8. Estruturas de repetição

8.1 Conceito no algoritmo

Como vimos, as listas são armazenadas com um índice, sempre iniciando com zero.

Lista_Funcionários =

0	1	2	3	4
Bruno	Luiz	Ana	João	Moisés

Para criar a lista acima, por exemplo, teríamos que usar o seguinte algoritmo:

```
[1] //criação da lista
[2] Lista_Funcionários = [“Bruno”, “Luiz”, “Ana”, “João”, “Moisés”]
[3]
[4] //imprime os nomes
[5] print (Lista_Funcionários[0])   //imprime “Bruno”
[6] print (Lista_Funcionários[1])   //imprime “Luiz”
[7] print (Lista_Funcionários[2])   //imprime “Ana”
[8] print (Lista_Funcionários[3])   //imprime “João”
[9] print (Lista_Funcionários[4])   //imprime “Moisés”
```

Antes de prosseguir, precisamos entender alguns conceitos sobre os índices das listas:

- As listas iniciam sempre em zero, por isso, apesar de ter 5 itens, o último índice é de número 4.

- Isto significa que para saber qual é o último índice, basta lembrar que ele é o índice de número ***n - 1***, sendo ***n*** a quantidade de itens da lista. Por exemplo, numa lista com 400 itens, o número do último índice é 399 (resultado de 400 - 1).

Agora que sabemos desse conceito inicial, vamos falar sobre ***loops***, que significa ***ciclo***. Para entender melhor, lembre-se de um aparelho de CD: quando você coloca um disco, ele inicia tocando na faixa 1, depois passa para a faixa 2 e assim sucessivamente, até a última faixa do disco, certo? Este ciclo desde a primeira faixa até a última é chamado de ***loop***.

Se você apertar a tecla "*repeat*" de seu aparelho de CD, assim que acabar a última música ele reinicia o ciclo e começa a tocar a primeira música do disco novamente. Isto nunca vai ter fim, sempre que chegar ao fim do disco ele volta para o início, por toda a vida. Este processo chamado de ***loop*** **infinito**, **enquanto não houver uma interrupção**.

Na lógica de programação temos 3 tipos diferentes de **estruturas de repetição**, que vamos conhecer agora:

a) *Loop* WHILE

O *loop* ***while*** funciona exatamente igual ao CD com repeat que explicamos agora, que continua seu ciclo (loop) **enquanto não houver** uma interrupção. Na verdade, ***while*** significa exatamente **enquanto** em inglês, não por coincidência.

Usamos o *loop while* quando **não sabemos** o número de eventos que precisam ser executados ou o número de dados de uma lista.

Vamos voltar naquele nosso algoritmo de cálculo de salário e imposto? Imagine que você precise calcular o valor do imposto dos funcionários de uma empresa, e você não sabe quantos funcionários existem nesta empresa.

Então todos os funcionários fazem uma fila e você inicia o procedimento, atende o primeiro funcionário, quando termina, pergunta-se: existe um próximo na fila? Se a resposta for SIM (***true***), você reinicia todo o processo.

O algoritmo do *loop while* deste exemplo fica da seguinte forma:

Algoritmo WHILE (português)	Algoritmo WHILE (inglês)
[1] Pergunta (próximo_funcionário)	[1] Prompt (próximo_funcionário)
[2] ENQUANTO (próximo_funcionário)	[2] WHILE (próximo_funcionário)
[3] INICIO	[3] BEGIN
[4] Salário = 1000	[4] Salário = 1000
[5] Tem_Filhos = FALSE	[5] Tem_Filhos = FALSE
[6] SE Tem_Filhos	[6] IF Tem_Filhos
[7] ENTÃO	[7] THEN
[8] Imposto = 50	[8] Imposto = 50
[9] SENÃO	[9] ELSE
[10] Imposto = 100	[10] Imposto = 100
[11] FIM SE	[11] END IF
[12] Salário_Final = Salário – Imposto	[12] Salário_Final = Salário – Imposto
[13] Pergunta (próximo_funcionário)	[13] Prompt (próximo_funcionário)
[14] FIM_ENQUANTO	[14] END

Repare que precisamos iniciar com a pergunta para que o loop inicie, afinal, precisa ter uma fila para iniciar o primeiro atendimento. Sem primeiro atendimento, o loop não inicia. Então, a pergunta precisa ser feita antes e ao final do *loop while*. A estrutura de repetição *while* serve para qualquer necessidade.

b) *Loop* FOR EACH

O *loop* ***for each*** é mais simples, se assemelha muito com o *play* normal do aparelho de CD que, quando tem um CD de 15 músicas, inicia na primeira música e para na última. E pronto. O *loop for each* ***não precisa de verificação*** como o *while.*

Para exemplificar este *loop* vamos usar o mesmo exemplo de salários e impostos, mas, desta vez, você recebe uma pilha de pastas com os documentos dos funcionários. Você não sabe quantas pastas de funcionários tem na pilha, mas sabe que é uma quantidade imutável.

