

# Introdução ao Fortran 90 - 1

Alexandre Diehl

Departamento de Física – UFPel

## Programa ou Código-fonte

Programa ou **código-fonte** é um conjunto de instruções, também chamadas de comandos, escritas de acordo com as regras sintáticas da linguagem.

- O programa deve ser escrito num editor de texto: **emacs**, **gedit**, **nano**, etc.
- Deve ser salvo num arquivo com a extensão **.f90**

## Tipos de Unidades de Programa

- 1 Programa Principal (ou *main program*).
- 2 Sub-Programas (Funções e Subrotinas).
- 3 Módulos.

## Programa ou Código-fonte

Programa ou **código-fonte** é a coleção de blocos (**unidades de programa** ou seções) que executam as diferentes instruções para a realização de uma tarefa computacional.

- O programa deve ser escrito num editor de texto: **emacs**, **gedit**, **nano**, etc.
- Deve ser salvo num arquivo com a extensão **.f90**

## Tipos de Unidades de Programa

- 1 **Programa Principal** (ou *main program*).
- 2 Sub-Programas (Funções e Subrotinas).
- 3 Módulos.

# Estrutura de um Programa Principal

**PROGRAM** <nome do programa>

**IMPLICIT NONE**

*[Campo de Declaração dos Identificadores]*

*[Campo de Execução de Comandos]*

**END PROGRAM** <nome do programa>

## Formato Livre

- Uma instrução (ou comando) por linha.
- Para colocar **mais de uma instrução por linha**, usamos o caracter **;** para separá-las.
- As instruções podem iniciar em qualquer caracter da linha, inclusive na coluna **1**, podendo ir até **132** caracteres.
- Se a instrução não cabe na linha, ela deve ser interrompida com o caracter **&**, e a **continuação da instrução** deve aparecer na linha seguinte.
- **Comentários** são identificados pelo caracter **!**, que pode aparecer em qualquer posição da linha.

# Estrutura de um Programa Principal

**IMPLICIT NONE**

*[Campo de Declaração dos Identificadores]*

*[Campo de Execução de Comandos]*

**END**

## Formato Livre

- Não é obrigatório o uso de um nome para o programa na primeira linha.
- Podemos inserir uma (ou várias dentro do programa) instrução **STOP**, que direciona o fluxo do programa para a instrução **END**.
- A **finalização da execução** do programa é definida pela instrução **END**. Só pode existir uma e ela é **obrigatória**. Nada pode ser escrito além dela.
- Use a instrução **IMPLICIT NONE**: ela evita declarações de identificadores de forma implícita, como em versões anteriores do **FORTAN (66 e 77)**, em geral com consequências desastrosas.

# Declaração dos Identificadores

## Definição de Identificador

**Identificador** é o nome dado para as variáveis, constantes, funções e outras entidades num programa.

O que é **permitido** nos nomes para os identificadores:

- o **primeiro** caracter deve ser uma **letra**
- podem ser usadas MAIÚSCULAS e minúsculas
  - **FORTRAN 90** não é **case sensitive**
  - **passos, Passos, pAssOs, PASSOS** são o mesmo identificador
- os outros caracteres podem ser letras, números ou o caracter **\_**
  - **const1, number\_of\_steps, hora1var** são permitidos.
- nomes com até **31 caracteres**: não exagere no tamanho dos nomes.
  - use nomes relacionados com o objetivo do programa.

# Declaração dos Identificadores

## Definição de Identificador

**Identificador** é o nome dado para as variáveis, constantes, funções e outras entidades num programa.

O que **não é permitido** nos nomes para os identificadores:

- o uso de espaços em branco  
→ `soma total` é errado. Use `soma_total`
- o uso de hífen  
→ `soma-total` é errado. Use `soma_total`
- o uso de **acentuação, pontuação, ç, !, &, %**  
→ `torção` é errado. Use `torcao` ou `torsion`
- Evite o uso** de nomes com palavras reservadas do F90:  
→ `END`, `READ`, `WRITE`, ...

## Tipo do Identificador

Todo identificador tem um **tipo**, associado à característica do dado associado ao nome do identificador.

## Tipos Intrínsecos

- **Numérico:**

**REAL** :: a

**INTEGER** :: i

**COMPLEX** :: b

- **Literal ou Character:**

**CHARACTER** :: answer

- **Lógico ou booleano:**

**LOGICAL** :: overlap

Podem ser listados **mais de um identificador por linha**, **separados por vírgula**, desde que tenham o **mesmo tipo** de dado.



