Fortran

Temas

- Estructura de un Programa en Fortran 90
- Compilar y Ejecutar un Programa Fortran 90
- Palabras Reservadas
- Constantes y Variables
  - Tipos de Datos
  - Declaración de Constantes
  - Declaración de Variables
  - Conversión entre Tipos
- Operaciones Básicas con Fortran 90
- Arreglos
  - Declaración
  - Vectores y Matrices
  - Acceso a los Elementos (Vectores y Matrices)
  - Operaciones con Matrices

Temas

- Arreglos Dinámicos
  - Declaración
  - Asignar Memoria
  - Acceso a los Elementos
  - Liberar Memoria
- Funciones Intrínsecas
- Loops
  - Do
  - Do While
- IF
- Operadores Lógicos
- Entrada y Salida de Datos
  - Obtener y Mostrar Datos
  - Manejo de Archivos
    - Crear un Archivo
    - Lectura y Escritura de un Archivo
    - Cierre de un Archivo

Temas

- Subprogramas
  - Funciones
  - Subrutinas
  - Módulos

Estructura de un Programa en Fortran 90

- Comenzar con la palabra reservada **program**
- Seguido del **Nombre del Programa**
- Declaración de Variables a utilizar
- Cuerpo del Programa
- Terminar con las Palabras reservadas **end program**
- Subprogramas

Elementos de comandos en Fortran 90 a Considerar

- Los espacios en blanco al principio de un comando son ignorados por el compilador
- Todo lo que le siga de un signo **!** se considera comentario y también es ignorado por el compilador
- **IMPORTANTE**: Fortran no distingue entre **MAYÚSCULAS** y **minúsculas**.

Ejemplo

Inicio del Progama

Declaración de Variables

Cuerpo del Programa

Fin del Programa

Subprogramas

```
1  program Funciones
2    !implicit none
3    real radio, area
4
5    print *, 'Dame el Valor del Radio del Círculo: '
6    read(*,*) radio
7
8    area = areaCirculo(radio)
9
10   print*
11   print*, 'El área del Círculo es ', area
12   print *
13 end program Funciones
14
15
16 !Función para Calcular el Área de un Círculo
17 real function areaCirculo(unRadio)
18   implicit none
19
20   real pi
21   real unRadio, unArea
22
23   pi = acos(-1.0)
24   unArea = pi * unRadio**2
25   return
26 end function areaCirculo
27
```

Compilar y Ejecutar un Programa Fortran 90

Compilar

- Para poder compilar un programa desde la Terminal se hace lo siguiente:
- Ir al Directorio donde tenemos el Código Fuente de Fortan 90.
- Una vez en el Directorio, escribir:

  gfortran nombrePrograma.f90 [*-o unNombre*]

  Si el compilador no marca ningún error, proseguir con la Ejecución del Programa, en caso contrario corregir los Errores.

Compilar y Ejecución de un Programa Fortran 90

Ejecución

- Si no se escribió el comando -o
  - Escribir en la Terminal **./a.out** para ejecutarlo
- Si se escribió el comando *-o unNombre*
  - *Escribir en la Terminal **./unNombre** para ejecutarlo*

Palabras Reservadas

ALLOCATE, ALLOCATABLE, CALL, CASE, CHARACTER, COMPLEX, CONTAINS, CYCLE, DEALLOCATE, DEFAULT, DIMENSION, DO, END, ELSE, ELSEWHERE, EXIT, EXTERNAL, FUNCTION, IF, IMPLICIT, IN, INOUT, INTEGER, INTENT, INTERFACE, INTRINSIC, KIND, LOGICAL, MODULE, NONE, ONLY, OPEN, OUT, PARAMETER, POINTER, PRINT, PROGRAM, READ, REAL, RECURSIVE, RESULT, RETURN, SAVE, SELECT, SIZE, STAT, STOP, SUBROUTINE, TARGET, THEN, TYPE, UNIT, USE, WHERE, WRITE

Constantes y Variables

Tipos de Datos

- Integer
- Logical
- Real
- Character
- Complex

Constantes y Variables

Declaración de Constantes

tipoDato, parameter :: identificador = unValor

Ejemplo:

real, parameter :: pi = acos(-1.0)

integer, parameter :: TAM = 50

Constantes y Variables

Declaración de Variables

Sintáxis

- TipoDeDato identificador

Ejemplos

- Integer edad
- Character(30) nombre
- Real aux
- Logical band

Constantes y Variables

Conversión entre Tipos

Muchas veces se comete el error de usar variables enteras con operaciones reales, para evitar valores erróneos se usa la conversión de Tipos. Los cuales son los siguientes:

- Int() Convierte un real a integer
- Nint() Redondea un real al entero más próximo
- Real() Convierte un integer a real
- Dble() Convierte un integer a double precision

Operaciones Básicas con Fortran 90

Operaciones Aritméticas

- Operador Operación Resultado
- Suma 2.0 + 5.0 7.0
- Resta 3.0 – 5.0 -2.0
- Multiplicación 5.0 * 5.0 25.0
- División 2.0/3.0 0.66666667
- Exponenciación 2.0**3.0 8.0

Operaciones Básicas con Fortran 90

Jerarquía de operadores en Fortran 90

- Términos entre paréntesis
- Exponenciación
- Multiplicación y División
- Sumas y Restas

Nota: A Fortran le toma mucho tiempo calcular las divisiones que las multiplicaciones. Así, que siempre que sea posible se deben evitar.

Arreglos

Que pasa cuando se quiere utilizar 100, 1000, o más variables del mismo tipo, pensaríamos hacer algo así:

real a1, a2, a3, ……., a100

integer b1, b2,b3, ……., b1000

Y esta correctamente válido pero eso nos llevaría mucho tiempo!!!

SOLUCIÓN

ARREGLOS

Arreglos

Declaración (Vectores)

- TipoDato :: identificador(Tamaño)

Ejemplos

integer :: vector(10)

real :: fx(100)

Declaración (Matrices)

- TipoDato :: identificador(Tamaño1, Tamaño2)

Ejemplos

integer :: MatrizA(10, 10)

real :: MatrizB(100, 100)

Arreglos

Acceso a los Elementos (Vectores)

identificador(índice)

Ejemplos

vector(1) = 22

fx(22) = 34.55

Acceso a los Elementos (Matrices)

identificador(índice1, índice2)

Ejemplos

MatrizA(10, 10) = 100

MatrizB(100, 100) = 234.666

Operaciones con Arreglos

Fortran nos permite hacer operaciones Aritméticas (suma, resta, multiplicación(elemento por elemento), división, exponenciación) directamente con los arreglos, **siempre y cuando éstos tengan la misma dimensión**.

Ejemplo:

Sean A, B, C Matrices de 5x5 las siguientes operaciones con válidas.

C = A + B

C = A – B

C = A * B

C = A / B

C = A ** B

Arreglos Dinámicos

En algunas ocasiones es útil declarar un arreglo sin asignar un número fijo de componentes, sino después (debido a que se lean datos del exterior). Para hacer esto se le hace la **Asignación Dinámica de Memoria**.

Arreglos Dinámicos

Declaración (Vectores)

tipoDeDato, allocatable, dimension(:) :: identificador

Ejemplo:

real, allocatable, dimension(:) :: vector

Declaración (Matrices)

tipoDeDato, allocatable, dimension(:,:) :: identificador

Ejemplo:

real, allocatable, dimension(:,:) :: Matriz

Arreglos Dinámicos

Asignar Memoria

Antes de realizar cualquier operación con arreglos de este tipo, se le debe asignar memoria de la siguiente forma:

Vectores

allocate(identificador(tamaño))

Ejemplo:

allocate(vector(100))

Matrices

allocate(identificador(tamaño1, tamaño2))

Ejemplo:

allocate(Matriz(100, 100))

Arreglos Dinámicos

Acceso a los Elementos

Acceso a los Elementos (Vectores)

identificador(índice)

