Introdução ao comando de repetição `for`

Nesta seção examinaremos um novo comando de repetição, bastante útil quando o número de *passos* (cada repetição de um bloco de comandos) é conhecido à priori, ou seja, conhecido ao iniciar o laço.

Por exemplo, para *somar os N primeiro naturais*, sabe-se a priori que serão necessários *N* passos. Isso é bem diferente de *somar os primeiros naturais até que obtenha um valor maior ou igual a K* (nesse caso seria muito mais adequado o comando `while`).

Em várias *linguagens de programação* (com *C* e em *Python*) esse comando para (repetir um número fixo de vezes) recebe o nome de **`for`** (*para*). Além disso ele deve sempre dispor de um **enumerador** (ou *iterador*), ou seja, uma variável que será utilizada para controlar o número de repetições do bloco de comandos subordinado ao *for*. Esse *enumerador* pode ser usado em comandos do bloco.

Geralmente os comandos de repetição do tipo *para* permitem um **inicializador** (`ini` na figura 1), e um **tamanho** (`tam` na figura 1). O *iterador* (`it` na figura 1) é iniciado com `ini`, incrementando-o a cada execução do bloco dentro do *laço*, fazendo isso `tam-ini` vezes, como indicado na figura 1.

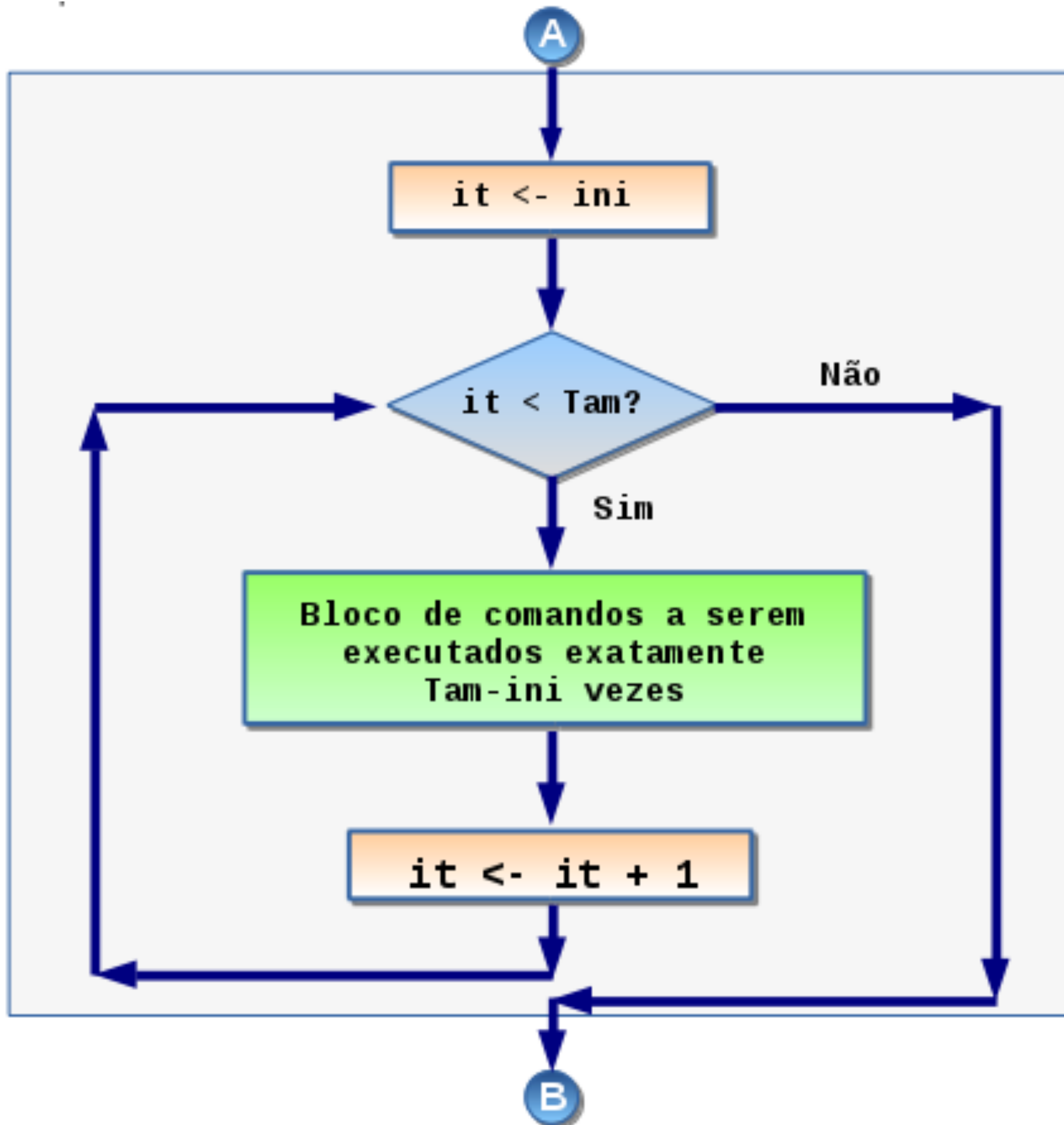

Fig. 1. Ilustração de um comando `for` *com iterador* `it`*, inicializador* `ini` *e tamanho de laço* `tam-ini`*.*

Se você está aprendendo a programar usando a linguagem *C* estude a seção 1.
Se você está aprendendo a programar usando a linguagem *Python* estude a seção 2.

1. Comando *for* em *C*

Em *C* a sintaxe do comando é:

Comando `for`	Equivalente usando `while`	`Exemplo: imprimir N primeiros naturais`
`for (it=valorIni, condicao, incremento)`	`it = valorIni;` `while  (condicao) {`	`for (it=0; it<N; it++) {`

comandoSubordinado;	comandoSubordinado;
	incremento; // geralmente algo como: it += passo;
	}
		// todos os comandos subordinados ao for
		printf("%d\n", it); // aqui apenas 1...
		}

Note que dentro da definição do comando também deve-se utilizar o separador ponto-e-vírgula (';').