# Declaração dos Identificadores

**Identificador do tipo INTEGER:** 4 bytes, 8 bytes, 16 bytes

→ são armazenadas apenas a **parte inteira** de um número e o **sinal**.

- **INTEGER(KIND=4)** ou **INTEGER(4)** ou **INTEGER**  
→ números inteiros entre  $-2^{31}$  até  $+2^{31}$
- **INTEGER(KIND=8)** ou **INTEGER(8)**  
→ números inteiros entre  $-2^{63}$  até  $+2^{63}$
- **INTEGER(KIND=16)** ou **INTEGER(16)**  
→ números inteiros entre  $-2^{127}$  até  $+2^{127}$

1 byte = 8 bits

sinal = 1 bit : - (1) / + (0)

# Declaração dos Identificadores

## Como converter um INTEIRO para a representação binária?

- Construa uma tabela com as potências de 2, começando com  $2^0$  à direita.

$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
512	256	128	64	32	16	8	4	2	1

- Pegue o número a ser convertido, 277 por exemplo, e procure na tabela o maior número que caiba em 277, no caso  $2^7 = 256$ . Coloque o número 1 abaixo deste número na tabela.

$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
256	128	64	32	16	8	4	2	1
1								

- Subtraia 256 de 277 para obter como resultado 21.

# Declaração dos Identificadores

## Como converter um INTEIRO para a representação binária?

- Procure na tabela o maior número que caiba em 21, no caso  $2^4 = 16$ . Coloque o número 1 abaixo deste número na tabela, completando com 0 os campos entre 16 e 256.

$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
256	128	64	32	16	8	4	2	1
1	0	0	0	1				

- Subtraia 16 de 21 para obter como resultado 5. Procure na tabela o maior número que caiba em 5, no caso  $2^2 = 4$ . Coloque o número 1 abaixo deste número na tabela, completando com 0 os campos entre 4 e 16.

$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
256	128	64	32	16	8	4	2	1
1	0	0	0	1	0	1		

# Declaração dos Identificadores

## Como converter um INTEIRO para a representação binária?

- Subtraia 4 de 5 para obter como resultado 1. Procure na tabela o maior número que caiba em 1, no caso  $2^0 = 1$ . Coloque o número 1 abaixo deste número na tabela, completando com 0 os campos entre 1 e 4.

$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
256	128	64	32	16	8	4	2	1
1	0	0	0	1	0	1	0	1

- O número será escrito com o número de bits associados ao tipo da dado inteiro. Por exemplo, se queremos 277 na representação com 4 bytes (32 bits) ou `INTEGER(KIND=4)`, temos que acrescentar 0 para completar os 32 bits:

277 → 000000000000000000000000100010101

- O bit mais a esquerda indica o sinal: 0 → +

Como converter um INTEIRO NEGATIVO para binário?

## Regra do Complemento de 2

- Considere o inteiro  $-277$ . Primeiro, represente o **equivalente positivo** em binário (por exemplo, usando 32 bits):

000000000000000000000000100010101

- Complemente cada bit, substituindo os 0 por 1 e vice-versa:

111111111111111111111111011101010

- Acrescente o número 1 (em binário). O resultado da soma é o número negativo procurado:

$-277 \rightarrow 111111111111111111111111011101011$

- O bit mais a esquerda indica o sinal:  $1 \rightarrow -$

# Declaração dos Identificadores

**Exemplo:** Programa com declaração de identificadores **inteiros**.

```
program declara_inteiros
  implicit none
  integer :: A, B      ! A e B tem 4 bytes
  integer(8) :: C
  integer(kind=8) :: D
  integer(16) :: E
end program declara_inteiros
```

- O **código-fonte** deve ser salvo num arquivo, com um nome qualquer (**teste.f90**, por exemplo) que o usuário escolhe.
- Use um terminal do linux e compile o código usando o **compilador gfortran** do GNU:  
$$\$ \text{gfortran teste.f90}$$
- Se não existirem erros de compilação, será criado um **arquivo executável** de nome **a.out**

