

Boas Práticas de Programação – Parte 1

Prof. Carlos Rodrigues

2023/02

Sumário

- Cuidar da nomenclatura
- Cuidar da indentação e deixar blocos de código claros
- Criar funções pequenas e com uma responsabilidade

Nomenclatura

- Escolham nomes que sejam claros e expressem o propósito da função, variável, estrutura, etc.
- Evite nomes genéricos
- Evite nomes ambíguos

Nomenclatura

- Exemplo de código sem padrão

```
function executar(param1, param2) {  
    const resposta = passo1(param1);  
  
    if (resposta === false) {  
        return passo2(param1, param2)[mensagem];  
    } else{  
        return 'Não foi possível cadastrar usuário';  
    }  
}
```

Nomenclatura

- Exemplo de código com padrão

```
function cadastrarUsuario(email, senha) {  
  const existe = existeEmailCadastrado(email);  
  
  if (existe === true) {  
    return 'Este email já tem cadastro no sistema';  
  } else {  
    const usuarioCriado = salvar(email, senha);  
  
    return usuarioCriado.mensagem  
  }  
}
```

Nomenclatura

- Evite nomes genéricos

```
int a = 18;
```

- Ao invés, descreva os nomes com mais clareza

```
int idadeMinima = 18;
```

Nomenclatura

- Evite nomes muito longos

```
int idadeMinimaParaIniciarProcessoDeCarteiraDeHabilitacao = 18;
```

- É possível resumir nomes dependendo do projeto, mas traga a maior clareza possível com o nome

```
int idadeMinimaHabilitacao = 18;
```

Nomenclatura

- Lembrem-se que códigos em linguagem de programação não são linguagem de máquina
- No mercado, é comum projetos terem um ciclo de vida de alguns anos e possuírem centenas de arquivos de código fonte
- Idealmente, um código feito há meses, ou até anos, deve continuar de fácil entendimento e manutenção

Nomenclatura

- Também é importante padronizar os nomes ao longo de um projeto
- Cada equipe vai ter seus próprios padrões, e isso é normal
- Adequar o código aos padrões aumentam a produtividade

Nomenclatura

- Evite código não padronizado

```
void novoAluno(char[] nome, char[] matricula);  
void criarDisciplina(char[] nome, int periodo);  
void gerarTurma(int periodo);
```

- Prefira manter a nomenclatura consistente

```
void novoAluno(char[] nome, char[] matricula);  
void novaDisciplina(char[] nome, int periodo);  
void novaTurma(int periodo);
```

Nomenclatura

- Utilize convenções de formatações de nomes

- camelCase

```
getNome(), criarTurma(), setNomeUsuario(char[] novoNome)
```

- PascalCase

```
TipoPessoa, DatabaseError
```

- under_score ou snake_case

```
nome_completo, data_nascimento
```

Indentação e blocos

- Evite código sem indentação

```
int main(){  
    bool eu_indento = false;  
    if(eu_indento){  
        printf("Parabéns pela boa prática");  
    }  
    else{  
        printf("Que pena!!");  
    }  
    return 0;  
}
```

Indentação e blocos

- Prefira

```
int main(){
    bool eu_indento = false;
    if(eu_indento){
        printf("Parabéns pela boa prática");
    }
    else{
        printf("Que pena!!");
    }
    return 0;
}
```

Indentação e blocos

- Evite blocos sem chaves

```
if (idade < idadeMinima) flag = 1;  
else {  
    flag = 0;  
    i++;  
}
```

- Prefira colocar explicitamente as chaves

```
if (idade < idadeMinima) {  
    flag = 1;  
} else {  
    flag = 0;  
    i++;  
}
```

Separação em funções

- Funções muito grandes normalmente possuem várias responsabilidades e são difíceis de se entender
- Idealmente, cada função deve ter apenas uma responsabilidade
- Evite muitos if-else ou loops aninhados
- As funções não devem possuir responsabilidades ocultas

Separação em funções

- Evite funções grandes ou com múltiplas responsabilidades


```
def buy_concert_ticket(user, ticket):  
    if not user.age >= 18:  
        print("You are not allowed to buy a ticket")  
        return  
  
    if not user.money >= ticket.price:  
        print("Sorry you don't have enough balance")  
        return  
  
    for seat in seats:  
        if seat.is_available():  
            seat.owner = user  
            user.money -= ticket.price  
            print("Congratulations, you have a seat")  
            return  
  
    print("There is no available seat")  
    return
```

Separação em funções

- Prefira funções menores

```
def buy_concert_ticket(user, ticket):  
    if not user_can_buy_a_ticket(user, ticket):  
        return  
  
    buy_available_seat(user)  
  
    return
```

```
def user_can_buy_a_ticket(user, ticket):  
    if not user_has_legal_age(user, ticket):  
        print("You are not allowed to buy a ticket")  
        return False  
  
    if not user_has_enough_balance(user, ticket):  
        print("Sorry you don't have enough balance")  
        return False  
  
    return True  
  
def user_has_legal_age(user):  
    if not user.age >= 18:  
        return False  
  
    return True  
  
def user_has_enough_balance(user, ticket):  
    if user.money >= ticket.price:  
        return True  
  
    return False
```

```
def buy_available_seat(user):
    available_seat = get_available_seat()

    if not available_seat:
        print("There is not available seat")

    buy_seat(user, available_seat)
    return

def get_available_seat():
    seats = stadium.seats
    for seat in seats:
        if seat.is_available():
            return seat



def buy_seat(user, seat):
    seat.owner = user
    user.money -= ticket.price
    print("Congratulations, you have a seat")
    return
```

Separação em funções

- Normalmente, separar em mais funções vai tornar o código total maior
- É melhor ter 10 funções de 10 linhas do que uma única função de 100 linhas

Separação em funções

- Evite “código hadouken”



```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_rep']
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user
                                $user = read_user($_POST['user_name']));
                                if (isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65
                                            if (filter_var($_POST['user_email
                                                create_user();
                                                $_SESSION['msg'] = 'You are n
                                                header('Location: ' . $_SERVE
                                                exit();
                                            } else $msg = 'You must provide a
                                        } else $msg = 'Email must be less tha
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-
                        } else $msg = 'Username must be between 2 and 64 char
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

Separação em funções

- Evite muitos if-elses aninhados

```
function userIsAdmin(user) {  
    if (user.role == 'admin') {  
        if (user.manager == true) {  
            return true;  
        }  
        else {  
            return false;  
        }  
    }  
    else {  
        return false;  
    }  
}
```

Separação em funções

- Prefira escrever *guard clauses*

```
function userIsAdmin(user) {  
    if (user.role !== 'admin') {  
        return false;  
    }  
  
    if (user.manager !== true) {  
        return false;  
    }  
  
    return true;  
}
```


Separação em funções

- Evite responsabilidades ocultas

```
public boolean validarSenha(Usuario usuario, String novaSenha) {  
    if (!novaSenha.equals(usuario.getSenha())) {  
        usuario.incrementarTentativaIncorreta(); // Efeito colateral  
        return false;  
    }  
    return true;  
}
```

Aplicando na prática

- Código original:

```
void insere(struct item **p0, int x) {
    struct item *p, *pa = NULL, *corr = *p0;
    int cont = TRUE;
    p = (struct item *)malloc(sizeof(struct item));
    p->info = x;
    p->prox = NULL;
    while (corr != NULL && cont) {
        if (x < corr->info) cont = FALSE;
        else {
            pa = corr;
            corr = corr->prox;
        }
    }
    p->prox = corr;
    if (pa == NULL) *p0 = p;
    else pa->prox = p;
}
```