# Resumen

- Funciones intrínsicas
- Estructuras de control de flujo: ciclos y condicionales.
- Subprogramas: Funciones, subrutinas y modulos.

# Funciones intrínsecas

Funciones matemáticas

- sqrt(x) Raíz cuadrada de x.
- abs(x) Valor absoluto de x.
- sin(x) Seno de x.
- cos(x) Coseno de x.
- tan(x) Tangente de x.
- asin(x) Arco-seno de x.
- acos(x) Arco-coseno de x.
- atan(x) Arco-tangente de x.
- exp(x) Exponencial de x
  alog(x) Logaritmo natural de x.
- alog10(x) Logaritmo en base 10 de x.
- max(x,y) Máximo entre x, y.
  min(x,y) Mínimo entre x, y

# Funciones intrínsecas

Funciones numéricas

- ABS (A) Valor absoluto
- AIMAG(Z)  parte imaginaria
- CMPLX (X [, Y, KIND]) Conversion a complejo
- DBLE (A) Conversion a doble precision
- INT (A [, KIND]) Conversion a entero
- REAL (A [, KIND]) Conversion a real
- MOD (A, P) Resto de A/P

# Funciones intrínsecas

Funciones de caracteres

- INDEX (STRING, SUBSTRING [, BACK])  Posición inicial de una subcadena en una cadena.
- LEN (STRING) Longitud de la cadena
- TRIM (STRING) Remueve los caracteres en blanco al final

Funciones de matrices

- DOT_PRODUCT (VECTOR_A, VECTOR_B) producto escalar
- MATMUL (MATRIX_A, MATRIX_B) multiplicacion de matrices

# Funciones intrínsecas

Funciones de arreglos

- MAXVAL (ARRAY, DIM [, MASK]). Valor máximo
- MINVAL (ARRAY, DIM [, MASK]) Valor mínimo.
- SIZE (ARRAY [, DIM])  Numero de elementos
- TRANSPOSE (MATRIX)  Matriz traspuesta

# Funciones intrínsecas: nombres genérico y específicos

Función SQRT

| Name | Argument | Return type | Standard |
|---|---|---|---|
| SQRT(X) | REAL(4) | REAL(4) | Fortran 95 and later |
| DSQRT(X) | REAL(8) | REAL(8) | Fortran 95 and later |
| CSQRT(X) | COMPLEX(4) | COMPLEX(4) | Fortran 95 and later |
| ZSQRT(X) | COMPLEX(8) | COMPLEX(8) | GNU extension |
| CDSQRT(X) | COMPLEX(8) | COMPLEX(8) | GNU extension |

# Ejercicio:

```
program raiz
print *,'sqrt(4.d0)=',sqrt(4.d0)
print *,'sqrt(4.0)=',sqrt(4.0)
print *,'dsqrt(4.d0)=',dsqrt(4.0)
print *,'csqrt((4.0,0.0))=',csqrt((4.0,0.0))
print *,'sqrt((4.0,0.0))=',sqrt((4.0,0.0))
print *,'zsqrt((4.d0,0.d0))=',zsqrt((4.d0,0.d0))
print *,'sqrt((4.d0,0.d0))=',sqrt((4.d0,0.d0))
end program raiz
```

# Conversión de tipo de datos
(fun.f90)

```
program funciones   ! fun.f90
implicit none
double precision :: x,y,z
real(4) :: x0,z0,y0
x=3.14d0
x0=3.14
z=3.14d0/2
z0=3.14/2
y=1/10.0d0
y0=1/10.0
print *,'cos(3.14d0)=', cos(x)
print *,'cos(3.14)  =', cos(x0)
print *,'tan(3.14d0/2)=',tan(z)
print *,'tan(3.14/2)  =',tan(z0)
print *,'1/10.0d0=',y
print *,'1/10.0  =',y0
y=1/10.0  ! division como real y asignado a double precision
print *,'1/10.0d0=',y
end program funciones
```

