Introdução ao comando de seleção e às expressões lógicas

[Expressão em C | Operadores | Seleção | Seleções | Exemplo | Par ou ímpar]

Como ler este texto. Aqui procuro explicar os fundamentos do comando de seleção `if-else` para a linguagem *C*, explicando *expressões lógicas* e também *aninhamento* de comandos `if-else`.

O que é um comando de seleção?

A existência de *comando de seleção* é essencial para a construção de *algoritmos flexíveis* no sentido de poder seguir uma sequência de instruções distintas de acordo com uma condição. Um exemplo simples poderia ser um algoritmo para computar a somar de todas as entradas de um extrato bancário correspondente aos créditos separadas dos lançamentos de débito. Desse modo, para cada entrada do extrato, deve-se verificar se é positiva, nesse caso acumula em uma variável para *crédito*, senão deve-se acumular em uma variável para débito. Ou seja, de acordo com o resultado da avaliação de uma **expressão lógica**, deve-se seguir determinado **fluxo de execução** (um **"caminho" de instruções**).
Qualquer *linguagem de programação* dispõe de um comando para *desvio de fluxo*, geralmente esse tipo de comando é demominado comando **`if-else`**. Outra característica comum das linguagens de programação é possibilitar definir **dois blocos** de comandos subordinados ao `if-else`, um deles devendo ser executado sempre que a condição resultar *verdadeira* e o outro, que é opcional, devendo ser executado sempre que resultar *falso*.

A figura 1 ilustra as duas possibilidades de comando `if-else`, a figura 1.a apresenta um comando de *seleção* apenas com o **bloco "*verdadeiro*"**, enquanto a 1.b ilustra um comando de *seleção* com os dois blocos.


Condição Verdadeira?
Não
Sim
Bloco de comandos a serem executados sob condição Verdadeira

