

# Padrão de Codificação para a Linguagem C++

Gustavo Yudi Bientinezi Matsuzake

May 19, 2015

## 1 Sobre este documento

Este<sup>1</sup> *padrão de codificação*<sup>2</sup> foi inspirado nas normas e padrões de codificação do Google®, do projeto GNU® e algumas preferências pessoais do autor. Essas organizações foram escolhidas porque representam uma parcela do mercado de software importante, e aderir os costumes e trejeitos de cada, pode ser de suma importância para garantir uma boa prática organizacional - uma vez que esses métodos sofreram diversos tipos de testes - e acolhida por um grande número de programadores no mundo todo.

## 2 Documentação

Todo tipo de documentação deve ser mantida atualizada, tanto a da interface gráfica quanto internamente e também a interface de texto, caso sua programação seja de acordo com os padrões *POSIX*<sup>3</sup>.

- Todas as opções de linha de comando (incluindo todos os argumentos `-arg`) devem ser documentados;
- Todas as mudanças devem ser documentadas;
- Em geral, a documentação de todos os aspectos documentados, tanto front-end e back-end devem ser sempre atualizados, e sempre que houver a possibilidade, arrumar e corrigir erros de documentação para evitar gaps de documentação.

## 3 C++

C++ é uma linguagem complexa. Tendo isso em mente, primeiramente, seja sensato! Use a linguagens e seus benefícios para o bem, em boas formas. Se você usa a linguagem de forma usual, pense bem antes de usar, pessoas podem fazer

---

<sup>1</sup>Disponível em [www.github.com/yudi-matsuzake/padrao-de-codificacao-cpp](http://www.github.com/yudi-matsuzake/padrao-de-codificacao-cpp)

<sup>2</sup>Do inglês, *coding standard*.

<sup>3</sup>Conjunto de regras para interfaces de texto e sistemas operacionais inspirado no UNIX.

a manutenção do seu código, então, priorize a legibilidade e a produtividade. Não estamos em uma competição de quem sabe mais "coisas loucas" sobre a essa linguagem.

## 4 Funções inline

Use funções inline apenas quando as funções forem muito pequenas e/ou for pouco reutilizadas. Funções de tamanho significativo pode deixar seu programa mais lento! Seja crítico e pense duas vezes antes de usar funções linearizadas em funções com loops.

## 5 Definição de classes

Somente defina classes para tipos *não-EDP*. *EDP*<sup>4</sup> (Estrutura de Dados Passiva) são tipos onde não são necessários as features de orientação a objetos.

## 6 Parâmetros de funções

Quando definir uma função, a ordem dos parâmetros é: entradas, então saídas.

Parâmetros de funções no c/c++ podem ser tanto entradas, quanto saídas (ou os dois!). Entradas são geralmente constantes<sup>5</sup> e saídas são poiteiros.

## 7 Namespaces

Namespaces são encorajados, porque são muito úteis em evitar colisões de nome no escopo global. *inline namespaces* são desencorajados, por outro lado, a não ser em caso de compatibilidade com códigos passados ou softwarares antigos.

inline namespaces são utilizados para casos onde não há diferença entre `X::Y::foo()` e `X::foo()` com o código abaixo:

```
1 namespace X{
2     inline namespace Y{
3         void foo();
4     }
5 }
```

---

<sup>4</sup>Os tipos EDP no C++ são definidos como um tipo escalar. Os tipos PDS não tem definidos operadores, destrutores e membros estáticos entre vários tipos do mesmo EDP. Não tem também construtores, membros privados, protegidos ou funções virtuais.

<sup>5</sup>É uma boa prática, ajuda na leitura do código e faz parte da documentação definir entradas de funções como 'const' (constantes).

## 7.1 Namespaces sem nome

Namespaces sem nome são encorajados em arquivos `{.cpp,cc}` mas não faça em `{.h,hpp}`. Em arquivos `{.c,cpp}` namespaces sem nome auxiliam no maior encapsulamento e na proteção de funções e membros desses arquivos, por isso, não faz sentido namespaces sem nome serem utilizados no `{.h,hpp}`.

Exemplo:

```
1 namespace {                                // Isso e um arquivo {cpp,cc}
2
3 // O conteudo do namespace nao e indentado
4 //
5 // Garantido que nao ira gerar simbolos que possam colidir
6
7 //Essa funcao nunca ira ser chamada de fora do .cc
8 bool UpdateInternals(Frobber* f, int newval) {
9     /*
10     ...
11     */
12 }
13
14 } // namespace
```

## 8 Arquivos de cabeçário

Em geral, todo arquivo `.cc` deve ter um arquivo de cabeçário<sup>6</sup> `.hpp`<sup>7</sup> associado. Tem algumas exceções, como pequenos `.cc` e/ou `.cc` contendo a função `main`. O uso correto desses arquivos pode fazer a diferença na leitura no tamanho e na performance do código.

