Array - Vetorizando as Operações

Matheus Roos

Universidade Federal de Santa Maria

26 de julho de 2022

Motivação

- **Alternativa** ao uso de arquivo de dados;
- A sistematização da álgebra linear;
- Necessidade de trabalhar com dados de natureza vetorial, multidimensional;
- Trazendo ferramentas muito **poderosas** para trabalhar com dados;
- O recurso array é comum as mais diversas linguagens;
- Praticamente todos os cálculos em Python são **vetorizados**;
- Arrays podem ser ferramentas extremamente poderosas,nos permitindo aplicar **o mesmo algoritmo** repetidamente a muitos itens de dados diferentes com um simples loop DO;
- Arrays são obviamente uma maneira muito mais limpa e curta de trabalhar com operações semelhantes e repetitivas.

Introdução

- Uma matriz é um grupo de variáveis ou constantes, todas de mesmo tipo, referenciadas por um único nome;
- Um valor individual dentro do array é chamado de **array element**;
- O elem. é identificado pelo nome do array junto com um subscrito apontando para o local específico dentro do array;
- O subscrito de uma matriz é do tipo INTEGER;

Declarando Arrays

- Primeiro devemos declarar o tipo e número de elementos de um array;

[1] Alocação dinâmica de memória

Declarando Arrays

- Primeiro devemos declarar o tipo e número de elementos de um array;
- A razão disso é informar ao compilar qual o tipo de dados que serão armazenados e o espaço na memória necessário[1];

[1] Alocação dinâmica de memória

Declarando Arrays

- Primeiro devemos declarar o tipo e número de elementos de um array;
- A razão disso é informar ao compilar qual o tipo de dados que serão armazenados e o espaço na memória necessário[1];
- Declaramos uma matriz **real,** contendo **n** elementos, de nome **name_array**:

$$\text{REAL, DIMENSION(n) :: name\_array}$$

n deve ser um inteiro, ele também é chamado de **extensão da matrix**.

[1] Alocação dinâmica de memória

Estrutura de um array

- Podemos definir uma matriz de constantes;
- Delimitadores - construtores de arrays
- Os delimitadores em Fortra 90 são a (/ e /)

(/ 1, 2 ,3 , 4 /)

- No Fortran 2003 os delimitadores são colchetes [ ]

[1, 2, 3, 4]

- Apesar dos colchetes serem um recurso trazido no Fortra 2003, o GFortran o traz ao compilar um .f90. Veja o **Exemplo 1**.

Trabalhando com arrays - Operações básicas

- Cada elemento de um array é uma **variável** como qualquer outra;
- Um elemento de array pode ser utilizado em **qualquer lugar** onde uma variável comum seria usada;
- Elementos de matriz podem ser incluídos em expressões aritméticas e lógicas,
- Os resultados de uma expressão podem ser atribuídos a um elemento de matriz.
- Os elementos de uma lista começam pelo índice 1, não pelo 0 como em outras linguagens. Vide **Exemplo 2**.
- Podemos alterar os valores dos elementos de uma lista. Veja o **Exemplo 3**.

Pausa para trabalhar com a formatação

- Podemos formatar a saída dos dados, uma das formas de se fazer isso é substituindo um número no caracter correspondente a formatação.
- No caso da declaração PRINT

PRINT *, value $\rightarrow$ PRINT 10, value

$\rightarrow$ 10 FORMAT(códigos de formatação)

- No caso do READ E WRITE será no segundo asterisco:

READ(*,10)

WRITE(*,10)

10 FORMAT (códigos de formatação)

Códigos de formatação

- Códigos de formatação:
  - A -> string de texto
  - D -> Nº de precisão dupla, notação exponencial
  - E -> Números reais, notação exponencial
  - F -> Números reais, formato de ponto fixo
  - I -> Número inteiro
  - X -> Salto horizontal (espaço em branco)
  - / -> Salto vertical (quebra de linha).
- Os códigos D, E e F possuem a forma geral D*w.d*, onde *w*=width (largura do campo) e *d*=digits (nº de dígitos significativos).
- Veja o **Exemplo 4** e **5**.

