



# Documento de Padronização de Código

## Sistema Acadêmico Colaborativo com IA - PIM II

**Projeto:** Sistema de Gerenciamento Acadêmico

**Disciplinas:** ADS - UNIP

**Versão:** 1.0

**Data:** Outubro/2025

---



## Índice

1. Convenções de Nomenclatura
  2. Estrutura de Arquivos
  3. Formatação e Indentação
  4. Comentários e Documentação
  5. Boas Práticas de C
  6. Padrões de Python
  7. Gerenciamento de Dados
- 

## 1. Convenções de Nomenclatura

### 1.1 Arquivos

#### Arquivos de Cabeçalho (.h)

- **Padrão:** `nome_modulo.h`
- **Exemplos:**
  - `structs.h`
  - `file_manager.h`
  - `aluno_manager.h`
  - `turma_manager.h`
  - `aula_manager.h`

#### Arquivos de Implementação (.c)

- **Padrão:** `nome_modulo.c`

- **Exemplos:**

- `file_manager.c`
- `aluno_manager.c`
- `turma_manager.c`
- `aula_manager.c`
- `main_test.c`

**Regra:** Nome do arquivo `.c` deve ser idêntico ao arquivo `.h` correspondente.

---

## 1.2 Constantes e Macros

### Definições de Constantes

- **Padrão:** `UPPER_CASE_COM_UNDERSCORE`
- **Exemplos:**

```
c
#define MAX_NOME 100
#define MAX_TURMA_NOME 50
#define MAX_CONTEUDO 500
#define MAX_PATH 200
#define MAX_ALUNOS 1000
#define MAX_TURMAS 500
#define MAX_AULAS 5000
```

### Definições de Arquivos

- **Padrão:** `ARQUIVO_NOME_ENTIDADE`
- **Exemplos:**

```
c
#define ARQUIVO_ALUNOS "data/alunos.csv"
#define ARQUIVO_TURMAS "data/turmas.csv"
#define ARQUIVO_AULAS "data/aulas.csv"
#define ARQUIVO_ALUNO_TURMA "data/aluno_turma.csv"
```

### Guards de Inclusão

- **Padrão:** `NOME_ARQUIVO_H`
- **Exemplos:**

c

```
#ifndef STRUCTS_H
#define STRUCTS_H
// ... conteúdo ...
#endif

#ifndef FILE_MANAGER_H
#define FILE_MANAGER_H
// ... conteúdo ...
#endif
```

## 1.3 Estruturas (Structs)

### Nome da Estrutura

- **Padrão:** `PascalCase` (primeira letra maiúscula)
- **Exemplos:**

c

```
typedef struct {
    int ra;
    char nome[MAX_NOME];
    char email[MAX_NOME];
    int ativo;
} Aluno;

typedef struct {
    int id;
    char nome[MAX_TURMA_NOME];
    char professor[MAX_NOME];
    int ano;
    int semestre;
} Turma;

typedef struct {
    int id;
    int id_turma;
    char data[11];
    char conteudo[MAX_CONTEUDO];
} Aula;
```

### Campos da Estrutura

- **Padrão:** `snake_case` (minúsculas com underscore)

- **Exemplos:**

```
c
int ra;
char nome[MAX_NOME];
int id_turma;
char path_arquivo[MAX_PATH];
```

**Exceção:** Campos com uma palavra única são escritos em minúsculas sem underscore:

```
c
int id;
int ano;
int ativo;
```

---

## 1.4 Variáveis

### Variáveis Locais

- **Padrão:** `snake_case`

- **Exemplos:**

```
c
int total_alunos = 0;
int total_turmas = 0;
int num_registros = 10;
char nome_arquivo[100];
Aluno *aluno_encontrado;
```

### Variáveis Globais/Estáticas

- **Padrão:** `snake_case` com prefixo `static`

- **Exemplos:**

```
c
```

```
static Aluno alunos[MAX_ALUNOS];
static int total_alunos = 0;
static Turma turmas[MAX_TURMAS];
static int total_turmas = 0;
```

## Ponteiros

- **Padrão:** Asterisco junto ao tipo
- **Exemplos:**

```
c
Aluno *aluno;           // ✅ Correto
Turma *turma_atual;     // ✅ Correto
const char *nome_arquivo; // ✅ Correto
```

## Contadores e Índices

- **Padrão:** Letras simples para loops curtos, descritivo para loops complexos
- **Exemplos:**

```
c
for (int i = 0; i < total; i++)      // ✅ Loop simples
for (int j = 0; j < count; j++)      // ✅ Loop aninhado

for (int idx_aluno = 0; idx_aluno < total_alunos; idx_aluno++) // ✅ Loop complexo
```

## 1.5 Funções

### Nomenclatura de Funções

- **Padrão:** `verboSubstantivo` ou `verboSubstantivoComplemento` (camelCase)
- **Exemplos:**

### Funções CRUD:

```
c
```

```
int cadastrarAluno(Aluno *aluno);
Aluno* buscarAlunoPorRA(int ra);
int listarAlunos(Aluno *destino, int max);
int atualizarAluno(Aluno *aluno);
int excluirAluno(int ra);
```

### Funções de Persistência:

```
c

int salvarDados(const char *nome_arquivo, void *dados, int num_registros, int tipo);
int carregarDados(const char *nome_arquivo, void *destino, int max_registros, int tipo);
```

### Funções Auxiliares:

```
c

int gerarProximoIDTurma(void);
int gerarProximoIDAula(void);
int validarData(const char *data);
int verificarMatricula(int ra, int id_turma);
```

### Funções de Associação:

```
c

int associarAlunoTurma(int ra, int id_turma);
int removerAlunoTurma(int ra, int id_turma);
int listarAlunosDaTurma(int id_turma, int *ras_destino, int max);
int listarTurmasDoAluno(int ra, int *ids_destino, int max);
```

### Funções de Relatório:

```
c

int gerarRelatorioTurma(int id_turma, const char *arquivo_destino);
int contarAulasDaTurma(int id_turma);
```

### Funções Privadas (Static)

- **Padrão:** Mesmo padrão, mas com `static` no início
- **Exemplos:**

```
c
```

```
static void carregarAlunosMemoria(void);
static void salvarAlunosArquivo(void);
static void carregarTurmasMemoria(void);
```

## 1.6 Enumerações

- **Padrão:** `PascalCase` para o tipo, `UPPER_CASE` para valores
- **Exemplos:**

```
c
enum TipoDado {
    TIPO_ALUNO = 1,
    TIPO_TURMA = 2,
    TIPO_AULA = 3,
    TIPO_ALUNO_TURMA = 4,
    TIPO_ATIVIDADE = 5
};
```

## 2. Estrutura de Arquivos

### 2.1 Organização de Diretórios

```
projeto_pim/
├── c_modules/      # Módulos em C
│   ├── structs.h
│   ├── file_manager.h
│   ├── file_manager.c
│   ├── aluno_manager.h
│   ├── aluno_manager.c
│   ├── turma_manager.h
│   ├── turma_manager.c
│   ├── aula_manager.h
│   ├── aula_manager.c
│   └── main_test.c
├── data/           # Arquivos de dados
│   ├── alunos.csv
│   ├── turmas.csv
│   ├── aulas.csv
│   └── aluno_turma.csv
├── python_app/     # Aplicação Python (UI)
│   └── main.py
```

```
| | └─ client_socket.py
| | └─ ...
| └─ Makefile          # Automação de compilação
└─ README.md          # Documentação do projeto
```

## 2.2 Estrutura de um Arquivo .h (Cabeçalho)

Template padrão:

```
c

#ifndef NOME_MODULO_H
#define NOME_MODULO_H

// Inclusões necessárias
#include "structs.h"

// Definições de constantes
#define MAX_ENTIDADES 1000
#define ARQUIVO_ENTIDADES "data/entidades.csv"

// ===== DECLARAÇÕES DE FUNÇÕES =====

// Descrição da função
// Retorna: tipo de retorno
tipo nomeFuncao(parametros);

// Mais funções...

#endif
```

Exemplo real:

```
c
```



```
#ifndef ALUNO_MANAGER_H
#define ALUNO_MANAGER_H

#include "structs.h"

#define MAX_ALUNOS 1000
#define ARQUIVO_ALUNOS "data/alunos.csv"

// ===== FUNÇÕES DE GERENCIAMENTO DE ALUNOS =====

// Função para cadastrar um novo aluno
// Retorna: 1 se sucesso, 0 se erro
int cadastrarAluno(Aluno *aluno);

// Função para buscar aluno por RA
// Retorna: ponteiro para o aluno ou NULL se não encontrado
Aluno* buscarAlunoPorRA(int ra);

#endif
```

## 2.3 Estrutura de um Arquivo .c (Implementação)

Template padrão:

c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "nome_modulo.h"
#include "file_manager.h"

// ===== VARIÁVEIS GLOBAIS/ESTÁTICAS =====
static Entidade entidades[MAX_ENTIDADES];
static int total_entidades = 0;

// ===== FUNÇÕES AUXILIARES PRIVADAS =====
static void funcaoPrivada(void) {
    // Implementação
}

// ===== IMPLEMENTAÇÃO DAS FUNÇÕES PÚBLICAS =====

int funcaoPublica(parametros) {
    // Implementação
    return resultado;
}
```

---

## 3. Formatação e Indentação

### 3.1 Indentação

- **Padrão:** 4 espaços (não tabs)
- **Editor:** Configurar VS Code para usar espaços

c

//  Correto


```
int cadastrarAluno(Aluno *aluno) {  
    if (aluno == NULL) {  
        return 0;  
    }  
  
    for (int i = 0; i < total; i++) {  
        if (alunos[i].ra == aluno->ra) {  
            return 0;  
        }  
    }  
  
    return 1;  
}
```

## 3.2 Chaves { }

- **Padrão:** Chave de abertura na mesma linha
- **Exemplos:**

c

//  Correto

```
if (condicao) {  
    // código  
}  
  
for (int i = 0; i < n; i++) {  
    // código  
}  
  
while (condicao) {  
    // código  
}  
  
//  Correto - Função  
int nomeFuncao(parametros) {  
    // código  
}
```

## 3.3 Espaçamento

### Operadores

c

//  Correto

```
int soma = a + b;  
if (x == 10) {  
int resultado = funcao(a, b, c);  
for (int i = 0; i < n; i++) {
```

//  Incorreto

```
int soma=a+b;  
if(x==10){  
int resultado=funcao(a,b,c);  
for(int i=0;i<n;i++){
```

## Vírgulas

c

//  Correto

```
funcao(param1, param2, param3);  
int array[] = {1, 2, 3, 4};
```

//  Incorreto

```
funcao(param1,param2,param3);  
int array[] = {1,2,3,4};
```

---

## 3.4 Linhas em Branco

c

//  Correto - Separar blocos lógicos

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define MAX 100
```

```
static int total = 0;
```

```
int funcao1() {
```

```
    // código
```

```
}
```

```
int funcao2() {
```

```
    // código
```

```
}
```

## 4. Comentários e Documentação

### 4.1 Comentários de Cabeçalho de Arquivo

c

```
/*
```

```
 * Arquivo: aluno_manager.c
```

```
 * Descrição: Gerenciamento de alunos (CRUD)
```

```
 * Autor: [Nome do Desenvolvedor]
```

```
 * Data: Outubro/2025
```

```
 * Projeto: PIM II - Sistema Acadêmico
```

```
 */
```

### 4.2 Comentários de Função (no .h)

c

```
// Função para cadastrar um novo aluno no sistema
```

```
// Parâmetros:
```

```
// - aluno: ponteiro para estrutura Aluno com dados preenchidos
```

```
// Retorna:
```

```
// - 1 se cadastro foi realizado com sucesso
```

```
// - 0 se houve erro (aluno NULL ou RA duplicado)
```

```
int cadastrarAluno(Aluno *aluno);
```

## 4.3 Comentários de Seção

```
c

// ===== FUNÇÕES DE GERENCIAMENTO DE ALUNOS =====


// ===== VARIÁVEIS GLOBAIS =====


// ===== FUNÇÕES AUXILIARES PRIVADAS =====

// ===== IMPLEMENTAÇÃO DAS FUNÇÕES PÚBLICAS =====
```

## 4.4 Comentários Inline

```
c

//  Correto - Comentário explicativo
for (int i = 0; i < total_alunos; i++) {
    // Verifica se o RA já existe no sistema
    if (alunos[i].ra == aluno->ra) {
        printf("Erro: RA %d já cadastrado.\n", aluno->ra);
        return 0;
    }
}

//  Correto - Marcar requisitos do PIM
// Estrutura de decisão (requisito obrigatório)
if (dados == NULL || num_registros <= 0) {
    return 0;
}

// Estrutura de repetição (requisito obrigatório)
for (int i = 0; i < num_registros; i++) {
    // código
}
```

## 4.5 Comentários de TODO/FIXME

```
c


// TODO: Implementar validação de CPF
// FIXME: Corrigir problema de buffer overflow
// NOTE: Esta função será refatorada na próxima sprint
```

---

## 5. Boas Práticas de C

### 5.1 Uso de const


c

```
//  Correto - Parâmetros que não serão modificados
int salvarDados(const char *nome_arquivo, void *dados, int num_registros);
int validarData(const char *data);
```

---

### 5.2 Verificação de Ponteiros

c

```
//  Sempre verificar ponteiros antes de usar
int cadastrarAluno(Aluno *aluno) {
    if (aluno == NULL) {
        printf("Erro: aluno inválido.\n");
        return 0;
    }
    // ... resto do código
}
```


---

### 5.3 Retorno de Funções

#### Padrão de retorno:

- `1` = Sucesso
- `0` = Erro
- `NULL` = Não encontrado (para ponteiros)
- Número positivo = Contagem/ID

c

```
//  Exemplos corretos

int cadastrarAluno(Aluno *aluno) {
    // ...
    return 1; // Sucesso
}


Aluno* buscarAlunoPorRA(int ra) {
    // ...
    return NULL; // Não encontrado
}


int listarAlunos(Aluno *destino, int max) {
    // ...
    return count; // Número de alunos listados
}
```

---

## 5.4 Uso de Arrays

```
c

//  Sempre verificar limites
if (total_alunos >= MAX_ALUNOS) {
    printf("Erro: limite de alunos atingido.\n");
    return 0;
}


//  Passar tamanho junto com array
int listarAlunos(Aluno *destino, int max) {
    // Garante que não excede o tamanho
    int count = (total_alunos < max) ? total_alunos : max;
    return count;
}
```


---

## 5.5 Strings

```
c
```



```
//  Usar funções seguras
strcpy(destino, origem);      // Simples
strncpy(destino, origem, MAX-1); // Mais seguro
destino[MAX-1] = '\0';      // Garantir terminação

//  Comparação
if (strcmp(str1, str2) == 0) {
    // strings iguais
}
```

## 6. Padrões de Python

### 6.1 Nomenclatura (a ser implementado)

```
python

# Variáveis e funções: snake_case
nome_completo = "João Silva"
total_alunos = 0

def cadastrar_aluno(dados):
    pass

# Classes: PascalCase
class AlunoManager:
    pass

class TurmaController:
    pass

# Constantes: UPPER_CASE
MAX_ALUNOS = 1000
ARQUIVO_CONFIG = "config.json"
```

### 6.2 Imports

```
python
```

```
#  Ordem de imports
```

```
# 1. Bibliotecas padrão
```

```
import os
```

```
import sys
```

```
from datetime import datetime
```

```
# 2. Bibliotecas de terceiros
```

```
import tkinter as tk
```

```
from tkinter import ttk
```

```
import pandas as pd
```

```
# 3. Módulos locais
```

```
from client_socket import ClientSocket
```

```
from utils import validar_ra
```

---

## 7. Gerenciamento de Dados

### 7.1 Formato CSV

**Padrão:** Primeira linha é o cabeçalho

csv

```
RA,Nome,Email,Ativo
```

```
12345,João Silva Santos,joao.silva@unip.br,1
```

```
12346,Maria Oliveira Costa,maria.oliveira@unip.br,1
```

csv

```
ID,Nome,Professor,Ano,Semestre
```



```
1,ADS-2A,Prof. Carlos Silva,2025,1
```

```
2,ADS-2B,Prof. Ana Paula,2025,1
```


---

### 7.2 Convenções de Dados

#### Campos Booleanos

-  = Verdadeiro/Ativo
-  = Falso/Inativo

#### Datas

- **Formato:** 

- Exemplo: 18/10/2025

## IDs

- Sempre inteiros positivos
  - Sequenciais (1, 2, 3...)
- 

## 8. Compilação e Build

### 8.1 Flags do GCC

makefile

```
CFLAGS = -Wall -Wextra -g -std=c99
```

*# -Wall: Ativa warnings importantes*

*# -Wextra: Ativa warnings extras*

*# -g: Inclui símbolos de debug*

*# -std=c99: Usa padrão C99*

---

### 8.2 Nomenclatura de Targets

makefile

```
all    # Compila tudo
```

```
clean  # Remove objetos e executável
```

```
run    # Compila e executa
```

```
rebuild # Limpa e recompila
```

