Estruturas de Controle do Fluxo de Execução: Repetição (Iteração)

1. Introducão

As Estruturas de Seleção, como vistas no capítulo anterior, tem o propósito de isolar um conjunto de instruções do fluxo normal de execução do algoritmo. Essas instruções que foram isoladas somente serão executadas caso uma condição seja satisfeita. Esse isolamento é feito através de palavras reservadas que marcam o início e o fim de bloco de instruções, também conhecido como escopo.

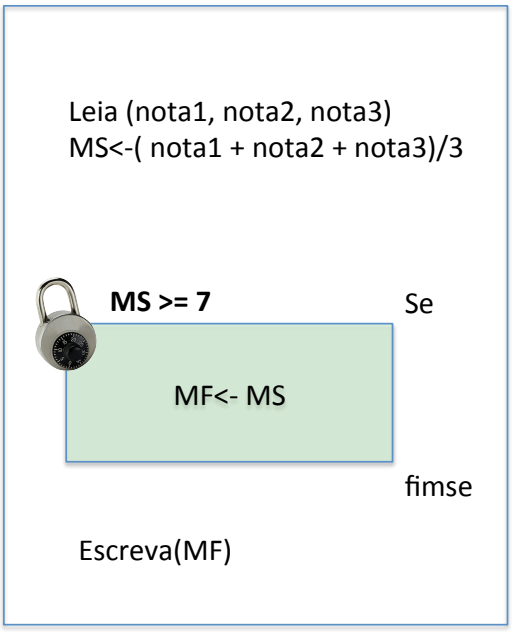
As Estruturas de Iteração (não confundir com interação) também possuem escopo, ou seja, possuem início e fim. Todos os comandos e estruturas que estiverem contidas dentro do escopo dessas estruturas serão repetidas. O número de repetições pode estar explicito vinculado a uma condição. Sendo assim, pode-se classificar as estruturas de iteração em dois grupos: a) Estruturas de Repetição com Condicional (abertas); e b) Estruturas de Repetição Pré-fixadas(fechadas).

A seguir veremos mais sobre esses dois tipos de estruturas de iteração/repetição.

2. Estruturas de Repetição Indeterminadas com Condicional

Se uma condição for verdadeira então um conjunto de ações são executadas. Esse conjunto de ações formam um bloco que tem início e fim bem definido, ou seja, pertencem ao escopo da estrutura. No caso de estruturas condicionais simples representamos o início e fim desse escopo com as palavras reservadas “se” e “fimse” respectivamente.

Vamos voltar ao exemplo do capítulo anterior no qual calculávamos a média final de um aluno. Nós sabemos que as notas dos alunos somente são válidas se estiveram no intervalo entre 0 e 10. Logo, qualquer nota que tiver sido digitada fora desse intervalo precisa ser novamente informada pelo usuário, pois deve ter havido um erro. Isso não está sendo feito no exemplo abaixo!




```
Inicio

  Leia (nota1, nota2, nota3)
  MS<-( nota1 + nota2 + nota3)/3

  Leia(provaFinal)
  MF<-(MS*0.6) + (provaFinal * 0.4)

  Se (MS >= 7) entao

    MF<-MS

  Fimse

  Escreva(MF)
Fimalgoritmo
```

Com as estruturas condicionais sem repetição somente é possível verificar a condição um única vez. Mesmo que seja possível identificar se o usuário digitou um valor fora da faixa especificada, não é possível predizer quantas vezes o usuário irá insistir no erro. Observe:

```
//. . . (algum código precede esse trecho)  . . .

leia(nota1)
Se (nota1 < 0)  OU (nota1 > 10) entao
Escreva( "Voce digitou um valor fora da faixa esperada. Tente
novamente:")
Leia(nota1)
Fimse
//. . . (algum código procede esse trecho) . . .
```

No exemplo acima, nós verificamos se o valor informado para a variável *nota1* é inferior a zero ou superior a dez. Se for o caso, então nós informamos a inconformidade ao usuário e tentamos mais **UMA** vez. O que acontecerá se o usuário digitar errado pela segunda vez? O número de vezes que o usuário pode errar a entrada da informação pode ser infinito! Como tratar isso?

