

Padrão de Codificação e Estilo para a Linguagem C

Introdução

A principal razão para a criação desse guia de codificação e construção de código em Linguagem C na FATEC Santo André é criar uma "linguagem" comum para as diferentes disciplinas de programação em linguagem C.

Além disso, o seu uso irá ajudar e reduzir o número de bugs presentes no software. Lembrando que, o uso de um padrão de codificação não representa uma regra de ouro que eliminará todos os problemas encontrados no desenvolvimento do software, mas permitirá que a construção do software siga um caminho padronizado.

Princípios na Construção

Este guia de codificação foi desenvolvido para eliminar as distrações durante o processo de criação de código, também conhecidos estilos pessoais de codificação:

- O código deve ser construído de forma estruturada;
- É mais fácil e barato (tempo) prevenir a entrada de um bug no código;
- Este padrão prioriza a codificação robusta e portátil ainda que represente perda de eficiência ou conveniência do programador;
- Existem diversas fontes de bugs nos programas, alguns são criados pelos programadores e outros nascem do não entendimento correto de pessoas que irão realizar a manutenção ou modificar o programa;
- O número e a severidades dos bugs introduzidos pelo programador pode ser reduzido drasticamente através do uso disciplinado de práticas de codificação.

Regras para a construção

Regra#1: Estrutura dos programas

A estruturação dos programas deverá seguir obrigatoriamente conforme Figura 1, para arquivos com extensão c, e conforme Figura 2 para arquivos com extensão h. forma abaixo:

```

/*****
 * Nome do Arquivo : <nome do arquivo>
 *
 * Descrição      : <Descrição da estrutura interna do arquivo>
 *
 * Ambiente       : <Descrição da ferramenta e versão, processador, etc>
 *
 * Responsável    : <responsável pelo arquivo, utilize o formato: sobrenome,
 *                  nome>
 *
 * Versão/Data    : <vxx.yy - dia/mes/ano - modificação>
 *
 *****/

/*****
 * HEADER-FILES (Somente os arquivos necessários nesse arquivo)
 *****/
#include "config.h"

/*****
 * Variáveis Globais
 *****/
unsigned int calcular_res;

/*****
 * Protótipos das funções
 *****/
void soma (unsigned char prim_dado, unsigned char seg_dado);

/*****
 * Funcao:      void main(void)
 * Entrada:     Nenhuma (void)
 * Saída:       Nenhuma (void)
 * Descrição:   Função principal
 *****/
void main(void)
{
    unsigned char prim_valor, seg_valor;
    prim_valor = 44;
    seg_valor  = 43;

    soma(prim_valor, seg_valor);
}

/*****
 * Funcao:      void soma (unsigned char prim_dado, unsigned char seg_dado)
 * Entrada:     unsigned char prim_dado - recebe valores de 0 a 65536
 *              unsigned char seg_dado  - recebe valores de 0 a 65536
 * Saída:       Nenhuma (void)
 * Descrição:   Realiza o calculo da soma dos parâmetros recebidos.
 *****/
void soma (unsigned char prim_dado, unsigned char seg_dado)
{
    calcular_res = prim_dado + seg_dado;
}

/*Final do Arquivo modelo.c *****/

```

Figura 1 - Estrutura para arquivos com extensão c

```

/*****
 * Nome do Arquivo : <nome do arquivo>
 *
 * Descrição      : <Descrição da estrutura interna do arquivo>
 *
 * Ambiente       : <Descrição da ferramenta e versão, processador, etc>
 *
 * Responsável    : <responsável pelo arquivo, utilize o formato: sobrenome,
 *                  nome>
 *
 * Versão/Data    : <vxx.yy - dia/mes/ano - modificação>
 *
 *****/
#ifndef __MODELO_H
#define __MODELO_H

/*****
 * HEADER-FILES (Somente os arquivos necessários nesse arquivo)
 *****/

/*****

/*****
 * Variáveis Globais
 *****/

/*****

/*****
 * Protótipos das funções
 *****/

/*****

#endif

```

Figura 2- Estrutura para arquivos com extensão h

Os arquivos modelo com extensão c/h, serão disponibilizados pelos professores das disciplinas de Linguagem e Técnicas de Programação, Microcontroladores, Carga e Partida, Sistemas de Conforto e Tópicos Avançados de Programação de Microcontroladores.

