(*,*) listado_de_variables
```

Sentencia y variables



Sentencias aritméticas

Son usadas para operar con dos variables. Ejemplos:

- ▶ (+ o -) Suma o diferencia.
- ▶ (* o /) Multiplicación o división
- ▶ (**) Potencia.

Variables

(=) $C = A + B$ Coloca el resultado de la operación que se realiza a la derecha de la variable indicada a la izquierda. Suma los números contenidos en las variables (A) y (B) y coloca el resultado en la variable (C)

Sentencia y variables



Variables

- ▶ Cada variable debe ser definida con una declaración. Esto indica el tipo de la variable.
- ▶ La lista de variables consisten en nombres de variables separadas por comas. Cada variable deberá ser declarada exactamente una vez.
- ▶ Si una variable no esta declarada, F77 usa un conjunto implícito de reglas para establecer el tipo. Antes esto se recomienda utilizar el comando 'implicit none' antes de declarar las variables.
- ▶ La probabilidad de errores en el programa crece exponencialmente si no se declaran las variables explícitamente.

<code>Integer</code>	Lista de variables
<code>Real</code>	Lista de variables
<code>Doubleprecision</code>	Lista de variables
<code>Complex</code>	Lista de variables
<code>Logical</code>	Lista de variables

Sentencias y variables

La sentencia lógica 'if'

- ▶ Si la expresión lógica es verdadera, ejecuta el bloque de sentencias **1** y si es falsa el bloque de sentencias **2**.
- ▶ Cuando la decisión a tomar radica entre ejecutar un bloque de sentencias o no, no es necesaria la sentencia ELSE.

```
if (expresion logica) then
    sentencia 1
else
    sentencia 2
end if
```

Expresion lógicas. Ejemplo, si **A** es menor que **B** (**A.LT.B**)

.LT.	<
.LE.	<=
.GT.	>
.GE.	>=
.EQ.	=
.NE.	/=



Sentencias y variables

La sentencia lógica 'do'

- ▶ El trabajo con vectores, matrices, sumatorias, requiere la confección de algoritmos en los que una serie de operaciones se repiten sucesivamente mientras una variable se modifica en una cantidad constante.
- ▶ El bloque de sentencias es ejecutado sucesivamente desde que la variable toma el valor $n1$ hasta el valor $n2$, incrementándose a cada paso (step) en $n3$. Si $n3$ es 1 no es necesario escribirlo.
- ▶ **La variable del ciclo do nunca deberá ser modificada por otras sentencias dentro del ciclo, ya que puede generar errores de lógica.**

```
do var=n1 ,n2 ,n3
    sentencias 1
end do
```

Sentencias y variables

- ▶ Ejemplo de uso del ciclo do:

```
program prueba_do
implicit none
integer i
real suma

suma=0.0

do i=1,10
    suma= i + suma
    write(*,*) 'i=', i
    write(*,*) 'suma=', suma
end do

write(*,*) 'ultimo valor guardado', i, suma

end program
```

Function



Es un programa escrito aparte del principal, con la siguiente estructura.

```
function nombre(var1, var2, ...)  
    sentencias  
return  
end function
```

- ▶ Toman un conjunto de variables de entrada (parámetros) y regresan un valor de algún tipo.
- ▶ Las funciones tienen un tipo. El tipo debe coincidir con el tipo de la variable que recibirá el valor.
- ▶ El valor que devolverá la función, deberá ser asignado en una variable que tenga el mismo nombre que la función. Ejemplo, si el nombre es '*vel(a, b)*', se usa dentro del programa como una variable de nombre '*vel(c, d)*'

Subroutine



Es un programa escrito aparte del principal, con la intención de entregar dos o más valores, o bien, leer o escribir datos.

```
subroutine nombre(var1,var2,...)
    sentencias
return
end subroutine
```

- ▶ Las subrutinas no tienen tipo y por consecuencia no pueden hacerse asignación al momento de llamar al procedimiento.
- ▶ El nombre de la subrutina se utiliza para identificarla en el programa. Cuando se precise utilizarla se usa la sentencia **call**, de la forma '**call** nombre(var1,var2,...)'

Escritura y Lectura



Cuando la lectura o escritura no tienen lugar en la pantalla se utilizan los ficheros. El programa interpreta una serie de unidades lógicas, cada una de las cuales identificada con un número unidad sobre las que se escribe o de las cuales lee.

Para utilizar un fichero es necesario abrirlo previamente, darle un nombre e identificarlo con un número. Se utiliza la sentencia **open**