### 8.1 Proteção contra múltiplos `#includes`

Todo cabeçário deve ser protegido contra múltiplos `#includes`<sup>8</sup>. Ou seja, todo cabeçário deve ter a seguinte estrutura ou algo parecido:

```
1 #ifndef _MEU_CABECARIO_H_
2 #define _MEU_CABECARIO_H_
3
4 /*      definicoes      */
5 /*      ...      */
6
7 #endif \_MEU_CABECARIO_H_
```

---

<sup>6</sup>Do inglês, *header*. São os arquivos `.h`, `.hpp` ou `.H` da linguagem `c/c++`.

<sup>7</sup>Mas nem todo `.hpp` tem um `.cc` associado, como classes com templates totalmente genérico. Mas não recomendamos o uso dessa também, só em algumas exceções.

<sup>8</sup>Proteção feita por uma definição, para evitar múltiplas definições de estruturas e funções do cabeçário.

## 8.2 Nomes e ordem

Use a seguinte ordem padrão para melhorar a leitura do código e para evitar dependências escondidas: Cabeçalhos relacionados, biblioteca padrão do c, biblioteca padrão do c++, outros `.{h,hpp}`, seus `.{h,hpp}`.

Exemplo:

```
1  #include "foo/server/fooserver.h" //relacionado
2
3  #include <sys/types.h> //c
4  #include <unistd.h> //c
5  #include <hash_map> //c++
6  #include <vector> //c++
7
8  #include "base/basicctypes.hpp" //hpp do projeto
9  #include "base/commandlineflags.hpp" //hpp do projeto
10 #include "foo/server/bar.hpp" //hpp seu
```

Todos os cabeçalhos devem ser listados decentemente no código do projeto sem o uso de atalhos do padrão UNIX, e.g., `.`<sup>9</sup> e `..`<sup>10</sup>. Por exemplo, o cabeçalho `life-change-project/src/base/logging.h` deverá ser incluído como:

```
1  #include "base/logging.h"
```

## 9 Opções do Compilador

O compilador deve compilar seu código sem nenhum warning com as tags `-Wall` `-Wextra`.

## 10 Convenções de formatação

### 10.1 Tamanho da linha

É recomendado que a linha possua no máximo 80 caracteres (80 colunas).

### 10.2 Nomes

MACROS devem ser definidas `TUDO_EM_MAIUSCULA`, quando são macros simples. Alguns *wrappers* eficientes para funções, podem ser definidos em letra minúscula. Outros nomes devem ser em letras minúsculas separados `_por_underscore`. É comum também, utilizar `_t` no final das estruturas, e.g. `minha_estrutura_t`.

---

<sup>9</sup>Autoreferência

<sup>10</sup>Referência do diretório pai

## 10.3 Expressões

Para	Use...	...ao invés de...
Negação lógica	!x	! x
Complemento bitwise	~x	~ x
Menos unário	-x	- x
Cast	(foo)x	(foo) x
Referencia de um ponteiro	*x	* x

## 10.4 Classes

Se a definição da classe cabe em uma única linha, deixe em uma única linha.

Senão, siga as seguintes regras.

Não indente rótulos de proteção (public, private, virtual).

Prefira colocar o cabeçalho da classe em uma única linha.

```
1 class myclass : base {
```

Se não, coloque o ponto-e-vírgula na outra linha:

```
1 class a_rather_long_class_name
2 : with_a_very_long_base_name, and_another_just_to_make_life_hard
3 {
4     int member;
5 };
```

Se as cláusulas de herança passar de uma linha, inicialize na próxima linha com a indentação de dois espaços:

```
1 class myclass
2 : base1 <template_argument1>, base2 <template_argument1>,
3   base3 <template_argument1>, base4 <template_argument1>
4 {
5     int member;
6 };
```

Quando definir uma classe (nessa ordem):

- Primeiro definir todos os tipos públicos depois os não-públicos;
- Então declarar todos os construtores públicos depois os destrutores públicos;
- Declarar todos os membros de função públicos;
- Declarar todos os membros de variáveis públicas;
- Declarar todos os construtores não-públicos, então, destrutores não públicos;
- Declarar todos os não públicos membros de função e
- Declarar todos os não públicos membros de variáveis