Introdução ao comando de repetição `while`

Nas seções sobre variáveis e sobre comando de seleção, pudemos perceber os conceitos básicos da computação que nos permite construir sistemas sofisticados. Com variáveis é possível armazenar e trocar valores e com a seleção (*if*) pode-se desviar o fluxo de execução de acordo com valores que as variáveis assumem.

Entretanto, falta um conceito essencial para permitir computação significativa, que é a *repetição condicionada* de comandos. Quer dizer, é possível *repetir um bloco de comandos enquanto uma condição se mantiver verdadeira*. Deve-se destacar que existem outros tipos de *laços de repetição*, como *repetir até determinada condição ser válida* ou *repetir um número de vezes*.

1. Estrutura do comando de *repetição com condição de entrada* (`while`)

O comando repita enquanto pode ser resumido em três passos, como ilustrado na figura 1, a saber:

1. testa a **condição lógica** (*Condição Verdadeira?*),
2. se resultar **verdadeiro**, então **entra no laço**, ou seja, segue o caminho indicado pela seta com rótulo **sim**;
    1. executa todos os comandos subordinados ao laço, o **bloco de repetição**;
    2. volta ao passo 1, ao início do laço (por isso este é classificado como **laço com condição de entrada**);
3. se resultar **falso**, então **sai no laço**, ou seja, segue o caminho indicado pela seta com rótulo **não** (segue para comandos *B*).

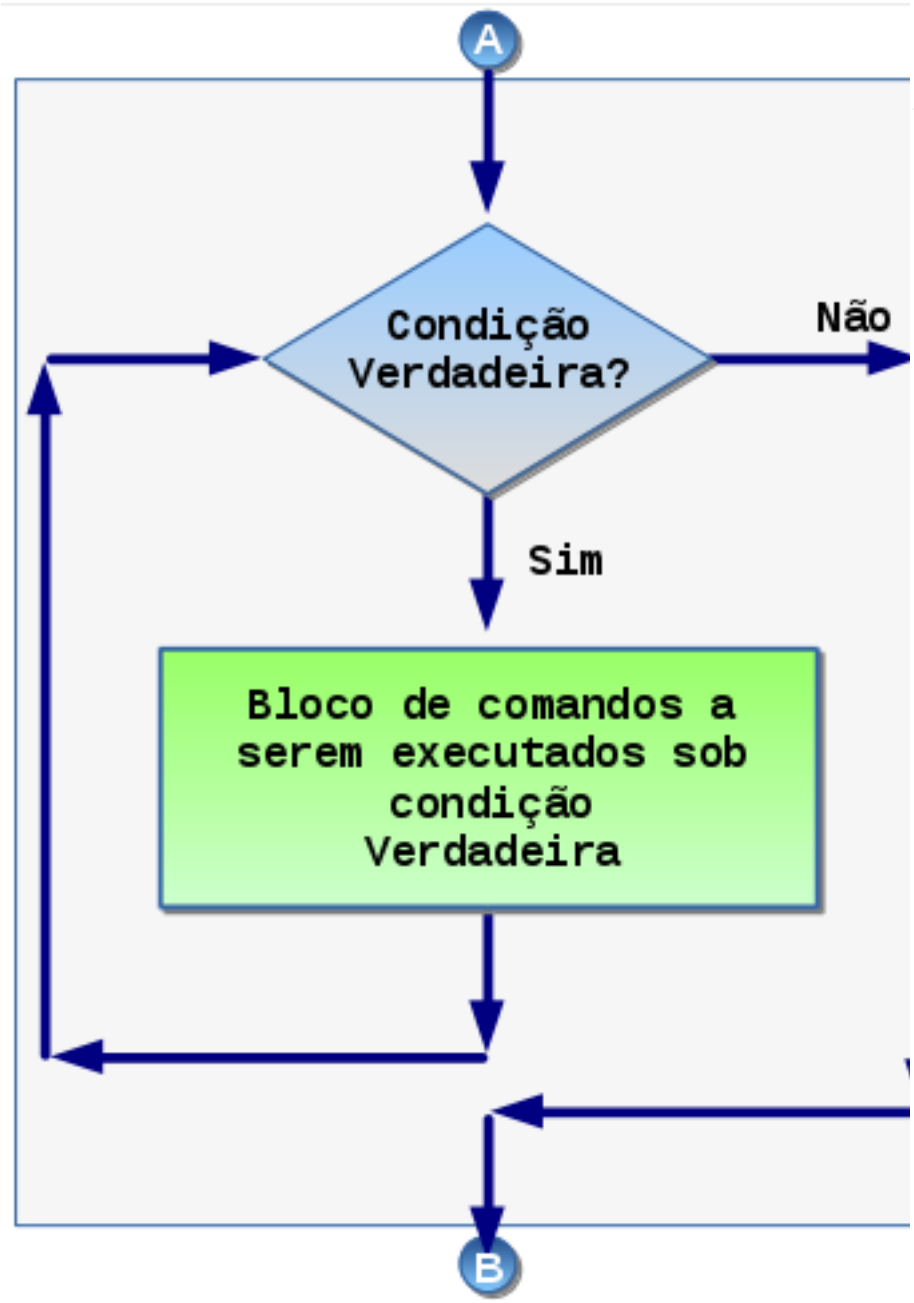

Fig. 1. Diagrama de fluxo de execução do laço enquanto com condição de entrada.

Tab. 1. Códigos ilustrativos para o fluxograma da figura 1.