# Atención
(conversion.f90)

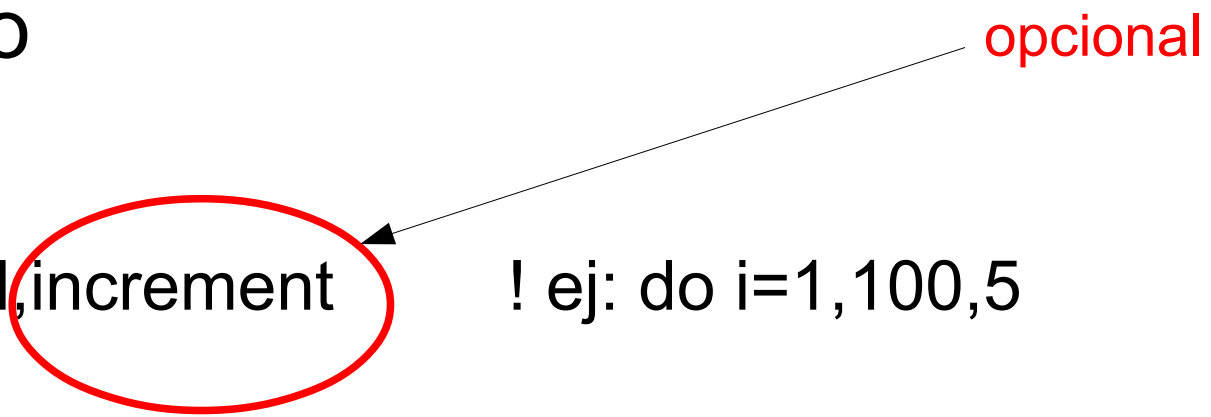
```
program test
integer(kind=1) :: i
real :: r=2.0d3
i=r
write(*,*) i,r
End

MacBook-Air-de-Eduardo:Tarea2 emenendez$ ./a.out
  -48   2000.00000
```

No asigne directamente  a un entero un real. Se truncaran los bits, y el entero no tiene nada que ver con lo que se squiere. Use int(numero).
Explore lo que ocrurre definiendo integer(kind=4) :: i

# Control de flujo: bucles, condicionales.

- Ciclos do




```
do i=start,end,increment        ! ej: do i=1,100,5
   comando 1
   comando 2
   comando 3
   ...
end do
```

Nota: Se admite **enddo** o **end do**

# Sintaxis antigua del ciclo do

```
C       Ejemplo:
        do 111 i=1,20
          comando 1
          comando 2
111     continue
```

# Ciclo do con nombre y ciclo sin fin

```
program testdo ! dof90.f90
! ciclo con nombre
primero: do i=1,10
          print *, i
         end do primero
! ciclo sin fin
do
 print *, 'Digite un numero'
 read (*,*) i
 if (i>3) exit
end do
end program testdo
```

# Estructuras condicionales

- Estructura IF...THEN...ELSE

```
if (expresión lógica) then
  comando 1
  comando 2
else
  comando 3
endif
```

Ejemplo:

```
if (i>3) then
  print *,'Se termina el ciclo'
  exit
else
  print *,'El cliclo continua'
endif
```

Operadores lógicos: ==, >, >=, <, <=, /=, .not. , .or. , .and.

En F77: .EQ., .GT., .GE., .LT., .LE., .NE., .NOT., .OR.,.AND.

