Sobre a estrutura básica de um programa em *C*

Como já citado a diferença entre uma *linguagem de alto-nível* e uma *linguagem de baixo-nível* é que a primeira pode mais facilmente ser "lida" por uma pessoa. Desse modo, é recomendável que você procure sempre deixar o código melhor organizado. Um dos recursos para isso é a utilização adequada de *deslocamento de espaços em branco para iniciar uma linha de comando*, o que é chamado de "indentação". Algo como:if (CONDICAO) comando1; // note o espacamento nessa linha para indicar que ela esta' subordinada 'a anterior

1. Comentários em meio ao código e o uso adequado de "indentação".

Toda linguagem de programação apresenta o conceito de **comentário**, que serve para indicar ao compilador *C* que o texto em *comentário* **não** deve ser *compilado* ou *interpretado*, respectivamente.

Geralmente as *linguagens de programação* oferecem dois modos para comentários, um para *comentar apenas o trecho final de uma linha* e outro para *comentar várias linhas* (ou *bloco de comentários*). Os comentário do primeiro tipo em *C* usam duas barras, como abaixo:

```
    // ignore o restante desta linha
```

Já os comentários em bloco, precisam de uma marca para abrir uma marca para indicar seu fechamento (nada dentro dele será compilado), como indicado abaixo:

```
 /* Ignore todas as linhas entre estas marcas
  printf("Esse comando de impressao sera' ignorado!\n");
  printf("Esse tambem...\n");
  ... */
```

Entretanto, sugiro que **evite** usar esse tipo de comentários, deixando-o reservado para ajudar na depuração do código!
A razão disso é que **não** é possível comentário dentro de comentário, como detalhado na próxima seção.

1.1 Comentários dentro de comentário não é possível em *C*.

Vale a pena ressaltar a impossibilidade "comentário dentro de comentário", ou seja, se abrir um bloco de comentário /*, então a próxima marca /* será ignorada e ao encontrar a primeira marca */, essa será interpretada como o fechamento da primeira abertur, **não** como fechamento da segunda tentativa de abertura de comentários.
Nesse sentido é **diferente** dos parênteses em expressões matemáticas, nas quais é perfeitamente válido "parênteses dentro de parênteses", como em "`(2 * (3+4))`".

Para ressaltar a impossibilidade de "comentário dentro de comentário", apresentamos o erro que o compilador *C* indicaria. Experimente digitar as 4 linhas seguintes (e depois executar seu código):

```
/*
```

```
Esta e' uma tentativa de colocar comentario dentro de comentario, mas NAO
funciona!
/* Ao encontrar o proximo fecha, ele correspondera' ao primeiro abre, */
logo esta linha NAO mais sera' comentario resultando erro de compilacao!
Assim, comentarios encaixados NAO sao permitidos!
*/
```

Cód. 1. Código ilustrando o erro ao supor que existe comentários aninhados em C.

Ao tentar fazer o computador interpretar o código acima, o interpretador indicará o erro, como indicado na figura 1. Experimente em seu interpretador.

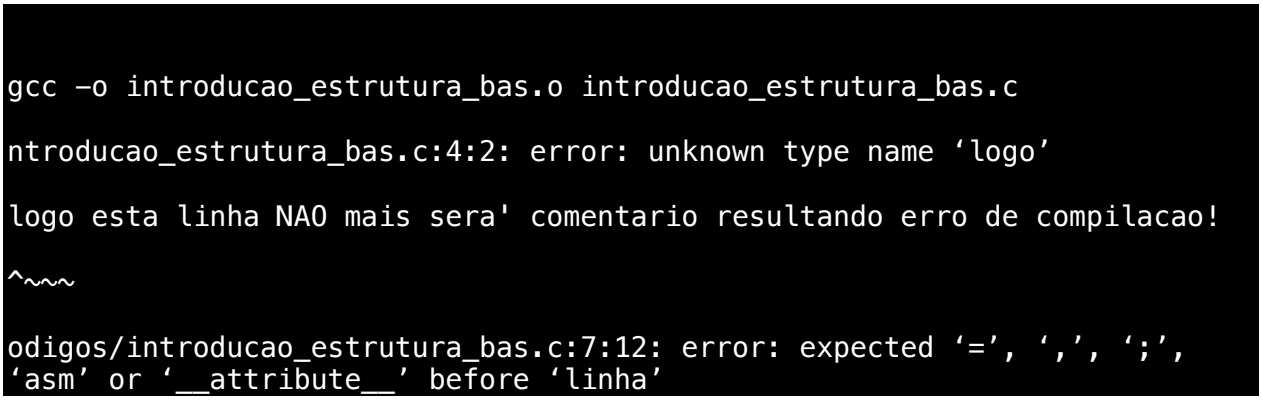

Fig. 1. Ilustração do erro ao supor que existe comentários aninhados em C.

A razão é que o **aninhamento** de comentários **não** existe na configuração padrão *C*, assim o segundo /* é ignorado por estar dentro de um bloco de comentários e, desse modo, o primeiro fecha */ a ser encontrado é associado à primeira abertura, gerando o erro na linha 4 `logo esta linha NAO mais sera' comentario resultando erro de compilacao!`.