2.1. Ciclo de repetição com verificação adiantada (Enquanto)

Para resolver o problema acima descrito basta trocar a estrutura condicional simples, por uma estrutura condicional indeterminada com condicional. A estrutura é dita indetermina porque o número de vezes que o usuário pode digitar um valor fora da faixa não é possível de ser determinado – isso vai ser

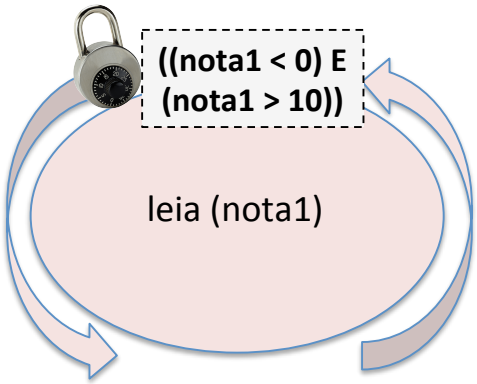
decidido por uma condição, que nesse caso, é a pergunta se o valor está entre 0 e 10; Observe:

```
//. . . (algum código precede esse trecho)  . . .

leia(nota1)
Enquanto ((nota1 < 0)  OU (nota1 > 10) )faca
   Escreva( "Voce digitou um valor fora da faixa esperada. Tente
novamente:")
   Leia(nota1)
Fimenquanto

//. . . (algum código procede esse trecho) . . .
```

Sintaxe do "Enquanto":

```
Enquanto (<expressão lógica>) faca
  <comandos>
fimenquanto
```

Observe que os comandos envoltos pela estrutura "enquanto" somente são executados se a condição previamente testada for verdadeira, sendo que uma vez que a execução termina, a condição é novamente verificada e, caso ainda seja verdadeira, os comandos são novamente executados. Esse processo ocorre indefinidamente até que a condição se torne falsa. Caso a condição nunca se torne falsa, então nunca será possível interromper o ciclo – uma anomalia conhecida como loop infinito. Logo, para que o ciclo possa se encerrar é muito importante observar se existe algum comando no escopo de execução do ciclo que altere a(s) variável(eis) da condição. Note que na ilustração abaixo o variável nota1 é novamente lida, dando-lhe a oportunidade de ter o valor modificado.

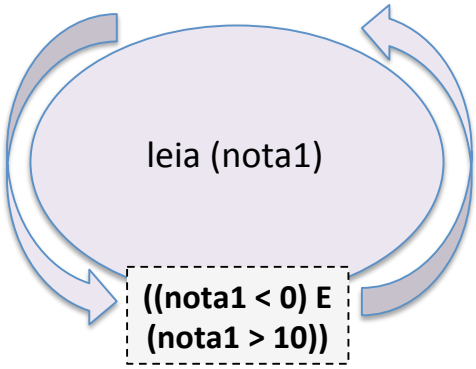


A notação gráfica acima proposta ilustra não somente o aspecto da repetição mas também o momento em que a condição é testada, nesse caso antes de

iniciar o ciclo. Logo, o ciclo só é iniciado se a condição for verdadeira e só se encerra quando a condição se tornar falsa. Para que seja possível tal verificação, as possíveis variáveis que compõem a expressão lógica, nesse caso *nota1*, devem contem um valor. Em outras palavras, antes de podermos determinar se a *nota1* se encontra no intervalo esperado, primeiro precisamos saber o valor de nota1.

2.2. Ciclo de repetição com verificação tardia (Repita)

Ao contrário da estrutura "Enquanto", em que a condição testada deve ser verdadeira para se iniciar o ciclo, a estrutura "Repita" testa a condição somente após já ter executado os comandos em seu escopo pelo menos uma vez. Se após a execução dos comandos a condição ainda for falsa, então os comando são novamente executados.



Sintaxe do "Repita":

```
Repita
  <comandos>
ate (<expressão lógica>)
```

Conceitualmente, o "Repita" e o "Enquanto" são os mesmos comandos. Eles são usados nas mesmas situações e sempre podem ser intercambiados um pelo outro, entretanto precisamos observar suas diferenças básicas.

