

Programación en FORTRAN

Nivel Básico - Sesión 4

Martin Josemaría Vuelta Rojas

23 de enero de 2018

SoftButterfly

1. Procedimientos
2. Subrutinas
3. Funciones

Procedimientos

- Qué es una unidad programática?

- Qué es una unidad programática?
- Existen cuatro tipos de unidades programáticas:

- Qué es una unidad programática?
- Existen cuatro tipos de unidades programáticas:
 - *program*, unidad programática principal.

Procedimientos

- Qué es una unidad programática?
- Existen cuatro tipos de unidades programáticas:
 - *program*, unidad programática principal.
 - *subroutine*, contempla instrucciones ejecutables.

Procedimientos

- Qué es una unidad programática?
- Existen cuatro tipos de unidades programáticas:
 - *program*, unidad programática principal.
 - *subroutine*, contempla instrucciones ejecutables.
 - *function*, tipo particular de subrutina.

Procedimientos

- Qué es una unidad programática?
- Existen cuatro tipos de unidades programáticas:
 - *program*, unidad programática principal.
 - *subroutine*, contempla instrucciones ejecutables.
 - *function*, tipo particular de subrutina.
 - *module*, contempla instrucciones de declaración de variables, su inicialización e interfaces entre funciones y subrutinas.

Procedimientos

- Qué es una unidad programática?
- Existen cuatro tipos de unidades programáticas:
 - *program*, unidad programática principal.
 - *subroutine*, contempla instrucciones ejecutables.
 - *function*, tipo particular de subrutina.
 - *module*, contempla instrucciones de declaración de variables, su inicialización e interfaces entre funciones y subrutinas.
- El programa principal, las subrutinas y funciones son llamados procedimientos, siendo estos de orden jerárquico.

Procedimientos

- Qué es una unidad programática?
- Existen cuatro tipos de unidades programáticas:
 - *program*, unidad programática principal.
 - *subroutine*, contempla instrucciones ejecutables.
 - *function*, tipo particular de subrutina.
 - *module*, contempla instrucciones de declaración de variables, su inicialización e interfaces entre funciones y subrutinas.
- El programa principal, las subrutinas y funciones son llamados procedimientos, siendo estos de orden jerárquico.
- Un procedimiento es interno si está definido dentro de una unidad programática, que puede ser "hospedante" o bien un módulo; mientras que los externos no están contenidas en unidad programática alguna.

Subrutinas

Subrutinas

- Una subrutina es un programa de menor orden jerárquico al principal.

Subrutinas

- Una subrutina es un programa de menor orden jerárquico al principal.
- Tiene como objetivo llevar a cabo instrucciones ejecutables, por un subprograma de nivel superior, a través de una instrucción que llame a la subrutina.

Subrutinas

- Una subrutina es un programa de menor orden jerárquico al principal.
- Tiene como objetivo llevar a cabo instrucciones ejecutables, por un subprograma de nivel superior, a través de una instrucción que llame a la subrutina.
- La sintaxis es la siguiente:

```
1  subroutine <nombre><argumentos (ficticios)>  
2  !instrucciones de declaración de los argumentos (ficticios)  
3  !instrucciones de declaración de los objetos locales  
4  !instrucciones ejecutables  
5  end subroutine <nombre>
```

donde los <argumentos (ficticios)>, en caso de uso, son los objetos sobre los cuales la subrutina trabajará preferentemente.

Subrutinas

- Una subrutina es un programa de menor orden jerárquico al principal.
- Tiene como objetivo llevar a cabo instrucciones ejecutables, por un subprograma de nivel superior, a través de una instrucción que llame a la subrutina.
- La sintaxis es la siguiente:

```
1  subroutine <nombre><argumentos (ficticios)>  
2  !instrucciones de declaración de los argumentos (ficticios)  
3  !instrucciones de declaración de los objetos locales  
4  !instrucciones ejecutables  
5  end subroutine <nombre>
```

donde los <argumentos (ficticios)>, en caso de uso, son los objetos sobre los cuales la subrutina trabajará preferentemente.