Por esta razão recomendo que em seus programas utilize comentários em várias linhas apenas no início do programa (para por exemplo, identificar-se e explicá-lo). Deste modo, fica fácil isolar um bloco grande de seu código (usando comentários em bloco), o que é útil para encontrar erros de sintaxe (mais simples) ou de semântica.

2. Uso adequado de "indentação".

Entenda-se por **"indentação"** o deslocamento horizontal de trechos de código, o que é usado para indicar claramente uma subordinação de comandos (ou **aninhamento**). Por exemplo, se um comando de seleção "`se`" tiver dois comandos subordinados à ele, pode-se usar 2 espaços em branco a mais nesses comandos subordinados. Veja como ficaria esse exemplo:

```
 if (x>0) {
   S++; // equivale a: S = S+1;
   printf("S=%d\n", S);
   } // Em C "indentacao" NAO e' obrigatorio, mas deixa o codigo muito
mais claro!
```

Cód. 1. Código ilustrativo de comandos subordinados ("indentação").

Para evitar que os códigos fiquem muito "longos" (não cabendo uma linha inteira na tela), sugerimos utilizar apenas 2 espaços para indicar subordinação (como no extrato de código acima).

Assim, se determinado comando começa na coluna *X* ("indentação" de *X* espaços), então todos os comandos subordinados a ele terão deslocamento de ao menos *X+2* espaços em branco. Digo "ao menos", pois dentre estes comandos pode existir outras subordinações. Isso é ilustrado no exemplo abaixo, no qual os comandos das linhas 2 até 5 estão subordinados ao "`repita enquanto`", tendo portanto deslocamento *X+2* ou mais espaços, mas o comando "`print`" da linha 4 tem deslocamento *X+4*, pois ele está subordinado ao "`if`" que já tinha deslocamento *X+2*.

A seguir um exemplo com mais linhas e mais subordinações.

```
      while (cont < N) {
         print("%d\n", cont);
         if (cont % 2 == 0) // "cont % 2" e' o resto da divisao inteira,
logo testa se 'cont' e' par"
           print("%d\n", cont);
         cont++;
         }
```

Cód. 2. Código ilustrativo de comandos subordinados ("indentação").

Note que no código acima, os comandos "`imprima(cont)`" e "`incremente(cont)`" estão deslocados 2 espaços à direita para indicar que ambos estão sobordinados ao comando de repetição "`while (cont < N)`" (laço de repetição será explicado em outra aula). Já o comando da linha 4 está subordinado ao comando da linha 3 ("`if`"), por isso o uso do deslocamento adicional de dois espaços em branco.

3. Estrutura básica de um programa em *C*.

Abaixo ilustro a estrutura básica de um programa em *C*, utilizando indentação para deixar o código mais claro, o que é importante para você detectar problemas em seu código e essencial para receber uma boa nota por parte do monitor que analisará seu programa.

```
/*
 O programa abaixo serve apenas para ilustrar a estrutura basica de um programa
em C.
 Neste primeiro exemplo existe:
 - Carrega o cabecalho para os comandos (ou funcoes) para entrada e saida de
dados: biblioteca "stdio.h"
 - Definicao do codigo principal (por onde devera ser iniciada a execucao de seu
codigo): funcao "main"
```

```
 - Comandos para leitura e para saida de dados: respectivamente, "scanf" e
"printf"
   * Para declarar 2 variaveis que armazenarao valores inteiros, utilizar o "int"
como abaixo
   * Para ler valores (usuario digita) e armazenar na variavel usando a funcao
predefinida "scanf"
   * O "scanf" deve sempre 2 grupos, o primeiro entre aspas e o segundo uma lista
de variaveis precedias do '&'
     dentro das aspas, '%d' indica que o valor lido sera' tratado como inteiro
     na lista de variaveis, o uso de '&' e' obrigatorio para conseguir guardar o
valor lido corretamente na variavel
     (experimente eliminar um dos '&' no programa abaixo e veja o que ocorre)
   * O "printf" adota a mesma estrutura do "scanf" com os 2 blocos, mas nele NAO
usar o '&' antes da variavel,
     senao sera' impresso o endereco de memoria da variavel, nao o valor que ela
guarda)
     O '\n' e' serve para que, apos imprimir "n1+n2", "pular" de linha (ou seja,
proxima impressao inicia-se na seguinte)
   * Os formatadores (como '%d') serao explicados noutra aula, eles podem ser
usados para imprimir tabelas
*/

#include <stdio.h> // coloque esta linha em TODOS seus programas C (a mesno que
ele NAO tenha nem entrada, nem saidas!)
int main (void) {  // obrigatorio em todo programa C (noutra aula explicamos por
que usar "void" aqui)
  int n1, n2;                 // declarando o uso de 2 variaves para armazenar
apenas valores inteiros
  scanf("%d %d\n", &n1, &n2); // leia 2 valores inteiros, guarde o primeiro em n1
e o outro em n2 (depois veremos '%d' e o '&')
  printf("%d\n", n1+n2);      // imprima um unico valor (soma dos valores
guardados em n1 e em n2)
  }
```

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

Alterações