# Declaração dos Identificadores

**Identificador do tipo REAL:** 32 bits, 64 bits, 128 bits

- **REAL(KIND=4)** ou **REAL(4)** ou **REAL**

→ 8 casas decimais: 3.40282347E+38 é o maior número

- **REAL(KIND=8)** ou **REAL(8)**

→ 16 casas decimais: 1.7976931348623157E+308 é o maior número

- **REAL(KIND=16)** ou **REAL(16)**

→ 35 casas decimais:

1.18973149535723176508575932662800702E+4932 é o maior número

→ também chamados de números com **ponto flutuante**

# Declaração dos Identificadores

Identificador do tipo `REAL(KIND=4)` ou `REAL(4)` ou `REAL`

Apresentado em duas formas: **decimal** e **exponencial**

- **Forma decimal:** conjunto de dígitos contendo um ponto decimal e um sinal (o sinal positivo é opcional).

Exemplo: 10.23 .57 20. -11.65 -.87 0.65 -3.1415926 0.

- **Forma exponencial (notação científica):** conjunto de dígitos que inicia com o sinal (positivo é opcional) e a parte inteira do número, seguido pelo ponto e a parte decimal do número, o símbolo da exponencial (**E** ou **e**) e finalizado com o sinal (positivo é opcional) e o inteiro indicando o expoente da potência de 10 do número.

Exemplo: 10.0E4 ( $10 \times 10^4$ ) -45.23e6 ( $-45.23 \times 10^6$ ) -2.1E-2 ( $-2.1 \times 10^{-2}$ )



# Declaração dos Identificadores

Identificador do tipo `REAL(KIND=8)` ou `REAL(8)`

Apresentado apenas em **notação científica**

- **Notação científica:** conjunto de dígitos que inicia com o sinal (positivo é opcional) e a parte inteira do número, seguido pelo ponto e a parte decimal do número, o símbolo da exponencial (**D** ou **d**) e finalizado com o sinal (positivo é opcional) e o inteiro indicando o expoente da potência de 10 do número.

Exemplo:    `10.0D4` ( $10 \times 10^4$ )    `-45.23D6` ( $-45.23 \times 10^6$ )    `-2.1d-2` ( $-2.1 \times 10^{-2}$ )  
              `0.D0` (número 0)    `0.9113D0` (número 0.9113)

Identificadores do tipo `REAL(8)` são ditos do tipo **DOUBLE PRECISION**

# Declaração dos Identificadores

**Exemplo:** Programa com declaração de identificadores **reais**.

```
program declara_reais
  implicit none
  real :: A ! precisao simples (32 bits)
  real(8) :: B
  real(kind=8) :: C
  real(16) :: D
end program declara_reais
```

- O **código-fonte** deve ser salvo num arquivo, com um nome qualquer (**teste.f90**, por exemplo) que o usuário escolhe.
- Use um terminal do linux e compile o código usando o **compilador gfortran** do GNU:  
$$\$ \text{gfortran teste.f90}$$
- Se não existirem erros de compilação, será criado um **arquivo executável** de nome **a.out**

**Identificador do tipo COMPLEX:** 32 bits, 64 bits, 128 bits

- `COMPLEX(KIND=4)` ou `COMPLEX(4)` ou `COMPLEX`
- `COMPLEX(KIND=8)` ou `COMPLEX(8)`
- `COMPLEX(KIND=16)` ou `COMPLEX(16)`

→ número complexo  $a + ib$  é representado pelo par  $(a,b)$

A representação do par  $(a,b)$  depende do tipo de declaração do identificador complexo: precisão *simples*, *dupla* ou *quádrupla*.

# Declaração dos Identificadores

**Identificador do tipo CHARACTER:** 1 byte por caracter

→ o conjunto de caracteres é chamado de *string*

- O número máximo de caracteres de um identificador do tipo CHARACTER é definido pela opção **LEN**

**CHARACTER(LEN=???)**

→ **CHARACTER(LEN=10) :: word**

→ **CHARACTER(10) :: word**

indicam que o identificador **word** tem no máximo 10 caracteres.

- Colocar os caracteres entre aspas (“..”) ou apóstrofes ('..')