Ejemplo:

vector(1) = 22.54

Acceso a los Elementos (Matrices)

identificador(índice1, índice2)

Ejemplo:

Matriz(100, 100) = 234.666

Arreglos Dinámicos

Liberar Memoria

Es importante que al terminar los cálculos libere la memoria Asignada, esto se hace con la siguiente función:

deallocate(identificador)

Ejemplo:

deallocate(vector)

deallocate(Matriz)

Funciones Intrínsecas

Fortran cuenta con una serie de funciones matemáticas pre-definidas llamadas ”funciones intrínsecas”. A continuación se presentan unas de ellas.

- max(a,b) Máximo entre a y b
- min(a,b) Mínimo entre a y b
- sqrt(x) Raíz Cuadrada de x
- abs(x) Valor Absoluto de x
- cos(x) Coseno de x
- sin(x) Seno de x
- tan(x) Tangente de x
- exp(x) Exponencial de x
- alog(x) Logaritmo natural de x
- asin(x) Arco-seno de x
- acos(x) Arco-coseno de x
- atan(x) Arco-tangente de x
- matmul(A, B) Retorna una Matriz del Producto matricial de A por B

Funciones Intrínsecas

Nota: Las funciones trigonométricas se usan siempre en Radianes, y no en Grados

Loops

Un “loop” es un conjunto de instrucciones que deben de realizarse muchas veces y tiene la forma estándar:

DO

do i = inicio, fin, incremento

Comando 1

Comando 2

Comando 3

……

end do

Nota: La variable **i** debe de ser de tipo **integer**.

Loops

DO WHILE

do while(expresión lógica)

Comando 1

Comando 2

Comando 3

….

end do

CUIDADO: Con este tipo de loop se corre el riesgo de caer en un ciclo infinito, cuando la condición lógica nunca se cumpla

IF

En ocasiones uno desea que una parte del programa solo sea ejecutada si cierta condición específica se satisface. Esto se logra utilizando los “condicionales”, que en Fortran se controlan con el omando IF. La sintaxis de este comando es:

if(expresión lógica) **then**

    Comando 1

    Comando 2

else

    Comando 3

    Comando 4

end if

Operadores Lógicos

Nombre	operador	ejemplo
igualdad	==	if(i == 0)
mayor que	>	if(i > 5)
mayor o igual	>=	if(i >= 5)
menor que	<	if(i < 5)
menor o igual	<=	if(i <= 5)
Diferente	/=	if(1 /= 5)
Negación Lógica	.not.	if(.not. Band)
“O” lógico	.or.	If((i==3) .or. (i==5))
“Y” lógico	.and.	If((i==3) .and. (i==5))

Entrada y Salida de Datos

Obtener y Mostrar Datos

En la mayoría de los códigos científicos es necesario obtener datos desde afuera y mostrarlos hacia afuera. Por default la captura de Datos es desde el Teclado, y la Salida de los Datos es la pantalla.

La entrada y salida de datos se manejan con los comandos:

- **Read(,)**
- **Write(,)**

Entrada y Salida de Datos

Ambos comandos tienen dos argumentos:

- El primer argumento indica la unidad de entrada o salida.
- El segundo argumento indica el formato en el que están los datos.

La versión más simple es:

  - Read(*, *)
  - Write(*, *)

Entrada y Salida de Datos

Sintaxis de write

write(*, *) ‘Cadena a mostrar en la pantalla’

Sintaxis de read

read(*, *) identificadorVariable

Entrada y Salida de Datos

Ejemplos del comando **write(*,*)**

- Write(*, *) ‘Inserte un valor para la Fracción de Volumen : ’
- Write(*, *) ‘Dame el valor del radio: ’

NOTA: El comando **write** se puede sustituir por: **print *** seguido de una coma.

