CODE STYLE

Olá a todos,

Bem-vindos ao canal,

vamos falar sobre code style, que é a forma como você codifica, alinha
parágrafos, coloca espaços, comenta o código; em fim, o seu jeito de codificar.

CODE STYLE

Evidentemente este é o meu padrão, se for útil utilize algumas coisas,
modifique e crie o seu para suas necessidades.

Então este vídeo será diferentes dos outros sendo mais didático.

CODE STYLE

O mais importante é adotar um padrão e segui-lo, assim as pessoas reconhecerão seu código e você poderá entender mais claramente códigos mais antigos que você tenha mexido.

CODE STYLE

Este code style foi criado utilizando a linguagem C, mas server como guia
para outras linguagens.

CODE STYLE

- Identação

  - utilizar 4 espaços

  - não utilizar tab

CODE STYLE

- declaração de variáveis

- cada variável deve estar sozinha em uma linha

```
int a,b; // errado
int a;
int b;
```

CODE STYLE

- nome das variáveis

- a variável deve começar com o primeiro nome em minúscula e o restante do nome deve ser capitalizado.

CODE STYLE

- nome das variáveis

```
int idadedofilho;     // errado
int idadeDoFilho;
int Contador;     // errado
int contador;
```

CODE STYLE

- nome das variáveis

- não colocar o tipo da variável na declaração.
Notação húngara, criada por Charles Simonyi.

```
int intContador;    // errado
int contador;
```

CODE STYLE

- inicialização de variáveis

- todas as variáveis devem ser inicializadas independente do escopo

```
int a( 0 );
Classe c( 0, "teste", 1.3 );
```

CODE STYLE

- chaves

- todo comando condicional deve conter chaves (iniciais e finais)

- para delimitar sua área de atuação.

CODE STYLE

- chaves

- as chaves iniciais e finais devem começar na próxima linha e separadas do código

- condições vazias, também devem ser delimitadas por chaves

CODE STYLE

- chaves

```
while( true ) // errado

while ( true )
{
}
```

CODE STYLE

- chaves

```
if ( true ) contador = 0; //
errado
else contador = -1;
```

CODE STYLE

- chaves

```
if ( true )
{
    contador = 0;
}
else
{
    contador = -1;
}
```

CODE STYLE

- parênteses

- utilizar para agrupar expressões ou isolar condições

```
if ( a && b || c )      // errado
    if ( ( a && b ) || c )
```

CODE STYLE

- parênteses

```
i = 10+y/x+2                    // errado
i = ( ( 10 + y ) / x ) + 2

while( a == true and x > 10 )
while( ( a == true ) && ( x > 10 )
)
```

CODE STYLE

- utilizar espaçamento entre os operadores e delimitadores [, {, (,), } e]

```
char texto[10] = {‘a’,’b’};
char texto[ 10 ] = { ‘a’, ‘b’ };
```

CODE STYLE

- classes

```
- o nome deve ser capitalizado
processarDados     // errado
ProcessarDados
BancoDeDados
Carregar
ConectarMaquina
```

CODE STYLE

- classes

  - o nome do arquivo deve ser totalmente em minúsculas.
  Ex.: conectarmaquina.cpp

  - cada classe deve ter seu próprio arquivo .cpp e .h

CODE STYLE

- enum

- todo nome de enum deve ser capitalizado assim como suas declarações.

CODE STYLE

- enum

- sempre iniciar o primeiro item do enum com o valor correspondente

```
enum Status
{
    Ready = 0,
    Loading,
    Error
};
```

CODE STYLE

- switch

```
switch( campo )
{
    case ‘A’:
        return 1;

    case ‘B’:
        return 2;
}
```

CODE STYLE

- namespace

```
namespace Controle
{
    class Seguranca : public QThread
    {
        Q_OBJECT
    }
}
```

CODE STYLE

- namespace

- especificar o namespace a que se refere

```
using namespace std;

int main()
{
    cout << "Hello << endl;
}
```

CODE STYLE

- namespace

```
int main()
{
    std::cout << "Hello <<
std::endl;
}
```

CODE STYLE

- cast
  evitar o uso de cast no padrão C, utilizar o padrão C++
  const_cast, static_cast, dynamic_cast

  - testar se o cast foi bem sucedido

CODE STYLE

- new / delete

- todo comando new deve ter sua exception testada.

CODE STYLE

- new / delete

```
try
{
    buffer = new char [ TAM_BUFFER
];
}
catch( std::bad_alloc )
{
    //erro
}
```

CODE STYLE

- new / delete

- todo comando new deve ter um delete correspondente implícito ou não.

CODE STYLE

- funções
- utilizar o padrão de documentação
- toda função deve ter somente um proposito
- todo parâmetro que não necessita ser alterado pela função deve conter const em sua declaração para evitar alterações incorretas.