- La subrutina es llamada por un subprograma por medio de la instrucción CALL

```
1  <identificacion unidad programatica>  
2  :  
3  call <nombre><argumentos (usados)>  
4  :  
5  end subroutine <unidad programática>
```


Subrutinas

```
PROGRAM Subroutines01
  IMPLICIT NONE

  REAL :: P, Q, R, Root1, Root2
  INTEGER :: IFail=0
  LOGICAL :: OK=.TRUE.

  CALL Interact(P,Q,R,OK)

  IF (OK) THEN
    CALL Solve(P,Q,R,Root1,Root2,IFail)
    IF (IFail == 1) THEN
      PRINT *, 'Complex roots' ,

```

```
SUBROUTINE Interact(A,B,C,OK)
  IMPLICIT NONE
  REAL , INTENT(OUT) :: A
  REAL , INTENT(OUT) :: B
  REAL , INTENT(OUT) :: C
  LOGICAL , INTENT(OUT) :: OK
  INTEGER :: IO_Status=0
  PRINT*, ' Type in the coefficients A, B AND C'
  READ(UNIT=*,FMT=*,IOSTAT=IO_Status)A,B,C
  IF (IO_Status == 0) THEN
    OK=.TRUE.
  ELSE
    OK=.FALSE.
  ENDIF
END SUBROUTINE Interact

SUBROUTINE Solve(E,F,G,Root1,Root2,IFail)
  IMPLICIT NONE
  REAL , INTENT(IN) :: E

```

Declaración de Argumentos ficticios

- Los argumentos ficticios pueden ser variables, funciones, subrutinas, arreglos, punteros o procedimientos de módulo.

Declaración de Argumentos ficticios

- Los argumentos ficticios pueden ser variables, funciones, subrutinas, arreglos, punteros o procedimientos de módulo.
- Existe un bloque de instrucciones que permite declarar argumentos ficticios de una subrutina:

Declaración de Argumentos ficticios

- Los argumentos ficticios pueden ser variables, funciones, subrutinas, arreglos, punteros o procedimientos de módulo.
- Existe un bloque de instrucciones que permite declarar argumentos ficticios de una subrutina:
 - *intent*(in): variables y arreglos cuyos valores permiten recibir información proporcionada por el subprograma que llama a la subrutina. Los valores no pueden ser modificados por la subrutina (comportamiento local).

Declaración de Argumentos ficticios

- Los argumentos ficticios pueden ser variables, funciones, subrutinas, arreglos, punteros o procedimientos de módulo.
- Existe un bloque de instrucciones que permite declarar argumentos ficticios de una subrutina:
 - *intent(in)*: variables y arreglos cuyos valores permiten recibir información proporcionada por el subprograma que llama a la subrutina. Los valores no pueden ser modificados por la subrutina (comportamiento local).
 - *intent(on)*: variables y arreglos cuyos valores permiten devolver información, además de albergar resultados de un subprograma. Estos pueden ser modificados por la subrutina.

Declaración de Argumentos ficticios

- Los argumentos ficticios pueden ser variables, funciones, subrutinas, arreglos, punteros o procedimientos de módulo.
- Existe un bloque de instrucciones que permite declarar argumentos ficticios de una subrutina:
 - *intent(in)*: variables y arreglos cuyos valores permiten recibir información proporcionada por el subprograma que llama a la subrutina. Los valores no pueden ser modificados por la subrutina (comportamiento local).
 - *intent(on)*: variables y arreglos cuyos valores permiten devolver información, además de albergar resultados de un subprograma. Estos pueden ser modificados por la subrutina.
 - *intent(inout)*: variables y arreglos cuyos valores a la entrada pueden ser modificados (argumentos mixtos).

Declaración de arreglos

Declaración de Argumentos ficticios

Declaración de arreglos

- Primera regla: el tamaño de los arreglos empleados debe ser mayor o igual que el de los ficticios.

```
1  REAL , ALLOCATABLE , DIMENSION &
2  (:,:)::One,Two,Three,One_T
3  INTEGER :: I,N
4  INTERFACE
5      SUBROUTINE Matrix_bits(A,B,C,A_T,N)
6          IMPLICIT NONE
7          INTEGER, INTENT(IN):: N
8              REAL, DIMENSION (:,:), INTENT(IN) :: A,B
9              REAL, DIMENSION (:,:), INTENT(OUT) :: C,A_T
10     END SUBROUTINE Matrix_bits
11 END INTERFACE
12 PRINT *, 'Input size of matrices'
13 :
```


