CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE LINS PROF. ANTONIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LAIS YUMI NAGAYAMA

PORTFÓLIO DE PROJETOS ACADÊMICOS

LINS/SP
2º SEMESTRE/2024

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE LINS PROF. ANTONIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LAIS YUMI NAGAYAMA

PORTFÓLIO DE PROJETOS ACADÊMICOS

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Lins Prof. Antônio Seabra, para obtenção do Título de Tecnólogos em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Felipe Maciel Rodrigues.

LINS/SP
2º SEMESTRE/2024

Nagayama, Lais Yumi

N147p Portfólio de projetos acadêmicos / Lais Yumi Nagayama. — Lins, 2024.

107f.

Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) — Faculdade de Tecnologia de Lins Professor Antonio Seabra: Lins, 2024.

Orientador(a): Me. Felipe Maciel Rodrigues

1. Algoritmo e lógica de programação. 2. Linguagem de programação. 3. Estruturas de dados. 4. Estruturas de dados. 5. Redes de computadores e Gestão de projetos. I. Rodrigues, Felipe Maciel. II. Faculdade de Tecnologia de Lins Professor Antonio Seabra. III. Título.

CDD 004.21

Gerada automaticamente pelo módulo web de ficha catalografica da FATEC Lins mediante dados fornecidos pelo(a) autor(a).

PORTFÓLIO DE PROJETOS ACADÊMICOS

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Lins Prof. Antônio Seabra, como parte dos requisitos para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas sob orientação do Prof. Me. Felipe Maciel Rodrigues.

Data de aprovação: ____/____/____

_____

Nome do Orientador

_____

Nome do Examinador 1

_____

Nome do Examinador 2

DEDICATÓRIA

Dedico este trabalho com gratidão à minha família e aos amigos, que me apoiaram incondicionalmente ao longo de toda a minha jornada acadêmica.

Lais Yumi Nagayama

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus por me conceder força e determinação para chegar até aqui.

Aos professores, meu sincero agradecimento por toda a dedicação e pelo compromisso em transmitir seus conhecimentos durante o curso, o que foi fundamental para a minha formação. Em especial, agradeço ao meu orientador, Felipe Maciel Rodrigues, por suas contribuições essenciais e pelo suporte constante durante o desenvolvimento deste trabalho.

À minha família, deixo minha profunda gratidão por todo o amor, apoio e compreensão inabaláveis. Vocês acreditaram em mim e me deram forças para continuar, mesmo nos momentos mais difíceis. Sou eternamente grata por serem minha maior fonte de motivação e por todo o suporte que me proporcionaram.

Aos amigos que me acompanharam nessa jornada acadêmica, agradeço pela amizade, pelas palavras de encorajamento e pela presença constante, que fizeram toda a diferença ao longo do caminho.

Lais Yumi Nagayama

RESUMO

Este trabalho de conclusão de curso aborda a trajetória de aprendizado nas disciplinas de Algoritmo e Lógica de Programação, Linguagem de Programação, Estruturas de Dados, Banco de Dados, Redes de Computadores e Gestão de Projetos no curso de Análise e Desenvolvimento de Sistemas (ADS). O objetivo é destacar a importância do desenvolvimento do raciocínio lógico e da aplicação prática dos conceitos aprendidos, visando à criação de sistemas de software eficientes e robustos. A metodologia inclui a análise das atividades práticas e teóricas realizadas ao longo do curso. Os resultados mostram a evolução do conhecimento e das habilidades técnicas dos alunos, com destaque para a criação de projetos aplicáveis ao mercado de trabalho. As conclusões ressaltam a importância da combinação de abordagens práticas e teóricas na formação de profissionais competentes em tecnologia da informação, preparados para enfrentar desafios complexos no desenvolvimento de software.

Palavras-chave: Algoritmo e Lógica de Programação. Linguagem de Programação. Estruturas de Dados. Banco de Dados. Redes de Computadores. Gestão de Projetos.

ABSTRACT

This final paper addresses the learning trajectory in the subjects of Algorithms and Logic Programming, Programming Languages, Data Structures, Databases, and Computer Networks within the Analysis and Systems Development (ADS) course. The objective is to highlight the importance of developing logical reasoning and the practical application of learned concepts, aiming at the creation of efficient and robust software systems. The methodology includes the analysis of practical and theoretical activities carried out throughout the course. The results demonstrate the evolution of students' knowledge and technical skills, with an emphasis on the creation of projects applicable to the job market. The conclusions underscore the importance of combining practical and theoretical approaches in training competent information technology professionals, prepared to face complex challenges in software development.

Keywords: Algorithms and Logic Programming. Programming Languages. Data Structures. Databases. Computer Networks. Project Management.

LISTA DE ILUSTRAÇÕES

Figura 2.1 - Exemplo de algoritmo em linguagem natural 17
Figura 2.2 - Principais formas geométricas utilizadas em fluxogramas 17
Figura 2.3 - Exemplo de um fluxograma convencional 18
Figura 2.4 - Algoritmo utilizando o pseudocódigo 19
Figura 2.5 - Algoritmo com estrutura simples 20
Figura 2.6 - Algoritmo com estrutura composta 21
Figura 3.1 - Estrutura básica da função main() em Linguagem C 25
Figura 3.2 - Atividade com if 26
Figura 3.3 - Atividade com if-else 27
Figura 3.4 - Atividade com For 28
Figura 3.5 - Atividade com While 28
Figura 3.6 - Atividade com Do-While 29
Figura 3.7 - Atividade com Vetor 30
Figura 3.8 - Atividade com Matriz 31
Figura 3.9 - Atividade com Structs 32
Figura 4.1 – Structs 34
Figura 4.2 - Cadastrar Livro 36
Figura 4.8 - Cadastrar cliente 46
Figura 4.9 - Listar Clientes 47
Figura 4.10 - Localizar livro 49
Figura 4.11 - Efetuar venda 51
Figura 4.12 - Fechar Pedido 53
Figura 4.13 - Relatório de vendas 55
Figura 4.14 - Relatório detalhado de vendas 56
Figura 4.15 - Cadastrar autor 58
Figura 4.16 - Listar autores 59
Figura 4.17 - Listar autores 61
Figura 4.18 - Relacionamento entre livro e autor 63
Figura 4.19 - Listar relacionamento 65
Figura 4.20 - Consultar livro por autor 67
Figura 4.21 - Menu principal 69

Figura 5.1 - Resolução atividade 75
Figura 5.2 - Resolução atividade 77
Figura 6.1 - Comunicação Simplex 78
Figura 6.2 - Comunicação Half-duplex 79
Figura 6.3 - Comunicação Full-duplex 80
Figura 6.4 - Topologia estrela conectando 4 estações 81
Figura 6.5 - Topologia barramento conectando 3 estações 82
Figura 6.6 - Topologia anel conectando 6 estações 82
Figura 6.7 - Topologia mesh completamente conectada (5 dispositivos) 83
Figura 6.8 - Exemplo prático de Switch 84
Figura 6.9 - Exemplo prático de DHCP 85
Figura 6.10 - Exemplo prático de Web Server 86
Figura 6.11 - Exemplo prático de DNS 87
Figura 6.12 - Exemplo prático de Router 88
Figura 6.13 - Exemplo prático de Bridge 89
Figura 7.1 - EAP 91
Figura 7.2 - Gerenciamento das partes interessadas 92
Figura 7.3 - Cronograma 93
Figura 7.4 - Matriz RACI 94
Figura 8.1 – Barra de navegação 95
Figura 8.2 – Site Fatec 96
Figura 8.3 – Página Inicial 97
Figura 8.4 – Seção Sobre mim 98
Figura 8.5 – Seção Educação 99
Figura 8.6 – Seção Experiência 100
Figura 8.7 – Seção Portfólio 101
Figura 8.8 – Botão de download do portfólio 101
Figura 8.9 – Seção Contato 102

LISTA DE ABREVIATURAS E SIGLAS

ab - append binary (Anexar Binário)

ADS - Análise e Desenvolvimento de Sistemas

C# - C Sharp

CSS - Cascading Style Sheets

DHCP - Dynamic Host Configuration Protocol

DNS - Sistema de Nomes de Domínio

EAP - Estrutura Analítica do Projeto

HTML - HyperText Markup Language

IDE - Integrated Development Environment

LAN - Local Area Network

MAN - Metropolitan Area Network

MER - Modelo Entidade-Relacionamento

PAN - Personal Area Network

PDF - Portable Document Format

PHP - Hypertext Preprocessor

RACI - Responsável, Aprovador, Consultado e Informado

rb - read binary (Ler binário)

rb+ - read binary + (Ler e escrever binário)

SQL - Structured Query Language

TCC - Trabalho de Conclusão de Curso

WAN - Wide Area Network

WLAN - Wireless Local Area Network

WMAN - Wireless Metropolitan Area Network

SUMÁRIO

RESUMO 7
ABSTRACT 8
1 INTRODUÇÃO 13
2 ALGORITMO E LÓGICA DE PROGRAMAÇÃO 16
2.1 ALGORITIMO 16
2.2 CONCLUSÃO 21
3 LINGUAGEM DE PROGRAMAÇÃO 23
3.1 LINGUAGEM C 23
3.2 COMPILADORES 24
3.3 FUNÇÕES 24
3.3.1 Função main() 24
3.4 ESTRUTURAS DE CONTROLE 25
3.4.1 Atividade com if 26
3.4.2 Atividade com if-else 26
3.4.3 Atividade com for 27
3.4.4 Atividade com while 28
3.4.5 Atividade com do-while 29
3.5.1 Atividade com vetor 30
3.5.2 ATIVIDADE COM MATRIZ 30
3.5.3 Atividade com structs 31
4 ESTRUTURA DE DADOS 33
4.1 SISTEMA DE GERENCIAMENTO DE LIVRARIA 34
4.1.1 Structs 34
4.1.2 Cadastrar Livro 35
4.1.3 Listar Livros 37
4.1.4 Consultar código de livros 38
4.1.5 Consultar título de livros 40
4.1.6 Alterar livros 41
4.1.7 Excluir livros 44
4.1.8 Cadastrar Cliente 46
4.1.9 Listar Clientes 47
4.1.10 Localizar clientes 48
4.1.11 Efetuar venda 50

4.1.12 Fechar pedido 52
4.1.13 Relatório de vendas 54
4.1.14 Relatório de vendas detalhado 55
4.1.15 Cadastrar autor 57
4.1.16 Listar autores 59
4.1.17 Listar autores 60
4.1.18 Relacionamento entre livro e autor 62
4.1.19 Listar relacionamento 64
4.1.20 Consultar livro por autor 66
4.1.21 Menu principal 68
4.2 CONCLUSÃO 70
5 BANCO DE DADOS 72
5.1 MODELO ENTIDADE-RELACIONAMENTO E MAPEAMENTO DE MODELOS DE DADOS 72
5.1.1 Atividades práticas do Modelo Entidade-Relacionamento (MER) 73
5.2 CONCLUSÃO 77
6 REDES DE COMPUTADORES 78
6.1 TIPOS DE COMUINICAÇÃO 78
6.2 TIPOS DE REDES 80
6.3 TOPOLOGIAS DE REDES 81
6.4 DISPOSITIVOS DE REDES 83
6.4.1 Switch 83
6.4.2 DHCP (Dynamic Host Configuration Protocol) 84
6.4.3 Web Server 85
6.4.4 Sistema de Nomes de Domínio (DNS) 86
6.4.5 Router 87
6.4.6 Bridge 88
6.5 CONCLUSÃO 89
7 GERENCIAMENTO DE PROJETOS 90
7.1 GESTÃO DE ESCOPO 90
7.2 GESTÃO DAS PARTES INTERESSADAS 91
7.3 GESTÃO DE CRONOGRAMA 92
7.4 GESTÃO DA COMUNICAÇÃO 93
7.5 CONCLUSÃO 94
8 MANUAL DO USUÁRIO 95

8.1 MENU PORTFÓLIO....95
8.2 PÁGINA INICIAL....96
8.3 SEÇÃO SOBRE MIM....97
8.4 SEÇÃO EDUCAÇÃO....98
8.5 SEÇÃO EXPERIÊNCIA PROFISSIONAL....99
8.6 SEÇÃO PORTFÓLIO....100
8.7 SEÇÃO CONTATO....101
9 CONCLUSÃO GERAL....103
REFERÊNCIAS....105

1 INTRODUÇÃO

Este trabalho de conclusão de curso tem como objetivo explorar e apresentar a trajetória de aprendizado nas disciplinas de Algoritmo e Lógica de Programação, Linguagem de Programação, Estruturas de Dados, Banco de Dados, Redes de Computadores e Gestão de Projetos destacando a aplicação prática dos conceitos estudados no desenvolvimento de sistemas de *software*.

No primeiro semestre, a disciplina de Algoritmo e Lógica de Programação foi crucial para a construção de uma base sólida em programação. Durante esse período, os alunos aprenderam a formular comandos em pseudocódigos e algoritmos usando linguagens como Portugol. A importância do raciocínio lógico foi constantemente enfatizada, assim como a compreensão de variáveis e constantes e o uso de estruturas de seleção simples e composta. O pseudocódigo se destacou como uma ferramenta essencial, permitindo a transição de conceitos abstratos para implementações práticas e compreensíveis.

No segundo semestre, a disciplina de Linguagem de Programação aprofunda o conhecimento com a exploração da Linguagem C, desenvolvida por Dennis M. Ritchie e Ken Thompson. Esta linguagem é conhecida por sua capacidade de criar códigos eficientes e robustos. O estudo envolveu a sintaxe e semântica da linguagem, bem como o processo de compilação, que traduz o código-fonte em linguagem de máquina executável. A IDE Dev C++ foi considerada um ambiente prático e intuitivo para o desenvolvimento de *software.*

A partir do terceiro semestre, a disciplina de Estruturas de Dados introduziu os alunos à representação e manipulação eficiente de dados e à implementação de algoritmos eficazes. Foram abordados conceitos fundamentais como listas, ponteiros, funções para manipulação de *strings* e gerenciamento de arquivos. A aplicação prática desses conceitos culminou no desenvolvimento de um sistema de gerenciamento de livraria, demonstrando a importância de selecionar e utilizar adequadamente diversas estruturas de dados para resolver problemas específicos. Este projeto consolidou o conhecimento teórico e preparou os alunos para desafios reais no desenvolvimento de sistemas.

