Lógica e ramificações

Lógica de programação e ramificações em um programa

Matheus Roos

Universidade Federal de Santa Maria

4 de agosto de 2022

Motivação

- Lógica está presente na matemática e em todas as linguagens de programação (semântica, lógica, sintaxe);
- Caráter generalista (conhecimento aplicável a outras linguagens);
- Economizador de condicionais if-else;
- Ferramenta a mais para contruir códigos, pois as ramificações são geralmente controladas através de valores lógicos;
- Loops trabalham com expressões lógicas;
- A variável do tipo LOGICAL (booleano).

Valores lógicos

- O dado de tipo lógico permite **apenas** dois valores[1] (duas constantes):

[1] Lógica paraconsistente para além do V ou F, Newton Costa

Valores lógicos

- O dado de tipo lógico permite **apenas** dois valores[1] (duas constantes):
  - .TRUE. $\rightarrow$Verdadeiro;

[1] Lógica paraconsistente para além do V ou F, Newton Costa

Valores lógicos

- O dado de tipo lógico permite **apenas** dois valores[1] (duas constantes):
  - .TRUE. $\rightarrow$Verdadeiro;
  - .FALSE.$\rightarrow$Falso.

[1] Lógica paraconsistente para além do V ou F, Newton Costa

Valores lógicos

- O dado de tipo lógico permite **apenas** dois valores[1] (duas constantes):
  - .TRUE. →Verdadeiro;
  - .FALSE.→Falso.
- Eles são do tipo LOGICAL (booleano);

```
PROGRAM example
LOGICAL :: var1, var2,...
!(bloco de execução)
```

[1] Lógica paraconsistente para além do V ou F, Newton Costa

Valores lógicos

- O dado de tipo lógico permite **apenas** dois valores[1] (duas constantes):
  - .TRUE. →Verdadeiro;
  - .FALSE.→Falso.
- Eles são do tipo LOGICAL (booleano);

```
PROGRAM example
LOGICAL :: var1, var2,...
!(bloco de execução)
```

- Raramente vamos utilizar constantes lógicas, mas sim, expressões lógicas para controlar a execução;

[1] Lógica paraconsistente para além do V ou F, Newton Costa

Cálculo lógico

- Assim como no caso aritmético, atribuímos uma declaração aos cálculos lógicos que serão executados, tal que,

$$\text{logical\_var\_name} = \text{logical\_expr}$$

Cálculo lógico

- Assim como no caso aritmético, atribuímos uma declaração aos cálculos lógicos que serão executados, tal que,

$$\text{logical\_var\_name} = \text{logical\_expr}$$

- Um operador lógico é um operador em dados numéricos, de caracteres ou lógicos que geram um resultado lógico;

Cálculo lógico

- Assim como no caso aritmético, atribuímos uma declaração aos cálculos lógicos que serão executados, tal que,

$$\text{logical\_var\_name} = \text{logical\_expr}$$

- Um operador lógico é um operador em dados numéricos, de caracteres ou lógicos que geram um resultado lógico;
- Existem operadores relacionais e operadores combinacionais.

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;
- O resultado depende da **relação** entre estes dois valores que estão sendo comparados;

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;
- O resultado depende da **relação** entre estes dois valores que estão sendo comparados;
- A forma geral de uma relação operador relacional é a seguinte:

A op B.

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;
- O resultado depende da **relação** entre estes dois valores que estão sendo comparados;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \text{ op } B.$$

- A e B podem ser uma:

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;
- O resultado depende da **relação** entre estes dois valores que estão sendo comparados;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \text{ op } B.$$

- A e B podem ser uma:
  - expressão aritmética;

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;
- O resultado depende da **relação** entre estes dois valores que estão sendo comparados;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \text{ op } B.$$

- A e B podem ser uma:
  - expressão aritmética;
  - variável;

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;
- O resultado depende da **relação** entre estes dois valores que estão sendo comparados;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \text{ op } B.$$

- A e B podem ser uma:
  - expressão aritmética;
  - variável;
  - constante;

Operadores relacionais

- São operadores com dois operandos numéricos ou de caracteres que vão produzir um resultado lógico;
- O resultado depende da **relação** entre estes dois valores que estão sendo comparados;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \text{ op } B.$$

- A e B podem ser uma:
  - expressão aritmética;
  - variável;
  - constante;
  - ou então cadeia de caracteres(string).