Declaración de Argumentos ficticios

- Segunda regla: establecer correctamente la relación entre los elementos de ambos arreglos.

```
1  SUBROUTINE Matrix_bits(A,B,C,A_T,N) !Véase subroutines06.f95
2  IMPLICIT NONE
3  INTEGER, INTENT(IN):: N
4  REAL, DIMENSION (:,:), INTENT(IN) :: A,B
5  REAL, DIMENSION (:,:), INTENT(OUT) :: C,A_T
6  INTEGER:: I,J, K
7  REAL:: Temp
8  !begining of matrix multiplication C = AB
9  DO I=1,N
10     DO J=1,N
11         Temp=0.0
12         DO K=1,N
13             Temp = Temp + A(I,K) * B (K,J)
14         END DO
15         C(I,J) = Temp
16     END DO
17 END DO
18 : !Calculate A_T transpose of A !set A_T to be transpose matrix A
19 END SUBROUTINE Matrix_bits
```

Declaración de Argumentos ficticios

Declaración de cadenas de caracteres

Declaración de Argumentos ficticios

Declaración de cadenas de caracteres

- La forma más conveniente es fijando la longitud de la cadena.

```
1  SUBROUTINE subrutina(ciudad)
2      CHARACTER(len=10)::ciudad
3      :
4  END SUBROUTINE subrutina
```

Declaración de Argumentos ficticios

Declaración de cadenas de caracteres

- La forma más conveniente es fijando la longitud de la cadena.

```
1 SUBROUTINE subrutina(ciudad)
2   CHARACTER(len=10)::ciudad
3   :
4 END SUBROUTINE subrutina
```

- De manera explícita, por medio de un argumento tipo INTEGER.

```
1 SUBROUTINE subrutina(n, ciudad)
2   INTEGER, INTENT(IN)::n
3   CHARACTER(len=n), INTENT(INOUT)::ciudad
4   :
5 END SUBROUTINE subrutina
```

Declaración de Argumentos ficticios

Declaración de cadenas de caracteres

- La forma más conveniente es fijando la longitud de la cadena.

```
1 SUBROUTINE subrutina(ciudad)
2   CHARACTER(len=10)::ciudad
3   :
4 END SUBROUTINE subrutina
```

- De manera explícita, por medio de un argumento tipo INTEGER.

```
1 SUBROUTINE subrutina(n, ciudad)
2   INTEGER, INTENT(IN)::n
3   CHARACTER(len=n), INTENT(INOUT)::ciudad
4   :
5 END SUBROUTINE subrutina
```

- De manera implícita, utilizando el símbolo "*"

```
1 SUBROUTINE subrutina(ciudad)
2   CHARACATER(len=*) INTENT(INOUT)::ciudad
3   :
4   n=len(ciudad) !da la longitud de la cadena del argumento en uso.
5   :
6 END SUBROUTINE subrutina
```

Declaración de Objetos locales

- Los objetos locales son de uso exclusivo de una subrutina y no utilizados por subprogramas de nivel superior.

Declaración de Objetos locales

- Los objetos locales son de uso exclusivo de una subrutina y no utilizados por subprogramas de nivel superior.
- Para conservar los valores de las variables y arreglos locales de una llamada a otra, es necesario el atributo SAVE.

```
1  SUBROUTINE subrutina(<argumentos>)  
2    : !declaración argumentos  
3    INTEGER, SAVE::i=0  
4    :
```

Declaración de Objetos locales

- Los objetos locales son de uso exclusivo de una subrutina y no utilizados por subprogramas de nivel superior.
- Para conservar los valores de las variables y arreglos locales de una llamada a otra, es necesario el atributo `SAVE`.

```
1  SUBROUTINE subrutina(<argumentos>)
2    : !declaración argumentos
3    INTEGER, SAVE::i=0
4    :
```

- En caso los otros objetos, como procedimientos (subrutinas y funciones) sean externos, se puede utilizar la instrucción bloque *INTERFACE* para declararlos.

```
1  INTERFACE
2    SUBROUTINE subrutina1 (<argumentos ficticios>)
3      !declaración de argumentos ficticios
4    END SUBROUTINE subrutina1
5    :
6    SUBROUTINE subrutinak (<argumentos ficticios>)
7      !declaración de argumentos ficticios
8    END SUBROUTINE subrutinak
9  END INTERFACE
```