No quarto semestre, a disciplina de Banco de Dados permitiu o aprofundamento em conceitos essenciais para o armazenamento e recuperação de informações de forma estruturada. Foram abordados tópicos como o Modelo Entidade-

Relacionamento (MER) e o mapeamento de modelos de dados. Através da modelagem de dados, os alunos aprenderam a criar representações lógicas de informações que refletem a realidade de um contexto específico, garantindo a integridade e eficiência no armazenamento e acesso aos dados. As atividades práticas consolidaram o conhecimento teórico e aplicaram técnicas de abstração e mapeamento de dados, essenciais para o desenvolvimento de sistemas.

Durante o quinto semestre, a disciplina de Redes de Computadores se mostrou fundamental para o desenvolvimento de competências essenciais na área de tecnologia da informação. Foram abordados os tipos de comunicação, tipos de redes, topologias e dispositivos de rede como switches, roteadores, bridges e servidores. A prática com o Cisco Packet Tracer permitiu a simulação de redes e a experimentação com diferentes configurações e topologias, facilitando a compreensão dos conceitos e a aplicação prática deles.

No sexto semestre, a disciplina de Gestão de Projetos foi muito importante para desenvolver habilidades de planejamento e organização. Os alunos aprenderam sobre gestão de escopo, cronograma, partes interessadas e utilizaram ferramentas como a matriz RACI, EAP e ProjectLibre para criar e monitorar cronogramas. Esse aprendizado promoveu uma compreensão prática da organização de projetos e preparou os estudantes para lidar com as responsabilidades e expectativas de stakeholders, aplicando métodos fundamentais para o sucesso de projetos.

A escolha destes temas se justifica pela importância desses conhecimentos na formação de profissionais de computação. Conceber algoritmos eficientes, estruturar o pensamento lógico e traduzir problemas em soluções algorítmicas são habilidades muito importantes em qualquer campo relacionado à ciência da computação. Além disso, o domínio de linguagens de programação, estruturas de dados, banco de dados, redes de computadores e gestão de projetos capacita os estudantes a enfrentarem os complexos desafios do desenvolvimento de software, promovendo uma abordagem organizada e sistemática na resolução de problemas.

A metodologia utilizada uniu teoria e prática por meio de atividades em sala de aula e projetos aplicados, o que consolidou o aprendizado. Os resultados obtidos mostram a importância de uma base sólida em algoritmos, lógica de programação, estruturas de dados, banco de dados, redes de computadores e gestão de projetos, destacando como esses conhecimentos são essenciais para o sucesso no desenvolvimento de soluções eficazes.

Para consolidar esse conhecimento e torná-lo acessível, foi desenvolvido um portfólio digital, estruturado em HTML, CSS e JavaScript, que organiza e expõe de maneira responsiva os principais projetos e realizações ao longo do curso. Com o intuito de orientar o usuário na navegação e utilização das funcionalidades do portfólio, elaborou-se um manual que detalha cada seção e descreve os recursos de interatividade e navegação, tornando-o acessível e fácil de usar.

2 ALGORITMO E LÓGICA DE PROGRAMAÇÃO

A disciplina de Algoritmo e Lógica de Programação, ministrada no primeiro semestre do Curso de Análise e Desenvolvimento de Sistemas (ADS), proporcionou uma sólida base para entendermos a importância do raciocínio lógico nas práticas de programação. Na disciplina adquirimos a habilidade de formular comandos em pseudocódigos ou em algoritmos escritos em linguagens como Portugol. O Portugol é uma linguagem estruturada criada para tornar o processo de aprendizagem de programação mais acessível e compreensível. Esta linguagem torna as instruções do algoritmo passíveis de compreensão por parte dos programadores humanos. Ao longo deste curso, não apenas aprendemos os conceitos básicos de algoritmos, mas também internalizamos a importância crucial do raciocínio lógico na construção de soluções eficazes em programação.

2.1 ALGORITIMO

> Informalmente, um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída. Portanto, um algoritmo é uma sequência de etapas computacionais que transformam a entrada na saída (Cormen et al., 2012, p.3).

Há diversas formas de representar um algoritmo, dentre estas formas podemos citar a Descrição Narrativa, Fluxograma Convencional e Pseudocódigo.

Na Descrição Narrativa os algoritmos são expressos em linguagem natural, conforme a figura 2.1.

Figura 2.1 - Exemplo de algoritmo em linguagem natural

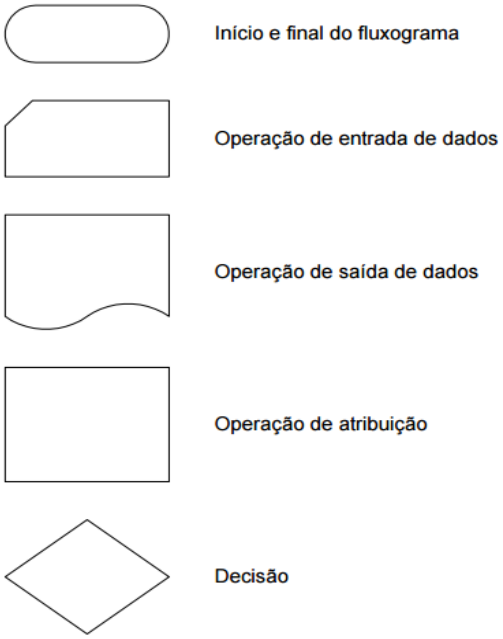
```
Algoritmo: Preparar Miojo
          1. Inicialização:
             1.1. Reúna todos os ingredientes e utensílios necessários: pacote de miojo, água, panela, fogão, colher e tigela.
             1.2. Certifique-se de ter acesso a um cronômetro ou relógio para controlar o tempo de cozimento.
          2. Preparação:
             2.1. Encha a panela com água suficiente para cobrir o miojo.
             2.2. Coloque a panela no fogão e acenda o fogo.
             2.3. Aguarde até que a água atinja o ponto de fervura.
          3. Cozimento:
             3.1. Abra o pacote de miojo e quebre os blocos de massa, se necessário.
             3.2. Adicione o miojo à água fervente, mexendo ocasionalmente com a colher para evitar que grude no fundo da
          panela.
             3.3. Cozinhe o miojo por aproximadamente 3 a 5 minutos, ou conforme as instruções do pacote.
            3.4. Monitore o tempo usando um cronômetro.
          4. Finalização:
            4.1. Desligue o fogão e escorra a água da panela.
            4.2. Transfira o miojo cozido para uma tigela.
            4.3. Adicione o tempero do pacote, misturando bem.
            4.4. Deixe o miojo descansar por alguns minutos para absorver o sabor.
          5. Servir:
            5.1. Sirva o miojo em uma tigela.
            5.2. Aproveite sua refeição!
Fim do Algoritmo.
```

Fonte: Elaborada pela autora, 2024

O Fluxograma Convencional usa símbolos gráficos em vez de palavras para representar processos de forma visual e simplificada. De acordo com Forbellone e Eberspacher (2005, p.9) “As formas gráficas são mais puras por serem mais fiéis ao raciocínio original.”.

A figura 2.2 ilustra as principais formas geométricas utilizadas em fluxogramas.

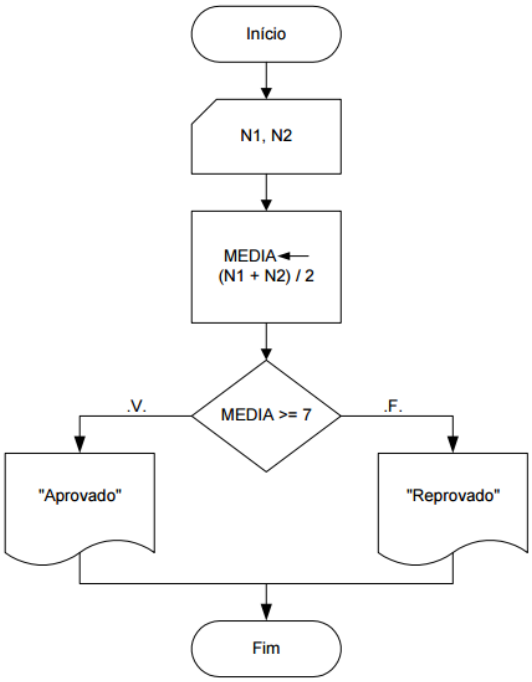
Figura 2.2 - Principais formas geométricas utilizadas em fluxogramas



Fonte: DCA 800 – Eng. Química, 2004, p.4

A figura 2.2 exemplifica o fluxograma do algoritmo de cálculo de média de um aluno.

Figura 2.3 - Exemplo de um fluxograma convencional



Fonte: DCA 800 – Eng. Química, 2004, p.5

O pseudocódigo é uma forma de representação de algoritmos rica em detalhes. Com esta linguagem, aprendemos não apenas a definir os tipos de variáveis, mas também a estruturar de maneira precisa e compreensível as operações e decisões que compõem um algoritmo. Além disso, o pseudocódigo nos capacita a expressar eficientemente a lógica subjacente e facilita a transição suave do conceito abstrato para a implementação prática em linguagens de programação mais formais.

Durante a disciplina aprendemos que um dado é considerado uma variável quando apresenta a capacidade de ser modificado em algum momento ao longo do tempo, ou seja, durante a execução do algoritmo no qual é empregado. Compreendemos também que um dado é considerado constante quando seu valor permanece inalterado ao longo do tempo, ou seja, desde o início até o término da execução do algoritmo.

Através da representação detalhada e ao mesmo tempo flexível do Pseudocódigo, foi possível experimentar e ajustar nossas lógicas de maneira mais

fluída para posteriormente traduzir estes conceitos para linguagens de programação mais formais.

Pode-se concluir que o pseudocódigo emerge como uma ferramenta valiosa no universo da programação, proporcionando não apenas uma representação detalhada e acessível dos algoritmos, mas também servindo como um meio eficaz para traduzir conceitos abstratos em implementações concretas. Ao compreender a dinâmica das variáveis e constantes, torna-se possível não apenas conceber algoritmos eficientes, mas também adaptá-los de maneira flexível ao longo da execução, resultando em soluções mais robustas e adaptáveis às demandas práticas da programação.

A figura 2.4 exemplifica um algoritmo desenvolvido em sala de aula com o intuito de estimular nosso raciocínio lógico.

Figura 2.4 - Algoritmo utilizando o pseudocódigo

```
Faça um algoritmo que possa entrar com o saldo de uma aplicação e o valor da taxa de juros. Calcule e
imprima o valor do rendimento e o valor total depois do rendimento.


Algoritmo: Cálculo de rendimento;
        Var
        saldo, txJuros, vlRend, vlTotal: Real;
        Início
                Escreva ('Digite o saldo:');
                Leia (saldo);
                Escreva ('Digite a taxa de juros:');
                Leia (txJuros);
                vlRend:= (saldo*txJuros)/100;
                vlTotal:= saldo+vlRend;
                Escreva ('O valor do rendimento é:', vlRend);
                Escreva ('O valor total do rendimento é:', vlTotal);
        Fim.
```

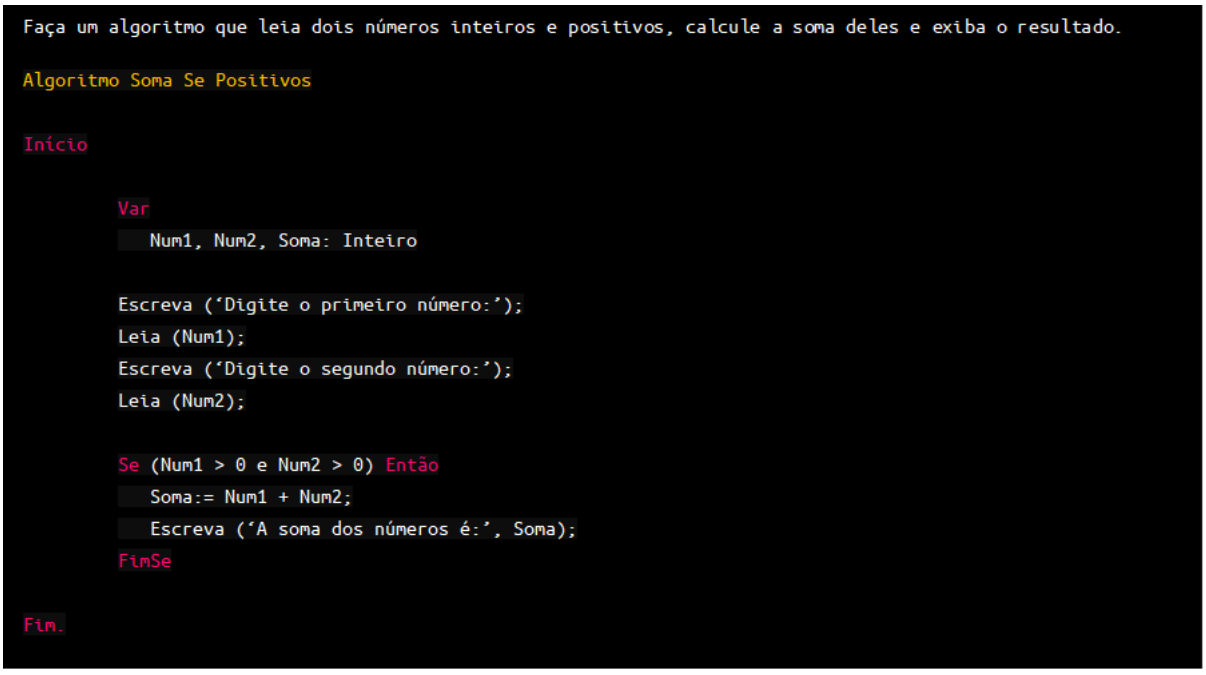
Fonte: Elaborada pela autora, 2024

Ainda na disciplina de “Algoritmo e Lógica de Programação”, foi possível conhecer a estrutura de seleção simples e composta. A seleção simples, muitas vezes expressa por meio da instrução 'se-então', é essencial para condicionar a execução de um bloco de código com base em uma única condição. Isso oferece aos programadores a capacidade de direcionar o fluxo do algoritmo de acordo com uma

avaliação lógica específica. Por outro lado, a seleção composta, frequentemente representada pela instrução 'se-então-senão', amplia essa capacidade ao permitir que dois caminhos distintos sejam tomados com base em uma condição. Dessa forma, o desenvolvedor pode fornecer soluções mais flexíveis e adaptáveis, adicionando caminhos alternativos para situações em que múltiplas condições podem ser satisfeitas. O entendimento profundo dessas estruturas de seleção é fundamental para a construção de algoritmos robustos e eficientes, onde a tomada de decisões é uma parte intrínseca do processo.

