Departamento de Informática – UFPR

Algoritmos e Estruturas de Dados I

Prof.ª Dr.ª Mariane Cassenote

mariane@inf.ufpr.br

Aninhamento de desvios condicionais

Objetivos da aula

1. Apresentar o aninhamento de desvios condicionais
2. Explicar problemas que podem ocorrer
3. Conceito de programação estruturada

Aula elaborada com base na bibliografia da disciplina e nos materiais dos Profs. Marcos Castilho e André Vignatti, disponíveis no link

https://www.inf.ufpr.br/cursos/ci055/

Conceitos elementares de linguagens de programação

- Fluxo de execução de um programa
- Comandos que manipulam dados e permitem interação com o usuário
- Expressões aritméticas e lógicas
- **Comandos que permitem alteração do fluxo de execução do programa**
  - Repetição
  - **Condicional**

O comando if, segunda versão

```
1    if { expressao booleana } then
2        { algum comando }
3    else
4        { outro comando }
```

- Lembrando a sintaxe e semântica
- Por quê tem ser ser “outro” comando?
- Estes comandos podem ser outro `if` ?

Aninhamento de desvios

```
1  program bhaskara_v6;
2  var a, b, c, delta: real;
3
4  begin
5      read (a, b, c);
6      delta:= b * b - 4 * a * c;
7      if delta = 0 then
8          writeln (-b / (2 * a))
9      else
10         if delta > 0 then
11         begin
12             writeln ((-b - sqrt(delta)) / (2 * a));
13             writeln ((-b + sqrt(delta)) / (2 * a));
14         end
15         else (* aqui delta eh negativo *)
16             writeln ('nao existem raizes reais');
17 end.
```

Cuidado com aninhamento de `if-then-else`

Problema: Ler dois números do teclado e se o segundo for positivo, imprimir a divisão dele pelo primeiro. O programa deve imprimir uma mensagem se houver uma divisão por zero.

- Este problema parece muito simples
- Mas permite explorar um erro muito comum quando se aninha uma sequência de **`if-then-else`**
- Vejamos a solução errada primeiro

Solução errada

```
1  program dividir_b_por_a_errado;
2  var a, b: longint;
3
4  begin
5      read (a, b);
6      if a <> 0 then
7          if b > 0 then
8              writeln (b / a)
9      else
10         writeln ('divisao por zero');
11 end.
```

Solução errada

```
1  program dividir_b_por_a_errado;
2  var a, b: longint;
3
4  begin
5      read (a, b);
6      if a <> 0 then
7          if b > 0 then
8              writeln (b / a)
9      else
10         writeln ('divisao por zero');
11 end.
```

- O erro está em acreditar que o compilador reconhece indentação
- Neste programa, o **else** é do segundo **if**, não do primeiro!
- Percebe-se a intenção do programador, que fez a indentação que ele achou certa
- Mas o compilador não reconhece intenções!

Solução errada, com indentação certa

```
1  program dividir_b_por_a_errado_v2;
2  var a, b: longint;
3
4  begin
5      read (a, b);
6      if a <> 0 then
7          if b > 0 then
8              writeln (b / a)
9          else
10             writeln ('divisao por zero')
11 end.
```

- Mesmo programa com a identação certa
- O **`else`** é do **`if`** “mais próximo”!
- Mas o programa continua errado!!!

Solução errada, com indentação certa

```
mrscassenote@swift3:~/algoritmos$ ./dividir_b_por_a_errado
0 2
mrscassenote@swift3:~/algoritmos$
```

- Deveria ter impresso a mensagem de erro
- O que prova que o else está errado

Correção

- Existem algumas maneiras de corrigir isso
- Veremos duas delas

Correção com begin-end

```
1  program dividir_b_por_a_certo_v1;
2  var a, b: longint;
3
4  begin
5      read (a, b);
6      if a <> 0 then
7      begin
8          if b > 0 then
9              writeln (b / a)
10     end
11     else
12         writeln ('divisao por zero')
13 end.
```

- Usando **`begin-end;`** se cria um bloco que separa claramente o **`else`**

Correção por lógica de programação

```
1  program dividir_b_por_a_certo_v3;
2  var a, b: longint;
3
4  begin
5      read (a, b);
6      if a = 0 then
7          writeln ('divisao por zero')
8      else
9          if b > 0 then
10             writeln (b / a);
11 end.
```

