

Programación en FORTRAN

Nivel Básico - Sesión 2

Martin Josemaría Vuelta Rojas

5 de enero de 2018

SoftButterfly

1. Variables y tipos de datos
2. Operaciones elementales
3. Estructuras de control
4. Instrucciones básicas de lectura y escritura de datos

Variables y tipos de datos

- Un lenguaje de programación permite identificar los datos que se manipulan y almacenan en grandes cantidades en un ordenador.

- Un lenguaje de programación permite identificar los datos que se manipulan y almacenan en grandes cantidades en un ordenador.

Variables

- Un lenguaje de programación permite identificar los datos que se manipulan y almacenan en grandes cantidades en un ordenador.

Variables

- Una variable es un objeto que representa un tipo de dato, susceptible de modificarse, nombrado por cadenas de caracteres.

Tipos de datos

Tipos de datos

1. **character:** cadena de uno o varios caracteres.

Tipos de datos

1. **character:** cadena de uno o varios caracteres.
2. **integer:** números enteros, positivos o negativos.

Tipos de datos

1. **character:** cadena de uno o varios caracteres.
2. **integer:** números enteros, positivos o negativos.
3. **logical:** valores lógicos o booleanos, es decir, toman uno de los dos valores, `.true.` (verdadero) o `.false.` (falso).

Tipos de datos

1. **character:** cadena de uno o varios caracteres.
2. **integer:** números enteros, positivos o negativos.
3. **logical:** valores lógicos o booleanos, es decir, toman uno de los dos valores, `.true.` (verdadero) o `.false.` (falso).
4. **real:** números reales, positivos o negativos.

Tipos de datos

1. **character:** cadena de uno o varios caracteres.
2. **integer:** números enteros, positivos o negativos.
3. **logical:** valores lógicos o booleanos, es decir, toman uno de los dos valores, `.true.` (verdadero) o `.false.` (falso).
4. **real:** números reales, positivos o negativos.
5. **complex:** números complejos, compuestos de una parte real y otra imaginaria, ambas de tipo real.

Declaración de variables

Declaración de variables

- La declaración de una o más variables del mismo tipo está dada por la sintaxis

`<tipo> , [<atributo(s)>] [::] <variable(s)> [= <valor>]`

Declaración de variables

- La declaración de una o más variables del mismo tipo está dada por la sintaxis

`<tipo> , [<atributo(s)>] [::] <variable(s)> [= <valor>]`

- Algunos atributos son:
parameter, save, pointer, target, allocatable, dimension, public, private, external, intrinsic, optional.

Declaración de variables

- La declaración de una o más variables del mismo tipo está dada por la sintaxis

`<tipo> , [<atributo(s)>] [::] <variable(s)> [= <valor>]`

- Algunos atributos son:
parameter, save, pointer, target, allocatable, dimension, public, private, external, intrinsic, optional.

```
1 CHARACTER(len= 4), PARAMETER :: prompt = ">>> "  
2 CHARACTER(len= *), PARAMETER :: message = "Ingresa tu primer nombre [máx 20 car]:"
```


Declaración de variables

- La declaración de una o más variables del mismo tipo está dada por la sintaxis

`<tipo> , [<atributo(s)>] [::] <variable(s)> [= <valor>]`

- Algunos atributos son:
parameter, save, pointer, target, allocatable, dimension, public, private, external, intrinsic, optional.

```
1 CHARACTER(len= 4), PARAMETER :: prompt = ">>> "  
2 CHARACTER(len= *), PARAMETER :: message = "Ingresa tu primer nombre [máx 20 car]:"
```

Véase strings.f95

Declaración de constantes

Declaración de constantes

- Si se requiere que una variable que tome un valor definido no susceptible de cambio, se utiliza el atributo `parameter`.

```
1 CHARACTER, PARAMETER :: NewLine = CHAR(10)
```

Declaración de constantes

- Si se requiere que una variable que tome un valor definido no susceptible de cambio, se utiliza el atributo `parameter`.

```
1 CHARACTER, PARAMETER :: NewLine = CHAR(10)
```

Véase `strings.f95`

Declaración de constantes

- Si se requiere que una variable que tome un valor definido no susceptible de cambio, se utiliza el atributo `parameter`.

```
1 CHARACTER, PARAMETER :: NewLine = CHAR(10)
```

Véase `strings.f95`

- Las variables pueden ser definidas en función de constantes mediante el atributo `parameter`.