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;
- Ele vai permitir que uma lista de argumentos seja escrita muitas vezes em função de uma variável de índice;

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;
- Ele vai permitir que uma lista de argumentos seja escrita muitas vezes em função de uma variável de índice;
- Cada argumento na lista de argumentos é escrito uma vez para cada valor da variável de índice no loop DO implícito;

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;
- Ele vai permitir que uma lista de argumentos seja escrita muitas vezes em função de uma variável de índice;
- Cada argumento na lista de argumentos é escrito uma vez para cada valor da variável de índice no loop DO implícito;
- A estrutura geral para esse loop implícito é a seguinte:

WRITE (unit, format) (name_list1, index=istart, iend, istep)

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;
- Ele vai permitir que uma lista de argumentos seja escrita muitas vezes em função de uma variável de índice;
- Cada argumento na lista de argumentos é escrito uma vez para cada valor da variável de índice no loop DO implícito;
- A estrutura geral para esse loop implícito é a seguinte:

WRITE (unit, format) (name_list1, index=istart, iend, istep)

- Onde name_list1 são os valores a serem escritos/lidos;

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;
- Ele vai permitir que uma lista de argumentos seja escrita muitas vezes em função de uma variável de índice;
- Cada argumento na lista de argumentos é escrito uma vez para cada valor da variável de índice no loop DO implícito;
- A estrutura geral para esse loop implícito é a seguinte:

WRITE (unit, format) (name_list1, index=istart, iend, istep)

- Onde name_list1 são os valores a serem escritos/lidos;
- index devem ser inteiros;

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;
- Ele vai permitir que uma lista de argumentos seja escrita muitas vezes em função de uma variável de índice;
- Cada argumento na lista de argumentos é escrito uma vez para cada valor da variável de índice no loop DO implícito;
- A estrutura geral para esse loop implícito é a seguinte:

WRITE (unit, format) (name_list1, index=istart, iend, istep)

- Onde name_list1 são os valores a serem escritos/lidos;
- index devem ser inteiros;
- Veja o **Exemplo 6**;

Loop DO implícito

- O loop DO implícito também é permitido em instruções de i/o;
- Ele vai permitir que uma lista de argumentos seja escrita muitas vezes em função de uma variável de índice;
- Cada argumento na lista de argumentos é escrito uma vez para cada valor da variável de índice no loop DO implícito;
- A estrutura geral para esse loop implícito é a seguinte:

WRITE (unit, format) (name_list1, index=istart, iend, istep)

- Onde name_list1 são os valores a serem escritos/lidos;
- index devem ser inteiros;
- Veja o **Exemplo 6**;
- Também podemos fazer loop implícitos aninhados (um loop dentro doutro). Veja o **Exemplo 7** e **8**.

Diferença entre loop implícito e explícito

- Um loop padrão

```
1 lista = [1, 2, 3]
2 do i=1,3
3   WRITE(*,*) lista(i), 2.*lista(i)
4 end do
```

Diferença entre loop implícito e explícito

- Um loop padrão

```
1 lista = [1, 2, 3]
2 do i=1,3
3   WRITE(*,*) lista(i), 2.*lista(i)
4 end do
```

- se comportará como **vários** WRITE/READ com alguns argumentos

```
1 WRITE (*,*) lista(1), 2.*lista(1)
2 WRITE (*,*) lista(2), 2.*lista(2)
3 WRITE (*,*) lista(3), 2.*lista(3)
```

Diferença entre loop implícito e explícito

- Um loop padrão

```
1 lista = [1, 2, 3]
2 do i=1,3
3   WRITE(*,*) lista(i), 2.*lista(i)
4 end do
```

- se comportará como **vários** WRITE/READ com alguns argumentos

```
1 WRITE (*,*) lista(1), 2.*lista(1)
2 WRITE (*,*) lista(2), 2.*lista(2)
3 WRITE (*,*) lista(3), 2.*lista(3)
```

- Já com o loop implícito,

```
1 WRITE(*,*) (lista(i), 2.*lista(i), i= 1, 5)
```

Diferença entre loop implícito e explícito

- Um loop padrão

```
1 lista = [1, 2, 3]
2 do i=1,3
3   WRITE(*,*) lista(i), 2.*lista(i)
4 end do
```

- se comportará como **vários** WRITE/READ com alguns argumentos

```
1 WRITE (*,*) lista(1), 2.*lista(1)
2 WRITE (*,*) lista(2), 2.*lista(2)
3 WRITE (*,*) lista(3), 2.*lista(3)
```