→ a palavra (ou string) é lida literalmente, inclusive os espaços:

**'bom Dia', "bom Dia", 'meio-dia'**

## Identificador do tipo LOGICAL

→ também chamados de **booleanos**

- O identificador lógico pode ser verdadeiro (**.true.**) ou falso (**.false.**)
- Identificadores lógicos não podem ser usados em expressões aritméticas.
- Exemplo de declaração:

**LOGICAL** :: overlap

# Declaração dos Identificadores

Definindo identificadores **CONSTANTES** usando **PARAMETER**

```
program declara_constantes
  implicit none
  integer , parameter :: n = 1000, j = 1
  real(4) , parameter :: e = 2.17828
  real(8) , parameter :: pi = 3.14159274D0
  logical , parameter :: overlap = .false.
  real :: i = 4      ! identificador VARIÁVEL
end program declara_constantes
```

- Um **identificador** definido através do **PARAMETER** não muda de valor durante a execução do programa.
  - Sem o **PARAMETER**, o identificador não terá valor constante, podendo mudar durante a execução do programa: **Identificador VARIÁVEL**

# Declaração dos Identificadores

Definindo identificadores **CONSTANTES** usando **PARAMETER**

```
program declara_constantes
  implicit none
  real(8), parameter :: pi = 3.14159274D0
  real(8), parameter :: doispi = 2.d0*pi, &
    trespi = 3.0*pi
  real(8) :: quatropi = 4.d0*pi
end program declara_constantes
```

- Podem ser usadas **expressões aritméticas** na definição de identificadores **CONSTANTES** e **VARIÁVEIS**.

→ todos os identificadores envolvidos na expressão devem ter sido declarados e com valores previamente inicializados usando **PARAMETER**.

## Por que usar **IMPLICIT NONE**?

```
program use_implicit
  implicit none
  . . .
end program use_implicit
```

- Se não for usada a instrução **IMPLICIT NONE** todos os identificadores que começam com as letras **i, j, k, l, m** e **n** são considerados do tipo **INTEGER**.  
Herança de versões antigas do Fortran !!!
- **IMPLICIT NONE** significa que os identificadores **não têm tipo implícito**, ou seja, devem ser **declarados de forma explícita** no campo de declaração.
- **Identificadores não declarados**, e eventualmente usados nas instruções ao longo da unidade do programa, produzirão um **erro de compilação**.



## Definição

No campo de execução colocamos todas as **instruções** (ou comandos) necessárias para a execução das tarefas computacionais descritas pelo algoritmo.

## Tipos de instruções

- Atribuição de valores
- Expressões aritméticas
- Entrada e saída de dados
- Estruturas condicionais
- Estruturas de repetição
- Desvios de fluxo para subprogramas

# Atribuição de valores

```
program teste
  implicit none
  real :: a
  real(8) :: b
  complex :: c
  logical :: flag , overlap
  character(LEN=20) :: nome
  a = 3.0;    b = 10.0d0
  c = (1.0 ,2.0)
  flag = .true.    ! verdadeira
  overlap = .false. ! falsa
  nome = 'Alexandre_Diehl'
end program teste
```

- Um identificador previamente declarado pode ter seu valor inicializado usando uma atribuição com o caracter =
- Atribuído desta forma, o **identificador** é do tipo **variável**, ou seja, pode ter seu valor modificado ao longo do programa.
- Deve ser respeitado o tipo da variável.

## Tipos de operadores

- **Aritmético:** potenciação, multiplicação, divisão, soma, subtração
- **Relacional:** menor, menor ou igual, maior, maior ou igual, igual, diferente
- **Lógico:** negação, conjunção, disjunção, equivalência, não-equivalência

Tipo	Operador						Associatividade
Aritmético	**						Direita para Esquerda
	*					/	Esquerda para Direita
	+					-	Esquerda para Direita
Relacional	<	<=	>	>=	==	/=	Nenhuma
Lógico	.NOT.						Direita para Esquerda
	.AND.						Esquerda para Direita
	.OR.						Esquerda para Direita
	.EQV.			.NEQV.			Esquerda para Direita

Ordem de Prioridade

↓

Maior

Menor

- Caracter

## Definição

Expressões aritméticas envolvem o uso de identificadores, constantes e variáveis, através dos **operadores aritméticos** usuais. Podem envolver também **Funções Intrínsecas** do F90 e subprogramas do tipo **FUNCTION**.

Podem ser usadas nas seguintes estruturas

- 1 **Declaração de atribuição:** o resultado da expressão aritmética (**dado**) é armazenado no endereço de memória de um identificador.
- 2 Como parte de uma estrutura condicional.
- 3 Como parte de uma estrutura de repetição.