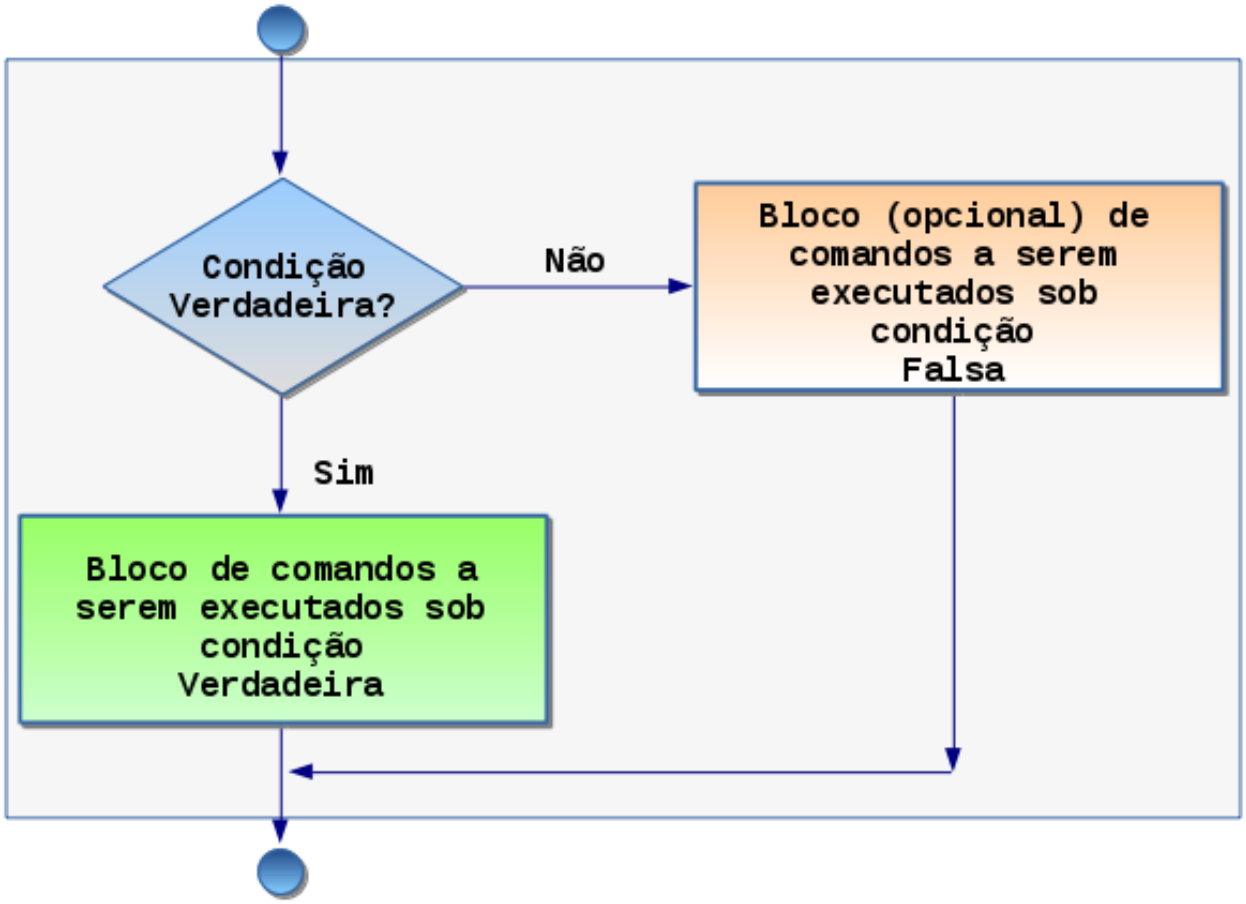

Fig. 1. Diagrama de fluxo de execução (a) sem o ramo de senão e (b) com o ramo de senão ..

Na figura 1.a, se o resultado da **condição lógica** for **verdadeiro**, segue o caminho indicado pela seta com rótulo **sim** e se **falso** segue o caminho da seta com rótulo **não**, que é a próxima instrução após o *bloco "verdadeiro"*.
Na figura 1.b, está o esquema do `if-else` com os dois blocos: se o resultado da **condição lógica** for **verdadeiro**, segue o caminho para o *bloco "verdadeiro"* (verde), mas se resultar **falso**, segue para o *bloco "falso"* (laranja).

De modo esquemático, o comando é do tipo `if (EXPRLOG) blocoV else blocoF`. Assim, devemos destacar que nos dois blocos do comando `if-else` pode haver outros comando `if-else`, como examinado na seção sobre comandos `if` aninhados.

Expressão lógica em *C*

Na linguagem *C* não existe um tipo específico para resultados de *expressões lógicas*, essas são obtidas interpretando-se o resultado final de uma *expressão aritmética*, se *diferente de zero* é *verdadeiro*, se *igual à zero* é *falso*. Desse modo, a expressão lógica mais simples é qualquer valor numérico.

Já uma *expressão lógica* não trivial deve envolver duas *expressões aritméticas* conectadas por um *operador relacional* (`EL = EA1 <= EA2`) ou duas outras *expressões lógicas* conectadas por

um *operador lógico* (`EL = EL1 and EL2`). Os operadores *relacionais* e *lógicos* são explicados na seção sobre operadores relacionais e lógicos.

Os códigos 1 e 2 a seguir ilustram essa particularidade da linguagem *C*: qualquer *expressão aritmética* pode ser interpretada como uma *expressão lógica*.

```
if (i!=N) printf("%d == %d\n", i, N); while (i < N) { printf("i = %d\n", i); i++; }
```

Cód. 1. Exemplos de expressão lógica em C.

No código 1 existem duas expressões lógicas, uma subordinada a um comando `if-else` e a outra subordinada a um comando `while`.

Mas vale a pena notar que a linguagem *C* permite construções cujo processamento é um pouco mais rápido, mas menos intuitivo, é permitido usar qualquer expressão aritmética (como `a-b`) dentro do contexto de expressão lógica! Isso é possível a partir do seguinte "truque": calcula-se o resultado da expressão aritmética, se o seu resultado for diferente de *0*, ele é interpretado como sendo o valor lógico *verdeiro*, caso contrário como *falso*. Desse modo, a condição lógica abaixo é equivalente à codição do `if` do código 1, pois sempre que a expressão *i-N* resultar valor não nulo, será "impresso" *OK* e se o valor for nulo será "impresso" *NAO*:

```
if (i-N) printf("OK\n"); else printf("NAO\n");
```

Cód. 2. Exemplos de expressão aritmética tratada como expressão lógica em C.

Desse modo, pode-se substituir a linha 1 do código 1 por: `if (i-N) printf("%d == %d\n", i, N);`. Ou seja, o tratamento de expressões é mais um exemplo de que o *contexto* pode mudar a forma como os dados são interpretados.

Negação em *C*. Pode-se usar o operador `!` (exclamação) para *negar* o resultado da expressão lógica (que esteja à sua direita). Por exemplo, os seguintes trechos de código são equivalentes em *C*:

```
if (a==b) printf("OK");
else printf("NAO");
```

```
if (!(a-b)) printf("OK");
else printf("NAO");
```

Cód. 3. Exemplos de uso de negação em expressão lógica em C.

