Introdução ao Fortran 90 - 1

Alexandre Diehl

Departamento de Física – UFPel

Definição de Programa em Fortran 90

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.

Definição de Programa em Fortran 90

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 FORTRAN (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, ...

Declaração dos Identificadores

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 Caracter:**

  CHARACTER :: answer

- **Lógico ou booleno:**

  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)

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.

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		

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:

$$277 \rightarrow 00000000000000000000000100010101$$

- O bit mais a esquerda indica o sinal: $0 \rightarrow +$

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):

$$00000000000000000000000100010101$$

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

$$11111111111111111111111011101010$$

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

$$\text{-277} \rightarrow 11111111111111111111111011101011$$

- 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:

  $ 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:

  1.189731495357231765085759326628007 02E+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 (10×10^4) -45.23e6 (-45.23×10^6) -2.1E-2 (-2.1×10^{-2})

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 (10×10^4) -45.23D6 (-45.23×10^6) -2.1d-2 (-2.1×10^{-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:

  $ gfortran teste.f90

- Se não existirem erros de compilação, será criado um arquivo executável de nome a.out

Declaração dos Identificadores

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)

$\longrightarrow$ 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 boolenos

- 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 VARIAVEL
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 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.

Campo de Execução

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

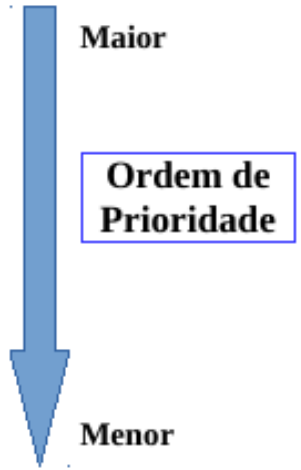
```
program teste
  implicit none
  real :: a
  real(8) :: b
  complex :: c
  logical :: flag, overlap
  character(LEN=20) :: nome
  a = 3.0;    b = 10.0d0
  c = (1.0,2.0)
  flag = .true.        ! verdadeira
  overlap = .false.    ! falsa
  nome = 'Alexandre Diehl'
end program teste
```

- 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



- Caracter

Expressões Aritméticas

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.

Expressões Aritméticas: declaração de atribuição

Como é avaliada a expressão abaixo?

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

- 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

Expressões Aritméticas: declaração de atribuição

Como é avaliada a expressão abaixo?

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

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
```

Expressões Aritméticas: declaração de atribuição

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

→ 5 * (11.0 - 5.0) ** 2 / 4 + 9

→ 5 * 6.0 ** 2 / 4 + 9

→ 5 * 36.0 / 4 + 9

→ 5.0 * 36.0 / 4 + 9

→ 180.0 / 4 + 9

→ 180.0 / 4.0 + 9

→ 45.0 + 9

→ 45.0 + 9.0

→ 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
```

Expressões Aritméticas: declaração de atribuição

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).

Expressões Aritméticas: declaração de atribuição

```
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 ≠ a**(b**c)