Forma dos operadores relacionais

- Existem duas formas de se escrever um operador relacional;

Forma dos operadores relacionais

- Existem duas formas de se escrever um operador relacional;
- A forma da esquerda foi introduzida no Fortran 90, enquanto que a outra vem de versões anteriores. Vejamos o Exemplo 1;

Novo	Velho	Significado
==	.EQ.	Igual a?
/=	.NE.	Diferente de
>	.GT.	Maior que
>=	.GE.	Maior ou igual a
<	.LT.	Menor que
<=	.LE.	Menor ou igual a

Forma dos operadores relacionais

- Existem duas formas de se escrever um operador relacional;
- A forma da esquerda foi introduzida no Fortran 90, enquanto que a outra vem de versões anteriores. Vejamos o Exemplo 1;

Novo	Velho	Significado
==	.EQ.	Igual a?
/=	.NE.	Diferente de
>	.GT.	Maior que
>=	.GE.	Maior ou igual a
<	.LT.	Menor que
<=	.LE.	Menor ou igual a

- Expressões matemáticas são desenvolvidas antes dos operadores relacionais. Veja o 2;

Forma dos operadores relacionais

- Existem duas formas de se escrever um operador relacional;
- A forma da esquerda foi introduzida no Fortran 90, enquanto que a outra vem de versões anteriores. Vejamos o Exemplo 1;

Novo	Velho	Significado
==	.EQ.	Igual a?
/=	.NE.	Diferente de
>	.GT.	Maior que
>=	.GE.	Maior ou igual a
<	.LT.	Menor que
<=	.LE.	Menor ou igual a

- Expressões matemáticas são desenvolvidas antes dos operadores relacionais. Veja o 2;
- Ao comparar um valor real com um inteiro, o valor inteiro será convertido para real. Veja o Exemplo 3.

Operadores combinacionais

- São operadores com um ou dois operandos lógicos que produzem um resultado lógico;

Operadores combinacionais

- São operadores com um ou dois operandos lógicos que produzem um resultado lógico;
- O resultado depende da **combinação** entre estes dois operandos lógicos;

Operadores combinacionais

- São operadores com um ou dois operandos lógicos que produzem um resultado lógico;
- O resultado depende da **combinação** entre estes dois operandos lógicos;
- A forma geral de uma relação operador relacional é a seguinte:

A .op. B

Operadores combinacionais

- São operadores com um ou dois operandos lógicos que produzem um resultado lógico;
- O resultado depende da **combinação** entre estes dois operandos lógicos;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \ .op. \ B$$

- A e B podem ser:

Operadores combinacionais

- São operadores com um ou dois operandos lógicos que produzem um resultado lógico;
- O resultado depende da **combinação** entre estes dois operandos lógicos;
- A forma geral de uma relação operador relacional é a seguinte:

$$A\ .op.\ B$$

- A e B podem ser:
  - expressões lógicas;

Operadores combinacionais

- São operadores com um ou dois operandos lógicos que produzem um resultado lógico;
- O resultado depende da **combinação** entre estes dois operandos lógicos;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \ .op. \ B$$

- A e B podem ser:
  - expressões lógicas;
  - variáveis;

Operadores combinacionais

- São operadores com um ou dois operandos lógicos que produzem um resultado lógico;
- O resultado depende da **combinação** entre estes dois operandos lógicos;
- A forma geral de uma relação operador relacional é a seguinte:

$$A \ .op. \ B$$

- A e B podem ser:
  - expressões lógicas;
  - variáveis;
  - ou constantes.