Como é avaliada a expressão abaixo?

$$A = 1 + \frac{2 \times 3}{(6 \times 6 + 5 \times 44)^{1/4}}$$

- Expressões são avaliadas da **esquerda para a direita**.
- Toda vez que um operador é encontrado, sua **prioridade** é comparada com o próximo operador:
  - se o **próximo** operador tem **prioridade menor**, o operador é executado.
  - se o **próximo** operador tem **prioridade igual**, a relação de associatividade é usada para determinar qual operador é executado primeiro.
  - se o **próximo** operador tem **prioridade maior**, passamos para o operador seguinte para estabelecer a ordem de execução.

# Expressões Aritméticas: declaração de atribuição

Como é avaliada a expressão abaixo?

$$A = 1 + \frac{2 \times 3}{(6 \times 6 + 5 \times 44)^{1/4}}$$

`A = 1.0 + 2.0 * 3.0 / ( 6.0*6.0 + 5.0*44.0 ) ** 0.25`

Seguindo as regras anteriores:

`1.0 + 2.0 * 3.0 / ( 6.0*6.0 + 5.0*44.0 ) ** 0.25`

→ `1.0 + 6.0 / ( 6.0*6.0 + 5.0*44.0 ) ** 0.25`

→ `1.0 + 6.0 / ( 36.0 + 5.0*44.0 ) ** 0.25`

→ `1.0 + 6.0 / ( 36.0 + 220.0 ) ** 0.25`

→ `1.0 + 6.0 / 256.0 ** 0.25`

→ `1.0 + 6.0 / 4.0`

→ `1.0 + 1.5`

→ `2.5`

Como é avaliada a expressão abaixo?

$$A = 1 + \frac{2 \times 3}{(6 \times 6 + 5 \times 44)^{1/4}}$$

Programa exemplo:

```
program teste
  implicit none
  real :: A
  A = 1.0 + 2.0*3.0/(6.0*6.0 + 5.0*44.0)**0.25
end program teste
```

Como é avaliada a expressão abaixo?

$$B = 5 \times \frac{(11 - 5)^2}{4} + 9$$

- Expressões que envolvem **REAL** e **INTEGER** são ditas no **modo misto**.
- Operação com **REAL** e **INTEGER** produz um resultado **REAL**: o **INTEGER** é convertido para **REAL** antes da operação.  
→  $3.5 * 4$  é convertido para  $3.5 * 4.0$
- Exceções a regra:  
→  $x^{**}3$  é realizado como  $x * x * x$   
→  $x^{**}(-3)$  é realizado como  $1.0 / (x * x * x)$



# Expressões Aritméticas: declaração de atribuição

Como é avaliada a expressão abaixo?

$$B = 5 \times \frac{(11 - 5)^2}{4} + 9$$

$$B = 5 * (11.0 - 5) ** 2 / 4 + 9$$

$$\rightarrow 5 * (11.0 - 5.0) ** 2 / 4 + 9$$

$$\rightarrow 5 * 6.0 ** 2 / 4 + 9$$

$$\rightarrow 5 * 36.0 / 4 + 9$$

$$\rightarrow 5.0 * 36.0 / 4 + 9$$

$$\rightarrow 180.0 / 4 + 9$$

$$\rightarrow 180.0 / 4.0 + 9$$

$$\rightarrow 45.0 + 9$$

$$\rightarrow 45.0 + 9.0$$

$$\rightarrow 54.0$$

Note que  
 $6.0 ** 2$   
é calculado como  
 $6.0 * 6.0$

# Expressões Aritméticas: declaração de atribuição

Como é avaliada a expressão abaixo?

$$B = 5 \times \frac{(11 - 5)^2}{4} + 9$$

```
program teste
  implicit none
  real :: B
  B = 5*(11.0 - 5)**2/4 + 9
end program teste
```

Será que podemos usar o programa abaixo?

```
real :: B
B = 5*(11 - 5)**2/4 + 9
end
```

Alguns cuidados importantes:

- Se os **tipos** da **variável** e a da **expressão** são os mesmos, o resultado da expressão é armazenado no endereço de memória da **variável**.
- Se os **tipos** da **variável** e a **expressão não são os mesmos**, o resultado da expressão é convertido ao tipo da **variável** e armazenado no endereço de memória desta.
- Se o resultado da **expressão** é **REAL** e o tipo da **variável** associada à expressão é **INTEGER**, o resultado é **truncado** (apenas a **parte inteira** da expressão é armazenada no endereço da variável).

```
program teste
  implicit none
  real :: A, C, D, E
  integer :: E
  A = 1.0/4.0; C = 1.0/4; E = 1/4.0  ! = 0.25
  D = 1/4                               ! = 0, truncamento
  B = 5*(11 - 5)**2/4 + 9                ! = 54, truncamento
end program teste
```