No "Enquanto" repetimos o bloco de comandos enquanto a condição for verdadeira. Já no "Repita" os comando são repetidos até a condição se tornar verdadeira, ou seja, assume-se que inicialmente a condição deve ser falsa. Observe o mesmo algoritmo escrito com o "Enquanto" e com o "Repita":

1.	Algortimo "ExemploEnquanto"	Algortimo "ExemploRepita"
2.	Var	Var
3.	numero : inteiro	numero : inteiro
4.	Inicio	Inicio

5.	leia (numero)	~~leia (numero)~~
6.		
7.	**enquanto (numero%2 = 0) faca**	**Repita**
8.	leia(numero)	leia(numero)
9.	**fimenquanto**	**ate (numero%2 <> 0)**
10.		
11.	escreva(numero , " é impar")	escreva(numero , " é impar")
12.		
13.	fimalgoritmo	fimalgoritmo

O algoritmo acima faz a validação de um número inteiro que deve ser impar para ser considerado válido, e consequentemente, escrito na tela (linha 11). Desse modo, não podemos deixar que o fluxo normal do algoritmo chegue a linha 11 sem termos a certeza que o número informado será impar. Se o número informado for par, então o mesmo deverá ser digitado novamente até que seja impar. O que o algoritmo deve garantir é que o comando de saída da linha 11 seja executado sem que o número informado pela última vez seja impar.

Note que no algoritmo "ExemploEnquanto", antes de testar a condição (linha 7) foi preciso ler a variável *numero* (linha 5). Isso não acontece com o algoritmo "ExemploRepita", pois é característica do comando "Repita" permitir a execução dos comandos no seu escopo pelo menos uma vez antes de testar a condição (linha 9). Essa é a primeira grande diferença;

A segunda diferença está no ponto de vista da condição. Observe as frases: a) "**Enquanto** está chovendo eu uso o guarda-chuva"; b) "Eu uso o guarda-chuva **até** parar de chover". Apesar de as frases carregarem a mesma informação - de que é preciso usar o guarda chuvas quando está chovendo- ela o faz de maneira opostas. Enquanto na primeira frase é preciso "estar chovendo" para a ação "de usar o guarda chuvas" ser realizada, na segunda, a ação de "usar o guarda-chuva" somente será interrompida quando "parar de chover". Isso reflete na inversão de operadores observada no algoritmo acima.

3. Estruturas de Repetição Pré-fixadas

Há momentos na programação em que é desejável que um bloco de instruções seja executado um determinado número de vezes pré-fixado, sem que necessariamente haja uma condição intrínseca do problema que possa ser usada nas estruturas do tipo "Repita" e "Enquanto".

Imagine que gostaríamos de saber qual valor da soma de todos os números múltiplos de 5 no intervalo entre 1 e 5000. Para resolver esse problema poderíamos fazer um algoritmo como o abaixo:

```
Algoritmo "SomaMultiplos5"
Var
  soma: inteiro
```

```
 num: inteiro
Inicio
 Soma <- 0
 Num<- 0

//Primeiro Número
Num<-num + 5      //0 + 5 = 5
Soma <- soma + num // 0+5 = 5

//Segundo Número
Num<-num + 5      //5 + 5 = 10
Soma <- soma + num // 5 + 10 = 15

//Terceiro Número
Num<-num + 5      //10 + 5 = 15
Soma <- soma + num // 15 + 15 = 30

....  (repetir o bloco acima mais 5000/5 = 1000 -4 = 9996 vezes.

// quinto milésino
Num<-num + 5      //4995 + 5 = 5000
Soma <- soma + num // 2497500 + 5 = 2497505

escreva(soma)


fimalgoritmo
```

Apesar de funcional, a abordagem acima não é nada prática. Repetir o mesmo bloco de instruções 1000 vezes manualmente não é sensato. É justamente para isso que temos as estruturas de repetição. Observe então como pode-se resolver o exemplo acima usando tanto as estruturas do tipo "enquanto" como as estruturas do tipo "repita".

1.	Algortimo "Somando5Enquanto"	Algortimo "Somando5Repita"
2.	Var	Var
3.	num, soma : inteiro	num, soma : inteiro
4.	Inicio	Inicio
5.	Soma<-0	Soma<-0
6.	Num<-0	Num<-0
7.		
8.	**enquanto (num < 5000) faca**	**Repita**
9.	num<- num + 5	num<- num + 5
10.	soma<- soma + num	soma<- soma + num
11.	**fimenquanto**	**ate (num >=5000)**
12.		
13.	escreva(soma)	escreva(soma)
14.		