Por exemplo, o programa abaixo que imprime a soma dos `n` primeiros naturais, mas somando-os em ordem invertida, começando com o `n`, depois somando-se `n-1` e, assim por diante, até o `1`.

```
#include <stdio.h>
int main (void) {
  int n, i; // declarando o iterador i do comando for e o tamanho do laco n
  int soma = 0;
  scanf("%d", &n); // ler a quantia desejada de naturais
  for (i=n; i>0; i--) { // tamanho = -1 * (0 - n) = n
    soma += i; // acumula o atual natural em i
    printf("%d : %d\n", i, soma); // imprime contador e a soma parcial
    // apos ultimo comando subordinado ao for, executa-se o incremento do contador (ou seja, i <- i-1), entao
    // a execucao retorna ao inicio do for e testa se continua (i≥0),
    // se for repete-se os comandos subordinados, senao finaliza o for
    }
  printf("Soma dos %d primeiros naturais eh igual a %d\n", n, soma);
  return 1; // se foi algum outro programa que invocou este, devolva o valor 1
  }
```

Cód. 1. Somar os n primeiros naturais, em ordem invertida, imprimindo cada parcial.

2. Comando *for* em *Python*

Em *Python* é necessário usar um *iterador*, isso pode ser feito com a função `range(...)` que pode ter 1, 2 ou 3 parâmetros do tipo número natural. Se tiver apenas um parâmetro significa que deve gerar os n primeiros naturais. Se tiver dois parâmetros naturais, neste caso significa os naturais entre ambos, mas também pode ter três parâmetros e neste caso o último é o tamanho do passo. Abaixo a forma geral do comando `for`, com um exemplo de `range(...)` usando apenas um parâmetro.

```
for it in range(numeroIteracoes) comandoSubordinado
```

Nesse exemplo, o `comandoSubordinado` será executado `numeroIteracoes`, na primeira com `i` valendo `0`, depois `i` valendo `1` e assim por diante, até `i` valendo `numeroIteracoes-1`. Também pode-se fazer o `i` ter outro valor inicial e tamanho de passo diferente, mesmo passos "negativos". Por exemplo, para imprimir os `n/2` primeiros ímpares pode fazer:

```
for i in range(1, n, 2) : print("%d " % i, end="");
print();
```

Se `n=10`, esse código geraria `1 3 5 7 9`, na mesma linha. Ou seja, o segundo parâmetro do `range(...)` (o controle `n`), é o limite, "repete-se enquanto" `i ≤10`.