Forma dos operadores combinacionais

- Os operadores e seu respectivo significado se encontram na tabela abaixo:

Operador	Função	Retorno TRUE se
A .AND. B	E lógico	A e B forem verdadeiros
A .OR. B	OU Lógico	A ou B forem verdadeiros
A .EQV. B	Equivalência lógica	A é o **mesmo** que B
A .NEQV. B	Não equivalência lógica	A é Verdadeiro e B é Falso
.NOT. B	Negação lógica	B é Falso

Forma dos operadores combinacionais

- Os operadores e seu respectivo significado se encontram na tabela abaixo:

Operador	Função	Retorno TRUE se
A .AND. B	E lógico	A e B forem verdadeiros
A .OR. B	OU Lógico	A ou B forem verdadeiros
A .EQV. B	Equivalência lógica	A é o **mesmo** que B
A .NEQV. B	Não equivalência lógica	A é Verdadeiro e B é Falso
.NOT. B	Negação lógica	B é Falso

- Vejamos o Exemplo 4;

Tabela verdade

- Os operadores lógicos abrem várias possibilidades de se construir a lógica de um programa;

Tabela verdade

- Os operadores lógicos abrem várias possibilidades de se construir a lógica de um programa;
- Uso do artifício da tabela verdade-sentenças matemáticas;

A	B	.AND.	.OR.	.EQV.	.NEQV.
V	V	V	V	V	F
V	F	F	V	F	V
F	V	F	V	F	V
F	F	F	F	V	F

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:
  1. Todos os operadores aritméticos;

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:
  1. Todos os operadores aritméticos;
  2. Todos os operadores relacionais, da esq. para a dir.;

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:
  1. Todos os operadores aritméticos;
  2. Todos os operadores relacionais, da esq. para a dir.;
  3. Todos os operadores de negação, .NOT.;

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:
  1. Todos os operadores aritméticos;
  2. Todos os operadores relacionais, da esq. para a dir.;
  3. Todos os operadores de negação, .NOT.;
  4. Todos os operadores .AND., da esq. para a dir.;

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:
  1. Todos os operadores aritméticos;
  2. Todos os operadores relacionais, da esq. para a dir.;
  3. Todos os operadores de negação, .NOT.;
  4. Todos os operadores .AND., da esq. para a dir.;
  5. Todos os operadores .OR., da esq. para a dir.;

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:
  1. Todos os operadores aritméticos;
  2. Todos os operadores relacionais, da esq. para a dir.;
  3. Todos os operadores de negação, .NOT.;
  4. Todos os operadores .AND., da esq. para a dir.;
  5. Todos os operadores .OR., da esq. para a dir.;
  6. Todos os operadores .EQV. e .NEQV., da esq. para a dir.;

Hierarquia geral dos operadores lógicos

- A ordem com que os operadores em uma expressão são avaliados é a seguinte:
  1. Todos os operadores aritméticos;
  2. Todos os operadores relacionais, da esq. para a dir.;
  3. Todos os operadores de negação, .NOT.;
  4. Todos os operadores .AND., da esq. para a dir.;
  5. Todos os operadores .OR., da esq. para a dir.;
  6. Todos os operadores .EQV. e .NEQV., da esq. para a dir.;
- Podemos também utilizar os parênteses para alterar a ordem natural.

Controle de ramificações

- As ramificações (branches) serão instruções que nos permitiram selecionar e executar seções específicas de código (chamados de blocos) enquanto pula para outras seções de código;

Controle de ramificações

- As ramificações (branches) serão instruções que nos permitiram selecionar e executar seções específicas de código (chamados de blocos) enquanto pula para outras seções de código;
- Há essencialmente dois tipos dessas instruções, o construtor de bloco IF e o construtor SELECT CASE;

Controle de ramificações

- As ramificações (branches) serão instruções que nos permitiram selecionar e executar seções específicas de código (chamados de blocos) enquanto pula para outras seções de código;
- Há essencialmente dois tipos dessas instruções, o construtor de bloco IF e o construtor SELECT CASE;
- Podemos também tornar a construção destes blocos mais semântica;

Controle de ramificações

- As ramificações (branches) serão instruções que nos permitiram selecionar e executar seções específicas de código (chamados de blocos) enquanto pula para outras seções de código;
- Há essencialmente dois tipos dessas instruções, o construtor de bloco IF e o construtor SELECT CASE;
- Podemos também tornar a construção destes blocos mais semântica;
- Além do controle intríseco dos construtores IF e SELECT CASE, podemos adicionar as intruções EXIT e STOP para direcionar/interromper o fluxo.