Para isso, você recebe a seguinte ordem: **para cada pasta**, execute o cálculo de imposto. Vamos ver como fica o algoritmo do *loop for each*, que significa literalmente “para cada”.

Algoritmo FOR EACH (português)	Algoritmo FOR EACH (inglês)
[1] PARA CADA (Funcionário) [2] INICIO [3] Funcionário.Salário = 1000 [4] Funcionário.Tem_Filhos = FALSE [5] SE Funcionário.Tem_Filhos [6] ENTÃO [7] Imposto = 50 [8] SENÃO [9] Imposto = 100 [10] FIM SE [11] Funcionário.Salário_Final = Funcionário.Salário – Imposto [12] FIM_ENQUANTO	[1] FOR EACH (Funcionário) [2] BEGIN [3] Funcionário.Salário = 1000 [4] Funcionário.Tem_Filhos = FALSE [5] IF Funcionário.Tem_Filhos [6] THEN [7] Imposto = 50 [8] ELSE [9] Imposto = 100 [10] END IF [11] Funcionário.Salário_Final = Funcionário.Salário – Imposto [12] END

Diferente do *while* que serve para cada ocasião, o *loop for each* **funciona apenas para listas**, pois ele faz uma iteração para cada um de seus itens.

c) *Loop* FOR

O *loop FOR* é a estrutura de repetição mais organizada das 3 existentes, pois possui uma estrutura mais rígida que precisa ser seguida. Seu ciclo de repetição é específico e limitado, com conhecimento prévio de quando inicia e quando termina.

Vamos voltar para a nossa lista de funcionários:

Lista_Funcionários =

0	1	2	3	4
Bruno	Luiz	Ana	João	Moisés

Aqui conhecemos a estrutura da lista e seus índices, sabemos que tem 5 itens com índices de zero a 4. Como temos conhecimento de toda sua estrutura, podemos

usar a seguinte lógica estruturada: inicie em zero, até todos os números menor que 5, contando de 1 em 1.

O *loop* FOR precisa ser configurado com 3 informações, e possui o seguinte formato ***for*** **(de; até; passo).** Vamos conhecer cada parte:

- **de:** informação da posição com o qual índice será iniciado, ou seja, sua posição inicial.
- **até:** informação da posição final do índice.
- **passo:** são os passos que o índice deve percorrer entre o início e o fim.

Lista_Funcionários =

0	1	2	3	4
Bruno	Luiz	Ana	João	Moisés

Na lista acima de 5 elementos, podemos verificar que o nosso loop tem que percorrer cada item da lista, para isso, precisa:

- Iniciar no índice 0 (zero).
- Finalizar no índice 4.
- Ter um passo de 1, que significa que precisa incrementar de 1 em 1 desde seu início. Ou seja, inicia em zero, o próximo é o índice 1, em seguida o índice 2... até chegar o último índice.

Sua estrutura fica então com a seguinte forma:

FOR *(índice = 0; índice < 5; índice = índice + 1)*

O índice iniciando em zero faz sentido, pois é o primeiro índice mesmo. O índice finalizando em "menor que 5" (< 5) também faz sentido, lembra daquele conceito que vimos que o último índice é o ***n-1***? Então, todos os números dos índices são de 0 a 4, ou seja, todos são menores que 5. Com isso, o loop vai percorrer apenas nessa faixa, parando em 4. Afinal, para que serve este passo?

Para entendermos o passo, aqui representado por ***índice = índice + 1***, que significa que quando o índice for 0 (zero), o próximo será o índice 0 + 1, então o próximo índice será o 1. O índice seguinte será, então, o atual mais um, ou seja, 1 + 1, então será 2. E assim segue até o último. Isto faz todo o sentido, mas porque precisamos informar este passo?

Vamos imaginar 2 situações. Na primeira, eu preciso saber apenas os dados da lista que estão nos índices pares. Para isso, posso simplesmente colocar o passo como *índice = índice + 2*. Como o índice inicia em zero, o próximo será o índice 2, o seguinte será o índice 4. Só números pares. Bacana, não é?

