Guia de Boas Práticas de Programação em C

Este guia visa fornecer regras e recomendações para melhorar a qualidade, legibilidade e manutenção do código em C. As práticas são baseadas em documentos reconhecidos e adaptadas à nomenclatura "camelCase", outros padrões de nomenclatura podem ser utilizados.

Estrutura e Formatação

Indentação

Use 4 espaços por nível de indentação. Isso melhora a consistência e facilita a leitura em diferentes editores. Evite usar caracteres TAB.

Exemplo:

```
void exampleFunction()
{
    int variable = 0;
    if (variable > 0)
    {
        variable++;
    }
}
```

Largura de Linha

Limite as linhas a 80 caracteres. Alguns projetos permitem até 120 caracteres, mas 80 é um bom padrão.

Chaves

Sempre use chaves, mesmo para blocos com apenas uma linha.

Exemplo ruim:

```
if (condition)
    executeFunction();
```

Exemplo bom:

```
if (condition)
{
```

```
    executeFunction();
}
```

Espaçamento

Use espaços antes e depois de operadores binários e de atribuição.

Exemplo ruim:

```
int sum=a+b;
```

Exemplo bom:

```
int sum = a + b;
```

Nomeação

Variáveis Globais

Prefixe variáveis globais com `g_` para identificá-las rapidamente. Outros prefixos como `s_` para estáticas também são comuns.

Exemplo:

```
int g_totalItems;
```

Funções

Use nomes descritivos que reflitam a ação executada.

Exemplo ruim:

```
void doStuff();
```

Exemplo bom:

```
void calculateAverage();
```

Ponteiros

Prefixe nomes de ponteiros com `p_` , coloque o `*` próximo ao nome do ponteiro.

Exemplo:

```
int *p_value;
```

Constantes e Macros

Use letras maiúsculas com separadores `_`.

Exemplo:

```
#define MAX_BUFFER_SIZE 1024

const int MAX_ATTEMPTS = 5;
```

Nomenclatura de estruturas, enumerações e uniões

Structs

- Nomenclatura: use nomes descritivos que reflitam o propósito da estrutura. Os nomes normalmente utilizam letras minúsculas.
- Declaração:

```
struct book
{
    char title[100];
    char author[100];
    int yearPublished;
};
```

- Declaração de variáveis:

```
struct book book1;
```

- Usando `typedef` (opcional), normalmente a primeira letra é maiúscula, como em `Book`, no exemplo:

```
typedef struct
{
    char title[100];
    char author[100];
    int yearPublished;
} Book;
Book book1;
```

Unions

- Nomenclatura: use nomes descritivos que reflitam o propósito da união. Os nomes normalmente utilizam letras minúsculas.
- Declaração:

```
union data
{
    int i;
    float f;
    char c;
};
```

- Declaração de variáveis:

```
union data dataVar;
```

Enums

- Nomenclatura: use nomes descritivos que reflitam o propósito da enumeração. Os nomes normalmente utilizam letras minúsculas. As constantes da enumeração por sua vez, devem usar letras maiúsculas nos nomes.
- Declaração:

```
enum day
{
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
};
```

- Declaração de variáveis:

```
enum day today = MONDAY;
```

Estruturas e Declarações

Declaração de Variáveis

Declare variáveis o mais próximo possível de onde serão utilizadas. Os nomes devem começar com letras minúsculas.
Sempre inicialize variáveis ao declarar.

Exemplo ruim:

```
int count;
...
count = 0;
```

Exemplo bom:

```
int count = 0;
```

Minimizar Variáveis Globais

Prefira variáveis locais ou parâmetros de função.

Exemplo ruim:

```
int g_count;
...
void function()
{
    // uso da variável count
}
```

Exemplo bom:

```
void function(int count)
{
    // uso da variável count
}
```

Controle de Fluxo

Instruções Condicionais

Use parênteses para agrupar expressões em `if`, `while` e `for`.

Exemplo:

```
if ((a > b) && (c != 0))
{
    printf("Condição atendida.");
}
```

Documentação e Comentários

Comentários Sucintos e Diretos

Comente o "porquê" e não o "como" quando o código já for autoexplicativo.