Declaración de cadenas de caracteres

Declaración de cadenas de caracteres

- La declaración de una variable de tipo character está dada por la sintaxis

CHARACTER[(len=<long>)], [<atributos>][:]<variables>[=<valor>]

Declaración de cadenas de caracteres

- La declaración de una variable de tipo character está dada por la sintaxis

CHARACTER[(len=<long>)], [<atributos>][:]<variables>[=<valor>]

```
1 CHARACTER(kind=ascii, len=26) :: Alphabet
2 CHARACTER(kind= ucs4, len=30) :: HelloWorld
```

Véase `kind_character.f95`

Cadenas de caracteres y valores lógicos

Declaración de cadenas de caracteres

- La declaración de una variable de tipo character está dada por la sintaxis

CHARACTER[(len=<long>)], [<atributos>][:]<variables>[=<valor>]

```
1 CHARACTER(kind=ascii, len=26) :: Alphabet
2 CHARACTER(kind= ucs4, len=30) :: HelloWorld
```

Véase `kind_character.f95`

Declaración de valores lógicos

Cadenas de caracteres y valores lógicos

Declaración de cadenas de caracteres

- La declaración de una variable de tipo character está dada por la sintaxis

CHARACTER[(len=<long>)], [<atributos>][:]<variables>[=<valor>]

```
1 CHARACTER(kind=ascii, len=26) :: Alphabet
2 CHARACTER(kind= ucs4, len=30) :: HelloWorld
```

Véase kind_character.f95

Declaración de valores lógicos

- La declaración de una variable lógica está dada por
LOGICAL <variable(s)>

Tipos de enteros

Tipos de enteros

- La representación de valores enteros se declara con `INTEGER`.

Tipos de enteros

- La representación de valores enteros se declara con `INTEGER`.
- Los valores pueden ser guardados usualmente con precisión simple, doble o cuádruple.

Tipos de enteros

- La representación de valores enteros se declara con `INTEGER`.
- Los valores pueden ser guardados usualmente con precisión simple, doble o cuádruple.

```
1  INTEGER, PARAMETER :: K02 = SELECTED_INT_KIND(2)
2  INTEGER, PARAMETER :: K04 = SELECTED_INT_KIND(4)
3  INTEGER, PARAMETER :: K08 = SELECTED_INT_KIND(8)
4  INTEGER, PARAMETER :: K16 = SELECTED_INT_KIND(16)
5
6  INTEGER(kind=K02) :: I02
7  INTEGER(kind=K04) :: I04
8  INTEGER(kind=K08) :: I08
9  INTEGER(kind=K16) :: I16
```

Véase `kind_integers.f95`

Tipos de reales

Tipos de reales

- La representación de número reales se declara con `REAL` y puede ser de precisión estándar o simple precisión (`sp`) y de precisión superior, doble (`dp`) o cuádruple (`qp`) precisión en adelante.

Tipos de reales

- La representación de número reales se declara con `REAL` y puede ser de precisión estándar o simple precisión (`sp`) y de precisión superior, doble (`dp`) o cuádruple (`qp`) precisión en adelante.
- La sintaxis para el tipo real es
`REAL (kind=<np>)`

Tipos de reales

- La representación de número reales se declara con `REAL` y puede ser de precisión estándar o simple precisión (`sp`) y de precisión superior, doble (`dp`) o cuádruple (`qp`) precisión en adelante.
- La sintaxis para el tipo real es

`REAL (kind=<np>)`

```
1  REAL(kind=p04) :: X04
2  REAL(kind=p08) :: X08
3  REAL(kind=p16) :: X16
4  REAL(kind=p32) :: X32
```

Véase `kind_real.f95`

- La notación científica para los reales viene dada por los identificadores "e" (sp), "d" (dp) y "q" (qp).

- La notación científica para los reales viene dada por los identificadores "e" (sp), "d" (dp) y "q" (qp).

```
1  REAL(kind=4)  :: x = 2.e0      !simple precisión
2  REAL(kind=8)  :: y = 4.d-6     !doble precisión
3  REAL(kind=16) :: z = -8.q-1000 !cuadruple precisión
```

- La notación científica para los reales viene dada por los identificadores "e" (sp), "d" (dp) y "q" (qp).

```
1  REAL(kind=4)  :: x = 2.e0      !simple precisión
2  REAL(kind=8)  :: y = 4.d-6     !doble precisión
3  REAL(kind=16) :: z = -8.q-1000 !cuadruple precisión
```

- Sin embargo, en muchos casos es útil predefinir la clase kind al cambiar de tipo de real variando el valor de np, como por ejemplo

- La notación científica para los reales viene dada por los identificadores "e" (sp), "d" (dp) y "q" (qp).

```
1 REAL(kind=4)  :: x = 2.e0      !simple precisión
2 REAL(kind=8)  :: y = 4.d-6     !doble precisión
3 REAL(kind=16) :: z = -8.q-1000 !cuadruple precisión
```

- Sin embargo, en muchos casos es útil predefinir la clase kind al cambiar de tipo de real variando el valor de np, como por ejemplo

```
1 INTEGER, PARAMETER :: np=16      !np = 4, 8 o 16
2 REAL (kind=np)  :: X = 2.e-10_np !e = e, d o q
```

Tipos de complejos

Tipos de complejos

- La representación de números complejos se declara con `COMPLEX`, e igualmente que los reales, puede presentar una precisión simple, doble o cuádruple.