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.
  - A declaração END, ou então END IF, é obrigatória neste caso;

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.
  - A declaração END, ou então END IF, é obrigatória neste caso;
  - Bem como a declaração THEN logo após a condição lógica.

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.
  - A declaração END, ou então END IF, é obrigatória neste caso;
  - Bem como a declaração THEN logo após a condição lógica.
- Mas esta não é a única forma, podemos construir a instrução IF em apenas uma linha também, o que é totalmente equivalente a nível de compilador;

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.
  - A declaração END, ou então END IF, é obrigatória neste caso;
  - Bem como a declaração THEN logo após a condição lógica.
- Mas esta não é a única forma, podemos construir a instrução IF em apenas uma linha também, o que é totalmente equivalente a nível de compilador;
- Podemos também adicionar critérios de parada. Existem dois tipos, são eles:

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.
  - A declaração END, ou então END IF, é obrigatória neste caso;
  - Bem como a declaração THEN logo após a condição lógica.
- Mas esta não é a única forma, podemos construir a instrução IF em apenas uma linha também, o que é totalmente equivalente a nível de compilador;
- Podemos também adicionar critérios de parada. Existem dois tipos, são eles:
  - EXIT: O programa não executará o que estiver dentro do bloco IF, saltando para a proxima linha abaixo do bloco

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.
  - A declaração END, ou então END IF, é obrigatória neste caso;
  - Bem como a declaração THEN logo após a condição lógica.
- Mas esta não é a única forma, podemos construir a instrução IF em apenas uma linha também, o que é totalmente equivalente a nível de compilador;
- Podemos também adicionar critérios de parada. Existem dois tipos, são eles:
  - EXIT: O programa não executará o que estiver dentro do bloco IF, saltando para a proxima linha abaixo do bloco
  - STOP: Para (interrompe) todo o programa. Deve ser usado com cautela. Não é recomendado utilizar em sub-rotinas. Pode vir acompanhado de uma mensagem (recomendado principalmente quando houverem diversos STOP).

Bloco IF - instrução EXIT e STOP

- A forma mais comum de da instrução IF é através da construção de um bloco, que contém expressões que serão executadas se uma dada condição for verdadeira.
  - A declaração END, ou então END IF, é obrigatória neste caso;
  - Bem como a declaração THEN logo após a condição lógica.
- Mas esta não é a única forma, podemos construir a instrução IF em apenas uma linha também, o que é totalmente equivalente a nível de compilador;
- Podemos também adicionar critérios de parada. Existem dois tipos, são eles:
  - EXIT: O programa não executará o que estiver dentro do bloco IF, saltando para a proxima linha abaixo do bloco
  - STOP: Para (interrompe) todo o programa. Deve ser usado com cautela. Não é recomendado utilizar em sub-rotinas. Pode vir acompanhado de uma mensagem (recomendado principalmente quando houverem diversos STOP).
- Veja o Exemplo 6.

Cláusula ELSE e ELSE IF

- Por vezes, gostaríamos tanto de executar um bloco de código quando uma condição IF é verdadeira, como também outro bloco de código quando for falsa;

Cláusula ELSE e ELSE IF

- Por vezes, gostaríamos tanto de executar um bloco de código quando uma condição IF é verdadeira, como também outro bloco de código quando for falsa;
- A cláusula ELSE resolve este problema, podemos interpretá-la como significando: senão;

Cláusula ELSE e ELSE IF

- Por vezes, gostaríamos tanto de executar um bloco de código quando uma condição IF é verdadeira, como também outro bloco de código quando for falsa;
- A cláusula ELSE resolve este problema, podemos interpretá-la como significando: senão;
- Mas também em outras ocasiões, gostaríamos de executar mais do que dois blocos distintos de código;

Cláusula ELSE e ELSE IF

- Por vezes, gostaríamos tanto de executar um bloco de código quando uma condição IF é verdadeira, como também outro bloco de código quando for falsa;
- A cláusula ELSE resolve este problema, podemos interpretá-la como significando: senão;
- Mas também em outras ocasiões, gostaríamos de executar mais do que dois blocos distintos de código;
- Ou seja, quando gostaríamos de continuar testando depois do SENÃO (ELSE);