Declaración de Objetos locales

- A diferencia del programa principal, en las subrutinas, tanto arreglos, como cadenas de caracteres, pueden ser declarados a partir del valor un argumento (ficticio) de tipo INTEGER.

```
1 SUBROUTINE subrutina(n, <otros argumentos>)
2   INTEGER, INTENT(IN)::n
3   :   !declaración argumentos
4   INTEGER, SAVE::i = 0
5   REAL(KIND=8), DIMENSION(0:n)::A
6   CHARACTER(LEN=n)::ciudad
7   :
```

Declaración de Objetos locales

- A diferencia del programa principal, en las subrutinas, tanto arreglos, como cadenas de caracteres, pueden ser declarados a partir del valor un argumento (ficticio) de tipo INTEGER.

```
1 SUBROUTINE subrutina(n, <otros argumentos>)
2   INTEGER, INTENT(IN)::n
3   :   !declaración argumentos
4   INTEGER, SAVE::i = 0
5   REAL(KIND=8), DIMENSION(0:n)::A
6   CHARACTER(LEN=n)::ciudad
7   :
```

- Cuando se declara un arreglo o una cadena de caracteres utilizando el valor de un argumento, la situación es similar a la de un arreglo dinámico, es decir, existe cuando la subrutina se ejecute más no cuando se emplee SAVE.

- Es parte de un subprograma de nivel jerárquico superior, por lo tanto, de uso local.

Subrutinas Internas

- Es parte de un subprograma de nivel jerárquico superior, por lo tanto, de uso local.
- La estructura sintáctica es la siguiente:

```
1  <identificación unidad programática>
2  :
3      CALL subrutina [(<argumentos de uso>)]
4      :
5      CONTAINS
6          SUBROUTINE subrutina[(<argumentos ficticios>)]
7          :
8          END SUBROUTINE subrutina
9      :
10 END <unidad programática>
```

- Una subrutina interna tiene acceso a todos los objetos de la unidad programática que la contiene; excepto que sean declaradas como locales o argumentos ficticios.

- Una subrutina interna tiene acceso a todos los objetos de la unidad programática que la contiene; excepto que sean declaradas como locales o argumentos ficticios.
- Una subrutina puede contener procedimientos internos, a condición de que no se trate de una subrutina interna; utilizando la estructura CONTAINS.

- Una subrutina interna tiene acceso a todos los objetos de la unidad programática que la contiene; excepto que sean declaradas como locales o argumentos ficticios.
- Una subrutina puede contener procedimientos internos, a condición de que no se trate de una subrutina interna; utilizando la estructura CONTAINS.
- Cuando son empleadas como argumentos de uso, no deben ser declaradas a través de *interface*.

Ventajas y desventajas de uso

Ventajas

Ventajas y desventajas de uso

Ventajas

- No es necesario construir una *interface*.

Ventajas y desventajas de uso

Ventajas

- No es necesario construir una *interface*.
- La subrutina interna tiene acceso a todos los objetos del procedimiento "hospedante". No es necesario pasarlos por argumentos o vía módulos.

Ventajas y desventajas de uso

Ventajas

- No es necesario construir una *interface*.
- La subrutina interna tiene acceso a todos los objetos del procedimiento "hospedante". No es necesario pasarlos por argumentos o vía módulos.

Desventajas

Ventajas y desventajas de uso

Ventajas

- No es necesario construir una *interface*.
- La subrutina interna tiene acceso a todos los objetos del procedimiento "hospedante". No es necesario pasarlos por argumentos o vía módulos.

Desventajas

- Existe un riesgo de modificar accidentalmente los valores de las variables del procedimiento "hospedante".

Ventajas y desventajas de uso

Ventajas

- No es necesario construir una *interface*.
- La subrutina interna tiene acceso a todos los objetos del procedimiento "hospedante". No es necesario pasarlos por argumentos o vía módulos.

Desventajas

- Existe un riesgo de modificar accidentalmente los valores de las variables del procedimiento "hospedante".
- Una subrutina interna solamente puede ser accedida por la unidad programática que la contiene. No puede ser utilizada por otras.

