CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho e Bruno Müller Jr

Departamento de Informática/UFPR

18 de julho de 2020

Resumo

Aninhamento de desvios condicionais

Objetivos da aula

- apresentar o aninhamento de desvios condicionais
- explicar problemas que podem ocorrer
- conceito de programação estruturada

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. Mas 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.
```

- o erro está em acreditar que o compilador reconhece identação
- neste programa, o `else` é do segundo `if`, não do primeiro!
- percebe-se a intenção do programador, que fez a identação que ele achou certa
- mas o compilador não reconhece intenções

Solução errada, com identação 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

```
marcos@tuco ~ $ ./dividir_b_por_a_errado
0 2
marcos@tuco ~ $
```

- 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-se `begin-end;`
- cria-se 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`
- para mim, 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.
```

- do ponto de vista lógico, estes programas são equivalentes
- o da esquerda é mais elegante

Apêndice: desvio incondicional

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

```
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.
```

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 saída
    - 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

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`