Padrão de Codificação para a Linguagem C++

Gustavo Yudi Bientinezi Matsuzake

May 19, 2015

1 Sobre este documento

Este[1] *padrão de codificação*[2] 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*[3].

- 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

[1] Disponível em `www.github.com/yudi-matsuzake/padrao-de-codificacao-cpp`
[2] Do inglês, *coding standard*.
[3] 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 Arquivos de cebeçário

Em geral, todo arquivo .cc deve ter um arquivo de cabeçário[4] .hpp[5] associoado. 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 performace do código.

4.1 Proteção contra múltiplos #includes

Todo cabeçário deve ser protegido contra múltiplos #includes [6]. 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_
```

4.2 Nomes e ordem

Use a seguinte ordem padrão para melhorar a leitura do código e para evitar dependencias escondidas: Cabeçários 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/basictypes.hpp" //.hpp do projeto
9  #include "base/commandlineflags.hpp" //.hpp do projeto
10 #include "foo/server/bar.hpp" //.hpp seu
```

[4] Do inglês, *header*. São os arquivos .h, .hpp ou .H da linguagem c/c++.

[5] Mas nem todo .hpp tem um .cc associoado, como classes com templates totalmente genérico. Mas não recomendamos o uso dessa também, só em algumas exeções.

[6] Proteção feita por uma definição, para evitar múltiplas definições de estruturas e funções do cabeçário.

Todos os cabeçários devem ser listadas decentemente no código do projeto sem o uso de atalhos do padrão UNIX, e.g., .[7] e ..[8]. Por exemplo, o cabeçário life-change-project/src/base/logging.h deverá ser incluído como:

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

5 Opções do Compilador

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

6 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.

7 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[9] e saídas são poiteiros.

8 Convenções de formatação

8.1 Tamanho da linha

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

8.2 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_underline. É comum também, utilizar _t no final das estruturas, e.g. minha_estrutura_t.

[7] Autoreferência

[8] Referência do diretório pai

[9] É uma boa prática, ajuda na leitura do código e faz parte da documentação definir entradas de funções como 'const' (constantes).

8.3 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