A figura 2.5 apresenta um exemplo de estrutura simples que foi desenvolvida durante o curso de Algoritmo e Lógica de Programação.

Figura 2.5 - Algoritmo com estrutura simples



```
Faça um algoritmo que leia dois números inteiros e positivos, calcule a soma deles e exiba o resultado.

Algoritmo Soma Se Positivos

Início

        Var
           Num1, Num2, Soma: Inteiro

        Escreva ('Digite o primeiro número:');
        Leia (Num1);
        Escreva ('Digite o segundo número:');
        Leia (Num2);

        Se (Num1 > 0 e Num2 > 0) Então
           Soma:= Num1 + Num2;
           Escreva ('A soma dos números é:', Soma);
        FimSe

Fim.
```

Fonte: Elaborada pela autora, 2024

A figura 2.6 apresenta um exemplo de estrutura composta.

Figura 2.6 - Algoritmo com estrutura composta

```
Faça um algoritmo que leia dois números inteiros e identifique se eles são iguais ou diferentes. Caso eles
sejam iguais imprima uma mensagem dizendo que eles são iguais. Caso contrário, imprima uma mensagem que eles
são diferentes.


Algoritmo: Identificando números;
        Var
        N1, n2: Inteiro;
        Início
                Escreva (‘Digite o primeiro número:’);
                Leia (n1);
                Escreva (‘Digite o segundo número:’);
                Leia (n2);
                Se (n1=n2) Então
                   Escreva (‘Os números são iguais’);
                Senão
                   Escreva (‘Os números são diferentes’);
                FimSe
        Fim.
```

Fonte: Elaborada pela autora, 2024

Essas representações visuais ilustram de maneira concreta a aplicação prática das estruturas aprendidas, destacando a compreensão adquirida durante o processo de estudo.

2.2 CONCLUSÃO

Com tudo o conteúdo abordado na disciplina Algoritmo e Lógica de Programação, é possível concluir que a disciplina desempenha um papel fundamental na formação de profissionais de computação e áreas afins. Ao proporcionar uma compreensão sólida dos fundamentos lógicos e algorítmicos, ela prepara os estudantes para enfrentar os desafios complexos do desenvolvimento de software. A capacidade de conceber algoritmos eficientes, estruturar o pensamento lógico e traduzir problemas em soluções algorítmicas é essencial em qualquer campo relacionado à ciência da computação. Além disso, a disciplina serve como alicerce para o aprendizado de linguagens de programação mais avançadas, facilitando a transição para conceitos mais complexos. O domínio desses princípios não apenas

capacita os estudantes a escreverem código de maneira eficaz, mas também promove uma abordagem sistemática e organizada na resolução de problemas, habilidade crucial em qualquer ambiente profissional relacionado à tecnologia da informação.

3 LINGUAGEM DE PROGRAMAÇÃO

Durante o segundo semestre do Curso de ADS, houve a oportunidade de explorar os fundamentos da Linguagem C na disciplina de Linguagem de Programação, a fim de aprofundar os conhecimentos em conceitos e estruturas essenciais. Para enriquecer ainda mais as atividades práticas, foi utilizado a IDE Dev C++, que proporcionou aos alunos um ambiente de desenvolvimento robusto e intuitivo.

3.1 LINGUAGEM C

A linguagem C, desenvolvida por Dennis M. Ritchie e Ken Thompson no laboratório de Bell em 1972, é reconhecida como um marco significativo no campo da programação de sistemas. Desde sua criação, ela tem sido fundamental na construção de diversos aplicativos e sistemas computacionais.

Uma das razões principais para a popularidade da linguagem C é sua capacidade de gerar códigos eficientes e robustos. A sintaxe da linguagem é simples, mas poderosa, permitindo o desenvolvimento rápido e eficaz de sistemas complexos.

Além disso, C proporciona uma compreensão profunda da programação de baixo nível, sendo frequentemente escolhida por desenvolvedores que desejam conhecer os detalhes internos do sistema. Por exemplo, a manipulação direta de ponteiros em C possibilita um gerenciamento preciso da memória, resultando em programas mais eficientes e robustos.

Como mencionado por Jandl Junior (2019, p. 02) “A linguagem C foi seguramente a mais utilizada no mundo para desenvolvimento profissional. Embora esta não seja a situação atual, em razão do surgimento de outras plataformas de programação.”.

O legado da linguagem C continua a ter uma grande influência no desenvolvimento de software. Linguagens como Java, C#, PHP, JavaScript, C++, Python, entre outras, foram influenciadas por C. Elas herdaram muitas características de C, incluindo tipagem estática, estruturas de controle, alocação de memória e gerenciamento de ponteiros. Essa influência cruzada demonstra a flexibilidade e a adaptabilidade da linguagem C ao longo do tempo.

3.2 COMPILADORES

O conhecimento da linguagem C abrange mais do que apenas a sua sintaxe e semântica. Compreender o processo de compilação é essencial para o uso eficaz dessa linguagem. Os compiladores desempenham um papel vital nesse processo, traduzindo o código fonte em linguagem de máquina executável.

Explorar como os compiladores operam e otimizam o código é fundamental para maximizar o desempenho e a eficiência dos programas em C. Além de garantir que o código seja traduzido corretamente para a linguagem de máquina, os compiladores também implementam diversas otimizações que melhoram a performance dos programas.

Uma maneira prática de aplicar esse conhecimento é através de ambientes de desenvolvimento integrado (IDEs), como o Dev C++. Esses IDEs oferecem não apenas uma interface amigável para escrever e depurar código, mas também incorporam compiladores internos que transformam o código em programas executáveis. Assim, compreender o papel dos compiladores no desenvolvimento de software é essencial para os desenvolvedores, e os IDEs como o Dev C++ proporcionam uma plataforma conveniente para aplicar esse conhecimento na prática.

3.3 FUNÇÕES

Segundo Schildt (1996, p.151) “Em C, cada função é um bloco discreto de código”. Os blocos de códigos executam tarefas específicas. As funções são fundamentais para organizar e modularizar o código, tornando-o mais legível, reutilizável e fácil de manter. Todo programa em C deve incluir, no mínimo, uma função principal chamada main().

3.3.1 Função main()

Durante o estudo da disciplina, foi possível compreender que a função main() desempenha um papel fundamental no início da execução de um programa em C. Sendo a primeira função a ser chamada, marca o ponto de partida do programa. A

presença da função main() é obrigatória em algum ponto do código, pois é através dela que o compilador C determina o início da execução do programa.

A figura 3.1 exemplifica a estrutura básica da função main().

Figura 3.1 - Estrutura básica da função main() em Linguagem C

```c
int main() {
    // Corpo da função main
    return 0;
}
```

Fonte: Elaborada pela autora, 2024

A linguagem C, além da função principal main(), inclui diversas funções padrão para entrada e saída. Duas das mais conhecidas são printf() e scanf(). A função printf() serve para exibir dados formatados na tela, enquanto a função scanf() captura a entrada do usuário e a armazena em variáveis. Essas funções são essenciais para tornar os programas em C mais interativos e funcionais, representando apenas algumas das muitas opções disponíveis para entrada e saída na linguagem.

3.4 ESTRUTURAS DE CONTROLE

As estruturas de controle desempenham um papel fundamental na programação, ao permitir que os programadores controlem o fluxo de execução de um programa e tomem decisões com base em determinadas condições. Essas estruturas são essenciais para criar programas dinâmicos e adaptáveis que respondam de maneira apropriada a diferentes situações. Com as estruturas de controle, pode-se determinar quais partes do código serão executadas e em que ordem, dependendo das circunstâncias.

Na Linguagem C, diversas estruturas de controle estão disponíveis para auxiliar na tomada de decisões e no controle do fluxo de execução do programa. Durante o segundo semestre desta disciplina, foram exploradas algumas dessas estruturas, incluindo if, if-else e os loops (for, while, do-while).

Atividades desenvolvidas em sala de aula foram seguidas para promover a fixação do conteúdo, além de estimular o raciocínio lógico dos alunos.

3.4.1 Atividade com if

Desenvolva um programa em Linguagem C que solicita ao usuário informações pessoais, como nome, sexo (F/M) e estado civil (S/C), bem como o ano de casamento, se aplicável. Em seguida, o programa calcula e exibe o tempo de casamento apenas para mulheres casadas (sexo 'F' ou 'f') e que tenham informado o estado civil como casadas (estado civil 'C' ou 'c').

A resolução da atividade é apresentada na figura 3.2.

Figura 3.2 - Atividade com if

```c
#include<stdio.h>
#include<stdlib.h>
#define AnoAtual 2022

main()

{
char Nome [20];
char Sexo, EstCivil;
int AnoCasamento, TempoCasamento=0;

printf ("\n Digite o nome: ");
fflush (stdin);
gets(Nome);
printf ("\n Digite o sexo  (F/M):");
fflush (stdin);
Sexo=getchar ();
printf ("\n Digite o estado civil: ");
fflush (stdin);
EstCivil=getchar ();
printf ("\n Digite o ano de casamento ");
scanf ("%d",&AnoCasamento);
if((Sexo=='F'||Sexo=='f')&&(EstCivil=='C'||EstCivil=='c'))
{
TempoCasamento=AnoAtual-AnoCasamento;
printf ("\n Tempo de casamento %d", TempoCasamento);
system ("pause > null");
}
}
```

Fonte: Elaborada pela autora, 2024

3.4.2 Atividade com if-else

Escreva um programa em C que calcula o valor total a ser pago por uma determinada quantidade de combustível vendida, com base em descontos para volumes específicos de álcool ou gasolina. O programa deve solicitar ao usuário a quantidade de litros vendidos e o tipo de combustível (álcool ou gasolina). Se o tipo de combustível vendido for álcool, o programa aplica um desconto de 3% para volumes até 20 litros ou 5% para volumes acima de 20 litros. Se for gasolina, o

programa aplica um desconto de 4% para volumes até 20 litros ou 6% para volumes acima de 20 litros. O programa deve exibir o valor a ser pago.

A resolução da atividade é apresentada na Figura 3.3.

Figura 3.3 - Atividade com if-else

```c
#include<stdio.h>
#include<stdlib.h>
#include<locale.h>
#define PrecoG 4.30
#define PrecoA 3.34

main()

{setlocale( LC_ALL,"Portuguese");

float QtdeLitros,ValoraPagar;
char TipoCombustivel;

printf("\n Digite a quantidade de litros vendidos: ");
scanf("%f", &QtdeLitros);
printf("\n Digite o tipo de combustível vendido A para álcool ou G para gasolina: ");
fflush (stdin);
TipoCombustivel=getchar();
if (TipoCombustivel=='A')
    if(QtdeLitros<=20)
    ValoraPagar=(PrecoA*QtdeLitros*0.97);
    else
    ValoraPagar=(PrecoA*QtdeLitros*0.95);
if(TipoCombustivel=='G')
    if(QtdeLitros<=20)
    ValoraPagar=(PrecoG*QtdeLitros*0.96);
    else
    ValoraPagar=(PrecoG*QtdeLitros*0.94);

printf("\n O valor a ser pago é: %0.2f", ValoraPagar);
system("pause>null");
}
```

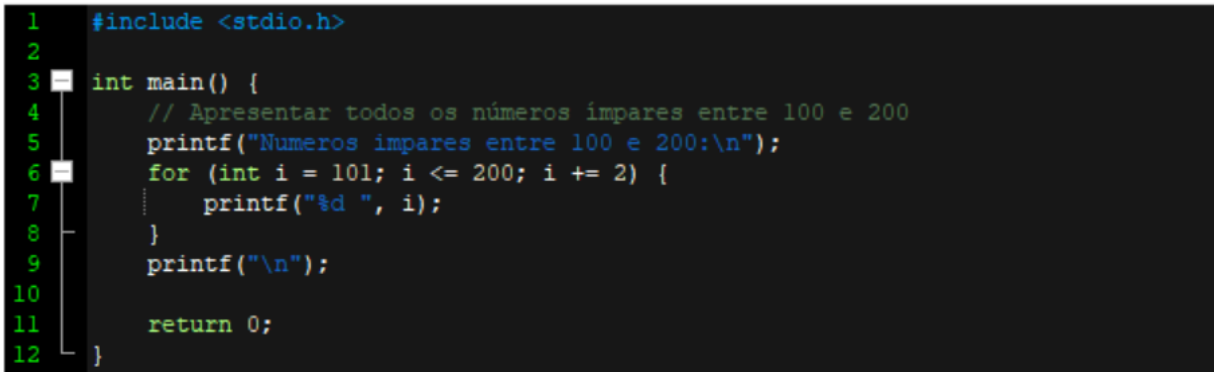
Fonte: Elaborada pela autora, 2024

3.4.3 Atividade com for

Escreva um programa para apresentar todos os valores numéricos inteiros ímpares situados na faixa de 100 a 200.

A resolução da atividade é apresentada na Figura 3.4.

