**Alunos**

Gabriel Augusto

Nathan Oliveira

Lucas Augusto

**Professor**

Roger Kreutz Immich

**Matéria**

Introdução às Técnicas de Programação

25 de outubro de 2019.

**Regras de estilo do código fonte**

Que organização de código é algo necessário para o dia a dia dos programadores todos nós já sabemos. Como requisitado, iremos detalhar ao longo deste documento sobre as regras que vamos seguir ao desenvolver o trabalho da terceira unidade.

# INDENTAÇÃO

O uso da indentação servirá para separar visualmente blocos lógicos dentro de um programa. Neste documento iremos considerar como blocos lógicos:

1. Corpo de função;
2. Corpo de estrutura condicional;
3. Corpo de estrutura de repetição; e
4. Corpo de definição de estrutura (*struct*).

O espaço de indentação será de 4 espaços de caracteres:

|  |
| --- |
| //Exemplo 1.1  if(A == B){  printf("A igual a B");  } |

# 

As chaves “{ }” dos blocos lógicos se inicializarão na mesma linha da passagem de condição dos laços *if/else/for/while/switch*, sem um espaço após o “( )”. E a chave de fechamento ficará na linha seguinte após a última linha de comando do bloco e estará alinhado por indentação com o tipo de laço, como está exemplificado com o laço *for* abaixo.

|  |
| --- |
| //Exemplo 1.2  for(int i = 0; i < len; i++){  printf("%d \n", i); } |

# 

Ainda que não seja necessário, para fins de padronização, o *if/else* mesmo que contenha apenas uma linha de comando, ainda iremos utilizar as chaves de abertura e fechamento como mostra o exemplo 1.1.

# NOMES DE VARIÁVEIS

Será usado o formato *camel case* para nomenclatura de variáveis. A nomenclatura para as variáveis de iteração serão comumente consoantes: *i, j, k, w*.

|  |
| --- |
| //Exemplo 2.1  int tamanhoDaMatriz = 0;  for(int i = 0; i < tamanoDaMatriz; i++){  printf("%d", i); } |

# 

Para inicialização de variáveis do mesmo tipo, não utilizaremos variáveis com atribuições e sem atribuições na mesma linha. E, no caso de haver a inicialização de um ponteiro e uma variável do mesmo tipo, as duas declarações devem igualmente ficar em linhas distintas, como mostra o exemplo 2.2 abaixo.

|  |
| --- |
| //Exemplo 2.2  //correto int primeiro, segundo, terceiro;  //errado int primeiro, segundo = 0, terceiro = 0;  //correto int \*p; int n;  //errado int \*p, n; |

Ademais, a declaração de ponteiros deverá seguir esta formatação, com o caractere característico “\*” sem espaço com a respectiva variável:

|  |
| --- |
| int \*p; |

e não contendo espaços:

|  |
| --- |
| int\* p; int \* p; |

# 

# COMENTÁRIOS

Em implementações de funções, uma boa prática consiste em se introduzir um comentário antes do cabeçalho contendo as seguintes informações:

* descrição da função
* descrição dos parâmetros
* descrição do retorno

|  |
| --- |
| /\*  \* nomeFuncao: o que a função faz  \* param1: tipo e o que é  \* retorno: tipo e o que irá retornar  \*/ int nomeFuncao(int param1){  return param1 + 1; } |

Comentários de uma linha deverão ser precedidos de “//” e sem espaçamento entre estas e o começo da escrita, além de ser uma linha acima do bloco que estará sendo explicado.

# ESPAÇAMENTOS E QUEBRAS DE LINHA

Expressões envolvendo operadores lógicos e aritméticos, deverão conter espaços separando os operandos dos operadores. Em caso de uso de parênteses para separar uma expressão, não terá espaços entre os parênteses e a expressão digitad;.

|  |
| --- |
| int n = n + 1; int j = 5 + (2 \* 3); |

Em casos nos quais seja necessário quebrar uma expressão ou chamada de função em mais do que uma linha, utilizar uma das seguintes regras:

- quebrar após uma vírgula; e

- quebrar após um operador.

No caso de impressão de strings (utilizando *printf* ou equivalente) com uma grande quantidade de caracteres é interessante utilizar várias chamadas de função em sequência, evitando que uma linha de código contenha um número excessivo de caracteres. Esta providência facilita a navegação no código-fonte e também a sua impressão.

Não haverá espaço entre alguma instrução de controle ou repetição e os parênteses.

|  |
| --- |
| for(int i = 0; i < 10; i++){  printf("%d", i); } |

# TAMANHO DE LINHA

Para que não haja uma linha com número excedido de caracteres utilizaremos uma quantidade máxima de caracteres por linha de 100.

# MODULARIZAÇÃO

Nos arquivos de código “.c” estarão contidos propriamente dito o corpo do programa, com todos os blocos lógicos. Já no arquivos header “.h” estarão contidos as funções, *structs* e definição de constantes que serão utilizadas no programa principal “.c”. Teremos 1 arquivo “.c” contendo a função *main* e rodando todas as funções e variáveis definidas em outro arquivo “.h”.

# Trabalhos em que nos baseamos

<https://cs50.readthedocs.io/style/c/>

http://www.dainf.ct.utfpr.edu.br/~pbueno/Arquivos/Regras\_de\_Estilo\_em\_C.pdf