Padrão de Codificação para a Linguagem C++

Gustavo Yudi Bientinezi Matsuzake

May 19, 2015

1 Sobre este documento

Este *padrão de codificação*[1] 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*[2].

- 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 Arquivos de cebeçário

Em geral, todo arquivo .cc deve ter um arquivo de cabeçário[3] .hpp[4] associoado. Tem algumas exceções, como pequenos .cc e/ou .cc contendo a função main. O

[1] Do inglês, *coding standard*.

[2] Conjunto de regras para interfaces de texto e sistemas operacionais inspirado no UNIX.

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

[4] 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.

uso correto desses arquivos pode fazer a diferença na leitura no tamanho e na performace do código.

3.1 Proteção contra múltiplos #includes

Todo cabeçário deve ser protegido contra múltiplos #includes [5]. Ou seja, todo cabeçário deve ter a seguinte estrutura ou algo parecido:

```
#ifndef _MEU_CABECARIO_H_
#define _MEU_CABECARIO_H_

/*        definicoes        */
/*           ...            */

#endif \\_MEU_CABECARIO_H_
```

3.2 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:

```
#include "foo/server/fooserver.h" //relacionado

#include <sys/types.h> //c
#include <unistd.h> //c
#include <hash_map> //c++
#include <vector> //c++

#include "base/basictypes.hpp" //.hpp do projeto
#include "base/commandlineflags.hpp" //.hpp do projeto
#include "foo/server/bar.hpp" //.hpp seu
```

Todos os cabeçários devem ser listadas decentemente no código do projeto sem o uso de atalhos do padrão UNIX, e.g., .[6] e ..[7]. Por exemplo, o cabeçário life-change-project/src/base/logging.h deverá ser incluído como:

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

4 Opções do Compilador

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

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

[6] Autoreferência

[7] Referência do diretório pai

5 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.

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[8] e saídas são poiteiros.

7 Convenções de formatação

7.1 Tamanho da linha

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

7.2 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.

7.3 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

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