# Condicional simplificado

- Comando unico

If (condicion) comando

Ejemplo:

if (i>3) exit

- If sin else

```
If(condicion) then
  Comando 1
  Comando 2
endif
```

- Ejemplo de condicional compuesta:

```
if((i>3).and.(i<100)) then
 write(*,*) i
 i=i+1
end if
```

# Bloque if con nombre

```
nombre1: if (condición) then
            Comandos
         else
            Comandos
         endif nombre1
```

Nota: puede ser **endif** o **end if**

# Condiciones múltiples

```
If ( condición 1) then
    comando 1
else if (condición 2) then
   comando 2
else if (condición3) then
   comando 3
else
   comando 4
end if
```

# SELECT CASE

```
[name:] select case (expression)
        case (selector1)
        ! some statements
        ...
        case (selector2)
        ! other statements
        ...
        case default
        ! more statements ...
      end select [name]
```

Ejemplo:

```
Program ejemplo_case
Integer :: n
select case (n)
case (1)
  print*, "You entered 1"
case (2:6)
  print*, n
case default
print*, "Number >6”
end select
```

# Ejercicio

Escriba un programa que calcule las raíces de la ecuación cuadrática $ax^2 + bx + c = 0$. El programa debe distinguir entre los tres casos en que el discriminante $(b^2 - 4ac)$ es positivo, negativo, o cero. Use una construcción if . Puede requerir usar la función intrinseca cmplx.

(cuadratica.f90)

# Ciclos do while

```
do while(condición)
 comando 1
 comando 2
end do
```

- Ejemplo

```
i=1
do while (i <=10)
  i=i+1
end do
```

CUIDADO: Con este tipo de loops se corre el riesgo de caer en un ciclo eterno, donde la condición lógica nunca deja de satisfacerse.
En el ejemplo previo bastaría cambiar i+1 por i-1 .

# Subprogramas: funciones, subrutinas, módulos.

- Funciones definidas por el programador. Ejercicio: Escribir un programa que defina la función del area de un circulo como función del radio. El programa debe pedir el radio y entregar el área calculada mediante la función.
- Siga el ejemplo de la lámina siguiente (fun_prog.f90)

# Ejercicio ejemplo: FUNCTION

```
program fun_prog
implicit none
real:: r,A
print *,'digite el radio'; read(*,*) r
A=area(r)
print *,'El area es ',A
end program fun_prog

function area(r)
implicit none
real :: r,pi
pi=acos(-1.0)
area=pi*r*r
return
end function area
```

Nota: al compilar verá que hay que corregir algunos errores.

# Ejercicio ejemplo: FUNCTION

```
program fun_prog
implicit none
real:: r,A
real :: area
print *,'digite el radio'; read(*,*) r
A=area(r)
print *,'El area es ',A
end program fun_prog

function area(r)
implicit none
real :: area,pi,r
pi=acos(-1.0)
area=pi*r*r
return
end function area
```

El tipo numérico de la función debe declararse

Las variables son locales. En particular las variable r y pi.

Nota: este es el programa corregido

# SUBROUTINE

Una subrutina es similar a una función, pero más complicada, de la que no solo se espera un número, sino toda una secuencia de operaciones que pueden requerir regresar muchos números al programa principal (o ninguno).

# El ejemplo anterior como subrutina

```
program progarea
implicit none
real:: r,A
print *,'digite el radio'; read(*,*) r
call area(r,A)
print *,'El area es ',A
end program progarea

subroutine area(r,A)
implicit none
real :: A,pi,r
pi=acos(-1.0)
A=pi*r*r
return
end subroutine area
```

Las variables son locales, pero como r y A son argumentos de la funcion, modifican las del programa principal. En cambio, la variable pi solamente está definida en la subrutina.

Ejercicios:

1) Separar el programa principal en dos archivos, compilarlos y hacer el ejecutable. Usar gfortran -c  y después enlazar los archivos objeto.

2) Cambie el nombre de las variables r o A en la subrutina y vea que pasa.

3) Cambie las variables de la subrutina a double precision y vea que pasa.

# Mas diversión: jugar con los tipos de variable

```
program progarea
implicit none
real(8):: r,A
print *,'digite el radio'; read(*,*) r
call area(r,A)
print *,'El area es ',A
end program progarea

subroutine area(r,A)
implicit none
real :: A,pi,r
pi=acos(-1.0)
A=pi*r*r
return
end subroutine area
```

1) Cambie las variables del programa principal a real(8) y vea que pasa.