Prof. Wesley M. Torres

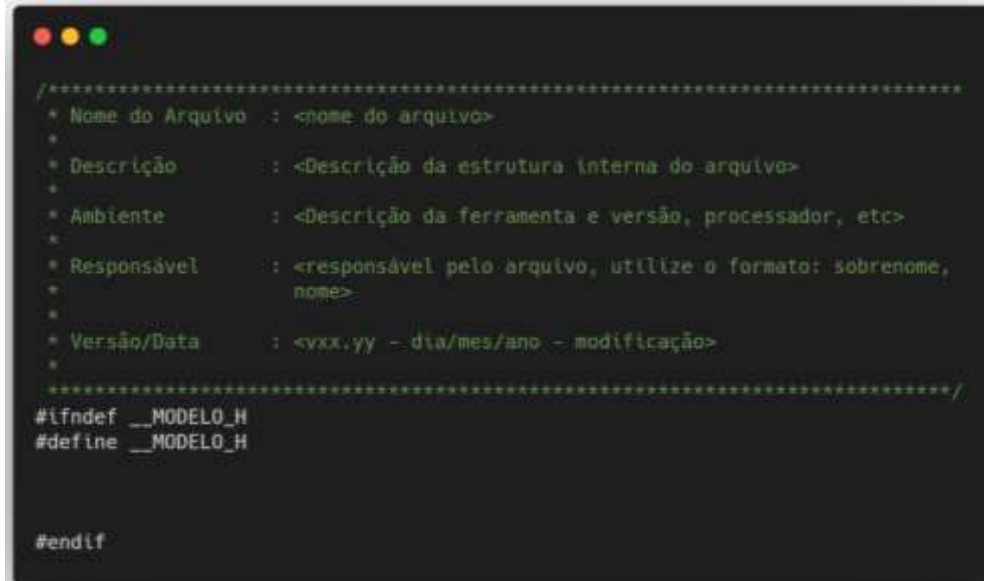
Data: 17/08/2020

Regra#2: Nomes de Variáveis

- a) Nenhuma variável deve possuir palavras reservadas da linguagem C (interrupt, true, false)
- b) O nome da variável não deve iniciar com um underscore (_);
- c) Seu nome não pode ser maior que 31 caracteres;
- d) Não pode conter letra maiúsculas em seu nome;
- e) Underscores devem ser utilizados para separar palavras no nome da variável;
- f) O nome da variável deve descrever claramente seu propósito;
- g) Variáveis globais devem começar com a letra 'g'. Por exemplo, g_calcula_soma;
- h) O nome de variáveis ponteiro deve começar com a letra 'p'. Por exemplo, p_recebe_dados;
- i) Variáveis que armazenam valores booleanos, devem começar com a letra 'b'. Por exemplo, b_retorno;
- j) Todas as variáveis devem ser inicializadas antes de serem utilizadas

Regra#3: Módulos

- a) O nome dos módulos deve ser em letra minúsculas e quando necessário, empregar underscores '_' para separar as palavras;
- b) Sempre deve existir um arquivo header para cada arquivo .c, e com o mesmo nome. Por exemplo, modelo.c, portanto, deve existir o arquivo modelo.h;
- c) Todos os arquivos header devem possuir uma proteção com o pré-processador, para evitar múltiplas inclusões do arquivo header. Exemplo:



```
/*=====
 * Nome do Arquivo : <nome do arquivo>
 *
 * Descrição      : <Descrição da estrutura interna do arquivo>
 *
 * Ambiente       : <Descrição da ferramenta e versão, processador, etc>
 *
 * Responsável    : <responsável pelo arquivo, utilize o formato: sobrenome,
 *                  nome>
 *
 * Versão/Data    : <vxx.yy - dia/mes/ano - modificação>
 *=====*/
#ifndef __MODELO_H
#define __MODELO_H

#endif
```

Figura 3 - Proteção de pré-processador.

- d) Dentro do arquivo header, somente devem estar funções, constantes, tipos definidos pelo desenvolvedor, que serão acessadas/ externalizadas para outros módulos;
- e) O arquivo .C deverá possuir somente a implementação de uma funcionalidade. Por exemplo, display_lcd.c, somente irá realizar o controle do display de LCD;

Regra#4: Funções

- a) O nome de nenhuma função deve possuir palavras reservadas da linguagem C (inline, true, false)
- b) O nome não deve iniciar com um underscore (_);
- c) Seu nome não pode ser maior que 31 caracteres;
- d) Não pode conter letra maiúsculas em seu nome;
- e) Nenhuma macro pode ter o nome em letras minúsculas;
- f) Underscores devem ser utilizados para separar palavras no nome da função;
- g) O nome da função deve descrever claramente seu propósito;
- h) O tamanho máximo em número de linhas de uma função não deve ser superior a 30 linhas. Isso ajuda na legibilidade do código e na sua estruturação;
- i) Todas as funções utilizadas somente pelo módulo, devem ser estáticas;
- j) Funções que são acessadas por outros módulos, devem ter seu protótipo declarado no arquivo H;
- k) As funções devem possuir somente um ponto de saída;
- l) Seus parâmetros de entrada devem possuir nomes significativos.

Regra#5: Interrupções

- a) As rotinas de interrupção (Interrupt Service Routines) não são funções normais, portanto, devem ser implementadas cuidadosamente, levando em consideração o seu tempo de processamento;
- b) Todas as funções de interrupção devem possuir o nome terminado com "_isr";
- c) Função de interrupção não deve possuir protótipo, dessa forma, não será possível chamá-la;

Regra#6: Expressões e Comandos

- a) Para todo comando if deverá ser adicionado abertura e fechamento de chaves. Ex.:

```
unsigned char dado = 0;

if(dado == 1)
{
}

```

Figura 4 - Comando if.

- b) Todos os comandos switch devem possuir um bloco default. Ex.:

```
switch(estado)
{
    default:
        .....
        break;

    case 1:
        .....
        break;
}

```

Figura 5- Comando switch

- c) O break de cada case deve estar alinhado;
d) Números mágicos (Figura 6) não devem estar presentes no código, substitua por #define ou constantes.



```
for(unsigned char contador = 0; contador < 10; contador++)  
{  
    .....  
}
```

Figura 6 - Uso de números mágicos.



```
#define CONTAGEM_MAX 10  
  
for(unsigned char contador = 0; contador < CONTAGEM_MAX; contador++)  
{  
    .....  
}
```

Figura 7 - Remoção dos números mágicos.

Referências

Barr, M., Embedded C Coding Standard, Barr Group

Misra-C: 2004 Guidelines for the use of C language in critical systems, MisraLimited, 2004