Exemplo:

```c
// Verifica se o arquivo foi aberto com sucesso
if (file == NULL)
{
    fprintf(stderr, "Erro: Não foi possível abrir o arquivo.");
}
```

Estrutura de Documentação

Utilize Doxygen para documentar funções.

Exemplo:

```c
/**
 * @brief Calcula a média de um array de inteiros.
 * @param valores Array de inteiros.
 * @param tamanho Número de elementos no array.
 * @return Média dos valores como float.
 */
float calculaMedia(const int *valores, size_t tamanho);
```

Abreviações Padronizadas

Não é recomendável abrevias nomes na programação, mas alguns termos são comuns na programação e costumam ser abreviados. A seguir estão os termos mais utilizados.

Termo	Abreviação
Minimum	min
Manager	mgr
Maximum	max
Mailbox	mbox
Interrupt Service Routine	isr
Initialize	init
Input/output	io
Handle	h_
Error	err
Current	curr
Configuration	cfg

Termo	Abreviação
Buffer	buf
Average	avg
Millisecond	msec
Message	msg
Nanosecond	nsec
Number	num
Transmit	tx
Receive	rx
Temperature	temp
Temporary	tmp
Synchronize	sync
String	str
Register	reg
Previous	prev
Priority	prio

Tratamento de Erros

Verificação de Retorno de Funções

Sempre cheque o valor de retorno de funções que podem falhar.

Exemplo:

```
FILE *file = fopen("data.txt", "r");
if (file == NULL)
{
    fprintf(stderr, "Erro ao abrir o arquivo.");
    return -1;
}
```

Mensagens de Erro Claras

Forneça mensagens úteis ao detectar erros.

Exemplo:

```
fprintf(stderr, "Erro: Conexão ao socket falhou.");
```

Estruturas de Código Modular

Modularidade

Separe o código em arquivos de cabeçalho (.h) e implementação (.c).

Exemplo:

- **header.h**

```
#ifndef HEADER_H
#define HEADER_H

void function(int param);

#endif
```

- **implementation.c**

```
 #include "header.h"

void function(int param)
{
    // implementação
}
```

Headers Guard

Utilize guardas para evitar inclusões múltiplas.

Exemplo:

```
#ifndef HEADER_FILE_H
#define HEADER_FILE_H

// conteúdo do cabeçalho

#endif
```

Outros Conceitos Importantes

Uso de `const`

Qualifique parâmetros e variáveis que não devem ser modificadas.

Exemplo:

```c
void printValue(const int value)
{
    printf("%d\n", value);
}
```

Evitar Números Mágicos

Defina constantes simbólicas para valores literais.

Exemplo ruim:

```c
for (int i = 0; i < 12; i++)
{
    // código
}
```

Exemplo bom:

```c
#define MONTHS 12
for (int i = 0; i < MONTHS; i++)
{
    // código
}
```

Uso de `static`

Use `static` para funções e variáveis com visibilidade limitada. Funções e variáveis globais declaradas como `static` são visíveis apenas no arquivo atual (com algumas exceções).

Exemplo:

```c
static void internalFunction()
{
    // função interna
}
```

Variáveis locais utilizadas em funções mantem seu valor entre as chamadas da função.

Exemplo:

```c
void function()
{
    static int counter = 0;
    count++;
}
```

Portabilidade

Quando possível evite dependências específicas de plataforma.

Verificação de Erros de Alocação de Memória

Sempre verifique se a alocação de memória foi bem-sucedida.

Exemplo:

```c
int *array = malloc(sizeof(int) * size);
if (array == NULL)
{
    fprintf(stderr, "Falha na alocação de memória.\n");
    return -1;
}
```

Uso de volatile

Use `volatile` para variáveis cujo valor pode ser alterado de forma imprevisível e fora do fluxo normal do programa. Isso é crucial para variáveis acessadas por rotinas de interrupção ou dispositivos de hardware, pois garante que o compilador não otimize as leituras e escritas dessa variável, garantindo sempre a leitura do valor mais atual.

Exemplo:

```c
volatile int flag = 0;
```