Tipos de complejos

- La representación de números complejos se declara con `COMPLEX`, e igualmente que los reales, puede presentar una precisión simple, doble o cuádruple.

```
1  INTEGER :: re = 25
2  REAL(kind= 4) :: im04 = 3.141592
3  REAL(kind= 8) :: im08 = 3.141592
4  REAL(kind=10) :: im10 = 3.141592
5  REAL(kind=16) :: im16 = 3.141592
6  COMPLEX(kind=16) :: z16 = (25, 3.141592)
```

Véase `complex.f95`

Tipos de complejos

- La representación de números complejos se declara con `COMPLEX`, e igualmente que los reales, puede presentar una precisión simple, doble o cuádruple.

```
1  INTEGER :: re = 25
2  REAL(kind= 4) :: im04 = 3.141592
3  REAL(kind= 8) :: im08 = 3.141592
4  REAL(kind=10) :: im10 = 3.141592
5  REAL(kind=16) :: im16 = 3.141592
6  COMPLEX(kind=16) :: z16 = (25, 3.141592)
```

Véase `complex.f95`

- La notación científica y las declaraciones empleando `kind` siguen las mismas reglas.

- Si existen variables que no han sido definidas, el tipo de variable depende de la letra inicial. Por lo cual...

La cláusula Implicit

- Si existen variables que no han sido definidas, el tipo de variable depende de la letra inicial. Por lo cual...
 - i, j, k, l, m, n representan variables enteras.

La cláusula Implicit

- Si existen variables que no han sido definidas, el tipo de variable depende de la letra inicial. Por lo cual...
 - i, j, k, l, m, n representan variables enteras.
 - las demás letras representan variables reales de precisión simple.

La cláusula Implicit

- Si existen variables que no han sido definidas, el tipo de variable depende de la letra inicial. Por lo cual...
 - i, j, k, l, m, n representan variables enteras.
 - las demás letras representan variables reales de precisión simple.
- El carácter implícito puede ser modificado empleando la instrucción IMPLICIT bajo la siguiente sintaxis:
IMPLICIT <tipo> (<caracter(es)_1>, ..., <caracter(es)_k>)

La cláusula Implicit

- Si existen variables que no han sido definidas, el tipo de variable depende de la letra inicial. Por lo cual...
 - i, j, k, l, m, n representan variables enteras.
 - las demás letras representan variables reales de precisión simple.
- El carácter implícito puede ser modificado empleando la instrucción IMPLICIT bajo la siguiente sintaxis:
IMPLICIT <tipo> (<caracter(es)_1>, ..., <caracter(es)_k>)
- Debido a que el compilador puede reconocer variables por defecto, se recomienda emplear la instrucción IMPLICIT NONE, especificando todas las variables y evitando errores con el código fuente.

La cláusula Implicit

- Disponibles a partir de Fortran90 (solo se podía trabajar con INTEGER, REAL, LOGICAL Y CHARACTER).

La cláusula Implicit

- Disponibles a partir de Fortran90 (solo se podía trabajar con INTEGER, REAL, LOGICAL Y CHARACTER).
- Son expresiones derivadas de datos, es decir, creadas a partir de los tipos intrínsecos.

La cláusula Implicit

- Disponibles a partir de Fortran90 (solo se podía trabajar con INTEGER, REAL, LOGICAL Y CHARACTER).
- Son expresiones derivadas de datos, es decir, creadas a partir de los tipos intrínsecos.
- La sintaxis para declarar un nuevo tipo es de la forma

La cláusula Implicit

- Disponibles a partir de Fortran90 (solo se podía trabajar con INTEGER, REAL, LOGICAL Y CHARACTER).
- Son expresiones derivadas de datos, es decir, creadas a partir de los tipos intrínsecos.
- La sintaxis para declarar un nuevo tipo es de la forma

```
1  type <Nombre del nuevo tipo>
2      <Instruccion declaracion variable(s)>
3      <Instruccion declaracion variable(s)>
4      :
5      <Instruccion declaracion variable(s)>
6  end type
```

La cláusula Implicit

- Disponibles a partir de Fortran90 (solo se podía trabajar con INTEGER, REAL, LOGICAL Y CHARACTER).
- Son expresiones derivadas de datos, es decir, creadas a partir de los tipos intrínsecos.
- La sintaxis para declarar un nuevo tipo es de la forma

```
1  type <Nombre del nuevo tipo>
2      <Instruccion declaracion variable(s)>
3      <Instruccion declaracion variable(s)>
4      :
5      <Instruccion declaracion variable(s)>
6  end type
```

- La sintaxis para declarar una o más variables de un tipo derivado se expresa de la forma
type (<nombre tipo>), [<atributos>]::<var_1,var_2...>=[<valor>]

Operaciones elementales

Reglas generales

Reglas generales

- Las operaciones siguen el orden tradicional de las operaciones matemáticas, es decir, primero los términos entre paréntesis, exponentes, multiplicación y adición.