2) Hagalo a la inversa: real(8) en la subrutina y real (4) en el programa principal.

# Módulos

- Como hemos visto, las variables se declaran localmente un cada función o subrutina, y pueden transferirse al programa principal como argumentos de los subprogramas.
- A veces es necesario definir variables y contantes globales, y es conveniente poder definirlas en un unico lugar del programa. La estructura de subprograma MODULE permite esto.

# Ejemplo en el programa del área

```
! Modulo "constantes".
module constantes
! Declarar parametros.
implicit none
real, parameter :: pi=3.141592,ee=2.71828
end module constantes

 subroutine area(r,A)
 use constantes, only: pi
 implicit none
 real :: A,r
 A=pi*r*r
 return
 end subroutine area
```

La opción **only** permite seleccionar la variable pi. Si se pone **use constantes** se usa todo el contenido del modulo.

# Alternativa: bloque COMMON

```
! Modulo "constantes".
program principal
implicit none
real pi=3.141592,ee=2.71828
COMMON /global/ pi,ee
.....

 subroutine area(r,A)
 use constantes, only: pi
 implicit none
 real :: A,r
COMMON /global/ pi,ee
 A=pi*r*r
 return
 end subroutine area
```

La instrucción COMMON permite transferir variables entre distintos subprogramas y con el programa principal.

Proviene del F77, su uso es obsoleto, pero está presente en muchos programas vigentes.

# Ejemplo de uso COMMON

```
      COMMON /A1/ ZNS(107),ZNP(107),ZND(107),ZND2(107),VAR(10),
     &            ANS(107),ANP(107),ANV(107),F2(17),G1(17),EA(107)
     &       /A2/ BE(107,2),U1(107,2),U2(107,2)
     &       /A3/ GE(107,3),UM(107,2)
      COMMON IR,IW,IC,JOB,IERR,NEX,NOCC,ICHGE,NR,KORD,INSP,IDUMB,
     &       AUI,AUII,AUEV,AUEVI,NO1(NATMAX)
      COMMON /ARR/ AR(3*NATMAX)
      COMMON /N11/ NAT(NATMAX)
```

Deben estar en todas las subrutinas o funciones que usan las variables globales.

# Modulos: en detalle

Un modulo contiene dos partes: declaraciones y subprogramas. La estructura es

**module module_name**

**! specification statements**

**contains**

**! procedures**

**end module module_name**

Ejemplo:

```
module constants
implicit none
real, parameter :: pi =
& 3.1415926536
real, parameter :: e =
& 2.7182818285
contains
subroutine show_consts()
print*, “pi = “, pi
print*, e=“.e
end subroutine show_consts
end module constants
```

# Modulos: en detalle

**Por defecto todo lo que hay en un modulo es publico y global. Es posible declarar como privadas algunas variables dentro de un modulo, de forma que no sean accesible fuera.**

**Se usa el atributo private. En el ejemplo anterios, se puede declarar asi la variable e.**

Ejemplo:

```
module constants
implicit none
real, parameter :: pi =
& 3.1415926536
real, parameter, private :: e =
& 2.7182818285
public:: show_consts
contains
subroutine show_consts()
print*, “pi = “, pi
print*, e=“.e
end subroutine show_consts
end module constants
```

# Resumen

- Funciones intrínsicas
- Estructuras de control de flujo: ciclos y condicionales.
- Subprogramas: Funciones, subrutinas y modulos.

# Ejercicio de tarea

- Hacer un programa que cree dos matrices de numeros reales de doble precisión, de dimension NxN, según cierta regla y haga su multiplicación. Estudiar la dependencia del tiempo de ejecución del programa con la dimension de la matriz.  Este ejercicio será usado en la próxima clase.
- Hagalo de dos maneras. Con el comando MATMUL, y haciendo la multiplicación explicitamente.