Figura 3.4 - Atividade com For



```c
#include <stdio.h>

int main() {
    // Apresentar todos os números ímpares entre 100 e 200
    printf("Numeros impares entre 100 e 200:\n");
    for (int i = 101; i <= 200; i += 2) {
        printf("%d ", i);
    }
    printf("\n");

    return 0;
}
```

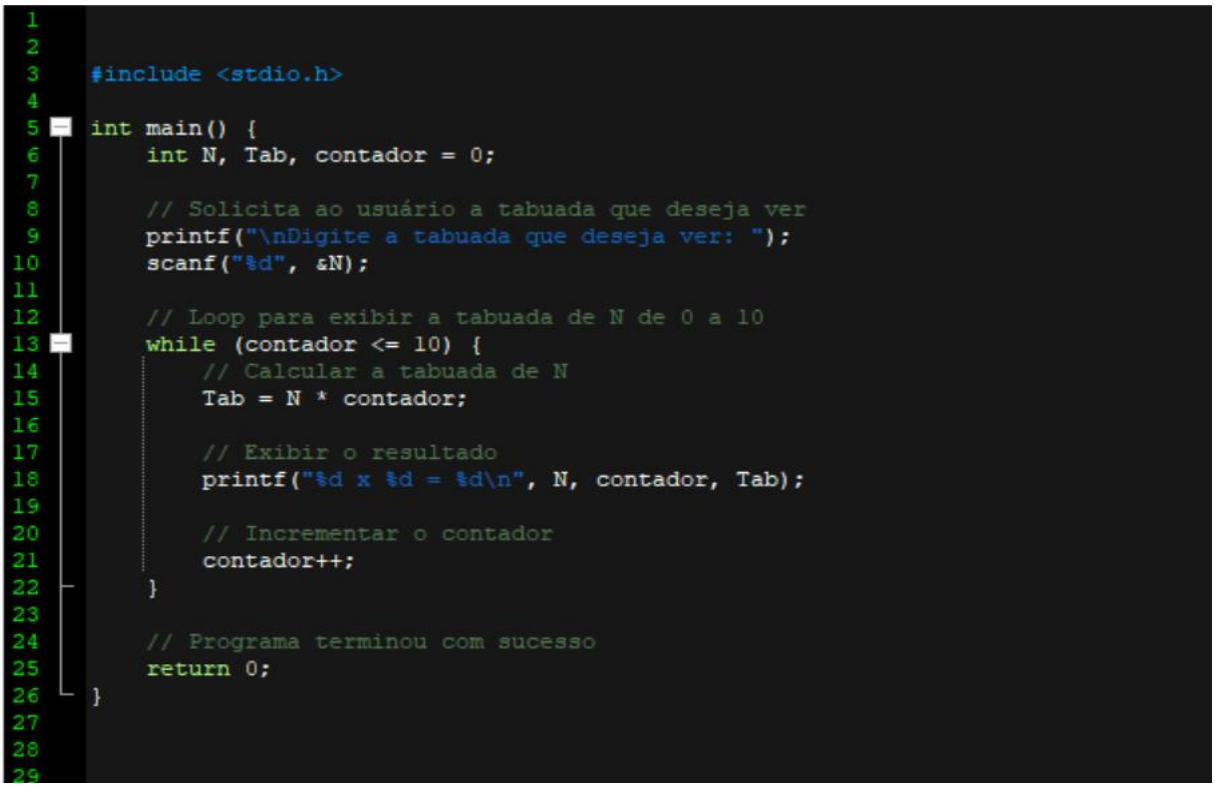
Fonte: Elaborada pela autora, 2024

3.4.4 Atividade com while

Escreva um programa em C que solicite ao usuário um número inteiro N e exiba a tabuada de N de 0 a 10. Para cada número de 0 a 10, o programa deve calcular e exibir a multiplicação de N pelo número atual, no formato "N x contador = resultado".

A resolução da atividade é apresentada na Figura 3.5.

Figura 3.5 - Atividade com While



```c


#include <stdio.h>

int main() {
    int N, Tab, contador = 0;

    // Solicita ao usuário a tabuada que deseja ver
    printf("\nDigite a tabuada que deseja ver: ");
    scanf("%d", &N);

    // Loop para exibir a tabuada de N de 0 a 10
    while (contador <= 10) {
        // Calcular a tabuada de N
        Tab = N * contador;

        // Exibir o resultado
        printf("%d x %d = %d\n", N, contador, Tab);

        // Incrementar o contador
        contador++;
    }

    // Programa terminou com sucesso
    return 0;
}


```

Fonte: Elaborada pela autora, 2024

3.4.5 Atividade com do-while

Desenvolva um programa que exiba a tabuada de um número escolhido pelo usuário.

A resolução da atividade é apresentada na Figura 3.6.

Figura 3.6 - Atividade com Do-While

```c
#include <stdio.h>

int main() {
    int numero;
    int contador = 0;

    // Solicita ao usuário que insira um número para a tabuada
    printf("Digite um número para calcular a tabuada: ");
    scanf("%d", &numero);

    // Loop do-while para calcular a tabuada do número de 0 a 10
    do {
        int resultado = numero * contador;
        printf("%d x %d = %d\n", numero, contador, resultado);
        contador++;
    } while (contador <= 10);

    return 0;
}
```

Fonte: Elaborada pela autora, 2024

3.5 ESTUDO DE MATRIZES, VETORES E STRUCTS

Durante o semestre, foi iniciado o estudo de estruturas de dados, abordando conceitos como vetores, matrizes e structs. Vetores e matrizes são estruturas que armazenam um número fixo de elementos do mesmo tipo de forma sequencial, permitindo acesso através de índices. Estas estruturas são amplamente usadas na programação para gerenciar conjuntos de dados relacionados, como listas de números, arrays multidimensionais e tabelas de informações. As structs possibilitam o agrupamento de variáveis de diferentes tipos em uma única estrutura, o que facilita a organização e o manuseio de dados complexos. Eles são particularmente úteis para modelar entidades com múltiplos atributos, como registros de alunos ou informações de produtos, contribuindo para a clareza e a organização do código. Para consolidar o entendimento desses conceitos, foram realizadas atividades práticas, exemplificadas a seguir.

3.5.1 Atividade com vetor

Faça um programa que leia um vetor de 10 números inteiros e calcule e mostre o maior e o menor elemento do vetor.

A resolução da atividade é apresentada na Figura 3.7.

Figura 3.7 - Atividade com Vetor

```c
#include <stdio.h>

int main() {
    int vetor[10];
    int maior, menor;
    int i;

    // Solicita ao usuário que insira 10 números inteiros
    printf("Digite 10 numeros inteiros:\n");
    for (i = 0; i < 10; i++) {
        printf("Numero %d: ", i + 1);
        scanf("%d", &vetor[i]);
    }

    // Inicializa os valores de maior e menor com o primeiro elemento do vetor
    maior = vetor[0];
    menor = vetor[0];

    // Percorre o vetor para encontrar o maior e menor valor
    for (i = 1; i < 10; i++) {
        if (vetor[i] > maior) {
            maior = vetor[i];
        }
        if (vetor[i] < menor) {
            menor = vetor[i];
        }
    }

    // Exibe o maior e menor valor
    printf("\nMaior elemento do vetor : %d\n", maior);
    printf("Menor elemento do vetor: %d\n", menor);

    return 0;
}
```

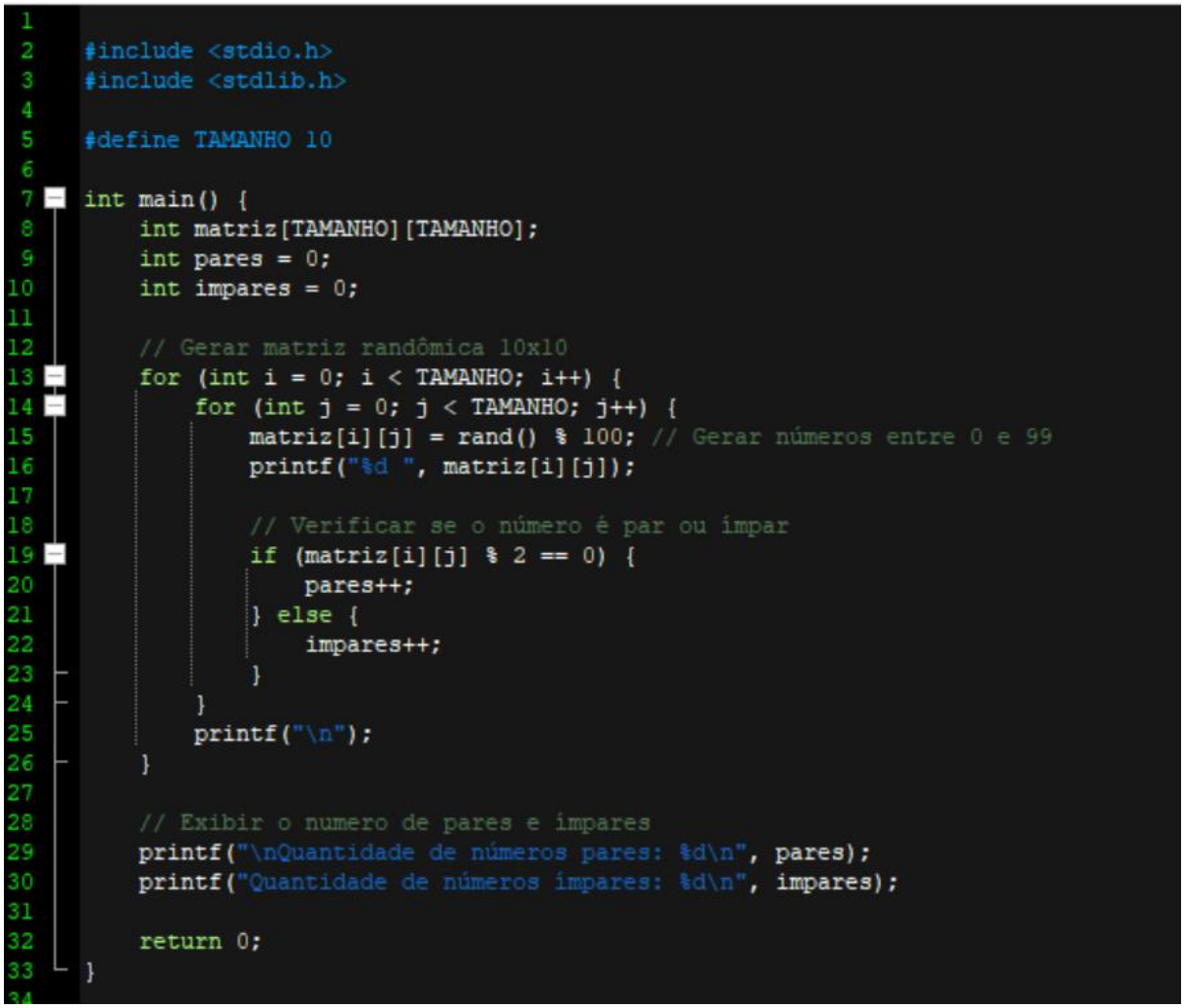
Fonte: Elaborada pela autora, 2024

3.5.2 Atividade com matriz

Faça um programa que monte uma matriz randômica A10x10 e verifique quantos números pares e quantos ímpares existem na matriz.

A resolução da atividade é apresentada na Figura 3.8.

Figura 3.8 - Atividade com Matriz



```c

#include <stdio.h>
#include <stdlib.h>

#define TAMANHO 10

int main() {
    int matriz[TAMANHO][TAMANHO];
    int pares = 0;
    int impares = 0;

    // Gerar matriz randômica 10x10
    for (int i = 0; i < TAMANHO; i++) {
        for (int j = 0; j < TAMANHO; j++) {
            matriz[i][j] = rand() % 100; // Gerar números entre 0 e 99
            printf("%d ", matriz[i][j]);

            // Verificar se o número é par ou ímpar
            if (matriz[i][j] % 2 == 0) {
                pares++;
            } else {
                impares++;
            }
        }
        printf("\n");
    }

    // Exibir o numero de pares e ímpares
    printf("\nQuantidade de números pares: %d\n", pares);
    printf("Quantidade de números ímpares: %d\n", impares);

    return 0;
}

```

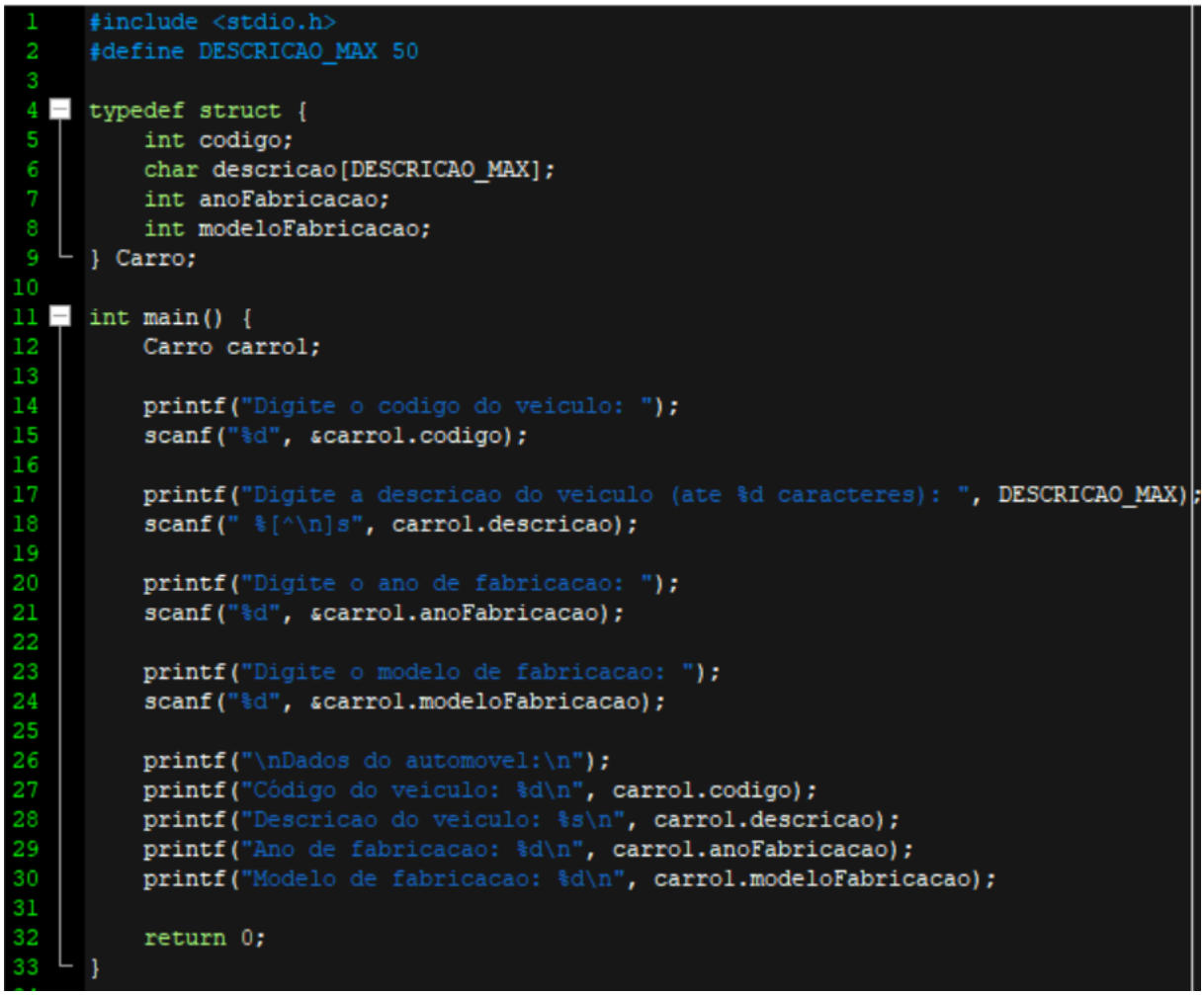
Fonte: Elaborada pela autora, 2024

3.5.3 Atividade com structs

Elabore um programa que armazene os dados relativos à compra de um automóvel, tais como: código do veículo, descrição, ano e modelo de fabricação. Utilize Structs para armazenar estes valores.

