CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

13 de agosto de 2020

Resumo

Outros comandos de repetição

Objetivos da aula

- Apresentar os comandos de repetição `for` e `repeat`
- Compará-los com o `while`

Comandos de repetição em *Pascal*

- `while/do`
- `repeat/until`
- `for/do`

Comando *while*

- Como já vimos, o `while` provoca uma repetição controlada dos comandos sob seu escopo *enquanto* uma expressão booleana for verdadeira.

```
1  (* inicializacao de variaveis *)
2  while (* condicao verdadeira *) do
3  begin
4      (* comandos *)
5  end;
```

Comando *repeat*

- O comando `repeat` provoca uma repetição controlada dos comandos sob seu escopo *até que* uma expressão booleana se torne verdadeira.

```
1   (* inicializacao de variaveis *)
2   repeat
3       (* comandos *)
4   until (* condicao verdadeira *);
```

Diferenças entre *while* e *repeat*

- No `while/do`:
  - É necessário colocar `begin/end;` para repetir mais de um comando
  - A expressão booleana é avaliada no início do laço
  - Por isso, os comandos podem não ser executados nenhuma vez
- No `repeat/until`:
  - Não é necessário colocar `begin/end;` para repetir mais de um comando
  - A expressão booleana é no final do laço
  - Por isso, os comandos sempre serão executados pelo menos uma vez

Comparando *while* com *repeat*

- A expressão booleana do `repeat` é a negação daquela do `while`
- Isto captura a diferença semântica entre *enquanto* e *até que*

Estes dois programas imprimem os números entre 1 e $n \geq 1$:

```
1  program exemplo_while;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      while i <= n do
7      begin
8          writeln (i);
9          i:= i + 1;
10     end;
11 end.
```

```
1  program exemplo_repeat_v0;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      repeat
7          writeln (i);
8          i:= i + 1;
9      until i > n;
10 end.
```

Comparando *while* com *repeat*

- Os comandos do `while` podem não ocorrer
- Os comandos do `repeat` sempre ocorrem pelo menos uma vez

Estes dois programas não fazem a mesma coisa se $n = 0$:

```
1  program exemplo_while;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      while i <= n do
7      begin
8          writeln (i);
9          i:= i + 1;
10     end;
11 end.
```

se $n = 0$ não imprime nada

```
1  program exemplo_repeat_v0;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      repeat
7          writeln (i);
8          i:= i + 1;
9      until i > n;
10 end.
```

se $n = 0$ imprime 1

Comparando *while* com *repeat*

Estes dois programas são equivalentes

```
1  program exemplo_while;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      while i <= n do
7      begin
8          writeln (i);
9          i:= i + 1;
10     end;
11 end.
```

```
1  program exemplo_repeat_v1;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      if i <= n then
7          repeat
8              writeln (i);
9              i:= i + 1;
10         until i > n;
11 end.
```

Comando *for*

- O comando `for` provoca uma repetição controlada dos comandos sob seu escopo baseada em uma enumeração

```
1   for VAR := INI to FIM do
2   begin
3       (* comandos *)
4   end;
```

Comando *for*

- *VAR* é uma variável do tipo ordinal (integer,longint, . . .)
- *INI* e *FIM* são expressões aritméticas que resultam em um valor do tipo ordinal
- Os comandos do laço serão executados *exatamente* $FIM - INI + 1$ vezes
- *VAR* é a variável de controle do laço:
  - ela é inicializada com o valor de *INI*
  - a cada iteração *VAR* é incrementada em uma unidade
  - o laço termina quando $VAR > FIM$ (isto é pré avaliado, conforme veremos)

```
1   for VAR := INI to FIM do
2   begin
3       (* comandos *)
4   end;
```

Comparando *while* com *for*

- O `for` pode ser facilmente simulado por um `while`

Estes dois programas imprimem os números entre 1 e n e são equivalentes:

```
1  program exemplo_while;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      while i <= n do
7      begin
8          writeln (i);
9          i:= i + 1;
10     end;
11 end.
```

```
1  program exemplo_for;
2  var i, n: integer;
3  begin
4      read (n);
5      for i:= 1 to n do
6          writeln (i);
7  end.
```

Comparando *while* com *for*

- O `for` nem sempre pode substituir um `while`
- *Pascal* não permite alterar a variável de controle do laço, sequer compila
- *Pascal* pré determina quantas vezes o laço será executado
- não adianta tentar forçar a saída do laço em *Pascal*

Comparando *while* com *for*

Exemplo: Entre com $n = 5$, na primeira iteração n é alterado para zero, mas o laço executa 5 vezes!

```
1  program exemplo_for_v1;
2  var i, n: integer;
3  begin
4      read (n);
5      for i:= 1 to n do
6      begin
7          writeln ('i= ',i,' n= ',n);
8          //i:= n+1; (* nao compila *)
9          n:= 0;    (* nao adianta forcar a saida *)
10     end;
11 end.
```

Comparando *while* com *for*

Em *Pascal* estes programas não são equivalentes se $n \geq 1$!

- Se $n = 5$, o `while` executa uma vez
- Se $n = 5$, o `for` executa cinco vezes

```
1  program exemplo_while_v2;
2  var i, n: integer;
3  begin
4      read (n);
5      i:= 1;
6      while i <= n do
7      begin
8          writeln (i);
9          n:= 0;
10         i:= i + 1;
11     end;
12 end.
```

```
1  program exemplo_for;
2  var i, n: integer;
3  begin
4      read (n);
5      for i:= 1 to n do
6          writeln (i);
7  end.
```

Comparando *while* com *for*

- O `for` nem sempre pode substituir um `while`
- O `for` só existe para laços que rodam um número exato de vezes
- Ele não se aplica quando este número não é conhecido

Este programa não pode ser feito usando-se *for*

```
1  program exemplo_while_v2;
2  var i, n: integer;
3  begin
4      read (n);
5      while n <> 0 do
6      begin
7          writeln (n);
8          read (n);
9      end;
10 end.
```

Outras observações

- O `for` pode não executar os comandos do laço se *INI* $>$ *FIM*
- *INI* e *FIM* podem ser expressões aritméticas do tipo ordinal
- Elas são avaliadas *antes* do laço executar

```
1  program exemplo_for_v2;
2  var i, n: integer;
3  begin
4      read (n);
5      for i:= (n div 2) to trunc (sqrt(n) + 3) do
6          writeln (i);
7  end.
```

Outra forma do *for*

- O `for` pode fazer decremento de uma unidade

Este programa imprime os números de n até 1

```
1  program exemplo_for_downto;
2  var i, n: integer;
3  begin
4      read (n);
5      for i:= n downto 1 do
6          writeln (i);
7  end.
```

Conclusão

- O comando mais genérico de todos é o `while`, com ele é possível fazer qualquer programa
- O comando `repeat` é útil quando se quer executar os comandos pelo menos uma vez
- O comando `for` em *Pascal* é diferente de outros `for` em outras linguagens de programação
- Se quiser incrementar de 2 em 2? Não use `for` !
- Outras linguagens de programação possuem ainda outras variantes de comandos de repetição, tais como um `do/while` ou um `until/do`

Fim do tópico

- este material não está no livro, é um tópico adicional

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/`