CODE STYLE

- comentários
  - todo comentário que explica uma codificação deve estar na linha anterior a que se refere e alinhado.

```
// commment
if ( isFoo )
```

CODE STYLE

- comentários
  - para comentários multiplos utilizar:

```
// commment
// commment
// commment
if ( isFoo )
```

CODE STYLE

- regras gerais
  - quanto a clareza de código:

```
QSlider * slider = new
QSlider( 12, 18, 3, 13,
Qt::Vertical, 0, "volume" );
```

CODE STYLE

- regras gerais
  - é menos claro que:

```
QSlider * slider = new
QSlider( Qt::Vertiçãl );
slider->setRange( 12, 18 );
slider->setPageStep( 3 );
slider->setValue( 13 );
slider->stObjectName( "volume" );
```

CODE STYLE

- regras gerais

- sempre pensar no mínimo uso da memoria, crie um conteúdo uma única vez e repasse o ponteiro ou referência para a próxima etapa do processamento (evite a criação de cópias do mesmo conteúdo).

CODE STYLE

- regras gerais

- utilize funções virtuais somente se você deseja reimplantá-las, caso contrário, não utilize a diretiva virtual pois, ela cria uma lista desnecessária em memória de ponteiro de função.

CODE STYLE

- regras gerais

- utilizar prefixo ou sufixo comuns ao mesmo tipo de categoria.
input_file
transaction_file
reject_file
file_input
file_transaction
file_reject

CODE STYLE

- regras gerais

- evitar variação de nomes padrões:
index
idx
ind
index2
indx

CODE STYLE

- regras gerais

- evitar a escolha de nomes parecidos:

categoriaHomemAlto
categoriaHomemAlvo

CODE STYLE

- regras gerais

- identificar o programa/classe: nome, data/hora da criação, versão, proposito, comentários relevantes, referencia de documentação

CODE STYLE

- regras gerais

- manter a documentação simples, mas descritiva

- evite comentários óbvios:
// adicionar 1 ao contador
contador++

CODE STYLE

- regras gerais

- o alinhamento dos operadores de ponteiro(*), e referência (&) deve ficar no meio da declaração.

```
char * foo1;
char & foo1;
```

CODE STYLE

- regras gerais

- ler o código demora mais tempo do que escrevê-lo.

- cada função deve executar uma e somente uma tarefa

- entenda o problema completamente antes de codificar

CODE STYLE

- Documentação

- recomenda-se o padrão Doxygen, algumas tags estao relacionadas abaixo:

```
// @brief < descricao da atividade >
// @author < nome do criador >
// @date < data da criacao >
```

CODE STYLE

- Documentação

```
// @version < número da versão >
// @param[ in/out/intout ]
    < descrição do parâmetro >
// @see < referência documentação
>
// @warnings < avisos >
```

CODE STYLE

- Documentação

```
// @remarks < comentários que pode ter mais de uma linha >
// @return < descrição do tipo de retorno >
```

CODE STYLE

- Regras para Desenvolvimento

Ken Thompson
Unix philosophy

CODE STYLE

- Regras para Desenvolvimento

- Regra da Modularidade: Escreva códigos simples conectadas por interfaces limpas.

CODE STYLE

- Regras para Desenvolvimento

- Regra de clareza: clareza é melhor do que inteligência.

- Regra de composição: Projete programas para serem conectados a outros programas.

CODE STYLE

- Regras para Desenvolvimento

- Regra de separação: Separe a política do mecanismo; interfaces separadas dos motores.

- Regra de Simplicidade: Design para simplicidade; adicione complexidade apenas onde você deve.

CODE STYLE

- Regras para Desenvolvimento

- Regra de parcimônia: Escreva um programa grande apenas para demonstração, ele não servirá para mais nada.

CODE STYLE

- Regras para Desenvolvimento

- Regra de Transparência: Projete visibilidade para tornar a inspeção e depuração mais fáceis.

CODE STYLE

- Regras para Desenvolvimento

- Regra de robustez: Robustez é filha da transparência e da simplicidade.
Robustez significa suportar os testes sem resultados estranhos e sem encerrar o programa com erros indefinidos.

CODE STYLE

- Regras para Desenvolvimento

- Regra da menor surpresa: No design de interface, sempre faça a coisa menos surpreendente.

CODE STYLE

- Regras para Desenvolvimento

- Regra do Silêncio: Quando um programa não tem nada a informar, ele não deve informar nada.

- Regra de reparo: quando você tiver que falhar, falhe ruidosamente e o mais rápido possível.

CODE STYLE

- Regras para Desenvolvimento

- Regra de economia: o tempo do programador é caro; mantenha-o em preferência ao tempo da máquina.

CODE STYLE

- Regras para Desenvolvimento

- Regra de geração: evite trabalhos manualmente; escreva programas para escrever programas quando puder; automacao é um exemplo

CODE STYLE

- Regras para Desenvolvimento

- Regra de otimização: faça funcionar corretamente antes de otimizá-lo.

- Regra de extensibilidade: projete para o futuro, porque ele estará aqui mais cedo do que você pensa.

CODE STYLE

Por favor, não esqueça de dar um like

inscreva-se no canal e clique no sino

para receber notificações.

Obrigado.