Subrutinas Internas

```
1  PROGRAM Subroutines03      !Véase subroutine03.f95
2  IMPLICIT NONE
3  REAL :: P, Q, R, Root1, Root2
4  INTEGER :: IFail=0
5  LOGICAL :: OK=.TRUE.
6      CALL Interact(P,Q,R,OK)
7      IF (OK) THEN
8          CALL Solve(P,Q,R,Root1,Root2,IFail)
9          IF (IFail == 1) THEN
10             PRINT *, ' Complex roots, calculation abandoned'
11             :
12      END IF
```

Subrutinas Internas

```
CONTAINS
SUBROUTINE Interact(A,B,C,OK)
  IMPLICIT NONE
  REAL , INTENT(OUT) :: A
  REAL , INTENT(OUT) :: B
  REAL , INTENT(OUT) :: C
  LOGICAL , INTENT(OUT) :: OK
  INTEGER :: IO_Status=0
  PRINT*, ' Type in the coefficients A, B AND C'
  READ(UNIT=*,FMT=*,IOSTAT=IO_Status)A,B,C
  IF (IO_Status == 0) THEN
    :
  END SUBROUTINE Interact
```

Funciones

Funciones

- Una función es un tipo particular de subrutina, a la que se accede directamente sin utilizar la instrucción CALL.

Funciones

- Una función es un tipo particular de subrutina, a la que se accede directamente sin utilizar la instrucción CALL.
- Algunas ventajas del uso de funciones son

Funciones

- Una función es un tipo particular de subrutina, a la que se accede directamente sin utilizar la instrucción CALL.
- Algunas ventajas del uso de funciones son
 - Permite el uso de una acción, incluidas las funciones intrínsecas.

Funciones

- Una función es un tipo particular de subrutina, a la que se accede directamente sin utilizar la instrucción CALL.
- Algunas ventajas del uso de funciones son
 - Permite el uso de una acción, incluidas las funciones intrínsecas.
 - Permite dedicar una parte del código exclusivamente a resolver un problema.

Funciones

- Una función es un tipo particular de subrutina, a la que se accede directamente sin utilizar la instrucción CALL.
- Algunas ventajas del uso de funciones son
 - Permite el uso de una acción, incluidas las funciones intrínsecas.
 - Permite dedicar una parte del código exclusivamente a resolver un problema.
 - Permite construir una biblioteca de funciones o módulos para resolver sub-problemas particulares para un mejor orden y mayor efectividad.

Funciones

- Una función es un tipo particular de subrutina, a la que se accede directamente sin utilizar la instrucción CALL.
- Algunas ventajas del uso de funciones son
 - Permite el uso de una acción, incluidas las funciones intrínsecas.
 - Permite dedicar una parte del código exclusivamente a resolver un problema.
 - Permite construir una biblioteca de funciones o módulos para resolver sub-problemas particulares para un mejor orden y mayor efectividad.
- La estructura sintáctica de una función es la siguiente:

```
1  [<tipo>] FUNCTION <nombre> [(<argumentos ficticios>)] [result (<resultado>)]  
2  : ! declaración función (nombre o resultado)  
3  : ! declaración argumentos ficticios  
4  : ! declaración objetos locales  
5  : ! instrucciones ejecutables  
6  END FUNCTION <nombre>
```

Funciones

Función	Descripción	Ejemplo
abs	$ x , z , n $	$Y=ABS(X)$
sqrt	$(\sqrt{x}, x \geq 0)$,	$Y=SQRT(X)$
int	Parte entera de un real x	$J=INT(X)$
cos	$(\cos x, x \in \mathbb{R})$	$Y=COS(X)$
sin	$(\sin x, x \in \mathbb{R})$	$Y=SIN(X)$
tan	$(\tan x, x \in \mathbb{R})$	$Y=TAN(X)$
acos	$(\arccos x, x \in \mathbb{R})$	$Y=ACOS(X)$
asin	$(\arcsin x, x \in \mathbb{R})$	$Y=ASIN(X)$
atan	$(\arctan x, x \in \mathbb{R})$	$Y=ATAN(X)$
atan2	$(\arctan(A/B), A, B \in \mathbb{R})$	$Y=ATAN2(A,B)$
exp	$(\exp x, x \in \mathbb{R})$	$Y=EXP(X)$
log	$(\log x, x > 0)$	$Y=LOG(X)$
log10	$(\log_{10} x, x > 0)$	$Y=LOG10(X)$

Funciones intrínsecas

Funciones genéricas

Funciones genéricas

- Son aquellas funciones intrínsecas que pueden ser llamadas con cualquier argumento numérico de cualquier tipo KIND.

```
1  PROGRAM Functions01
2  IMPLICIT NONE
3  COMPLEX :: C=(1,1)
4  REAL    :: R=10.9
5  INTEGER :: I=-27
6      PRINT *,ABS(C)
7      PRINT *,ABS(R)
8      PRINT *,ABS(I)
9  END PROGRAM Functions01
```

Funciones elementales

Funciones elementales

- Estas funciones trabajan con argumentos escalares y arreglos, es decir, ya sean simple o multiple valuados.