Podemos explorar equivalências para entender a razão dos dois códigos serem equivalentes. Observe que

!(a-b) é a mesma coisa que **negar** que *a-b* seja **não nulo**, ou seja, *!(a-b != 0)* e,
!(a-b != 0) é a mesma coisa que afirmar que *a-b == 0*, e
a-b == 0 é a mesma coisa que *a == b*.

Atenção *programadores C*, um **erro bastante comum** ao tentar usar o operador relacional `==` é acabar **digitando apenas =**. Em função da explicação acima, não existe qualquer erro sintático, o código será gerado, mas ao rodar, pode ser muito difícil detectar esse **erro semântico**.
O resultado de um comando como `if (a=b) printf("Sao iguais\n");` é que, se `a` estiver com qualquer valor diferente de zero, a mensagem será impressa.
Para descobrir este tipo de erro uma boa técnica é usar bandeiras.

Operadores relacionais e lógicos em *C*

Os **operadores relacionais** (*relações entre expressões aritméticas*) em *C* são os seguintes:

Tab. 1. Tabela dos operadores relacionais.

Operação	Operador	Exemplo
iguais	==	n1 == 4
diferentes	!=	n1 != 4
maior	>	n1 > n2
menor	<	n2 < n1
maior ou igual	>=	n1 >= 3
menor ou igual	<=	n1 <= 33

Os **operadores lógicos** principais em *C* são

Tab. 2. Tabela dos operadores lógicos em C.

Operação	Operador	Exemplo
conjunção (`e`)	&&	(n1 > 1 && n1 < 10)
disjunção (`ou`)	||	(n1 > 1) || (n1 < 10)
negação (`não`)	!	!(n1 > 1 || n1 < 10)

Comando de seleção `if-else` em *C*

O **comando de seleção** é essencial para qualquer *software*, é ele que permite que o programa desvie seu fluxo de execução de acordo com os valores dos dados (as já citadas *variáveis*). Ele está presente em qualquer linguagem de programação e está baseado no cômputo de uma *expressão lógica* (**EXPLOG**), se o resultado dela for *verdadeiro*, o fluxo segue por um "caminho", senão segue por outro. Essa estrutura está ilustrada na figura abaixo:

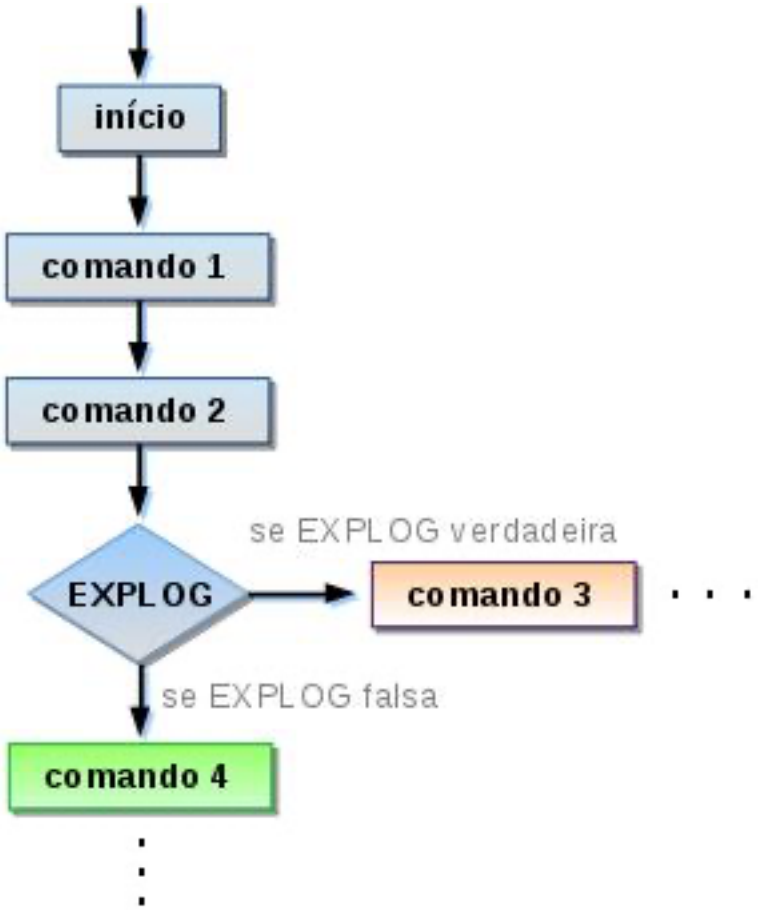

Fig. 2. Diagrama de fluxo de execucação do comando de seleção.