- Cuidado na **divisão** com **denominador inteiro**.
- Cuidado com a colocação de parênteses em expressões aritméticas:

→  $a^{**}b^{**}c$  é a mesma coisa que  $(a^{**}b)^{**}c$

→  $(a^{**}b)^{**}c \neq a^{**}(b^{**}c)$

## Definição

**Funções Intrínsecas** são **subprogramas** disponibilizados pela linguagem **FORTRAN 90**, sem que o usuário precise defini-las. Podem ser usadas em expressões matemáticas ou em qualquer parte do programa.

## O que precisamos saber para usar uma **Função Intrínseca**

- O **nome** e o **significado** da Função.
- O **número de argumentos** usados pela Função.
- O **tipo** e as **restrições** de **cada argumento** usados pela Função.
- O **tipo do valor retornado** pela Função.

## Trigonométricas

Função	Significado	Restrições		Tipo de Argumento	
		Entrada	Saída	Entrada	Saída
<b>SIN (X)</b>	Seno do ângulo X	X em radianos	$-1 \leq \text{SIN}(X) \leq 1$	REAL	REAL
<b>COS (X)</b>	Cosseno do ângulo X	X em radianos	$-1 \leq \text{SIN}(X) \leq 1$	REAL	REAL
<b>TAN (X)</b>	Tangente do ângulo X	X em radianos	nenhuma	REAL	REAL
<b>ASIN (X)</b>	Arco seno de X	$ X  \leq 1$	$-\pi/2 \leq \text{ASIN}(X) \leq \pi/2$	REAL	REAL
<b>ACOS (X)</b>	Arco cosseno de X	$ X  \leq 1$	$0 \leq \text{ACOS}(X) \leq \pi$	REAL	REAL
<b>ATAN (X)</b>	Arco tangente de X	nenhuma	$-\pi/2 \leq \text{ATAN}(X) \leq \pi/2$	REAL	REAL
<b>ATAN2 (Y,X)</b>	Arco tangente para pares X e Y. Retorna o valor principal do número complexo (X,Y), expresso em radianos	X e Y não podem ser simultaneamente nulos	$-\pi \leq \text{ATAN2}(Y,X) \leq \pi$	REAL	REAL
<b>SINH (X)</b>	Seno hiperbólico de X	Nenhuma	nenhuma	REAL	REAL
<b>COSH (X)</b>	Cosseno hiperbólico de X	Nenhuma	nenhuma	REAL	REAL
<b>TANH (X)</b>	Tangente hiperbólica de X	Nenhuma	Nenhuma	REAL	REAL

```
real :: A, pi = 4.0*ATAN(1.0)
A = 3*COS(pi) - SIN(pi/2.0)
end
```

## Matemáticas comuns

Função	Significado	Restrições		Tipo de Argumento	
		Entrada	Saída	Entrada	Saída
ABS (X)	Valor absoluto de X	nenhuma	$ABS(X) \geq 0$	INTEGER	INTEGER
		nenhuma	$ABS(X) \geq 0$	REAL	REAL
SQRT (X)	Raiz quadrada de X	$X > 0$	nenhuma	REAL	REAL
		nenhuma	nenhuma	COMPLEX	COMPLEX
EXP (X)	Exponencial de X	nenhuma	nenhuma	REAL	REAL
		nenhuma	nenhuma	COMPLEXO	COMPLEXO
LOG (X)	Logaritmo natural de X	$X > 0$	nenhuma	REAL	REAL
		$X > 0$	$-\pi < \text{parte imaginária de } X < \pi$	COMPLEXO	COMPLEXO
LOG10 (X)	Logaritmo de base 10 de X	$X > 0$	nenhuma	REAL	REAL

```
real(8), parameter :: pi = 4.0*ATAN(1.0d0)
real(8) :: A, B
B = -10.0d0
A = 5.d0*ABS(B+3.0d0) + 1.0d0/(EXP(B*pi))
end
```