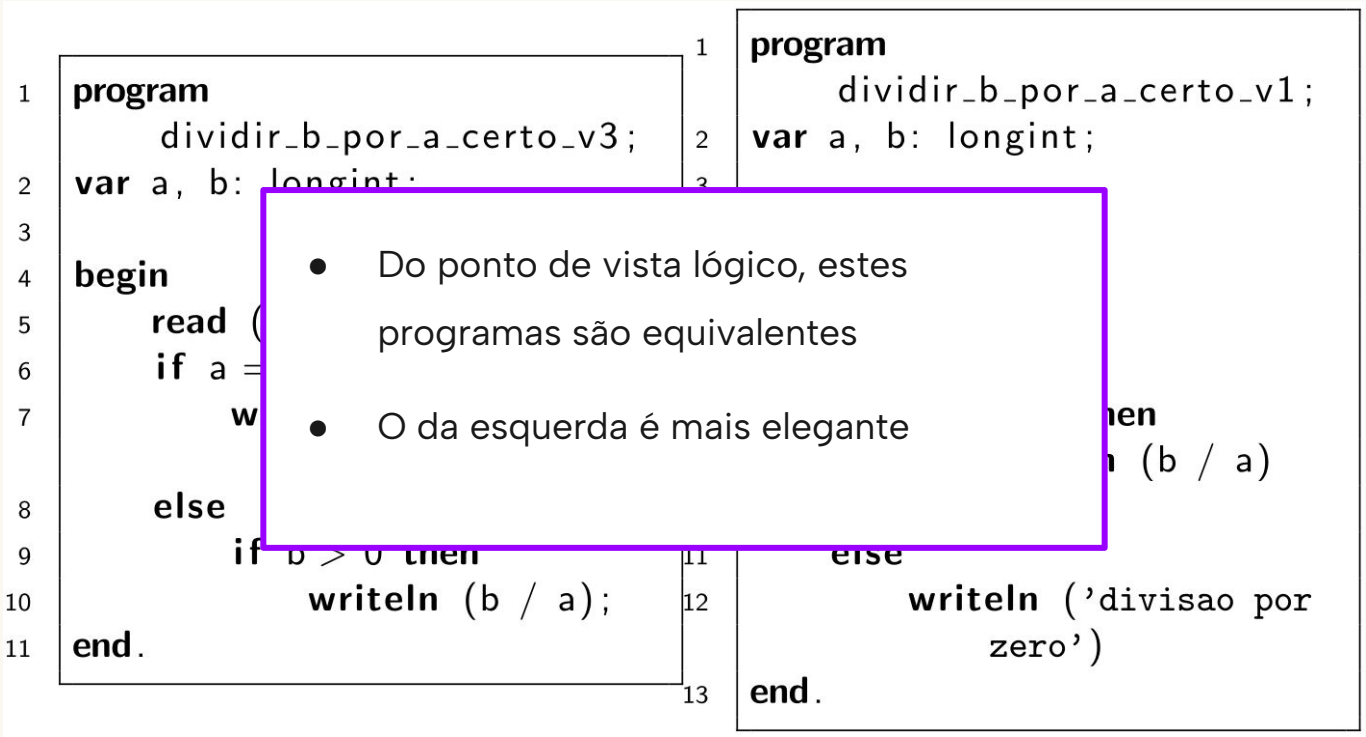
- Usa-se lógica de programação
- Basta inverter o primeiro **`if`**, testando-se a negação da primeira condição
- O que era **`else`** agora é o **`then`**
- É a solução mais elegante de todas

Comparando as soluções

```
1  program
       dividir_b_por_a_certo_v3;
2  var a, b: longint;
3
4  begin
5      read (a, b);
6      if a = 0 then
7          writeln ('divisao por
               zero')
8      else
9          if b > 0 then
10             writeln (b / a);
11 end.
```

```
1  program
       dividir_b_por_a_certo_v1;
2  var a, b: longint;
3
4  begin
5      read (a, b);
6      if a <> 0 then
7      begin
8          if b > 0 then
9              writeln (b / a)
10     end
11     else
12         writeln ('divisao por
               zero')
13 end.
```

Comparando as soluções



```
1  program
       dividir_b_por_a_certo_v3;
2  var a, b: longint;
3
4  begin
5      read (
6      if a =
7          w
8      else
9          if b > 0 then
10             writeln (b / a);
11 end.
```

```
1  program
       dividir_b_por_a_certo_v1;
2  var a, b: longint;
3
                                  hen
                                  n (b / a)
11     else
12         writeln ('divisao por
               zero')
13 end.
```

- Do ponto de vista lógico, estes programas são equivalentes
- O da esquerda é mais elegante

Apêndice: desvio incondicional

```
1  program contar_com_goto;
2  label 10;
3  var i: longint;
4
5  begin
6      i:= 1;
7  10: writeln (i);
8      i:= i + 1;
9      if i <= 30 then
10       goto 10;      (* Nunca usaremos! *)
11 end.
```

- Desvantagem: confunde compreensão do programa
- Vantagem: útil em raríssimas situações

Com e sem `goto`

```
1  program contar_com_goto;
2  label 10;
3  var i: longint;
4
5  begin
6      i:= 1;
7  10: writeln (i);
8      i:= i + 1;
9      if i <= 30 then
10       goto 10;      (* Nunca
             usaremos! *)
11 end.
```

```
1  program contar_sem_goto;
2  var i: longint;
3
4  begin
5      i:= 1;
6      while i <= 30 do
7      begin
8          writeln (i);
9          i := i + 1;
10     end;
11 end.
```

Conclusão

- Já temos todos os elementos básicos de linguagens de programação
  - Fluxo de execução de programas
  - Expressões aritméticas e booleanas
  - Comandos básicos
    - Atribuição
    - Entrada e Saıída
    - Repetição
    - Desvio condicional
- Com estes elementos é possível fazer qualquer algoritmo

Fim do tópico

- A parte do `goto` está no apêndice do capítulo 5 do livro
- Na próxima aula veremos técnicas elementares de construção de algoritmos

Departamento de Informática – UFPR

Algoritmos e Estruturas de Dados I

Prof.ª Dr.ª Mariane Cassenote

mariane@inf.ufpr.br

Aninhamento de desvios condicionais