Existem outras duas características de comando de seleção (em qualquer linguagem) que são:

1. permitir a execução de comandos mutuamente exclusivo, ou seja, se a expressão lógica resultar verdadeira executa-se determinado comando, senão executa-se outro comando ("comando alternativo") - mas a existência do "senão" é opcional;
2. possível subordinar mais de uma instrução ao comando "se" ou ao comando "senão", mas cada linguagem tem a sua sintaxe particular para isso (marcadores '{' e '}' em *C*).

A linguagem *C*, assim como várias outras, usam as masmas *palavras reservadas* (palavras que *não* podem ser usados como nomes para variáveis): as palavras `if` e o `else`. Sua sintaxe é ilustrada na tabela abaixo.

Além disso, as linguagens permitem que vários comandos (**bloco de comandos**) sejam subordinados à uma seleção. Em *C* deve-se usar abre e fecha chaves (`{...}`) para delimitar os blocos.

A tabela a seguir ilustra a sintaxe do comando de seleção em *C*:

Tab. 3. Comandos de seleção em C.

Apenas 1 comando	Vários comandos

`if (EXPR) comando1;`
`else comando2;`
	`if (EXPR) { comando1;`
	`  comando2; }`
	`else { comando3;`
	`  comando4; }`

Vale destacar que o comando `else` é opcional (como já citado), ou seja, não existindo ação a ser executada caso a condição seja falsa, não é necessário usar um `else` com algum comando. Assim é correto em *C* escrever: `if (1==1) printf("ok");`

Comandos de seleção `if-else` aninhados em *C*

Em geral, toda linguagem de programação de alto-nível permite que os comandos `if-else` sejam utilizados de modo **aninhado**, ou seja, qualquer dos comandos acima (`comando1`... `comando4`) podem ser um novo `if-else`. Por exemplo:

Tab. 4. Comandos de seleção aninhados em C.

Linguagem	if aninhado

C	`if (a<b) { // este exemplo nao trata todos os casos...`
	`  if (b<c) printf("O menor e': %d\n", a);`
	`  }`
	`else if (b>=c)`
	`  printf("O maior e': %d\n", a);`

Note que a regra para associar um `else` a um `if` é por proximidade (o `else` é associado ao `if` mais próximo), e em caso que precisamos que o `else` seja associado a um `if` mais "distante", somos obrigados a usar os blocos de comandos. Isso está ilustrado nos exemplos acima, o primeiro `else` está associado ao `if (a<b)` e não ao mais "próximo".

Um exemplo para decidir se uma variável inteira armazena um valor par ou ímpar

Utilizando a divisão inteira (quer dizer `7/2` resultando `3`) pode-se conseguir determinar se um inteiro é ou não par. Por exemplo, dispondo de uma variável inteira `n1`, pode-se fazer a divisão inteira por *2* e multiplicar seu resultado também por *2*, se recuperar o valor integral de `n1`, então esse é um valor par (e.g. *7/2* é *3* que multiplicado por *2* resulta "apenas" *6*, concluindo daí que *7* não é par - vide explicação estendida).

No código abaixo ilustramos este método e no seguinte com outro recurso.

Tab. 5. Exemplo de expressão aritmética em expressão lógica para decidir se valor é par em C.

Código *C*
`if (n1/2*2==n1)`

```
  printf("%d eh par\n", n1);
else
  printf("%d eh impar\n", n1);
```

Mas pode-se obter o mesmo resultado utilizando o operador binário `%` que é "resto da divisão inteira". Este operador devolve o resto da divisão, por exemplo, `7%2` resulta `1` (o quociente é `3`: `3 * 2 + 1 = 7`). Assim o código fica:

Tab. 6. Exemplos de comandos para decidir se valor é ímpar ou par em C.

Código *C*

```
if (n1%2==0)
   printf("%d eh par\n", n1);
else
   printf("%d eh impar\n", n1);
```

Mas por que razão "n1/2*2==n1" é suficiente para decidir se n1 é ou não par?

Bem, isso é um *truque computacional* baseado em como o computador interpreta e trata os valores numéricos de acordo com seu tipo. Quanto todos os valores (variáveis) são inteiros, é utilizada uma aritmética de precisão inteira, como examinamos ao final da seção sobre variáveis e expressões.

Lembre-se que em *C* sempre que fizer divisão entre inteiros o resultado é inteiro (ou seja, `7/2` resulta `3`). Novamente, é o contexto determinando o resultado.

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

Alterações