- Ejemplo

  print *, ‘Dame el valor del radio: ’

  print*, ‘Inserte un valor para la Fracción de Volumen : ’

Entrada y Salida de Datos

Ejemplos del comando **read(*, *)**

- read(*, *) fraccionVolumen
- read(*, *) radio

Entrada y Salida de Datos

Manejo de Archivos

Además de mostrar los datos en pantalla y obtenerlos desde el Teclado, muchas veces es necesario leer y guardar los datos obtenidos desde un Archivo. Por fortuna, Fortran nos brinda la facilidad de Manejar Archivos.

Entrada y Salida de Datos

Manejo de Archivos

Creación de un Archivo

- Sintaxis:

open (unidadAsignaSalida/EntradaDatos, **file=** ‘nombreArchivo/PATH’)

NOTA: Para el nombre del Archivo que no esta en el mismo Directorio que el Programa, se debe de usar la Ruta Completa (PATH) seguido del nombre.

Entrada y Salida de Datos

Crear un Archivo

Ejemplo:

Open(10, file='archivo.dat')

Crea un archivo en el mismo directorio donde se encuentra el programa, con el nombre archivo.dat.

Open (11, file='/home/andres/Documentos/archivo.dat')

Crea un archivo en la Ruta especificada, con el nombre archivo.dat.

Entrada y Salida de Datos

Lectura y Escritura de un Archivo

Escritura Formato Libre

write (NumeroArchivo, *) var1, var 2, … , varN

Ejemplo:

write(10, *) k, deltaK, fraccionVolumen

Escritura con Formato

write(NumeroArchivo, ‘formato’) var1, var2, …, varN

Ejemplo:

write(10, ‘(2f10.4)’) k, deltaK, fraccionVolumen

Entrada y Salida de Datos

Lectura y Escritura de un Archivo

Lectura Formato Libre

read (NumeroArchivo, *) var1, var 2, … , varN

Ejemplo:

read(10, *) k, deltaK, fraccionVolumen

Lectura con Formato

read(NumeroArchivo, ‘formato’) var1, var2, …, varN

Ejemplo:

read(10, ‘(2f10.4)’) k, deltaK, fraccionVolumen

Entrada y Salida de Datos

Cierre de un Archivo

Una vez que se termine de usar el Archivo, independientemente de que se haya creado para Leer o Escribir, se debe de Cerrar el archivo con el comando **close**.

Sintaxis

close(Número de la Unidad)

Ejemplo:

close(10)

Subprogramas

En la mayoría de las veces existen tareas que se deben de realizar varias veces en el Progama. Por lo que resulta conveniente que sean subprogramas en si mismos. Esto también hace que el código sean más modular y permita que diferentes personas puedan enternder su contenido.

En Fortran 90 hay 3 tipos básicos de subprogramas :

- Funciones
- Subrutinas
- Módulos

Subprogramas

Funciones

EL objetivo de una función es obtener un número a partir de los argumentos.

Sintaxis

```
tipoDato function nombreFuncion(arg1, arg2, … , argN)
    ...
    ...
    return
end function
```

Llamado de la Función

```
identificador = nombreFuncion(arg1, arg2, …, argN)
```

Subprogramas

Subrutinas

Una subrutina es similar a una función, pero no solo se espera un número a regresar, sino toda una secuencia de operaciones que requieren regresar muchos números al programa principal o ninguno.

Sintaxis

```
subroutine nombre(arg1, arg2, …, argN)
    …
    ...
    return
end subroutine
```

Llamado de la subrutina

```
call nombreSubrutina(arg1, agr2, …, argN)
```

Nota: A diferencia de las funciones, la subrutina no tiene tipo de Dato.

Subprogramas

Módulos

Los módulos sirven para declarar variables que se usan en muchos subprogramas o para agrupar muchos subprogramas en una sola unidad.

Sintaxis

```
module nombreModulo
    .......
    .......
end module nombreModulo
```

A diferencia de las funciones y subrutinas, si el módulo esta en el mismo archivo que el programa principal, debe estar antes que este.

Llamado de los módulos

```
use NombreModulo
```

Este debe ser llamado inmediatamente después del nombre del subprograma.

Nota: En el los módulos no se usa el comando return.