Por exemplo, o programa abaixo que imprime a soma dos `n` primeiros naturais, mas somando-os em ordem invertida, começando com o `n`, depois somando-se `n-1` e, assim por diante, até o `1`.

```
def main () :
  #inteiros: n, i :: usaremos como iterador i para o comando for e o tamanho
do laco n
  soma = 0;
  n = int(input()); # ler a quantia desejada de naturais
  for i in range(n-1, -1, -1) : # tamanho = -1 * (-1 - (n-1)) = (n+1)-1 = n
    soma += i; # acumula o atual natural em i
    print("%d : %d" % (i,soma)); # imprime contador e a soma parcial
    # desse ponto, incrementa-se i (ou seja, i <- i-1), entao a execucao
    # retorna ao inicio do for e se i≥0, repete-se os comandos subordinados
ao for
  print("Soma dos %d primeiros naturais eh igual a %d" % (n,soma));
main()
```

Cód. 2. Somar os n primeiros naturais, em ordem invertida, imprimindo cada parcial.

Note que no exemplo acima usamos o marcador **`%d`** dentro da função `print`, isso significa que no lugar do primeiro `%d` deverá ser impresso o valor da primeira variável, encontrada após fechar aspas e após o caractere `%` (i.e., `i`) e no lugar do `%d` deverá ser impresso o valor da segunda variável (i.e., `soma`). O `%d` está associado à inteiros, se precisar de caractere deve-se usar o formator `%c`, se precisar de ponto flutuante deve-se usar o `%f` e se for usar uma variável que guarda caracteres (*string*), deve-se usar o formatador `%s`.

3. Exercitando a compreensão: simulando o código

É interessante usar **simulações** para melhor compreender o desenvolvimento dos algoritmos, mais ainda se você detectar algum erro em seu código, que portanto precisa ser corrigido. Adotaremos o seguinte esquema de simulação: usaremos uma tabela com as variáveis do código e registraremos cada alteração em uma linha, seguindo o *fluxo de execução* de cima para baixo, de modo que o último valor de uma variável (em qualquer ponto/linha) será o primeiro valor, na coluna da variável, encontrado "olhando" para "cima". Indicaremos as atribuições com ":=".

```
                                                      n | soma |
 i     impressoes
                                                     ---+------+--
 --    ----------
 soma := 0                                            ? |    0
 | ?
 * leitura para inteiro n                             3 |      |
 i := 0 (primeiro da lista (0,1,2)                      |      |
 0
 * entra no laco "for" pois 0=i<n=3                     |      |
 soma := 0 (pois: soma recebe soma+i=0+1)               |    0 |
```

```
* saida: 0 : 0                                              |      |
0 : 0
i := 1 (segundo da lista (0,1,2))                           |      |
1
* entra no laco "for" pois 1=i<n=3                          |      |
soma := 1 (pois: soma recebe soma+i=0+1)                    |    1 |
* saida: 1 : 1                                              |      |
1 : 1
i := 2 (terceiro da lista (0,1,2))                          |      |
2
* entra no laco "for" pois 2=i<n=3                          |      |
soma := 3 (pois: soma recebe soma+i=1+2)                    |    3 |
* saida: 2 : 3                                              |      |
2 : 3
* nao tem elemento na lista (0,1,2) apos o 2                |      |
* final do laco "for" (sem mais elementos)                  |      |
* executa primeira instrucao apos laco "for"                |      |
* saida: Soma dos 3 primeiros naturais eh igual a 3         |      |
Soma dos 3 primeiros naturais eh igual a 3
```

Em relação à função `range(...)` existe uma diferença interessante entre o *Python 2* e o *Python 3*. No 2 ao usar por exemplo, `range(10)` é gerada uma lista com os 10 primeiros naturais, assim imediatamente após invocar a função é necessário alocar espaço de memória para os 10 elemento. Já no *Python 3* os elementos são gerados a medida que se precisa deles.

Vejamos como funcionam os parâmetros em `range`:

# parâmetros	função	resultado
1 parâmetro	`lst = range(10)`	Gera uma lista com os naturais 0,1,2,3,4,5,6,7,8,9
2 parâmetros	`lst = range(0,10)`	Gera uma lista com os naturais 0,1,2,3,4,5,6,7,8,9
2 parâmetros	`lst = range(1,10)`	Gera uma lista com os naturais 1,2,3,4,5,6,7,8,9,10
3 parâmetros	`lst = range(0,10,2)`	Gera uma lista com os naturais 0,2,4,6,8

[Leônidas de Oliveira Brandão](#)
http://line.ime.usp.br

Alterações: