

# Estruturas de repetição

Marina Andretta

ICMC-USP

13 de março de 2016

# Comando **enquanto**

Em pseudo-código, a sintaxe do comando **enquanto** é:

**enquanto** (<condição>) **faca**

    <comando 1>

    <comando 2>

    ⋮

    <comando k>

**fimenquanto**

# Comando enquanto

A <condição> deve ser uma expressão lógica, ou seja, deve ter como resultado um valor verdadeiro ou falso.

Enquanto o resultado da expressão for **verdadeiro**, os comandos <comando 1>, <comando 2>, ..., <comando k> são executados (nesta ordem).

Quando o resultado da expressão é **falso**, passa-se ao **fimenquanto**.

Ou seja, se o resultado da expressão é verdadeiro, são executados os comandos <comando 1>, <comando 2>, ..., <comando k> e, por fim, verifica-se novamente a <condição>. Se ela for verdadeira, os comandos são executados novamente. Este processo é repetido até que a <condição> se torne falsa.

Um exemplo do uso do comando **enquanto** é o seguinte:

```
x ← 1
enquanto (x <= 5) faca
    escreva(x, ‘\n’)
    x ← x + 1
fimenquanto
```

# Comando **while**

Em linguagem C, o equivalente ao **enquanto** é o comando **while**, que tem a seguinte sintaxe:

```
while (<condição>) {  
    <comando 1>;  
    <comando 2>;  
    :  
    <comando k>;  
}
```

O exemplo dado em pseudo-código traduzido para linguagem C fica:

```
x = 1;
while (x <= 5) {
    printf(“%d\n”, x);
    x = x + 1
}
```

# Comando **for**

Em linguagem C, podemos usar o comando **for** para substituir o comando **while**. Ele tem a seguinte sintaxe:

```
for (<comandos iniciais>; <condição>; <comandos finais>) {  
    <comando 1>;  
    <comando 2>;  
    ⋮  
    <comando k>;  
}
```

# Comando **for**

No comando **for**, primeiro são executados os <comandos iniciais>.

Depois, enquanto a <condição> for verdadeira, são executados os comandos <comando 1>, <comando 2>, ..., <comando k>, seguidos dos <comandos finais>.

Se a <condição> for falsa, o laço encerra-se e passam-se aos comandos que estão depois da chave }.



Tanto os <comandos iniciais> como os <comandos finais> são compostos por 0 ou mais comandos em linguagem C, separados por vírgulas.

# Comando **for**

O equivalente ao comando **for**, usando o comando **while** é

```
<comandos iniciais>;  
while (<condição>) {  
    <comando 1>;  
    <comando 2>;  
    ⋮  
    <comando k>;  
    <comandos finais>;  
}
```

Neste caso, os comandos contidos tanto nos <comandos iniciais> como nos <comandos finais> são separados por ponto-e-vírgulas.

O exemplo anterior pode ser escrito com o comando **for** da seguinte maneira:

```
for (x = 1; x <= 5; x = x + 1) {  
    printf(“%d\n”, x);  
}
```

A sintaxe do comando **repita**, em pseudo-código é:

## **repita**

<comando 1>

<comando 2>

⋮

<comando k>

**enquanto** (<condição>)

# Comando **repita**

A <condição> deve ser uma expressão lógica, ou seja, deve ter como resultado um valor verdadeiro ou falso.

Primeiramente são executados os comandos <comando 1>, <comando 2>, ..., <comando k> são executados (nesta ordem).

Depois, verifica-se o resultado da expressão <condição>. Se for **verdadeiro**, <comando 1>, <comando 2>, ..., <comando k> são executados novamente.

Quando o resultado da expressão é **falso**, passa-se aos comandos após o **enquanto**.

# Comando **repita**

Basicamente, a diferença entre os comandos **enquanto** e **repita** está no momento em que é verificado o valor da expressão <condição>.

No comando **enquanto**, primeiro verifica-se se a <condição> é verdadeira, para depois serem executados os comandos <comando 1>, <comando 2>, ..., <comando k>.

No comando **repita**, são executados os comandos <comando 1>, <comando 2>, ..., <comando k> para depois verificar-se se a <condição> é verdadeira.

Um exemplo do uso do comando **repita** é o seguinte:

```
repita
```

```
    leia(nota)
```

```
enquanto (nota < 0)
```

# Comando **do..while**

Em linguagem C, o equivalente ao **repita** é o comando **do..while**, que tem a seguinte sintaxe:

```
do {  
    <comando 1>;  
    <comando 2>;  
    ⋮  
    <comando k>;  
} while (<condição>;
```



O exemplo dado em pseudo-código traduzido para linguagem C fica:

```
do {  
    scanf("%f", &nota);  
} while (nota < 0);
```

# Comando **para**

Em pseudo-código, a sintaxe do comando **para** é:

```
para <var> ← <v.inicial> ate <v.final> passo <v.passo>  
  <comando 1>  
  <comando 2>  
  ⋮  
  <comando k>  
fimpara
```

# Comando para

`<var>` deve ser uma variável inteira, `<v.inicial>`, `<v.final>` e `<v.passo>` devem ser valores inteiros.

Primeiramente, é atribuído o valor `<v.inicial>` à variável `<var>`.

Enquanto o valor da variável `<var>` for menor ou igual a `<v.final>`, são executados os comandos `<comando 1>`, `<comando 2>`, ..., `<comando k>` (nesta ordem).

Ao fim da execução destes comandos, é atribuído à variável `<var>` o valor `<var> + <v.passo>` e o processo se repete.

Quando o valor da variável `<var>` é maior do que `<v.final>`, passa-se ao **fimpara**.

Um exemplo do uso do comando **para** é o seguinte:

```
para x ← 1 ate 10 passo 2  
    escreva(x, ‘\n’)  
fimpara
```

# Comando **for**

Em linguagem C não existe uma estrutura específica para representar o comando **para**, mas ele pode ser facilmente simulado usando-se o comando **for**:

```
for (<var> = <v.inicial>; <var> <= <v.final>;  
<var>=<var>+<v.passo>) {  
    <comando 1>;  
    <comando 2>;  
    ⋮  
    <comando k>;  
}
```

O mesmo exemplo pode ser escrito da seguinte maneira em linguagem C:

```
for (x = 1; x <= 10; x = x + 2) {  
    printf(“%d\n”, x);  
}
```