A resolução da atividade é apresentada na Figura 3.9.

Figura 3.9 - Atividade com Structs



```c
#include <stdio.h>
#define DESCRICAO_MAX 50

typedef struct {
    int codigo;
    char descricao[DESCRICAO_MAX];
    int anoFabricacao;
    int modeloFabricacao;
} Carro;

int main() {
    Carro carro1;

    printf("Digite o codigo do veiculo: ");
    scanf("%d", &carro1.codigo);

    printf("Digite a descricao do veiculo (ate %d caracteres): ", DESCRICAO_MAX);
    scanf(" %[^\n]s", carro1.descricao);

    printf("Digite o ano de fabricacao: ");
    scanf("%d", &carro1.anoFabricacao);

    printf("Digite o modelo de fabricacao: ");
    scanf("%d", &carro1.modeloFabricacao);

    printf("\nDados do automovel:\n");
    printf("Código do veiculo: %d\n", carro1.codigo);
    printf("Descricao do veiculo: %s\n", carro1.descricao);
    printf("Ano de fabricacao: %d\n", carro1.anoFabricacao);
    printf("Modelo de fabricacao: %d\n", carro1.modeloFabricacao);

    return 0;
}
```

Fonte: Elaborada pela autora, 2024

3.6 CONCLUSÃO

Ao explorar estruturas de controle, funções e estruturas de dados em C, é possível adquirir habilidades práticas fundamentais para resolver problemas de programação com eficiência. A utilização de IDEs como o Dev C++ proporciona uma experiência de desenvolvimento integrada que facilita a aplicação prática dos conceitos aprendidos. As atividades práticas, como trabalhar com if, if-else, for, while, do-while, vetores, matrizes e structs, consolidaram os conhecimentos e aprimoraram as habilidades de programação.

4 ESTRUTURA DE DADOS

As estruturas de dados desempenham um papel fundamental na representação e manipulação eficiente de dados e possibilitam a implementação de algoritmos eficazes. Por exemplo, ao organizar dados em estruturas como arrays, listas ligadas, pilhas, filas, árvores e grafos, os programadores podem otimizar o desempenho de seus algoritmos para uma ampla gama de problemas computacionais.

Conforme enfatizado por Szwarcfiter e Markenzon (2010, p.01): “O estudo de estruturas de dados não pode ser desvinculado de seus aspectos algorítmicos.”. Esta conexão é essencial para compreender como escolher a estrutura de dados mais adequada para resolver um determinado problema. Por exemplo, enquanto uma lista ligada pode ser ideal para operações de inserção e remoção frequentes, uma árvore binária de busca pode ser mais eficiente para operações de busca em conjuntos de dados ordenados.

Cada estrutura de dados tem suas próprias vantagens e desvantagens, e a escolha correta depende das características específicas do problema em questão. Portanto, um conhecimento abrangente das diferentes estruturas de dados e de seus algoritmos associados é essencial para desenvolver soluções otimizadas e eficazes em computação.

Durante o terceiro semestre do curso ADS, os alunos foram apresentados à disciplina de Estrutura de Dados, cujo propósito era dotá-los de habilidades práticas na aplicação de conceitos fundamentais. Um dos projetos emblemáticos desse período foi o desenvolvimento de um sistema para Livraria, exigindo uma compreensão aprofundada de estruturas de dados e algoritmos.

Durante o curso, os estudantes foram imersos em conceitos fundamentais de programação em C, que incluem manipulação de listas, uso de ponteiros, aplicação de funções para manipulação de strings e gerenciamento de arquivos. Esses conceitos não apenas formam a base teórica da disciplina, mas também foram habilmente empregados na execução do projeto da Livraria. Por exemplo, a utilização de listas possibilitou uma organização eficiente dos livros disponíveis, enquanto os ponteiros desempenharam um papel importante na gestão dinâmica da memória. Além disso, as funções para manipulação de strings foram essenciais para lidar com informações relativas aos livros, e a manipulação de arquivos viabilizou a persistência dos dados da livraria.

4.1 SISTEMA DE GERENCIAMENTO DE LIVRARIA

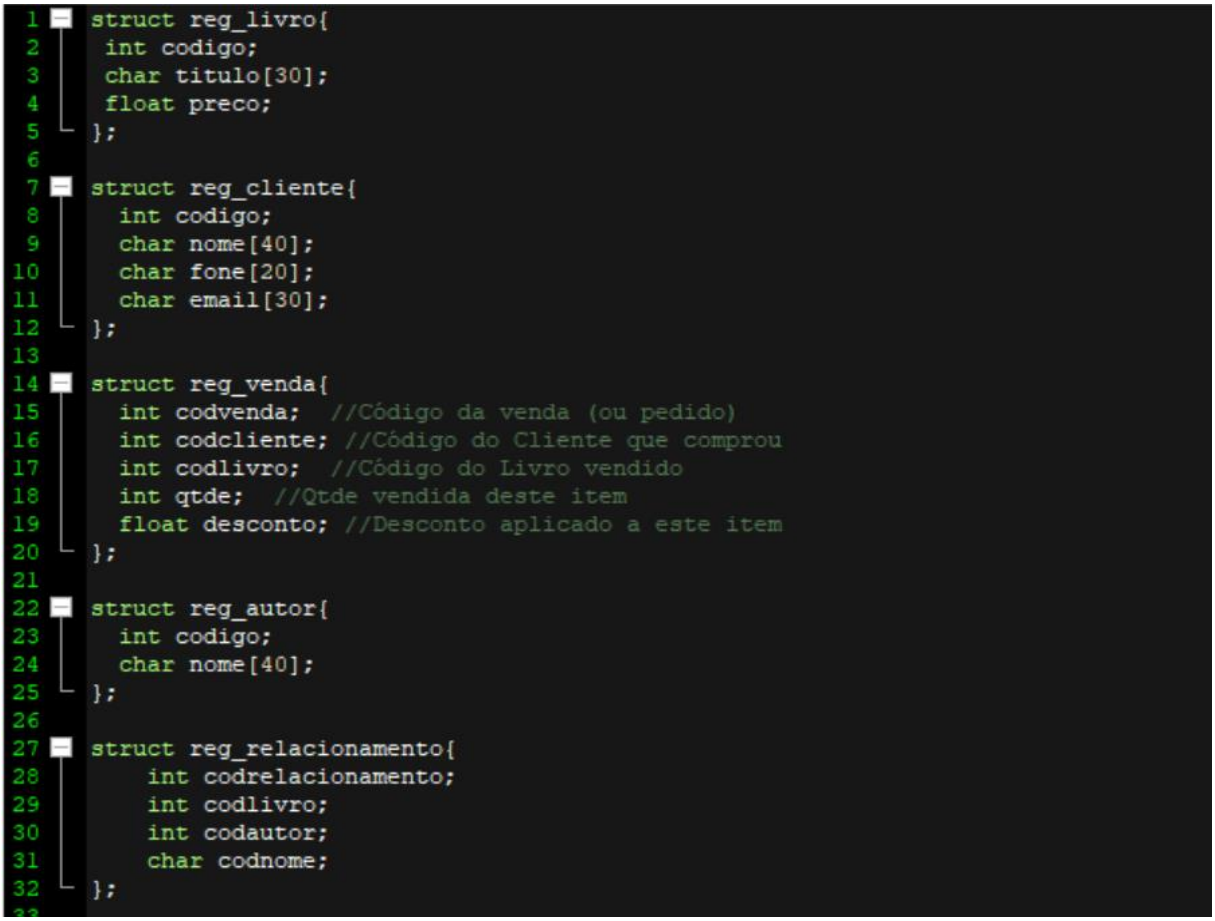
O projeto do sistema de gerenciamento de livraria foi desenvolvido para automatizar as operações de uma loja de livros. Este sistema possibilita a inclusão, listagem, consulta, atualização e remoção de registros relacionados a livros, clientes, autores e vendas.

4.1.1 Structs

As structs são usadas para definir os tipos de dados que representam diversas entidades dentro do sistema de gerenciamento de livraria.

A figura 4.1 ilustra as structs criadas para armazenar informações sobre clientes, vendas, autores e a relação entre livros e autores.

Figura 4.1 – Structs



```
struct reg_livro{
 int codigo;
 char titulo[30];
 float preco;
};

struct reg_cliente{
  int codigo;
  char nome[40];
  char fone[20];
  char email[30];
};

struct reg_venda{
  int codvenda;  //Código da venda (ou pedido)
  int codcliente; //Código do Cliente que comprou
  int codlivro;  //Código do Livro vendido
  int qtde;  //Qtde vendida deste item
  float desconto; //Desconto aplicado a este item
};

struct reg_autor{
  int codigo;
  char nome[40];
};

struct reg_relacionamento{
    int codrelacionamento;
    int codlivro;
    int codautor;
    char codnome;
};
```

Fonte: Elaborada pela autora, 2024

struct reg_livro: Armazena informações como código do livro, título e preço.

struct reg_cliente: Armazena informações como código do cliente, nome, número de telefone e endereço de e-mail.

struct reg_venda: Armazena informações como código da venda, código do cliente, código do livro vendido, quantidade vendida deste item e desconto aplicado.

struct reg_autor: Armazena informações como código do autor e nome do autor.

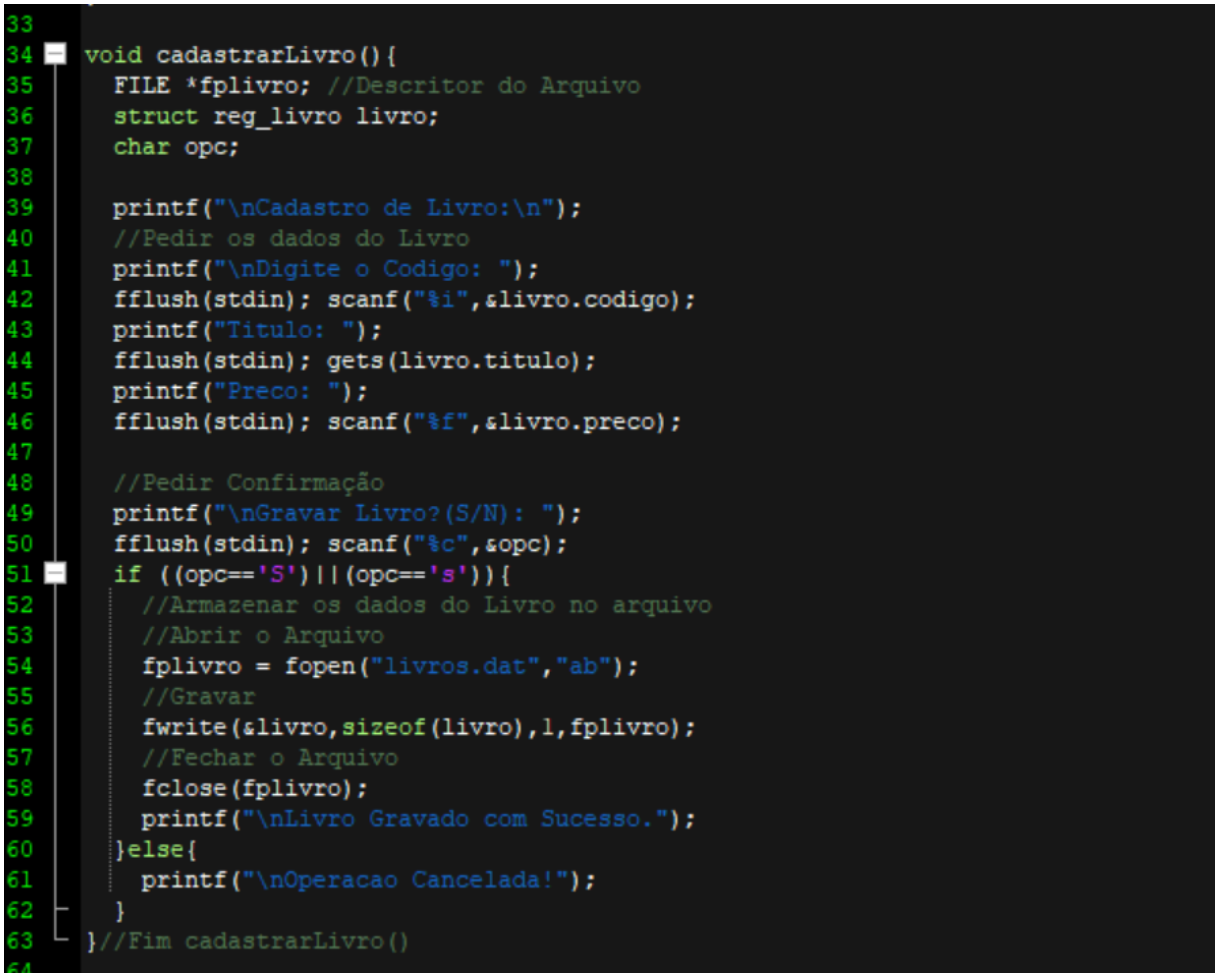
struct reg_relacionamento: Esta struct representa o relacionamento entre livros e autores. Ela armazena informações como código do relacionamento, código do livro, código do autor e um código de nome.