A segunda situação é a seguinte: estou com uma empresa com 400 funcionários e todos estão numa lista. Eu preciso fazer uma auditoria nos dados destes funcionários, mas só em alguns. Então, posso programar meu loop for para ter um passo de 10. Assim, vou pegar o funcionário que está na posição 0, depois o funcionário que está na posição 10, o próximo que está na posição 20.

Dessa forma, vou seguindo coletando dados de amostras de funcionários, não de todos.

Agora vamos ver seu funcionamento *loop* FOR no algoritmo. Para isso, vamos expandir os dados dos funcionários em 3 listas diferentes, além da lista de "nomes", vamos acrescentar também uma lista de "salários" e uma outra lista com a informação de "filhos". Vamos acrescentar também uma 4ª lista vazia para receber o resultado dos cálculos. Essas listas, quando colocadas **lado a lado**, ficam com a seguinte forma:

Índice	0	1	2	3	4
Lista Nome =	Bruno	Luiz	Ana	João	Moisés
Lista Salário =	1000	1500	1300	1250	1600
Lista Filhos =	False	True	True	False	True
Salário Final =					

O índice é sempre o mesmo para todas as listas. Ou seja, na posição 0 é sempre relacionada ao "Bruno", com salário de "1000" e "sem filhos", até a última posição. Para cada item, do primeiro ao último, precisamos calcular seu salário final. O algoritmo FOR para este caso ficaria com a seguinte estrutura:

Algoritmo FOR (português)	Algoritmo FOR (inglês)
[1] Funcionários = [“Bruno”, “Luiz”, “Ana”, “João”, “Moisés”] [2] Salários = [1000, 1500, 1300, 1250, 1600] [3] Filhos = [False, True, True, False, True] [4] Salário_Final = [ ] [5] [6] PARA (ind = 0; ind < 5; ind = ind +1) [7] INICIO [8] SE Filhos[ind] [9] ENTÃO [10] Imposto = 50 [11] SENÃO [12] Imposto = 100 [13] FIM SE [14] Salário_Final[ind] = Salários[ind] – Imposto [15] FIM_PARA	[1] Funcionários = [“Bruno”, “Luiz”, “Ana”, “João”, “Moisés”] [2] Salários = [1000, 1500, 1300, 1250, 1600] [3] Filhos = [False, True, True, False, True] [4] Salário_Final = [ ] [5] [6] FOR (ind = 0; ind < 5; ind = ind +1) [7] BEGIN [8] IF Filhos[ind] [9] THEN [10] Imposto = 50 [11] ELSE [12] Imposto = 100 [13] END IF [14] Salário_Final[ind] = Salários[ind] – Imposto [15] END_FOR

Vamos analisar agora nosso algoritmo:

- As listas com os dados de nome, salário, filhos e o resultado de salário final, conforme tabela, estão nas linhas de 1 a 4.

- Na linha 4, a lista de salário final é criada vazia, sem itens, pois é o que precisamos calcular.

- Começamos em seguida o nosso loop for, onde precisamos definir os índices iniciando em 0 (zero) e indo até o valor menor que 5 (ind < 5).

- Para a estrutura de repetição iniciar, ela começa definindo a variável **ind** como 0 (zero), que é o primeiro item da lista, e vai pegar os valores de salário e filhos no índice zero (salario[0] e filhos[0]), fazendo o cálculo e preenchendo o valor de Salário_Final[0] com o resultado do cálculo. Ao final do primeiro ciclo, a tabela vai ficar assim:

Índice	0	1	2	3	4
Lista Nome =	Bruno	Luiz	Ana	João	Moisés
Lista Salário =	1000	1500	1300	1250	1600
Lista Filhos =	False	True	True	False	True
Salário Final =	900				

- Para continuar, o loop precisa saber qual o próximo índice, por isso definimos a última parte com o valor incremental de 1 (ind = ind + 1).

Dos 3 tipos de estrutura de repetição que estudamos, o FOR é o que tem o formato mais rígido, como vimos. Vamos resumir estes 3 tipos de loop:

WHILE: usamos para qualquer situação em que precisamos repetir algo até que uma condição seja satisfeita. Pode ser usada para ler dados de uma lista ou para qualquer outro trecho de código que precisa ser repetido.

FOR EACH: usamos apenas para iterar (percorrer) itens de uma lista. Pudemos ver sua diferença com o *FOR* pois, para cada pasta da pilha de nosso exemplo, ele já tem os dados em si mesmo (Funcionário.Tem_Filhos).

FOR: usamos para percorrer os índices de uma lista ou repetir um trecho de código quando sabemos exatamente a quantidade de vezes que precisam ser repetidas.

Agora, vamos praticar estes conceitos.