Cláusula ELSE e ELSE IF

- Por vezes, gostaríamos tanto de executar um bloco de código quando uma condição IF é verdadeira, como também outro bloco de código quando for falsa;
- A cláusula ELSE resolve este problema, podemos interpretá-la como significando: senão;
- Mas também em outras ocasiões, gostaríamos de executar mais do que dois blocos distintos de código;
- Ou seja, quando gostaríamos de continuar testando depois do SENÃO (ELSE);
- Para isso, utilizaremos a cláusula ELSE IF;

Cláusula ELSE e ELSE IF

- Por vezes, gostaríamos tanto de executar um bloco de código quando uma condição IF é verdadeira, como também outro bloco de código quando for falsa;
- A cláusula ELSE resolve este problema, podemos interpretá-la como significando: senão;
- Mas também em outras ocasiões, gostaríamos de executar mais do que dois blocos distintos de código;
- Ou seja, quando gostaríamos de continuar testando depois do SENÃO (ELSE);
- Para isso, utilizaremos a cláusula ELSE IF;
- O programa percorrerá **todos os blocos** de cima para baixo até atingir a condição verdadeira;

Cláusula ELSE e ELSE IF

- Por vezes, gostaríamos tanto de executar um bloco de código quando uma condição IF é verdadeira, como também outro bloco de código quando for falsa;
- A cláusula ELSE resolve este problema, podemos interpretá-la como significando: senão;
- Mas também em outras ocasiões, gostaríamos de executar mais do que dois blocos distintos de código;
- Ou seja, quando gostaríamos de continuar testando depois do SENÃO (ELSE);
- Para isso, utilizaremos a cláusula ELSE IF;
- O programa percorrerá **todos os blocos** de cima para baixo até atingir a condição verdadeira;
- Veja o Exemplo 7.

Tornando os blocos IF mais semânticos

- Em alguns casos teremos muitos blocos IF-ELSE encadeados (aninhados), o que poderá tornar difícil visualizar o que está acontecendo ali;

Tornando os blocos IF mais semânticos

- Em alguns casos teremos muitos blocos IF-ELSE encadeados (aninhados), o que poderá tornar difícil visualizar o que está acontecendo ali;
- Para resolver isso, podemos nomear os blocos IF-ELSE;

```
[name:] IF (logical_expr_1) THEN
  !Declaração 1
  !Declaração 2
  !...
ELSE IF (logical_expr_2) THEN [name]
  !Declaração 1
  !Declaração 2
  !...
ELSE [name]
  !Declaração 1
  !Declaração 2
  !...
END IF [name]
```

- Em [nome] poderemos utilizar caracteres alfanuméricos, desde que o primeiro **seja uma letra**;

- Em [nome] poderemos utilizar caracteres alfanuméricos, desde que o primeiro **seja uma letra**;
- [nome] deve ser **único** dentro de cada unidade de programa;

- Em [nome] poderemos utilizar caracteres alfanuméricos, desde que o primeiro **seja uma letra**;
- [nome] deve ser **único** dentro de cada unidade de programa;
- O nome do bloco deve estar contido na declaração END IF;

- Em [nome] poderemos utilizar caracteres alfanuméricos, desde que o primeiro **seja uma letra**;
- [nome] deve ser **único** dentro de cada unidade de programa;
- O nome do bloco deve estar contido na declaração END IF;
- Nas declaração ELSE e ELSE IF o nome do bloco é opcional;

- Em [nome] poderemos utilizar caracteres alfanuméricos, desde que o primeiro **seja uma letra**;
- [nome] deve ser **único** dentro de cada unidade de programa;
- O nome do bloco deve estar contido na declaração END IF;
- Nas declaração ELSE e ELSE IF o nome do bloco é opcional;
- Nomear os blocos tornará a intenção de um programador explicitamente claras;

- Em [nome] poderemos utilizar caracteres alfanuméricos, desde que o primeiro **seja uma letra**;
- [nome] deve ser **único** dentro de cada unidade de programa;
- O nome do bloco deve estar contido na declaração END IF;
- Nas declaração ELSE e ELSE IF o nome do bloco é opcional;
- Nomear os blocos tornará a intenção de um programador explicitamente claras;
- Os nomes também poderão auxiliar o compilador a sinalizar erros, pois ele sempre irá associar com o IF mais recente.