Reglas generales

- Las operaciones siguen el orden tradicional de las operaciones matemáticas, es decir, primero los términos entre paréntesis, exponentes, multiplicación y adición.
- El uso del símbolo $=$ en el lenguaje Fortran tiene el sentido de asignación mientras que en el uso matemático tiene sentido de igualdad.

Reglas generales

- Las operaciones siguen el orden tradicional de las operaciones matemáticas, es decir, primero los términos entre paréntesis, exponentes, multiplicación y adición.
- El uso del símbolo $=$ en el lenguaje Fortran tiene el sentido de asignación mientras que en el uso matemático tiene sentido de igualdad.
- La asignación de una variable tiene la sintaxis
$$\langle \text{variable} \rangle = \langle \text{expresión} \rangle$$

- Están permitidas las operaciones entre valores tipo INTEGER, REAL y COMPLEX.

- Están permitidas las operaciones entre valores tipo INTEGER, REAL y COMPLEX.
- Las operaciones están dadas por adición (+), sustracción (-), multiplicación (*), división (/) y potenciación (**).

- Están permitidas las operaciones entre valores tipo INTEGER, REAL y COMPLEX.
- Las operaciones están dadas por adición (+), sustracción (-), multiplicación (*), división (/) y potenciación (**).
- Los operandos del mismo tipo y clase resultan en otro del mismo tipo y clase.

Operaciones de tipo INTEGER

Operaciones de tipo INTEGER

- Las operaciones de tipo INTEGER manejan números enteros dentro de un rango en \mathbb{Z} .

Operaciones de tipo INTEGER

- Las operaciones de tipo INTEGER manejan números enteros dentro de un rango en \mathbb{Z} .
- La división se obtiene con un resto; es decir

$$\frac{x}{y} = z \iff |x| = |z| \cdot |y| + \text{resto}$$

Operaciones de tipo INTEGER

- Las operaciones de tipo INTEGER manejan números enteros dentro de un rango en \mathbb{Z} .

- La división se obtiene con un resto; es decir

$$\frac{x}{y} = z \iff |x| = |z| \cdot |y| + \text{resto}$$

- La potenciación depende del tipo de variable del exponente.

$$x ** n = \begin{cases} \underbrace{x * x * \dots * x}_{n \text{ veces}} & \text{si } n > 0 \\ \frac{1}{x ** (-n)} & \text{si } n < 0 \\ 1 & \text{si } x = 0 \end{cases}$$

Conversión de tipos

Conversión de tipos

- Los enteros son convertidos en reales o complejos.

Conversión de tipos

- Los enteros son convertidos en reales o complejos.
- Los reales son convertidos en complejos.

Conversión de tipos

- Los enteros son convertidos en reales o complejos.
- Los reales son convertidos en complejos.
- Los reales o complejos son convertidos en la clase (kind) más alta.

Conversión de tipos

- Los enteros son convertidos en reales o complejos.
- Los reales son convertidos en complejos.
- Los reales o complejos son convertidos en la clase (kind) más alta.
- Al asignar valores ($=$), la parte derecha se evalúa en el tipo y clase correspondiente, luego es convertida al del tipo y clase de la variable al lado izquierdo.

Conversión de tipos

- Los enteros son convertidos en reales o complejos.
- Los reales son convertidos en complejos.
- Los reales o complejos son convertidos en la clase (kind) más alta.
- Al asignar valores ($=$), la parte derecha se evalúa en el tipo y clase correspondiente, luego es convertida al del tipo y clase de la variable al lado izquierdo.

```
1  INTEGER          :: n, m
2  REAL             :: a, b
3  REAL(kind=8)     :: x, y
4  COMPLEX          :: c
5  COMPLEX(kind=8)  :: z
6  :
7  a = (x*(n**c))/z
8  :
```

Conversiones de tipo más significativas

Conversión	Mecanismo de Conversión
$x = n$	$x = n$
$x = a$	$x = a$
$n = x$	$n = \begin{cases} m & \text{si } m \leq x \leq m+1 \text{ y } x \geq 0 \\ -m & \text{si } m \leq -x \leq m+1 \text{ y } x < 0 \end{cases}$
$a = x$	$a = \text{round}(x)$
$a = c$	$a = \mathbb{R}(z)$
$z = x$	$z = (x, 0)$

Operaciones de comparación

- La operación de comparación se expresa de la forma
 $\langle \text{expresión_1} \rangle \langle \text{operador} \rangle \langle \text{expresión_2} \rangle$

Operaciones de comparación

- La operación de comparación se expresa de la forma
 $\langle \text{expresión_1} \rangle \langle \text{operador} \rangle \langle \text{expresión_2} \rangle$
- Para variables de tipo COMPLEX solo son válidos los operadores
`==` y `/=`.

Operaciones de comparación

- La operación de comparación se expresa de la forma
 $\langle \text{expresión_1} \rangle \langle \text{operador} \rangle \langle \text{expresión_2} \rangle$
- Para variables de tipo COMPLEX solo son válidos los operadores `==` y `/=`.
- Los operadores de comparación son variables de tipo LOGICAL.