## Funções de Conversão

Função	Significado	Restrições		Tipo de Argumento	
		Entrada	Saída	Entrada	Saída
INT (X)	Converte X ao tipo inteiro	nenhuma	INT(X) = X	INTEGER	INTEGER
		nenhuma	valor truncado em sentido ao zero	REAL	INTEGER
		nenhuma	Somente a parte real de X tem valor truncado em sentido ao zero	COMPLEX	INTEGER
NINT (X)	Retorna o valor inteiro mais próximo de X	nenhuma	nenhuma	REAL	INTEGER
AINT (X)	Trunca o valor de X em sentido ao zero	nenhuma	Os valores decimais de X são igualados a zero	REAL	REAL
ANINT (X)	Retorna o valor real mais próximo de X, sem a parte fracionária	nenhuma	Os valores decimais de X são igualados a zero	REAL	REAL

```
real :: A = -3.65, B = 3.65, E, F
integer :: C, D
C = INT(A); D = NINT(A)
E = AINT(B); F = ANINT(B)
end
```

Resultado:

→ C = -3    D = -4    E = 3.00000000    F = 4.00000000



## Funções de Conversão

Função	Significado	Restrições		Tipo de Argumento	
		Entrada	Saída	Entrada	Saída
REAL (X)	Converte X ao tipo real	nenhuma	nenhuma	INTEGER	REAL
		nenhuma	nenhuma	REAL	REAL
		nenhuma	parte imaginária de X é ignorada	COMPLEX	REAL
FLOOR (X)	Retorna o <b>maior inteiro</b> menor ou igual a X	nenhuma	nenhuma	REAL	INTEGER
CEILING (X)	Retorna o <b>menor inteiro</b> maior ou igual a X	nenhuma	nenhuma	REAL	INTEGER
AIMAG (X)	Retorna a <b>parte imaginária</b> do complexo X	nenhuma	nenhuma	COMPLEX	REAL
CMPLX (X)	Converte X ao tipo complexo	nenhuma	nenhuma	INTEGER	COMPLEX
		nenhuma	nenhuma	REAL	COMPLEX
		nenhuma	nenhuma	COMPLEX	COMPLEX
CMPLX (X,Y)	Converte (X,Y) ao tipo complexo	nenhuma	nenhuma	INTEGER	COMPLEX
		nenhuma	nenhuma	REAL	COMPLEX

```
real :: a = 4.5; integer :: b = 4; complex :: c, d
c = CMPLX(a); d = CMPLX(b)
end
```

Resultado:

→ c = (4.50000000, 0.00000000)    d = (4.00000000, 0.00000000)

## Funções de Conversão

Função	Significado	Tipo de Argumento	
		Entrada	Saída
CONJG (X)	Retorna o conjugado de X	COMPLEX	COMPLEX
MAX (X1, X2, ..., Xn)	Retorna o máximo ( <b>maior valor</b> ) entre dois ou mais valores	INTEGER	INTEGER
		REAL	REAL
MIN (X1, X2, ..., Xn)	Retorna o mínimo ( <b>menor valor</b> ) entre dois ou mais valores	INTEGER	INTEGER
		REAL	REAL
MOD (X,Y)	Retorna o restante de X módulo Y: $X - \text{INT}(X/Y)*Y$ Se $Y = 0$ o resultado depende do processador	INTEGER	INTEGER
		REAL	REAL
MODULO (X,Y)	Retorna X módulo Y: $X - \text{FLOOR}(X/Y)*Y$ A divisão é a matemática ordinária. Se $Y = 0$ o resultado depende do processador	INTEGER	INTEGER
	Retorna X módulo Y: $X - \text{FLOOR}(X/Y)*Y$ Se $Y = 0$ o resultado depende do processador	REAL	REAL
SIGN (X,Y)	Retorna o valor absoluto de X vezes o sinal de Y. Se $Y = 0$ ( <b>INTEGER</b> ) o sinal é assumido positivo. Se $Y = 0.0$ ( <b>REAL</b> ) o sinal é assumido positivo	INTEGER	INTEGER
		REAL	REAL

```
real :: a, b
a = MAX(1.0, -4.5, 10.0); b = MIN(1.0, -4.5, 10.0)
end
```