- Em [nome] poderemos utilizar caracteres alfanuméricos, desde que o primeiro **seja uma letra**;
- [nome] deve ser **único** dentro de cada unidade de programa;
- O nome do bloco deve estar contido na declaração END IF;
- Nas declaração ELSE e ELSE IF o nome do bloco é opcional;
- Nomear os blocos tornará a intenção de um programador explicitamente claras;
- Os nomes também poderão auxiliar o compilador a sinalizar erros, pois ele sempre irá associar com o IF mais recente.
- Veja o Exemplo 8.

Construtor SELECT CASE

- Possui um mecanismo muito semelhante ao construtor IF, mas com uma diferença sutil;

Construtor SELECT CASE

- Possui um mecanismo muito semelhante ao construtor IF, mas com uma diferença sutil;
- Suporta os seguintes tipos de dados de entrada:

Construtor SELECT CASE

- Possui um mecanismo muito semelhante ao construtor IF, mas com uma diferença sutil;
- Suporta os seguintes tipos de dados de entrada:
  - um único número inteiro;

Construtor SELECT CASE

- Possui um mecanismo muito semelhante ao construtor IF, mas com uma diferença sutil;
- Suporta os seguintes tipos de dados de entrada:
  - um único número inteiro;
  - caractere;

Construtor SELECT CASE

- Possui um mecanismo muito semelhante ao construtor IF, mas com uma diferença sutil;
- Suporta os seguintes tipos de dados de entrada:
  - um único número inteiro;
  - caractere;
  - uma expressão lógica;

Construtor SELECT CASE

- Possui um mecanismo muito semelhante ao construtor IF, mas com uma diferença sutil;
- Suporta os seguintes tipos de dados de entrada:
  - um único número inteiro;
  - caractere;
  - uma expressão lógica;
  - ou um intervalo de valores.

Construtor SELECT CASE

- Possui um mecanismo muito semelhante ao construtor IF, mas com uma diferença sutil;
- Suporta os seguintes tipos de dados de entrada:
  - um único número inteiro;
  - caractere;
  - uma expressão lógica;
  - ou um intervalo de valores.
- Veja o Exemplo 9.

```
SELECT CASE (case_expr)
CASE (case_selector_1)
  !Declaração
CASE (case_selector_2)
  !Declaração
!...
CASE DEFAULT
  !Declaração
END SELECT
```

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.
- Portanto, é importante usufruir de ferramentas que verifiquem o código, pedaço por pedaço;

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.
- Portanto, é importante usufruir de ferramentas que verifiquem o código, pedaço por pedaço;
  - Uma delas é utilizar o depurador;

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.
- Portanto, é importante usufruir de ferramentas que verifiquem o código, pedaço por pedaço;
  - Uma delas é utilizar o depurador;
  - Ou então, incluindo declarações PRINT ou WRITE em pontos específicos do código.

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.
- Portanto, é importante usufruir de ferramentas que verifiquem o código, pedaço por pedaço;
  - Uma delas é utilizar o depurador;
  - Ou então, incluindo declarações PRINT ou WRITE em pontos específicos do código.
- O problema do depurador, é que as instruções e seu uso, dependerá do compilador em específico;

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.
- Portanto, é importante usufruir de ferramentas que verifiquem o código, pedaço por pedaço;
  - Uma delas é utilizar o depurador;
  - Ou então, incluindo declarações PRINT ou WRITE em pontos específicos do código.
- O problema do depurador, é que as instruções e seu uso, dependerá do compilador em específico;
- A princípio, o LINUX já vem com um depurador, o GDB;