Operaciones de comparación

- La operación de comparación se expresa de la forma
 <expresión_1> <operador> <expresión_2>
- Para variables de tipo COMPLEX solo son válidos los operadores
 == y /= .
- Los operadores de comparación son variables de tipo LOGICAL.

Fortran 90	Fortran 77	Significado
==	.eq.	es igual a
/=	.ne.	no es igual a
>	.gt.	es estrictamente mayor a
>=	.ge.	es mayor o igual a
<	.lt.	es estrictamente menor a
<=	.le.	es menor o igual a

Operadores de comparación

- La operación lógica se expresa de la forma
 $\langle \text{expresión_1} \rangle \langle \text{operador} \rangle \langle \text{expresión_2} \rangle$

Operaciones lógicas

- La operación lógica se expresa de la forma
 $\langle \text{expresión_1} \rangle \langle \text{operador} \rangle \langle \text{expresión_2} \rangle$
- Las operaciones lógicas se evalúan luego de las operaciones de comparación, de izquierda a derecha.

Operaciones lógicas

- La operación lógica se expresa de la forma
 $\langle \text{expresión}_1 \rangle \langle \text{operador} \rangle \langle \text{expresión}_2 \rangle$
- Las operaciones lógicas se evalúan luego de las operaciones de comparación, de izquierda a derecha.

Operador	Significado
.not.	No
.and.	y
.or.	o
.eqv.	equivalente
.neqv.	no equivalente

Operadores lógicos

Funciones intrínsecas *built in functions*

- Funciones predefinidas independientes del compilador.

Funciones intrínsecas *built in functions*

- Funciones predefinidas independientes del compilador.
- Una función intrínseca monoargumental se expresa de la forma
 <función> (<argumento>)

Funciones intrínsecas *built in functions*

- Funciones predefinidas independientes del compilador.
- Una función intrínseca monoargumental se expresa de la forma
 <función> (<argumento>)
- Las funciones intrínsecas pueden no tener argumento o tener varios.

Funciones intrínsecas *built in functions*

- Funciones predefinidas independientes del compilador.
- Una función intrínseca monoargumental se expresa de la forma
 <función> (<argumento>)
- Las funciones intrínsecas pueden no tener argumento o tener varios.
- Las funciones biargumentales más destacadas son la función CMPLX y la función MOD.

Funciones intrínsecas *built in functions*

- Funciones predefinidas independientes del compilador.
- Una función intrínseca monoargumental se expresa de la forma
 <función> (<argumento>)
- Las funciones intrínsecas pueden no tener argumento o tener varios.
- Las funciones biargumentales más destacadas son la función CMPLX y la función MOD.
- CMPLX asigna un valor complejo a partir de valores reales.

```
1  !Sea z = (x,y) una variable compleja
2  CMPLX (<expres_1>, <expres_2>) !expres son de tipo REAL
```

Funciones intrínsecas *built in functions*

- Funciones predefinidas independientes del compilador.
- Una función intrínseca monoargumental se expresa de la forma
 $\langle \text{función} \rangle (\langle \text{argumento} \rangle)$
- Las funciones intrínsecas pueden no tener argumento o tener varios.
- Las funciones biargumentales más destacadas son la función CMPLX y la función MOD.
- CMPLX asigna un valor complejo a partir de valores reales.

```
1  !Sea z = (x,y) una variable compleja
2  CMPLX (<expres_1>, <expres_2>) !expres son de tipo REAL
```

- MOD es el valor que corresponde al resto de una división, es decir

$$n = n/m + MOD(n, m)$$

```
1  !Sea n de tipo INTEGER o REAL y m del mismo tipo y distinto de 0
2  MOD (<expres_1>, <expres_2>) !expres son de tipo INTEGER o REAL
```

Funciones intrínsecas *built in functions*

Función	Argumento	Resultado	Descripción
abs	real, complex, integer	real, integer	$ x , z , n $
sqrt	real, complex	real, complex	$(\sqrt{x}, x \geq 0), (\sqrt{z}, z \in \mathbb{C})$
int	real	integer	Parte entera de un real x
fraccion	real	real	Parte fraccional de un real x
real	complex	real	$\operatorname{Re}(z), z \in \mathbb{C}$
aimag	complex	real	$\operatorname{Im}(z), z \in \mathbb{C}$
conjg	complex	complex	$\bar{z}, z \in \mathbb{C}$
cos	real complex	real complex	$(\cos x, x \in \mathbb{R}), (\cos z, z \in \mathbb{C})$
sin	real complex	real complex	$(\sin x, x \in \mathbb{R}), (\sin z, z \in \mathbb{C})$
tan	real complex	real complex	$(\tan x, x \in \mathbb{R}), (\tan z, z \in \mathbb{C})$
acos	real complex	real complex	$(\arccos x, x \in \mathbb{R}), (\arccos z, z \in \mathbb{C})$
asin	real complex	real complex	$(\arcsin x, x \in \mathbb{R}), (\arcsin z, z \in \mathbb{C})$
atan	real complex	real complex	$(\arctan x, x \in \mathbb{R}), (\arctan z, z \in \mathbb{C})$
exp	real complex	real complex	$(\exp x, x \in \mathbb{R}), (\exp z, z \in \mathbb{C})$
log	real complex	real complex	$(\log x, x > 0), (\log z, z \in \mathbb{C}, z \neq 0)$
log10	real	real	$(\log_{10} x, x > 0)$