15.	fimalgoritmo	fimalgoritmo

Note como foi preciso incrementar o valor da variável *num* de 5 em 5 unidades (linha 9) até que ele atingisse o valor limite estipulado de 5000, ou seja, a própria variável *num* funcionou como uma espécie de contador incrementando em 5 seu valor anterior repetidas vezes até chegar no limite estipulado de 5000. Essa estratégia de usar uma variável como contador para testar uma condição é bastante comum quando se trabalha com estruturas de repetição indeterminadas com condicionais, entretanto existe um outra estrutura mais adequada para essas situações.

```
Sintaxe  do "Para":

Para <variável contadora> <- <valor inicial> ate <valor final> faca
  <comandos>
fimpara
```

```
Algoritmo "SomaMultiplos5"
Var
  soma: inteiro
  num, contador: inteiro
Inicio
 Soma <- 0
 Num<- 0

Para contador<- 1 ate 1000 faca
    Num<-num + 5      //0 + 5 = 5
   Soma <- soma + num // 0+5 = 5
fimpara

fimalgoritmo
```

Sabemos que no intervalo entre 1..5000, existe 1000 números múltiplos de 5 e para acha-los basta que incrementar o número em cinco unidades 1000 vezes. Para manter controle da contagem, usamos uma variável chamada contador que tem o seu valor incrementado em uma unidade a cada ciclo. Esse incremento é feito pela própria estrutura "Para", por isso não devemos ordenar explicitamente o incremento como fizemos quando usamos o "enquanto" ou o "repita". Mas e isso quisermos alterar o tamanho desse incremento?

Alternativa de sintaxe do "Para" com "Passo" :

Para <variável > <- <valor inicial> ate <valor final> **passo <incremento>** faca

```
  <comandos>
fimpara
```

O passo padrão da estrutura "Para" é uma unidade; se for do seu desejo que o variável contadora seja incrementada de uma em uma unidade então não é preciso especificar o passo. Basta omitir esse informação! Por outro lado, se você desejar que o passo do incremento seja qualquer outro diferente de um, então o passo deve ser especificado, como no exemplo abaixo.

```
Algoritmo "SomaMultiplos5"
Var
  soma: inteiro
  num: inteiro
Inicio
 Soma <- 0

Para num<- 0 ate 5000 passo 5 faca
    Soma <- soma + num
fimpara

fimalgoritmo
```

Note como a estrutura "Para" usando o *passo 5* simplificou o algoritmo como um todo. Observe ainda que usou-se a própria variável *num* como a variável contadora que teve o valor inicial definido em 0 e o valor final em 5000. Porém, o mais importe nesse algoritmo, é o fato de o incremento automático realizado pelo "Para" ter sido alterado para 5.

Importante: Não altere o valor da variável contadora dentro do escopo do "para". Isso pode não ser suportado pela linguagem ou levar a erros de contagem.

4. Qual das estruturas usar?

Para saber qual a estrutura de repetição melhor se encaixa na solução algorítmica de um problema devemos observar alguns aspectos:

i. A quantidade de iterações é previamente conhecida? Caso positivo o "Para" aparece como a estrutura o mais adequada.
ii. A quantidade de iterações é fixa, ou possui uma quantidade mínima e máxima? Se for fixa, então o "Para" é o mais adequado, caso contrário, o "Para" não deve ser usado.
iii. É necessário interromper o ciclo de repetições se uma determinada situação ocorrer? Se for necessário, uso o Repita ou o Enquanto

iv. Existe uma condição (ou conjunto de condições) intrínsecas do problema de determina o número de vezes que algo deve ser feito? Use o Repita ou o Enquanto.
v. É necessário deixar que o bloco de comandos seja executado ao menos uma vez antes de verificar se é necessário repeti-lo? Use o “Repita”.
vi. Existe alguma condição que deva ser testada antes que as repetições ocorram? Use o “Enquanto”.

Note que existe ainda a preferencia pessoal e a lógica individual de cada programador. Na maioria dos casos, não existe somente uma única estrutura certa para ser usada! Tudo depende da lógica do programador. Como regra básica podemos afirmar que: a) sempre é possível intercambiar o uso de “Enquanto” e do “Repita”; b) também é sempre possível usar o “Repita” ou o “Enquanto” ao invés do “Para”, mas o contrário nem sempre é possível.