Com essas definições, o programa pode manipular essas entidades e realizar operações como adição, remoção, atualização e consulta de informações relacionadas a livros, clientes, vendas, autores e seus relacionamentos.

4.1.2 Cadastrar Livro

A figura 4.2 apresenta a parte do programa que implementa a função cadastrarLivro(), que permite ao usuário inserir informações de um livro e armazená-las em um arquivo binário chamado "livros.dat".

Figura 4.2 - Cadastrar Livro



```
33
34  void cadastrarLivro(){
35    FILE *fplivro; //Descritor do Arquivo
36    struct reg_livro livro;
37    char opc;
38
39    printf("\nCadastro de Livro:\n");
40    //Pedir os dados do Livro
41    printf("\nDigite o Codigo: ");
42    fflush(stdin); scanf("%i",&livro.codigo);
43    printf("Titulo: ");
44    fflush(stdin); gets(livro.titulo);
45    printf("Preco: ");
46    fflush(stdin); scanf("%f",&livro.preco);
47
48    //Pedir Confirmação
49    printf("\nGravar Livro?(S/N): ");
50    fflush(stdin); scanf("%c",&opc);
51    if ((opc=='S')||(opc=='s')){
52      //Armazenar os dados do Livro no arquivo
53      //Abrir o Arquivo
54      fplivro = fopen("livros.dat","ab");
55      //Gravar
56      fwrite(&livro,sizeof(livro),1,fplivro);
57      //Fechar o Arquivo
58      fclose(fplivro);
59      printf("\nLivro Gravado com Sucesso.");
60    }else{
61      printf("\nOperacao Cancelada!");
62    }
63  }//Fim cadastrarLivro()
```

Fonte: Elaborada pela autora, 2024

Declarações e inicializações: A função declara um ponteiro de arquivo fplivro para manipular o arquivo de livros e uma variável opc para armazenar a opção do usuário sobre a confirmação do cadastro.

Entrada de dados: O usuário é solicitado a inserir os dados do livro, como código, título e preço.

Confirmação do registro: O sistema pede ao usuário para confirmar a gravação dos dados do livro no arquivo. Se o usuário confirmar (digitando 'S' ou 's'), o processo continua; caso contrário, a operação é cancelada.

Gravação no arquivo: Se a confirmação for positiva, o arquivo "livros.dat" é aberto em modo de escrita binária ("ab" - append binary). Em seguida, os dados do livro são escritos no arquivo usando a função fwrite(). Depois que o registro é salvo, o arquivo é fechado.

Feedback ao usuário: Se o livro for gravado com sucesso, uma mensagem indicando isso é exibida. Caso contrário, se o usuário optar por cancelar a operação, uma mensagem informando sobre o cancelamento é exibida.

Essencialmente, esta função encapsula o processo de captura de dados de um livro, sua gravação em um arquivo binário e fornece feedback adequado ao usuário sobre o resultado da operação.

4.1.3 Listar Livros

A figura 4.3 apresenta a função listarLivros() que a finalidade de exibir um relatório contendo informações sobre todos os livros armazenados no arquivo de livros, juntamente com os códigos dos autores responsáveis por cada livro.

Figura 4.3 - Listar livros

```
64
65  void listarLivros(){
66    FILE *fplivro;
67    struct reg_livro livro;
68    FILE *fpautor;
69    struct reg_autor autor;
70    FILE *fprelacionamento;
71    struct reg_relacionamento relacionamento;
72
73    //Abrir o Arquivo
74    if ((fplivro = fopen(LIVROS,"rb"))==NULL ){
75      printf("\nErro ao abrir o Arquivo %s %s",LIVROS, RELACIONAMENTO);
76      return;  //Volta para o Programa Principal
77    }
78
79    fpautor = fopen(AUTORES,"rb");
80    fprelacionamento = fopen (RELACIONAMENTO, "rb");
81
82    printf("\n####  Relatorio de Todos os Livros  ####");
83    printf("\nCodigo Titulo                          Preco   Cod.Autor   Autor");
84    //Ler registro por registro e mostrar na tela
85    while (fread(&livro,sizeof(livro),1,fplivro)==1){
86      fread(&relacionamento, sizeof(relacionamento),1,fprelacionamento);
87      fread (&autor,sizeof(autor),1,fpautor);
88      if(relacionamento.codautor==autor.codigo){
89        printf("\n%-6i %-30s %5.2f       %-6i  %s",
90          livro.codigo,livro.titulo,livro.preco, autor.codigo);
91      }
92    }
93    //Fechar o Arquivo
94    fclose(fplivro);
95    fclose(fprelacionamento);
96    fclose(fpautor);
97  }//Fim listarLivros()
```

Fonte: Elaborada pela autora, 2024

Abertura dos arquivos: A função começa abrindo três arquivos: um para os livros (fplivro), outro para os autores (fpautor), e um terceiro para o relacionamento entre livros e autores (fprelacionamento).

Verificação de erros na abertura do arquivo: O código verifica se o arquivo de livros foi aberto com sucesso. Se não, exibe uma mensagem de erro e retorna imediatamente. Essa verificação de erro é importante para garantir a consistência e robustez do programa.

Loop de leitura e exibição de registros: A função entra em um loop enquanto há registros para serem lidos no arquivo de livros (fplivro). Durante cada iteração do loop, um registro de livro é lido do arquivo de livros. Em seguida, um registro de relacionamento é lido do arquivo de relacionamento (fprelacionamento) e um registro de autor é lido do arquivo de autores (fpautor). Se o código do autor no registro de relacionamento corresponder ao código do autor no registro atual, as informações sobre o livro e o autor são exibidas na tela.

Formatação da saída: As informações sobre os livros (código, título e preço) e os autores (código) são exibidas na tela em um formato tabular para facilitar a leitura.

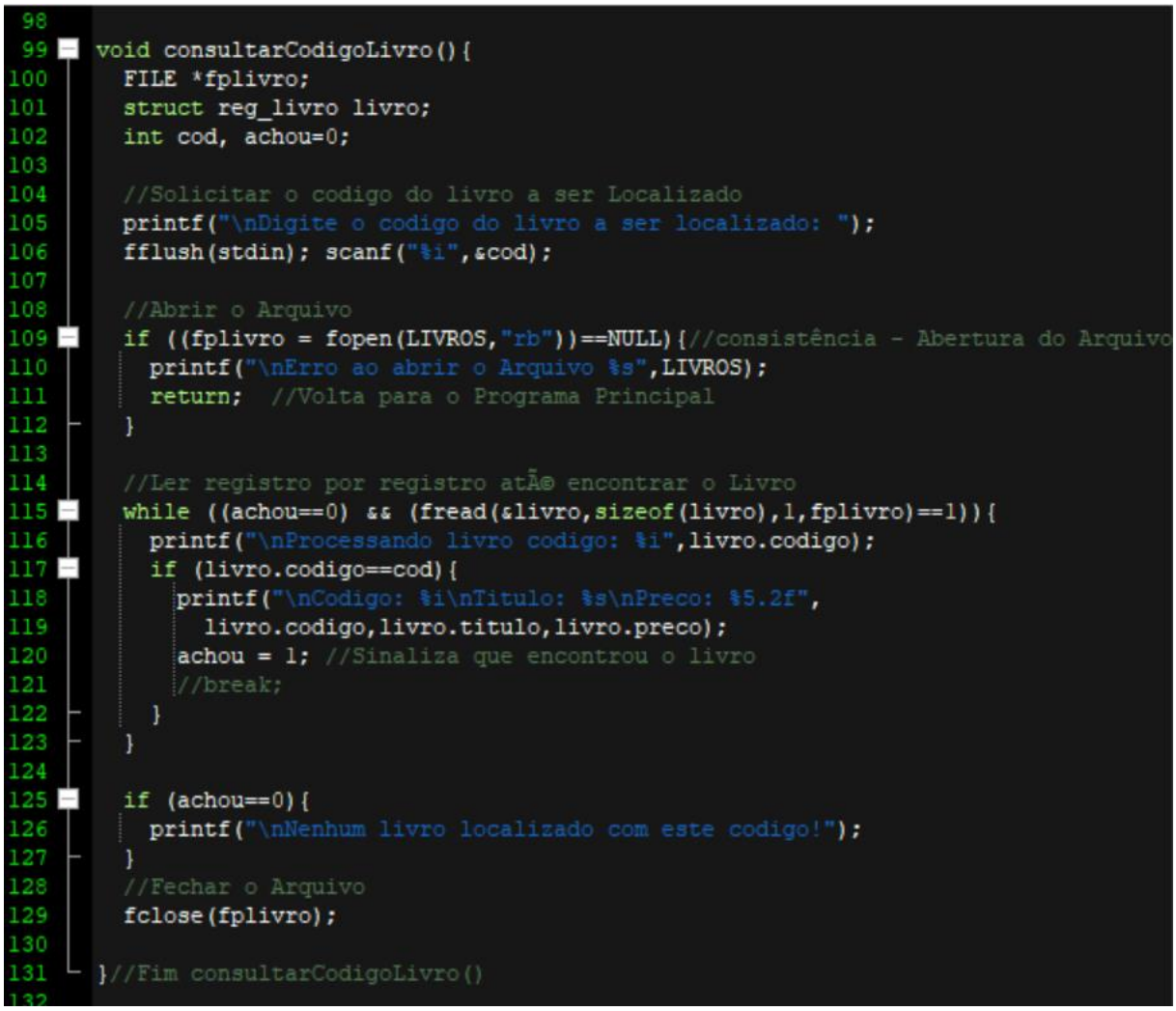
Fechamento dos arquivos: Após o término do loop, os arquivos são fechados para liberar os recursos do sistema.

Em suma, a função exemplifica boas práticas de programação para manipulação de arquivos, resultando em um código robusto e eficiente.

4.1.4 Consultar código de livros

A figura 4.4 apresenta a função consultarCodigoLivro() que tem como objetivo buscar e exibir informações de um livro a partir do seu código, dentro de um arquivo binário.

Figura 4.4 - Consultar código livro



```
98
99  void consultarCodigoLivro(){
100   FILE *fplivro;
101   struct reg_livro livro;
102   int cod, achou=0;
103
104   //Solicitar o codigo do livro a ser Localizado
105   printf("\nDigite o codigo do livro a ser localizado: ");
106   fflush(stdin); scanf("%i",&cod);
107
108   //Abrir o Arquivo
109   if ((fplivro = fopen(LIVROS,"rb"))==NULL){//consistência - Abertura do Arquivo
110     printf("\nErro ao abrir o Arquivo %s",LIVROS);
111     return;  //Volta para o Programa Principal
112   }
113
114   //Ler registro por registro atÃ© encontrar o Livro
115   while ((achou==0) && (fread(&livro,sizeof(livro),1,fplivro)==1)){
116     printf("\nProcessando livro codigo: %i",livro.codigo);
117     if (livro.codigo==cod){
118       printf("\nCodigo: %i\nTitulo: %s\nPreco: %5.2f",
119         livro.codigo,livro.titulo,livro.preco);
120       achou = 1; //Sinaliza que encontrou o livro
121       //break;
122     }
123   }
124
125   if (achou==0){
126     printf("\nNenhum livro localizado com este codigo!");
127   }
128   //Fechar o Arquivo
129   fclose(fplivro);
130
131 }//Fim consultarCodigoLivro()
```

Fonte: Elaborada pela autora, 2024

Declaração de variáveis: Começa declarando um ponteiro de arquivo fplivro, uma estrutura reg_livro chamada livro, e variáveis cod e achou para armazenar o código do livro a ser consultado e indicar se o livro foi encontrado, respectivamente.

Solicitação do código do livro: O sistema pede ao usuário que informe o código do livro que deseja encontrar.

Abertura do arquivo: Tenta abrir o arquivo binário onde os dados dos livros estão armazenados. Se a abertura falhar, exibe uma mensagem de erro e retorna para o programa principal.

Busca no arquivo: Utiliza um loop while para ler registros do arquivo binário até encontrar o livro com o código desejado. Durante cada iteração, ele lê um registro do arquivo e verifica se o código do livro corresponde ao código fornecido pelo usuário.

Se encontrar o livro, exibe suas informações (código, título e preço), define achou como 1 e sai do loop. Se não encontrar o livro, achou permanece 0.

Exibição de resultados: Se o livro não for encontrado, exibe uma mensagem indicando isso.

Encerramento do arquivo: O arquivo é fechado assim que a operação é finalizada.

Em suma a função consultarCodigoLivro() realiza a busca por um livro no arquivo binário usando o código do livro. Caso o livro seja localizado, suas informações serão apresentadas. Se não, uma mensagem informando que o livro não foi localizado é mostrada.

4.1.5 Consultar título de livros

A figura 4.5 apresenta a função consultarTituloLivro() que permite a consulta de um livro em um arquivo binário, procurando pelo título do livro.

Figura 4.5 - Consultar título de livros

```
133
134 void consultarTituloLivro(){
135   FILE *fplivro;
136   struct reg_livro livro;
137   char tit[30];
138   int achou=0;
139
140   //Solicitar o Titulo do livro a ser Localizado
141   printf("\nDigite o Titulo do livro a ser localizado: ");
142   fflush(stdin); gets(tit);
143
144   //Abrir o Arquivo
145   if ((fplivro = fopen(LIVROS,"rb"))==NULL){//consistência-Abertura do Arquivo
146     printf("\nErro ao abrir o Arquivo %s",LIVROS);
147     return;  //Volta para o Programa Principal
148   }
149
150   //Ler registro por registro até encontrar o Livro
151   while ((achou==0) && (fread(&livro,sizeof(livro),1,fplivro)==1)){
152     if (strcmp(livro.titulo,tit)==0){
153       printf("\nCodigo: %i\nTitulo: %s\nPreco: %5.2f"
154         ,livro.codigo,livro.titulo,livro.preco);
155       achou = 1; //Sinaliza que encontrou o livro
156       //break;
157     }
158   }
159
160   if (achou==0){
161     printf("\nNenhum livro localizado com este codigo!");
162   }
163   //Fechar o Arquivo
164   fclose(fplivro);
```

Fonte: Elaborada pela autora, 2024

Declaração de variáveis: fplivro: ponteiro para o arquivo que contém os registros dos livros, tit: array de caracteres para armazenar o título do livro que o usuário deseja procurar e achou: variável de controle para indicar se o livro foi encontrado ou não.

Entrada do título do livro: Solicita ao usuário que insira o título do livro a ser localizado.

Abertura do arquivo: O arquivo contendo os registros dos livros é aberto no modo de leitura binária ("rb"). Se a abertura falhar, exibe uma mensagem de erro e retorna para o programa principal.

Leitura e comparação de registros: Um loop while lê os registros do arquivo um por um utilizando fread, que lê blocos de dados binários.

A função fread lê um registro de cada vez e compara o título do livro lido (livro.titulo) com o título fornecido pelo usuário (tit) usando strcmp.

Se os títulos coincidirem, as informações do livro (codigo, titulo, preco) são exibidas e a variável achou é setada para 1, indicando que o livro foi encontrado.

Tratamento de livro não encontrado: Após o loop, se achou ainda for 0, significa que nenhum livro com o título fornecido foi encontrado, e uma mensagem é exibida.

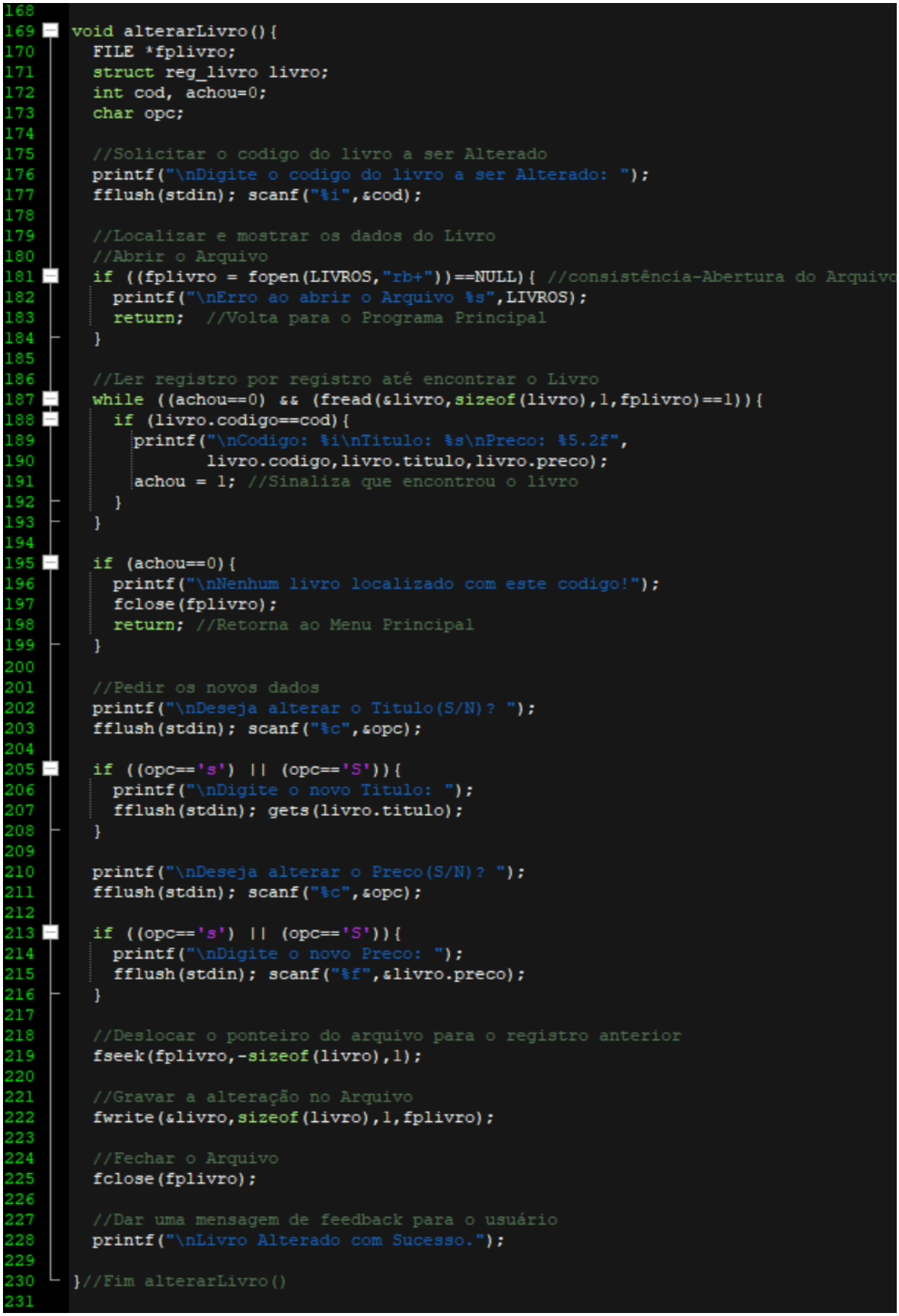
Encerramento do arquivo: O arquivo é fechado assim que a operação é finalizada.

Em resumo, a função consultarTituloLivro() é utilizada para buscar e exibir informações de um livro baseado no título fornecido pelo usuário, lendo registros de um arquivo binário de livros.

4.1.6 Alterar livros

A figura 4.6 apresenta a função alterarLivro() que tem o propósito de modificar os dados de um livro específico em um arquivo binário de registros de livros, usando o código do livro para localizar o registro a ser alterado.

Figura 4.6 - Alterar Livro



```
168
169 void alterarLivro(){
170   FILE *fplivro;
171   struct reg_livro livro;
172   int cod, achou=0;
173   char opc;
174
175   //Solicitar o codigo do livro a ser Alterado
176   printf("\nDigite o codigo do livro a ser Alterado: ");
177   fflush(stdin); scanf("%i",&cod);
178
179   //Localizar e mostrar os dados do Livro
180   //Abrir o Arquivo
181   if ((fplivro = fopen(LIVROS,"rb+"))==NULL){ //consistência-Abertura do Arquivo
182     printf("\nErro ao abrir o Arquivo %s",LIVROS);
183     return;  //Volta para o Programa Principal
184   }
185
186   //Ler registro por registro até encontrar o Livro
187   while ((achou==0) && (fread(&livro,sizeof(livro),1,fplivro)==1)){
188     if (livro.codigo==cod){
189       printf("\nCodigo: %i\nTitulo: %s\nPreco: %5.2f",
190              livro.codigo,livro.titulo,livro.preco);
191       achou = 1; //Sinaliza que encontrou o livro
192     }
193   }
194
195   if (achou==0){
196     printf("\nNenhum livro localizado com este codigo!");
197     fclose(fplivro);
198     return; //Retorna ao Menu Principal
199   }
200
201   //Pedir os novos dados
202   printf("\nDeseja alterar o Titulo(S/N)? ");
203   fflush(stdin); scanf("%c",&opc);
204
205   if ((opc=='s') || (opc=='S')){
206     printf("\nDigite o novo Titulo: ");
207     fflush(stdin); gets(livro.titulo);
208   }
209
210   printf("\nDeseja alterar o Preco(S/N)? ");
211   fflush(stdin); scanf("%c",&opc);
212
213   if ((opc=='s') || (opc=='S')){
214     printf("\nDigite o novo Preco: ");
215     fflush(stdin); scanf("%f",&livro.preco);
216   }
217
218   //Deslocar o ponteiro do arquivo para o registro anterior
219   fseek(fplivro,-sizeof(livro),1);
220
221   //Gravar a alteração no Arquivo
222   fwrite(&livro,sizeof(livro),1,fplivro);
223
224   //Fechar o Arquivo
225   fclose(fplivro);
226
227   //Dar uma mensagem de feedback para o usuário
228   printf("\nLivro Alterado com Sucesso.");
229
230 }//Fim alterarLivro()
231
```

Fonte: Elaborada pela autora, 2024

Declaração de Variáveis: fplivro: ponteiro para o arquivo que contém os registros dos livros, livro: estrutura que representa um registro de livro, cod: variável para armazenar o código do livro que o usuário deseja alterar, achou: variável de controle para indicar se o livro foi encontrado ou não e opc: variável para armazenar a escolha do usuário para alterar título ou preço.

Entrada do Código do Livro: Solicita ao usuário que insira o código do livro a ser alterado.

Abertura do Arquivo: O arquivo contendo os registros dos livros é aberto em modo de leitura e escrita binária ("rb+"). Se o arquivo não conseguir ser aberto, uma mensagem de erro será exibida e a função será encerrada**.**

Leitura e comparação de registros: Um loop while lê os registros do arquivo um por um utilizando fread. Cada registro é comparado com o código fornecido pelo usuário (cod). Se o registro é encontrado, as informações do livro são exibidas e achou é setado para 1.

Tratamento de Livro Não Encontrado: Se achou ainda for 0 após o loop, significa que nenhum livro com o código fornecido foi encontrado e uma mensagem de erro é exibida, o arquivo é fechado e a função retorna ao ponto de chamada.

Alteração dos Dados do Livro: O sistema pergunta ao usuário se deseja alterar o título do livro. Se a resposta for afirmativa ('s' ou 'S'), o novo título é solicitado e armazenado.

Alteração do Preço: Pergunta ao usuário se deseja alterar o preço do livro. Se a resposta for afirmativa ('s' ou 'S'), o novo preço é solicitado e armazenado.

Atualização do Registro no Arquivo: a função fseek reposiciona o ponteiro para o início do registro atual, permitindo sua substituição, enquanto fwrite escreve os dados modificados de volta no arquivo.

Fechamento do arquivo e feedback ao usuário: O arquivo é fechado para liberar os recursos e uma mensagem de confirmação é exibida ao usuário indicando que o livro foi alterado com sucesso.

Em resumo a função alterarLivro() é utilizada para localizar e alterar os dados de um livro específico em um arquivo binário.

4.1.7 Excluir livros

A figura 4.7 apresenta a função excluirLivro() que é utilizada para remover um registro de livro específico de um arquivo binário. A exclusão é realizada criando um novo arquivo temporário que contém todos os registros, exceto aquele que deve ser excluído, e depois substituindo o arquivo original pelo novo arquivo.

Figura 4.7 - Excluir livro

```
232
233 void excluirLivro(){
234   FILE *fplivro, *fplivrosnew;
235   struct reg_livro livro;
236   int cod, achou=0;
237   char opc;
238
239   //Solicitar o codigo do livro a ser Excluido
240   printf("\nDigite o codigo do livro a ser Excluido: ");
241   fflush(stdin); scanf("%i",&cod);
242
243   //Localizar e mostrar os dados do Livro
244   //Abrir o Arquivo
245   if ((fplivro = fopen(LIVROS,"rb"))==NULL){//consistência-Abertura do Arquivo
246     printf("\nErro ao abrir o Arquivo %s",LIVROS);
247     return;  //Volta para o Programa Principal
248   }
249
250   //Ler registro por registro até encontrar o Livro
251   while ((achou==0) && (fread(&livro,sizeof(livro),1,fplivro)==1)){
252     if (livro.codigo==cod){
253       printf("\nCodigo: %i\nTitulo: %s\nPreco: %5.2f",
254              livro.codigo,livro.titulo,livro.preco);
255       achou = 1; //Sinaliza que encontrou o livro
256     }
257   }
258
259   fclose(fplivro);
260
261   if (achou==0){
262     printf("\nNenhum livro localizado com este codigo!");
263     return; //Retorna ao Menu Principal
264   }
265
266   //Pedir Confirmação
267   printf("\nConfirma Exclusao(S/N)? ");
268   fflush(stdin); scanf("%c",&opc);
269
270   if ((opc=='n') || (opc=='N')){
271     printf("\nExclusao Cancelada!");
272     return; //Retorna ao Menu Principal
273   }
274
275   //Exclusao Confirmada - Excluir o Livro
276   fplivro = fopen(LIVROS,"rb");
277   fplivrosnew = fopen("livrosnew.dat","wb");
278
279   //Copiar todos os livros, menos o livro a ser excluido, para um
280     //arquivo temporário - livrosnew.dat
281   while(fread(&livro,sizeof(livro),1,fplivro)==1){
282     if (livro.codigo!=cod){ //Se o livro lido NÃfO for o livro sendo excluido
283       fwrite(&livro,sizeof(livro),1,fplivrosnew);//Grava no temporÃ¡rio
284      }
285   }
286   fclose(fplivro);
287   fclose(fplivrosnew);
288
289   //Excluir o arquivo antigo
290   system("del livros.dat");
291
292   //Renomear o arquivo temporÃ¡rio para livros.dat
293   system("ren livrosnew.dat livros.dat");
294
295   //Dar um msg de feedback para o usuÃ¡rio
296   printf("\nLivro Excluido com Sucesso.");
297 }//Fim excluirLivro()
```

Fonte: Elaborada pela autora, 2024

Declaração das Variáveis: fplivro é um ponteiro para o arquivo principal contendo os livros, enquanto fplivrosnew é um ponteiro para o arquivo temporário que manterá os registros não excluídos. Livro é uma estrutura usada para representar um registro de livro, cod armazena o código do livro que o usuário pretende remover, achou é uma variável de controle que indica se o livro foi encontrado e opc guarda a escolha do usuário para confirmar ou cancelar a exclusão.