Funciones intrínsecas más relevantes

- La asignación del tipo CHARACTER es de la forma
 $\text{<variable> = <expresión>}$

- La asignación del tipo CHARACTER es de la forma

$\langle \text{variable} \rangle = \langle \text{expresión} \rangle$

donde $\langle \text{variable} \rangle$ y $\langle \text{expresión} \rangle$ son de tipo CHARACTER y pueden tener longitud ($len = n$) o ($len = m$) respectivamente; $n, m \in \mathbb{Z}^+$

- La asignación del tipo CHARACTER es de la forma

$\langle \text{variable} \rangle = \langle \text{expresión} \rangle$

donde $\langle \text{variable} \rangle$ y $\langle \text{expresión} \rangle$ son de tipo CHARACTER y pueden tener longitud ($\text{len} = n$) o ($\text{len} = m$) respectivamente; $n, m \in \mathbb{Z}^+$

- Si $n \leq m$, se asigna a la $\langle \text{variable} \rangle$ los n primeros caracteres de la $\langle \text{expresión} \rangle$ de izquierda a derecha, eliminando la diferencia $m - n$.

- La asignación del tipo CHARACTER es de la forma

$\langle \text{variable} \rangle = \langle \text{expresión} \rangle$

donde $\langle \text{variable} \rangle$ y $\langle \text{expresión} \rangle$ son de tipo CHARACTER y pueden tener longitud ($len = n$) o ($len = m$) respectivamente; $n, m \in \mathbb{Z}^+$

- Si $n \leq m$, se asigna a la $\langle \text{variable} \rangle$ los n primeros caracteres de la $\langle \text{expresión} \rangle$ de izquierda a derecha, eliminando la diferencia $m - n$.
- Si $n > m$, se asigna a la $\langle \text{variable} \rangle$ de izquierda a derecha la cadena de caracteres de $\langle \text{expresión} \rangle$, completando los últimos $n - m$ caracteres de la derecha con espacios.

Operadores binarios

Operadores binarios

- Concatenación: dado por el operador `//`.

Operadores binarios

- Concatenación: dado por el operador `//`.
- Comparación: dado por los operadores `==` y `/=`.

Operadores binarios

- Concatenación: dado por el operador `//`.
- Comparación: dado por los operadores `==` y `/=`.

Partes de cadenas de caracteres

Operadores binarios

- Concatenación: dado por el operador `//`.
- Comparación: dado por los operadores `==` y `/=`.

Partes de cadenas de caracteres

- Si `<expresión>` representa una cadena de caracteres $c_1 \dots c_k \dots c_l \dots c_n$ de n caracteres, con $1 \leq k \leq l \leq n$, podemos obtener lo siguiente:

```
1 <expresion> (k:l) !cadena "c_k...c_l"
2 <expresion> (:l) !cadena "c_1...c_k...c_l"
3 <expresion> (k:) !cadena "c_k...c_l...c_n"
```

Operaciones con caracteres

Operaciones sobre cadenas de caracteres

Operaciones sobre cadenas de caracteres

- La sintaxis para instrucciones de tipo CHARACTER es
 <instrucción> (<expresión>)

Operaciones con caracteres

Operaciones sobre cadenas de caracteres

- La sintaxis para instrucciones de tipo CHARACTER es
 <instrucción> (<expresión>)

Instrucción	Resultado	Descripción
len	integer	Define la longitud una cadena de caracteres
trim	character	Suprime los espacios del final de una cadena
adjustl	character	Si hay espacios al inicio de una cadena, los suprime desplazando el resto de la cadena a la izquierda.

Instrucciones de cadenas de caracteres

La instrucción INDEX

La instrucción INDEX

- La instrucción INDEX proporciona como resultado un valor tipo INTEGER a partir de una cadena de tipo CHARACTER.

La instrucción INDEX

- La instrucción INDEX proporciona como resultado un valor tipo INTEGER a partir de una cadena de tipo CHARACTER.
- La sintaxis es de la forma
INDEX (<expresión_1>, <expresión_2>).

La instrucción INDEX

- La instrucción INDEX proporciona como resultado un valor tipo INTEGER a partir de una cadena de tipo CHARACTER.
- La sintaxis es de la forma
INDEX (<expresión_1>, <expresión_2>).
- Si <expresión_2> es parte de <expresión_1> entonces indica su primera posición; sino el valor es 0.