C	Python

`while (i<N) { // condicao "i<N"`
 	
`comandoW1 ;`
`  ...`
 	
`comandoWk ;`
`  } // final do laco`
`comandoF1 ; // primeiro comando fora do laco`
	`while (i<N) : # condicao "i<N"`
	
	`comandoW1 ;`
	`  ...`
	
	`comandoWk ;`
	`  # indentacao marca final do laco`
	`comandoF1 ; # primeiro comando fora do laco`

Vale destacar que os rótulos *A* e *B* na figura 1 procuram enfatizar que: rótulo *A* indica que **antes** do comando *repita enquanto condição verdadeira* pode existir comandos e, rótulo *B* indica que **depois** do comando *repita enquanto* também pode existir comandos.

Outro destaque deve ser feito: se após executar os comandos em *A*, a *condição lógica* resultar *falso*, então o bloco *subordinado* ao laço *repita enquanto* não será executado sequer uma vez! Ou seja, devido à sua característica de *laço com condição de entrada*, é possível que seus comandos **nunca** sejam executados.

A estrutura básica do comando de *repetição com condição de entrada* é formado por 3 partes, uma marca indicando o comando (`while`), seguido de uma expressão lógica (EXPL) depois o bloco de *comandos subordinados*. Assim, se tiver apenas um comando subordinado a sintaxe *C* e *Python* são:

Tab. 2. Códigos ilustrativos o formato geral do comando `while` em C e em Python.

C	Python
0. `// (A) pode haver comandos anteriores` 1. `while (EXPL) // em C e' obrigatorio os '(' e ')'` 2. `    comando1;` 3. `// (B) pode haver comandos posteriores`	0. `# (A) pode haver comandos anteriores` 1. `while (EXPL) : # note que em Python '(' e ')' sao opcionais` 2. `    comando1;` 3. `# (B) pode haver comandos posteriores`

O significado (semântica) do código acima, durante sua execução, é:

1. *Verifica-se a condição `EXPL`, se o seu resultado for verdadeiro executa-se o passo 2, senao vai para passo 3 (final do comando)*
2. *Executa-se o comando "comando1", depois volta-se a executar o passo 1.*
3. *Final do bloco de repetição!*

2. Vários comandos subordinados ao mesmo comando de repetição `while`

Se houver necessidade de vários comandos subordinados ao comando de repetição, então em C deve-se usar as marcas '{' e '}' para anotar o início e o fim do bloco. Em Python não é necessário devido à indentação ser obrigatória e por ser capaz de identificar todos os comandos subordinados. Assim, vejamos um trecho de código que gerar e imprimir os naturais entre 1 e N:

Linha	C	Python

1.
2.
3.
4.
5.
6.
7.
	`scanf("%d", &N); // ler valor limite N`
	`i = 0; // variavel para "contar"`
	`while (i < N) {`
	`   i = i+1; // leia-se como: i recebe o valor de i acrescido de 1`
	`   printf("i=%d\n", i);`
	`   } // final do bloco`
	`printf("Final!\n");`
		`N = int(input()); # ler valor limite N`
		`i = 0 # variavel para "contar"`
		`while (i < N) :`
		`   i = i+1; # a indentacao indica esta linha estar subordinada ao "while"`
		`   print("i=%d" % i) # e a proxima linha como sobordinadas 'a repeticao`
		`   # para indicar final do bloco basta a proxima linha estar alinhada com o "while"`
		`print("Final!")`

3. Exercitando a compreenção: simulando o código

*Para algoritmos "pequenos", após desenhá-lo, antes mesmo sem implementá-la em um computador, é interessante examinar sua execução, para isso pode-se empregar a técnica de **simulações**,*
A simulação também é interessante para melhor compreender algum algoritmo desenvolvido por outros ou para você detectar eventuais erros em seu código, que portanto precisarão ser corrigidos.

Para ilustrar a técnica de simulação e melhorar a compreensão sobre o comando repita enquanto, a aplicaremos sobre o algoritmos do código 1.
Adotaremos o seguinte esquema nessa simulação:

1. *na parte esquerda colocaremos texto explicativo, indicando a linha em execução;*
2. *na parte central a simulação propriamente dita, destacando as variáveis sendo examinadas;*
3. *na parte direita as saídas de impressão correspondente à linha sendo executada/simulada.*

Vale notar que, para identificar mais facilmente a linha que provocou a alteração de uma dada variável, usaremos linhas distintas para comando executado. Assim, se houver necessidade de verificar algum possivel engano na simulação, pode-se mais facilmente identificar o comando que interpretamos erroneamente.
Indicaremos as atribuições com ":=".

```
Linha/explicação                                    N | i
impressoes
-------------------------------------------    ---+---    ---
-------
1. leitura para variável N                          3 | ?
2. i := 0                                             | 0
2. entra no laco "while" pois 0=i<N=3                 |
4. i := 1 (pois: i recebe i+1=0+1)                    | 1
5. saida: i=1                                         |       i=1
6. final do laco "while" volta para condicao          |
3. entra no laco "while" pois 1=i<N=3                 |
4. i := 2 (pois: i recebe i+1=1+1)                    | 2
5. saida: i=2                                         |       i=2
6. final do laco "while" volta para condicao          |
3. entra no laco "while" pois 2=i<N=3                 |
4. i := 3 (pois: i recebe i+1=2+1)                    | 3
5. saida: i=3                                         |       i=3
6. final do laco "while" volta para condicao          |
7. sai no laco "while" pois 3=i<N=3 = falso!!!        |
Final!
```

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações