# RELATÓRIO TÉCNICO

## Sistema de Cadastro Escolar

**Disciplina**: Introdução às Técnicas de Programação
**Unidade**: 1
**Aluno**: Breno Igor da Silva
**Matrícula**: 20250065477
**Período**: 2025.2

## 1. INTRODUÇÃO E CONTEXTO

### 1.1 Nome do Projeto e Objetivo

O projeto desenvolvido é um Sistema de Cadastro Escolar. O objetivo é criar um programa que ajude a organizar as matrículas dos alunos do ensino fundamental de forma automática e prática.

### 1.2 Problema que o Projeto Resolve

Em muitas escolas, o cadastro de alunos ainda é feito de forma manual ou desorganizada, o que além de ser mais lento, pode gerar erros como:

- Informações incorretas (idade fora da faixa permitida, séries inválidas);
- Dificuldade para encontrar dados de um aluno específico;
- Falta de relatórios sobre quantos alunos há por série ou turno.

O sistema resolve esses problemas automatizando o cadastro, validando as informações e gerando relatórios de forma rápida.

### 1.3 Justificativa da Escolha

Conversei com uns amigos e decidi escolher esse projeto pela sua relevância prática no mercado de trabalho. Sistemas de cadastro e gerenciamento de dados são uma das aplicações mais comuns que programadores desenvolvem profissionalmente.

## 2. ANÁLISE TÉCNICA

### 2.1 Metodologia - Ferramentas Utilizadas

**Sistema Operacional:** Linux
**Compilador:** GCC versão 13.3.0
**Editor de Código:** Visual Studio Code
**Linguagem:** C

### 2.2 Aplicação dos Conceitos da U1

#### 2.2.1 As estruturas condicionais (if-else e switch) foram usadas em várias partes:

1. **Para validar os dados:**

- Verificar se a idade do aluno está entre 6 e 14 anos;
- Checar se a série digitada é válida (1ª a 9ª);
- Confirmar se o turno é M (manhã) ou T (tarde)

2. **Para definir a turma automaticamente:**

- Se o turno for M, a turma é A;
- Se o turno for T, a turma é B.

3. **No menu do sistema:** Usei um switch-case para escolher qual função executar baseado na opção do usuário (1=cadastrar, 2=listar, etc).

4. **Para separar alunos ativos de transferidos:** O sistema verifica o status de cada aluno e conta quantos estão matriculados ou transferidos.

#### 2.2.2 A lógica das estruturas de repetição usada no projeto

1. **Loop do menu principal (do-while):** O menu continua aparecendo até o usuário digitar 0 para sair. Escolhi do-while porque o menu precisa aparecer pelo menos uma vez.

2. **Loops de validação (do-while):** Quando o usuário digita algo errado (como uma série inválida), o programa pede novamente até receber um valor correto, impedindo que dados errados entrem no sistema.

3. **Loops para percorrer os alunos (for):** Sempre que preciso procurar um aluno, contar quantos há em cada série ou calcular estatísticas, uso um loop for que passa por todos os alunos cadastrados.

**2.2.3 A aplicação de vetores no projeto O projeto usa um array (vetor) principal que guarda até 100 alunos:** Cada posição do array guarda as informações completas de um aluno (nome, idade, série, etc). Também usei uma variável total_alunos para controlar quantas posições estão ocupadas, resolvendo assim, o problema de armazenar múltiplos alunos, deixando os dados organizados em um único lugar e facilitando o percorrimento dos loops.

OBS: não citei o uso da string porque não foi aplicado na U1.

**2.2.4 Organização e função das funções criadas** Criei 14 funções no total, separadas por tipo de tarefa:

**Funções de validação (3):**

- validar_idade(): verifica se a idade é válida;
- tem_vagas(): checa se ainda há espaço para cadastrar;

- definir_Turma_Automaticamente(): define a turma baseada no turno.

**Funções de cadastro e busca (3):**

- adicionar_aluno(): coloca um novo aluno no array;
- buscar_aluno_matriculado(): encontra um aluno pela matrícula;
- cadastro_alunos(): coleta os dados do usuário.

**Funções de estatísticas (5):**

- contar_status(): conta ativos e transferidos;
- calcular_idade_media(): faz a média de idade;
- contar_alunos_por_serie(): conta quantos há em cada série;
- contar_alunos_por_turno(): separa manhã e tarde;
- contar_alunos_por_turma(): conta por turma.

**Funções de exibição (3):**

listar_alunos(): mostra todos em uma tabela; buscar_alunos(): interface para buscar; calcular_estatisticas(): mostra relatório completo.

### 2.3 Estruturas de Dados

#### 2.3.1 O projeto usa principalmente a struct Aluno:

```
typedef struct {
    int matricula;
    char nome[25];
    int idade;
    int serie;
    char turma;
    char turno;
    char responsavel[20];
    int telefone;
    int cpf;
    int ativo;
} Aluno;
```

Essa struct agrupa todas as informações de um aluno em um único "pacote", tornando o código mais otimizado e sendo mais prático do que ter 10 variáveis saparadas para cada aluno.

#### 2.3.2 Também criei structs menores para organizar os retornos das funções de estatística:

1. **status_aluno:** guarda quantos estão matriculados e transferidos.
2. **contagem_turnos:** separa a contagem de manhã e tarde.

# 3. IMPLEMENTAÇÃO E REFLEXÃO

## 3.1 Dificuldades Encontradas

Algumas das dificuldades encontradas:

1. **Problema com o buffer do teclado:** Quando usava scanf() para ler texto, às vezes o programa pulava a leitura.
2. **Organizar tantas funções:** No começo foi confuso decidir o que cada função deveria fazer e como elas iam conversar entre si.

## 3.2 Soluções Implementadas

1. **Para o problema do buffer:** Usei getchar() antes de ler strings para consumir o Enter que sobrou: getchar(); // Limpa o buffer
   scanf("%24[^\n]", nome);
2. **Para organizar as funções:** Separei por categoria (validação, cadastro, estatísticas, exibição) e mantive cada função pequena e com um objetivo claro.

## 3.3 Organização do Código

Organizei o código na seguinte ordem:

1. **Definições no topo:** constantes e structs.
2. **Variáveis globais:** o array de alunos.
3. **Funções agrupadas por categoria:** primeiro as de validação, depois cadastro, depois estatísticas, e por último as de exibição.
4. **Função main no final:** só com o menu que chama as outras funções.

## 3.4 Conclusão

**Aprendizados obtidos:** O maior aprendizado foi entender melhor como dividir um programa grande em pedaços menores através de funções, tornando o código muito mais organizado e fácil de manter do que se fosse uma estrutura única e gigante. Outro ponto importante foi perceber que validar os dados do usuário é essencial, tendo em vista que sem essas verificações, o programa poderia travar ou aceitar informações incorretas, comprometendo todo o sistema. Além disso, entendi melhor o conceito e como Trabalhar com arrays. E por fim, aprendi na prática como gerenciar melhor o tempo e a trabalhar de forma mais organizada.

**Possíveis melhorias:** Para as próximas unidades, penso em adicionar:

**Editar cadastros:** Permitir mudar dados de alunos já cadastrados.
**Mais opções de busca:** Buscar por nome, série e turma ou turno, não só por matrícula.
**Sistema de notas:** Adicionar controle de notas e frequência.

**Melhorar a interface:** Usar cores no terminal.
**Remover o limite de 100 alunos:** Usar alocação dinâmica.