La instrucción INDEX

- La instrucción INDEX proporciona como resultado un valor tipo INTEGER a partir de una cadena de tipo CHARACTER.
- La sintaxis es de la forma
INDEX (<expresión_1>, <expresión_2>).
- Si <expresión_2> es parte de <expresión_1> entonces indica su primera posición; sino el valor es 0.

```
1  INDEX ('Fortran', 'tra') ! INDEX dará 4
```

Estructuras de control

- Permiten la ejecución secuencial de una o más instrucciones bajo ciertas condiciones.

- Permiten la ejecución secuencial de una o más instrucciones bajo ciertas condiciones.
- Emplea variables y expresiones comparativas y lógicas. (`==`, `/=`, ..., `.AND.`, `.OR.` ...)

- Permiten la ejecución secuencial de una o más instrucciones bajo ciertas condiciones.
- Emplea variables y expresiones comparativas y lógicas. (`==`, `/=`, ..., `.AND.`, `.OR.` ...)
- Permite la construcción de código en bloques. (`IF`, `THEN`, `ENDIF`)

Condiciones: IF

```
1  PROGRAM FizzBuzz
2  IMPLICIT NONE
3  INTEGER :: I
4  CHARACTER(len=40) :: CharacterI
5
6  WRITE(*, '(A)') "Ingrese un número entero:"
7  READ(*, *) I
8
9  IF (mod(I,3) == 0 .AND. mod(I,5) == 0) THEN
10     WRITE(*, '(A)') 'FizzBuzz'
11 ELSE IF (mod(I,3) == 0) THEN
12     WRITE(*, '(A)') 'Fizz'
13 ELSE IF (mod(I,5) == 0) THEN
14     WRITE(*, '(A)') 'Buzz'
15 ELSE
16     WRITE(CharacterI, '(I10)') I
17     WRITE(*, '(A)') ADJUSTL(CharacterI)
18 END IF
19 END PROGRAM FizzBuzz
```

Repeticiones: DO, DO WHILE, DO

- Permiten la ejecución repetitiva de una o más instrucciones bajo un número determinado de veces o una expresión lógica.

Repeticiones: DO, DO WHILE, DO

- Permiten la ejecución repetitiva de una o más instrucciones bajo un número determinado de veces o una expresión lógica.
- Entre estas estructuras destacan

Repeticiones: DO, DO WHILE, DO

- Permiten la ejecución repetitiva de una o más instrucciones bajo un número determinado de veces o una expresión lógica.
- Entre estas estructuras destacan

DO FROM X TO Y? (desde-hasta)

```
1 DO <variable> = <inicio>, <fin>, <incremento>  
2     bloque de declaraciones  
3 END DO
```

Repeticiones: DO, DO WHILE, DO

- Permiten la ejecución repetitiva de una o más instrucciones bajo un número determinado de veces o una expresión lógica.
- Entre estas estructuras destacan

DO FROM X TO Y? (desde-hasta)

```
1 DO <variable> = <inicio>, <fin>, <incremento>  
2     bloque de declaraciones  
3 END DO
```

DO WHILE (mientras)

```
1 DO WHILE <(expresiones logicas)>  
2     bloque de declaraciones  
3 END DO
```

Repeticiones: DO, DO WHILE, DO

- Permiten la ejecución repetitiva de una o más instrucciones bajo un número determinado de veces o una expresión lógica.
- Entre estas estructuras destacan

DO FROM X TO Y? (desde-hasta)

```
1 DO <variable> = <inicio>, <fin>, <incremento>
2   bloque de declaraciones
3 END DO
```

DO WHILE (mientras)

```
1 DO WHILE <(expresiones logicas)>
2   bloque de declaraciones
3 END DO
```

DO? (repetir-hasta)

```
1 DO
2   bloque de declaraciones
3   IF <expresion logica> EXIT
4 END DO
```


Repeticiones: DO, DO WHILE, DO

```
1  ! Do finito
2  ! Igual que un bucle FOR en otros lenguajes
3      DO I = 3, N, 1
4          Fibn = Fib2 + Fib1
5
6          IF (Fibn < 0) THEN
7              WRITE(*, '(I5,2X,A)') I, "Overflow error!"
8              EXIT
9          END IF
10
11         WRITE(strFibn, '(I64)') Fibn
12         WRITE(*, '(I5,2X,A)') I, ADJUSTL(strFibn)
13
14         Fib1 = Fib2
15         Fib2 = Fibn
16     END DO
```

Véase fibonacci.f95

Selección: SWITCH CASE

- Permite la ejecución selectiva de una o más instrucciones o alternativas en función de expresiones lógicas.

Selección: SWITCH CASE

- Permite la ejecución selectiva de una o más instrucciones o alternativas en función de expresiones lógicas.
- Se presenta una estructura simple (IF), doble (IF-ELSE) y multialternativa (SELECT CASE).