Resultado: → a = 10.00000000 b = -4.50000000

## Funções de Conversão

Função	Significado	Tipo de Argumento	
		Entrada	Saída
CONJG (X)	Retorna o conjugado de X	COMPLEX	COMPLEX
MAX (X1, X2, ..., Xn)	Retorna o máximo ( <b>maior valor</b> ) entre dois ou mais valores	INTEGER	INTEGER
		REAL	REAL
MIN (X1, X2, ..., Xn)	Retorna o mínimo ( <b>menor valor</b> ) entre dois ou mais valores	INTEGER	INTEGER
		REAL	REAL
MOD (X,Y)	Retorna o restante de X módulo Y: $X - INT(X/Y)*Y$ Se $Y = 0$ o resultado depende do processador	INTEGER	INTEGER
		REAL	REAL
MODULO (X,Y)	Retorna X módulo Y: $X - FLOOR(X/Y)*Y$ A divisão é a matemática ordinária. Se $Y = 0$ o resultado depende do processador	INTEGER	INTEGER
	Retorna X módulo Y: $X - FLOOR(X/Y)*Y$ Se $Y = 0$ o resultado depende do processador	REAL	REAL
SIGN (X,Y)	Retorna o valor absoluto de X vezes o sinal de Y. Se $Y = 0$ ( <b>INTEGER</b> ) o sinal é assumido positivo. Se $Y = 0.0$ ( <b>REAL</b> ) o sinal é assumido positivo	INTEGER	INTEGER
		REAL	REAL

```
complex :: a = (1.0,1.0), b
b = CONJG(a)
end
```

Resultado:  $\rightarrow b = (1.00000000, -1.00000000)$

## Entrada de Dados

- Os dados podem ser inseridos através do **teclado**
- Os dados podem ser inseridos através de arquivos (veremos depois)

## Usando o teclado

- Usamos o comando **READ\***, **A1**, **A2**, ..., **An**
  - os dados de entrada não são formatados (**formato livre**)
- ou usamos o comando **READ(\*,\*)****A1**, **A2**, ..., **An**
  - os dados de entrada não são formatados (**formato livre**)
  - este formato admite entrada de dados formatada (veremos depois)
- As variáveis **A1**, **A2**, ..., **An** devem ser definidas no programa. Não precisam ser do mesmo tipo.

## Entrada de Dados

- Os dados podem ser inseridos através do teclado

```
INTEGER :: idade
REAL    :: peso
CHARACTER(LEN=20) :: nome
READ*, nome, idade, peso
!READ(*,*)nome, idade, peso ! forma alternativa
END
```

Digitar no teclado:

'Alexandre Diehl', 51, 87.0 → OK

'Alexandre Roberto Diehl', 51, 87 → nome com mais de 20 caracteres

## Saída de Dados

### Saída de Dados

- Os dados podem ser disponibilizados através do **monitor**
- Os dados podem ser disponibilizados através de arquivos (veremos depois)

### Usando o monitor

- Usamos o comando **PRINT\*, A1, A2, ..., An**
  - os dados de saída não são formatados (**formato livre**)
  - este formato admite entrada de dados formatada (veremos depois)
- ou usamos o comando **WRITE(\*,\*)A1, A2, ..., An**
  - os dados de saída não são formatados (**formato livre**)
  - este formato admite saída de dados formatada (veremos depois)
- As variáveis **A1, A2, ..., An** devem ser definidas no programa. Não precisam ser do mesmo tipo.

## Saída de Dados

- Os dados podem ser disponibilizados através do **monitor**

```
INTEGER :: idade  
REAL :: peso  
CHARACTER(LEN=20) :: nome  
READ(*,*) nome, idade, peso  
PRINT*,nome, idade, peso  
END
```

Digitar no teclado:

'Alexandre Diehl', 51, 87.0 → OK

Saída no monitor:

Alexandre Diehl 51 87.00000000

## Saída de Dados

- Os dados podem ser disponibilizados através do **monitor**

```
INTEGER :: idade
REAL :: peso
CHARACTER(LEN=20) :: nome
READ(*,*) nome
READ*, idade
READ(*,*) peso
PRINT*, 'nome=', nome
WRITE(*,*) 'idade=', idade
PRINT*, 'peso=', peso
END
```

- Comentários (caracteres) são inseridos entre **".."** ou **'..'**



## Saída de Dados

- Os dados podem ser disponibilizados através do **monitor**

```
real :: theta
read *, theta
print *, 'valor do angulo=', theta
print *, 'valor do seno do angulo=', SIN(theta)
write (*,*) 'tangente=', SIN(theta)/COS(theta)
END
```

- Podem ser usadas **expressões aritméticas**
- Podem ser usadas **Funções Intrínsecas**