```
open(unit=u, file='nombre', status='status')
```

```
open(unit=23, file='datos.dat', status='replace')
```

Status puede ser 'new' si el archivo se va a crear, 'old' si ya existe o 'replace' si se va a sobrescribir en el.

Ejecución del programa

- ▶ Abrir una terminal e ingresar a la carpeta en la que está el programa.
- ▶ Escribir en la terminal 'gfortran nombreachivo'. Se creará un archivo ejecutable con el nombre 'a.out'. Es posible cambiar el nombre a éste escribiendo 'gfortran nombreachivo -o nombreejecutable'.
- ▶ Para ejecutar se escribe './a.out' o bien './nombreejecutable'
- ▶ Hecho esto se realizan todas las ordenes escritas en el programa y es posible visualizarlas en la pantalla o en los ficheros según corresponda.

```
Terminal
Archivo Editar Ver Terminal Ayuda
/home/clausiux %
/home/clausiux % cd Escritorio/Iniciofortran/
/home/clausiux/Escritorio/Iniciofortran % gfortran nombreachivo.f
/home/clausiux/Escritorio/Iniciofortran % ./a.out
hola mundo
/home/clausiux/Escritorio/Iniciofortran % gfortran nombreachivo.f -o nombreejecutable
/home/clausiux/Escritorio/Iniciofortran % ./nombreejecutable
hola mundo
/home/clausiux/Escritorio/Iniciofortran % ls
a.out nombreachivo.f nombreejecutable
/home/clausiux/Escritorio/Iniciofortran %
```



Contador simple

Programa

```
program simple_do
  implicit none
  integer :: counter, countMax=5
  do counter=1,countMax
    write(*,*) "Counter is", counter, "and counting."
  enddo
end program simple_do
```

Salida

```
Counter is    1 and counting.
Counter is    2 and counting.
Counter is    3 and counting.
Counter is    4 and counting.
Counter is    5 and counting.
```



Contador condicional

Entrada

```
program simple_do
  implicit none
  integer :: counter, countMax=5

  do counter=1, countMax
    if(counter < countMax) then
      write(*,*) "Counter is", counter, "and counting."
    else
      write(*,*) "Counter is", counter, "and not counting anymore."
    endif
  enddo
end program simple_do
```

Salida

```
Counter is      1 and counting.
Counter is      2 and counting.
Counter is      3 and counting.
Counter is      4 and counting.
Counter is      5 and not counting anymore.
```

Multiplicación de matrices

Programa

```
program matmultiply
  implicit none
  integer, parameter :: N=5 ! Las matrices seran N x N
  integer, dimension(N,N) :: A,B,C
  integer :: i,j,k
  ! Poner algunos valores en matrices
  do i=1,N
    do j=1,N
      A(i,j) = i; B(i,j) = j; C(i,j) = 0
    enddo
  enddo
  ! Calcular matriz resultante
  do i=1,N
    do j=1,N
      do k=1,N
        C(i,j) = C(i,j)+A(i,k)*B(k,j)
      enddo
    enddo
  enddo
```

Continúa



Multiplicación de matrices

Programa, última parte

```
write(*,*) "Result:"
call printSqrMatrix(C,N)
C = MATMUL(A,B) ! Calcular utilizando intrinseca

contains

subroutine printSqrMatrix(mat,matSize)
  implicit none
  integer, dimension(:,,:), intent(IN) :: mat
  integer, intent(IN) :: matSize
  integer :: p
  ! Print the result nicely
  do p=1,matSize
    write(*,*) mat(p,:)
  enddo
end subroutine printSqrMatrix
end program matmultiply
```



Referencias

1. S.J Chapman, Fortran 90/95 for Scientists and Engineers, 1998
2. M. Metcalf, J. Reid, Fortran 90/95 explained, 1999
3. Lawrence, Norman, Compaq Visual Fortran: A guide to creating windows applications, 2002



Diseño Computarizado 15274

Programación: Fortran

Claudio García Herrera

Cristian Catrilef Pozo

Sergio González Pesce

Universidad de Santiago de Chile (USACH)

Laboratorio de Biomecánica y Biomateriales

Facultad de Ingeniería - Departamento de Ingeniería Mecánica

Correo: claudio.garcia@usach.cl

cristian.catrilef@usach.cl