Selección: SWITCH CASE

- Permite la ejecución selectiva de una o más instrucciones o alternativas en función de expresiones lógicas.
- Se presenta una estructura simple (IF), doble (IF-ELSE) y multialternativa (SELECT CASE).

```
1 Selector: SELECT CASE (Operator)
2     CASE ('+') Selector
3         C = A + B
4     CASE ('-') Selector
5         C = A - B
6     CASE ('/') Selector
7         C = A / B
8     CASE ('*') Selector
9         C = A * B
10    CASE DEFAULT Selector
11        EXIT
12    END SELECT Selector
```

Véase calculator.f95

The infamous TO GO

- Permite la ramificación o división de una instrucción hacia otra parte específica del código.

The infamous TO GO

- Permite la ramificación o división de una instrucción hacia otra parte específica del código.
- Puede emplearse de forma lógica-condicional junto con IF.

The infamous TO GO

- Permite la ramificación o división de una instrucción hacia otra parte específica del código.
- Puede emplearse de forma lógica-condicional junto con IF.

```
1      PROGRAM GOTOER
2      1  WRITE(*, *) 0
3      GOTO 2
4      9  WRITE(*, *) 8
5      GOTO 10
6      .
7      .
8      7  WRITE(*, *) 6
9      GOTO 8
10     10 WRITE(*, *) 9
11     END PROGRAM GOTOER
```

Véase gotoer.f

Instrucciones básicas de lectura y escritura de datos

Escritura sobre la pantalla

Escritura sobre la pantalla

- Se utilizan dos instrucciones equivalentes: PRINT y WRITE.

Escritura sobre la pantalla

- Se utilizan dos instrucciones equivalentes: PRINT y WRITE.
- La sintaxis es de la forma

Escritura sobre la pantalla

- Se utilizan dos instrucciones equivalentes: PRINT y WRITE.
- La sintaxis es de la forma

```
1  print*, <expresion_1>, ..., <expresion_n>  
2  write(*,*) <expresion_1>, ..., <expresion_n>  
3  write(6,*) <expresion_1>, ..., <expresion_n>
```

Escritura sobre la pantalla

- Se utilizan dos instrucciones equivalentes: PRINT y WRITE.
- La sintaxis es de la forma

```
1  print*, <expresion_1>, ..., <expresion_n>  
2  write(*,*) <expresion_1>, ..., <expresion_n>  
3  write(6,*) <expresion_1>, ..., <expresion_n>
```

- El caracter * permite que la computadora predetermine un formato a los datos.

Escritura sobre la pantalla

- Se utilizan dos instrucciones equivalentes: PRINT y WRITE.
- La sintaxis es de la forma

```
1 print*, <expresion_1>, ..., <expresion_n>  
2 write(*,*) <expresion_1>, ..., <expresion_n>  
3 write(6,*) <expresion_1>, ..., <expresion_n>
```

- El caracter * permite que la computadora predetermine un formato a los datos.
- Cada expresión debe ser de tipo INTEGER, REAL, COMPLEX, CHARACTER o LOGICAL.

Lectura de datos utilizando el teclado? (entrada de dato por el usuario)

Lectura de datos utilizando el teclado? (entrada de dato por el usuario)

- Se utiliza la instrucción READ

Lectura de datos utilizando el teclado? (entrada de dato por el usuario)

- Se utiliza la instrucción READ
- La sintaxis es de la forma

Lectura de datos utilizando el teclado? (entrada de dato por el usuario)

- Se utiliza la instrucción READ
- La sintaxis es de la forma

```
1 READ*, <variable_1>, ..., <variable_n>  
2 READ(*,*) <variable_1>, ..., <variable_n>  
3 READ(5,*) <variable_1>, ..., <variable_n>
```

Lectura de datos utilizando el teclado? (entrada de dato por el usuario)

- Se utiliza la instrucción READ
- La sintaxis es de la forma

```
1 READ*, <variable_1>, ..., <variable_n>  
2 READ(*,*) <variable_1>, ..., <variable_n>  
3 READ(5,*) <variable_1>, ..., <variable_n>
```

- Cada variable debe ser de tipo INTEGER, REAL, COMPLEX, CHARACTER o LOGICAL.

Lectura de datos utilizando el teclado? (entrada de dato por el usuario)

- Se utiliza la instrucción READ
- La sintaxis es de la forma

```
1 READ*, <variable_1>, ..., <variable_n>  
2 READ(*,*) <variable_1>, ..., <variable_n>  
3 READ(5,*) <variable_1>, ..., <variable_n>
```

- Cada variable debe ser de tipo INTEGER, REAL, COMPLEX, CHARACTER o LOGICAL.
- En caso de las cadenas de caracteres es necesario delimitarlas con comillas.

Lectura de datos utilizando el teclado? (entrada de dato por el usuario) 2003

Lectura de datos utilizando el teclado? (entrada de dato por el usuario) 2003

-