- Já com o loop implícito,

```
1 WRITE(*,*) (lista(i), 2.*lista(i), i= 1, 5)
```

- teremos **um** WRITE/READ para muitos argumentos

```
1 WRITE(*,*) lista(1), 2.*lista(1), &
2            lista(2), 2.*lista(2), &
3            lista(3), 2.*lista(3)
```

Diferença entre loop implícito e explícito

- Um loop padrão

```
1 lista = [1, 2, 3]
2 do i=1,3
3   WRITE(*,*) lista(i), 2.*lista(i)
4 end do
```

- se comportará como **vários** WRITE/READ com alguns argumentos

```
1 WRITE (*,*) lista(1), 2.*lista(1)
2 WRITE (*,*) lista(2), 2.*lista(2)
3 WRITE (*,*) lista(3), 2.*lista(3)
```

- Já com o loop implícito,

```
1 WRITE(*,*) (lista(i), 2.*lista(i), i= 1, 5)
```

- teremos **um** WRITE/READ para muitos argumentos

```
1 WRITE(*,*) lista(1), 2.*lista(1), &
2            lista(2), 2.*lista(2), &
3            lista(3), 2.*lista(3)
```

- Além da formatação de saída. Vejamos como isso ocorre com o **Exemplo 9**.

Seção de Array

- Podemos selecionar fatias de uma lista, ou seja, seções;
- Escolhendo um start e um stop, tal que, **start:stop**;
- Podemos começar a contar a partir de um certo valor, se for a partir do 2^o elem., então **2:**
- Se então quisermos contar até um certo valor, se for até o 5^o elem., então **:5**. Vejamos o **Exemplo 10**;
- Podemos escolher o passo com que é percorrida uma matriz, da forma **start:stop:step**
- Ao chamarmos um array e atribuirmos um valor a ele, toda a matriz assumirá esse valor.Vejamos o **Exemplo 11**;
- Também podemos percorrer os valores de forma regressiva, ou seja, com um step negativo;

Quando Utilizar Arrays?

- Como podemos decidir se faz ou não sentido usar um array em um problema específico?
- Em geral, se muitos ou todos os dados de entrada devem estar na memória ao mesmo tempo para resolver um problema de forma eficiente, então o uso de arrays para armazenar esses dados será apropriado para este fim;
- Caso contrário, as matrizes não serão necessárias;
- Um programa para calcular a média e o desvio padrão não precisam de matrizes;
- Mas para encontrar a mediana de um conjunto de dados requer que os dados sejam ordenados de forma crescente. Como a ordenação requer que todos os dados estejam na memória, um programa que calcula a mediana deve usar uma matriz para armazenar todos os dados de entrada antes do início dos cálculos.

Quando Não Utilizar Array

- O que há de errado em usar um array dentro de um programa mesmo que não seja necessário?
  1. Matrizes desnecessárias desperdiçam memória. Arrays desnecessários podem consumir muita memória, tornando um programa maior do que o necessário. Um programa grande requer mais memória para executá-lo, o que torna o computador em que ele roda mais caro. Em alguns casos, o tamanho extra pode impossibilitar a execução em um computador específico de alguma forma.

Quando Não Utilizar Array

- O que há de errado em usar um array dentro de um programa mesmo que não seja necessário?
  1. Matrizes desnecessárias desperdiçam memória. Arrays desnecessários podem consumir muita memória, tornando um programa maior do que o necessário. Um programa grande requer mais memória para executá-lo, o que torna o computador em que ele roda mais caro. Em alguns casos, o tamanho extra pode impossibilitar a execução em um computador específico de alguma forma.
  2. Matrizes desnecessárias restringem os recursos do programa. Para entender este ponto, vamos considerar um programa que calcula a média e o desvio padrão de um conjunto de dados. Se o programa for projetado com uma matriz de entrada estática de 1.000 elementos, ele funcionará apenas para conjuntos de dados com até 1.000 elementos. Se encontrarmos um dado conjunto com mais de 1000 elementos, o programa teria que ser recompilado e revinculado com um tamanho de matriz maior. Por outro lado, um programa que calcula a média e o desvio padrão de um conjunto de dados à medida que os valores são inseridos não tem limite superior no tamanho do conjunto de dados.