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.
- Portanto, é importante usufruir de ferramentas que verifiquem o código, pedaço por pedaço;
  - Uma delas é utilizar o depurador;
  - Ou então, incluindo declarações PRINT ou WRITE em pontos específicos do código.
- O problema do depurador, é que as instruções e seu uso, dependerá do compilador em específico;
- A princípio, o LINUX já vem com um depurador, o GDB;
- Que é mais utilizado em linguagens como C/C++, mas que pode ser utilizado para o Fortran;

Debugging

- Quanto mais complexo vai ficando o código, maiores serão as chances de ter erros não esperados:
  - Seja erros de sintaxe, ao unir os pequenos blocos do código;
  - Seja erros de tempo de execução;
  - Seja também, a escrita correto, mas que não dá o resultado esperado.
- Portanto, é importante usufruir de ferramentas que verifiquem o código, pedaço por pedaço;
  - Uma delas é utilizar o depurador;
  - Ou então, incluindo declarações PRINT ou WRITE em pontos específicos do código.
- O problema do depurador, é que as instruções e seu uso, dependerá do compilador em específico;
- A princípio, o LINUX já vem com um depurador, o GDB;
- Que é mais utilizado em linguagens como C/C++, mas que pode ser utilizado para o Fortran;
- No editor VS CODE, há uma extenção, Fortran Breakpoint Support, do Ekibun.

Algumas dicas de checagem

- Se o problema estiver numa instrução IF, verique se utilizou a condição (a lógica) de forma correta. Expressões lógicas são complexas, pois são difíceis de entender, e muito fáceis para bagunçar o código;

Algumas dicas de checagem

- Se o problema estiver numa instrução IF, verique se utilizou a condição (a lógica) de forma correta. Expressões lógicas são complexas, pois são difíceis de entender, e muito fáceis para bagunçar o código;
- Utilize parênteses extras em condições relacionais para torná-las mais fáceis de se entender;

Algumas dicas de checagem

- Se o problema estiver numa instrução IF, verique se utilizou a condição (a lógica) de forma correta. Expressões lógicas são complexas, pois são difíceis de entender, e muito fáceis para bagunçar o código;
- Utilize parênteses extras em condições relacionais para torná-las mais fáceis de se entender;
- Se suas expressões lógicas estsão muito grandes, considere dividí-las em mais sentenças mais simples e curtas;

Algumas dicas de checagem

- Se o problema estiver numa instrução IF, verique se utilizou a condição (a lógica) de forma correta. Expressões lógicas são complexas, pois são difíceis de entender, e muito fáceis para bagunçar o código;
- Utilize parênteses extras em condições relacionais para torná-las mais fáceis de se entender;
- Se suas expressões lógicas estsão muito grandes, considere dividí-las em mais sentenças mais simples e curtas;
- Ao trabalhar com valores reais, substitua testes de igualdade, pelos de quase igualdade , por causa dos erros de arredondamento;

Algumas dicas de checagem

- Se o problema estiver numa instrução IF, verique se utilizou a condição (a lógica) de forma correta. Expressões lógicas são complexas, pois são difíceis de entender, e muito fáceis para bagunçar o código;
- Utilize parênteses extras em condições relacionais para torná-las mais fáceis de se entender;
- Se suas expressões lógicas estsão muito grandes, considere dividí-las em mais sentenças mais simples e curtas;
- Ao trabalhar com valores reais, substitua testes de igualdade, pelos de quase igualdade , por causa dos erros de arredondamento;
- Por exemplo, ao invés de teste se $x = 2$, considere testar

$$|x - 2| < 0,0001$$

Algumas dicas de checagem

- Se o problema estiver numa instrução IF, verique se utilizou a condição (a lógica) de forma correta. Expressões lógicas são complexas, pois são difíceis de entender, e muito fáceis para bagunçar o código;
- Utilize parênteses extras em condições relacionais para torná-las mais fáceis de se entender;
- Se suas expressões lógicas estsão muito grandes, considere dividí-las em mais sentenças mais simples e curtas;
- Ao trabalhar com valores reais, substitua testes de igualdade, pelos de quase igualdade , por causa dos erros de arredondamento;
- Por exemplo, ao invés de teste se $x = 2$, considere testar

$$|x - 2| < 0,0001$$

- Qualquer valor entre $1,9999$ e $2,0001$ será contemplado.