Funciones elementales

- Estas funciones trabajan con argumentos escalares y arreglos, es decir, ya sean simple o multiple valuados.

```
1 PROGRAM Functions02
2 REAL , DIMENSION(5) :: X = (/1.0,2.0,3.0,4.0,5.0/)
3 PRINT *, ' Sine of ', X , ' = ', SIN(X)
4 END PROGRAM Functions02
```

Funciones elementales

- Estas funciones trabajan con argumentos escalares y arreglos, es decir, ya sean simple o multiple valuados.

```
1 PROGRAM Functions02
2 REAL , DIMENSION(5) :: X = (/1.0,2.0,3.0,4.0,5.0/)
3 PRINT *, ' Sine of ', X , ' = ', SIN(X)
4 END PROGRAM Functions02
```

Funciones de transformación

Funciones elementales

- Estas funciones trabajan con argumentos escalares y arreglos, es decir, ya sean simple o multiple valuados.

```
1 PROGRAM Functions02
2 REAL , DIMENSION(5) :: X = (/1.0,2.0,3.0,4.0,5.0/)
3 PRINT *, ' Sine of ', X , ' = ', SIN(X)
4 END PROGRAM Functions02
```

Funciones de transformación

- Son aquellas cuyos argumentos son arreglos. Actúan sobre otros arreglos.

Funciones elementales

- Estas funciones trabajan con argumentos escalares y arreglos, es decir, ya sean simple o multiple valuados.

```
1 PROGRAM Functions02
2 REAL , DIMENSION(5) :: X = (/1.0,2.0,3.0,4.0,5.0/)
3 PRINT *, ' Sine of ', X , ' = ', SIN(X)
4 END PROGRAM Functions02
```

Funciones de transformación

- Son aquellas cuyos argumentos son arreglos. Actúan sobre otros arreglos.

```
1 PROGRAM Functions0304
2 IMPLICIT NONE
3 REAL , DIMENSION(5) :: X = (/1.0,2.0,3.0,4.0,5.0/)
4 ! Transformational function dotproduct
5 PRINT *, ' Dot product of X with X is '
6 PRINT *, ' ', DOT_PRODUCT(X,X)
7 ! Transformational function sum
8 PRINT *, ' Sum of ', X , ' = ', SUM(X)
9 END PROGRAM Functions0304
```

Funciones predefinidas

Funciones de consulta

Funciones predefinidas

Funciones de consulta

- Estas funciones retornan información sobre sus argumentos. Pueden subclasificarse en BIT, CHARACTER, NUMERIC, ARRAY, POINTER y ARGUMENT PRESENCE.

Bit	BIT_SIZE
Character	LEN
Numeric	DIGITS, EPSILON, EXPONENT, FRACTION, HUGE, KIND, MAXEXPONENT, MINEXPONENT, NEAREST, PRECISION, RADIX, RANGE, RRSPACING, SCALE, SET_EXPONENT, SELECTED_INT_KIND, SELECTED_REAL_KIND, SPACING, TINY
Array	ALLOCATED, LBOUND, SHAPE, SIZE, UBOUND
Pointer	ASSOCIATED, NULL
Argument present	PRESENT

Transferencia y conversión de funciones

Transferencia y conversión de funciones

- Estas funciones convierten datos de un tipo y KIND a otro tipo y otro KIND.

Transferencia y conversión	ACHAR, AIMAG, AINT, ANINT, CHAR, CMPLX, CONJG, DBLE, IACHAR, IBITS, ICHAR, INT, LOGICAL, NINT, REAL, TRANSFER.
---	--

Funciones computacionales

Funciones computacionales

- Estas funciones convierten datos de un tipo y KIND a otro tipo y otro KIND.

Transferencia y conversión	ACHAR, AIMAG, AINT, ANINT, CHAR, CMPLX, CONJG, DBLE, IACHAR, IBITS, ICHAR, INT, LOGICAL, NINT, REAL, TRANSFER.
---	--