---

## 9. Controle de Versão (Git)

### 9.1 Commits

Formato: tipo: descrição curta

bash

#  Exemplos corretos

```
git commit -m "feat: adicionar função cadastrarAluno"
```

```
git commit -m "fix: corrigir segmentation fault em listarTurmas"
```

```
git commit -m "docs: atualizar README com instruções"
```

```
git commit -m "refactor: melhorar validação de datas"
```

## Tipos:

- `feat`: Nova funcionalidade
  - `fix`: Correção de bug
  - `docs`: Documentação
  - `refactor`: Refatoração de código
  - `test`: Testes
  - `style`: Formatação
- 

## 9.2 Branches

```
bash

main          # Branch principal (código estável)
develop       # Branch de desenvolvimento
feat/crud-alunos # Branch de feature
fix/segfault   # Branch de correção
```

---

## 10. Checklist de Qualidade

Antes de fazer commit, verifique:

- ☐ Código compila sem warnings
  - ☐ Todas as funções estão documentadas
  - ☐ Não há variáveis com nomes ambíguos
  - ☐ Ponteiros são verificados antes do uso
  - ☐ Limites de arrays são respeitados
  - ☐ Código está indentado corretamente (4 espaços)
  - ☐ Guards de inclusão estão presentes nos .h
  - ☐ Comentários explicam "por quê", não "o quê"
  - ☐ Funções têm no máximo 50 linhas
  - ☐ Nomes seguem as convenções estabelecidas
- 

## 11. Exemplos Completos

### 11.1 Exemplo de Função Bem Documentada

```
c
```

```

// Função para associar um aluno a uma turma (realizar matrícula)
//
// Parâmetros:
// - ra: Registro Acadêmico do aluno (deve existir no sistema)
// - id_turma: Identificador da turma (deve existir no sistema)
//
// Retorna:
// - 1: Matrícula realizada com sucesso
// - 0: Erro (aluno já matriculado ou limite atingido)
//
// Observações:
// - Verifica se o aluno já está matriculado na turma
// - Respeita o limite máximo de matrículas
// - Atualiza o arquivo aluno_turma.csv automaticamente
int associarAlunoTurma(int ra, int id_turma) {
    carregarMatriculasMemoria();

    // Verifica se já está matriculado (evita duplicação)
    for (int i = 0; i < total_matriculas; i++) {
        if (matriculas[i].ra == ra && matriculas[i].id_turma == id_turma) {
            printf("Aviso: aluno já matriculado nesta turma.\n");
            return 0;
        }
    }

    // Adiciona matrícula se há espaço disponível
    if (total_matriculas < MAX_TURMAS * 10) {
        matriculas[total_matriculas].ra = ra;
        matriculas[total_matriculas].id_turma = id_turma;
        total_matriculas++;
        salvarMatriculasArquivo();

        printf("Aluno RA %d matriculado na turma ID %d.\n", ra, id_turma);
        return 1;
    }

    printf("Erro: limite de matrículas atingido.\n");
    return 0;
}

```

## 11.2 Exemplo de Struct Bem Definida

```
// Estrutura para representar um Aluno no sistema
// Utilizada para CRUD de alunos e associação com turmas
typedef struct {
    int ra;           // Registro Acadêmico (PK, único)
    char nome[MAX_NOME]; // Nome completo do aluno
    char email[MAX_NOME]; // Email institucional
    int ativo;        // Status: 1=ativo, 0=inativo
} Aluno;
```

## 12. Glossário de Termos

Termo	Significado
CRUD	Create, Read, Update, Delete
PK	Primary Key (Chave Primária)
FK	Foreign Key (Chave Estrangeira)
RA	Registro Acadêmico
CSV	Comma-Separated Values
UI	User Interface
API	Application Programming Interface

## Observações Finais

Este documento é **vivo** e deve ser atualizado conforme o projeto evolui. Todos os desenvolvedores devem seguir estes padrões para manter a **consistência** e **qualidade** do código.

**Última atualização:** Outubro/2025

**Responsável:** Equipe de Desenvolvimento PIM II