Solicitação do código do livro: o sistema solicita ao usuário o código do livro a ser excluído.

Abertura do arquivo original: O arquivo contendo os livros é acessado no modo de leitura binária ("rb").Se o arquivo não puder ser aberto, uma mensagem de erro é exibida e a função retorna ao menu principal.

Leitura e verificação dos registros: O arquivo é lido registro por registro até encontrar o livro com o código fornecido. Se o livro é encontrado, suas informações são exibidas e achou é setado para 1. Se o livro não é encontrado, uma mensagem de aviso é exibida, o arquivo é fechado, e a função retorna ao menu principal.

Confirmação da Exclusão: O programa solicita ao usuário a confirmação ou cancelamento da exclusão do livro.

Ações de Exclusão: Se a exclusão for cancelada, uma mensagem é exibida e a função retorna ao meu principal.

Exclusão Confirmada: Dois arquivos são abertos, o arquivo original para leitura e um novo arquivo temporário para escrita.

Cópia dos livros para o novo arquivo: Todos os livros, exceto o livro a ser excluído, são copiados do arquivo original para o novo arquivo temporário. Os arquivos são fechados após a operação de cópia.

Exclusão do arquivo antigo e renomeação do novo arquivo: O arquivo antigo é excluído e o novo arquivo temporário é renomeado para substituir o arquivo original.

Mensagem ao Usuário: Uma notificação é apresentada para informar que o livro foi removido com sucesso.

Em suma a função excluirLivro() possui funcionalidades básicas para excluir um livro de um arquivo binário.

4.1.8 Cadastrar Cliente

A figura 4.8 apresenta a função cadastrarCliente()tem como objetivo cadastrar um novo cliente em um arquivo binário chamado Cliente.

Figura 4.8 - Cadastrar cliente

```
298
299 void cadastrarCliente(){
300  struct reg_cliente cliente;
301  FILE *fpcliente; //VariÃ¡vel para manipular o Arquivo de Clientes
302  char op;
303
304  //Entrada dos dados do Livro
305  printf("\nCadastro de Cliente: ");
306  printf("\n Codigo: ");
307  fflush(stdin); scanf("%i",&cliente.codigo);
308  printf(" Nome: ");
309  fflush(stdin); gets(cliente.nome);
310  printf(" Fone: ");
311  fflush(stdin); gets(cliente.fone);
312  printf(" Email: ");
313  fflush(stdin); gets(cliente.email);
314
315
316  printf("\nGravar ?(S/N) ");
317  fflush(stdin); scanf("%c",&op);
318  if ((op=='S') || (op=='s')){
319     //Abrir o arquivo
320     fpcliente = fopen(CLIENTES,"ab"); //Modo "ab" para acrescentar
321                                       //um novo Cliente no arquivo binário
322     fwrite(&cliente,sizeof(cliente),1,fpcliente); //Gravar no arquivo
323     printf("\nCliente Cadastrado com sucesso.");
324     fclose(fpcliente); //Fecha o arquivo
325  }else{
326     printf("\nCadastro Cancelado!");
327  }
328 }//Fim cadastrarCliente()
329
```

Fonte: Elaborada pela autora, 2024

Declarações e definições: Declaração de uma variável do tipo struct reg_cliente para armazenar os dados do cliente. Declaração de um ponteiro fpclientedo tipo FILE para manipulação do arquivo. Declaração de uma variável op do tipo char para confirmar se o usuário deseja salvar o cadastro.

Entrada de dados: O programa solicita ao usuário que insira os dados do cliente (código, nome, telefone e email).

Confirmar gravação: O programa pergunta ao usuário se deseja gravar os dados no arquivo.

Confirmação da gravação: O programa solicita ao usuário a confirmação para salvar os dados no arquivo.

Gravação no arquivo: Se o usuário confirmar a gravação, o arquivo é aberto no modo "ab" (append binary). Os dados do cliente são gravados no arquivo usando fwrite().

Fechamento do arquivo: O arquivo é fechado após a gravação. Se o usuário não confirmar a gravação, aparecerá a mensagem Cadastro Cancelado!.

Em resumo a função cadastrarCliente() é desenvolvida para inserir novos dados de clientes de forma segura e eficiente em um arquivo binário.

4.1.9 Listar Clientes

A figura 4.9 apresenta função listarClientes() que é responsável por ler e exibir todos os registros de clientes armazenados em um arquivo binário. Ela exibe os dados na tela em um formato de relatório.

Figura 4.9 - Listar Clientes

```
330 void listarClientes(){
331   FILE *fpcliente;
332   struct reg_cliente cliente;
333
334   //Abrir o Arquivo
335   if ((fpcliente = fopen(CLIENTES,"rb"))==NULL){
336     printf("\nErro na abertura do arquivo!!");
337     return;
338   }
339
340   //Ler registo por registro e mostrar na tela
341   printf("\n##### Relatorio de Todos os Clientes ######");
342   printf("\nCodigo Nome                 Fone           Email           ");
343   while (fread(&cliente,sizeof(cliente),1,fpcliente)==1){
344     printf("\n%-6i %-30s %-15s %-20s",cliente.codigo,cliente.nome,
345                                       cliente.fone,cliente.email);
346   }
347
348   //Fechar o Arquivo
349   fclose(fpcliente);
350
351 }//Fim listarClientes()
352
```

Fonte: Elaborada pela autora, 2024

Declarações: Arquivo: FILE *fpcliente é um ponteiro para manipulação do arquivo que contém os dados dos clientes e struct reg_cliente cliente é uma estrutura que representa os dados de um cliente, contendo campos como código, nome, telefone e email.

Abertura do arquivo: A função tenta abrir o arquivo CLIENTES no modo "rb" (leitura binária). Se o arquivo não puder ser aberto fopen retorna NULL e uma mensagem de erro é exibida.

Leitura e exibição dos dados: A função imprime um cabeçalho para o relatório de clientes. Em um loop while, a função usa fread para ler cada registro de cliente do arquivo, fread(&cliente, sizeof(cliente), 1, fpcliente) lê um registro de tamanho sizeof(cliente) do arquivo e armazena os dados na variável cliente. O loop continua enquanto fread retornar 1, indicando que um registro foi lido com sucesso. Para cada cliente lido, a função imprime os dados formatados, alinhando os campos com largura fixa para uma exibição ordenada.

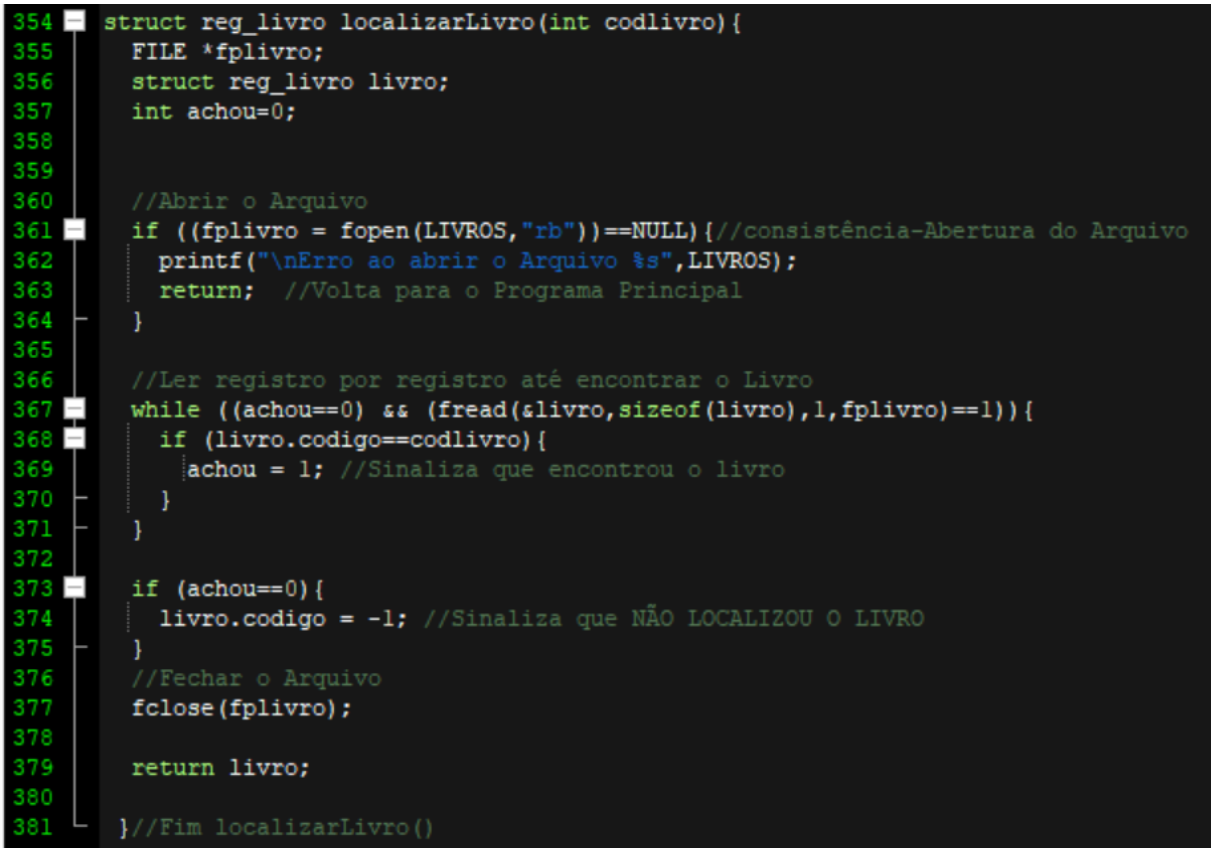
Fechamento do arquivo: Após ler todos os registros, a função fecha o arquivo usando fclose(fpcliente).

Em resumo a função listarClientes() é eficaz para seu propósito principal, ler e exibir registros de clientes de um arquivo binário.

4.1.10 Localizar clientes

A figura 4.10 A função localizarLivro() tem como objetivo localizar um livro em um arquivo binário, identificado pelo seu código, e retornar os dados do livro encontrado. Se o livro não for encontrado, a função sinaliza isso.

Figura 4.10 - Localizar livro



```
354 struct reg_livro localizarLivro(int codlivro){
355   FILE *fplivro;
356   struct reg_livro livro;
357   int achou=0;
358
359
360   //Abrir o Arquivo
361   if ((fplivro = fopen(LIVROS,"rb"))==NULL){//consistência-Abertura do Arquivo
362     printf("\nErro ao abrir o Arquivo %s",LIVROS);
363     return;  //Volta para o Programa Principal
364   }
365
366   //Ler registro por registro até encontrar o Livro
367   while ((achou==0) && (fread(&livro,sizeof(livro),1,fplivro)==1)){
368     if (livro.codigo==codlivro){
369       achou = 1; //Sinaliza que encontrou o livro
370     }
371   }
372
373   if (achou==0){
374     livro.codigo = -1; //Sinaliza que NÃO LOCALIZOU O LIVRO
375   }
376   //Fechar o Arquivo
377   fclose(fplivro);
378
379   return livro;
380
381 }//Fim localizarLivro()
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fplivro é um ponteiro para manipulação do arquivo que contém os dados dos livros, struct reg_livro livro é uma estrutura que representa os dados de um livro e achou é uma variável usada como flag para indicar se o livro foi encontrado ou não.

Abertura do arquivo: A função tenta abrir o arquivo LIVROS no modo "rb" (leitura binária). Se o arquivo não puder ser aberto fopen retorna NULL e uma mensagem de erro é exibida.

Leitura e pesquisa do livro: A função lê registros do arquivo um por um dentro de um loop while até encontrar o livro ou até que todos os registros tenham sido lidos, fread(&livro, sizeof(livro), 1, fplivro) lê um registro de tamanho sizeof(livro) do arquivo e armazena os dados na variável livro. Se o código do livro lido livro.codigo for igual ao código procurado codlivro, achou é definido como 1 para indicar que o livro foi encontrado.

Sinalização de livro não encontrado: Após o loop, se achou ainda for 0, isso indica que o livro não foi encontrado. Nesse caso, livro.codigo é definido como -1 para sinalizar que a busca falhou.

Fechamento do Arquivo: O arquivo é fechado usando fclose(fplivro) e a função retorna a estrutura livro.

Em resumo a função localizarLivro é responsável por procurar e retornar os dados de um livro específico a partir de um arquivo binário.

4.1.11 Efetuar venda

A figura 4.11 apresenta a função efetuarVenda() realiza o processo de registrar uma venda, solicita informações ao usuário, verifica a existência do cliente e do livro, confirma os dados e grava a venda em um arquivo. A função interage com o usuário para coletar e confirmar informações essenciais antes de registrar a venda.

Figura 4.11 - Efetuar venda

```c
void efetuarVenda(){
  FILE *fpvenda;
  struct reg_venda venda;
  struct reg_cliente cliente;
  struct reg_livro livro;
  char opc;

  //Entrada de Dados - Pedir o código da venda (nro. do pedido)
    printf("\nDigite o Codigo da Venda: ");
    fflush(stdin); scanf("%i",&venda.codvenda);
    //Pedir o Códio do cliente
    printf("Digite o Codigo do Cliente: ");
    fflush(stdin); scanf("%i",&venda.codcliente);
    //Localizar o Cliente, mostrar e pedir confirmação
    cliente = localizarCliente(venda.codcliente);
    if (cliente.codigo==-1){
        printf("\nNenhum Cliente localizado com este codigo!");
        return; //Retorna para o menu principal
    }
    printf("\nCliente: %s",cliente.nome);
    printf("\nConfirma Cliente(S/N)? ");
    fflush(stdin); scanf("%c",&opc);
    if ((opc=='n')||(opc=='N')){
        printf("\nCliente NAO Confirmado!");
        return; ////Retorna para o menu principal
    }
    //Pedir o Código do Livro
    printf("Digite o Codigo do Livro: ");
    fflush(stdin); scanf("%i",&venda.codlivro);
    //Localizar o Livro, mostrar e pedir confirmação
    livro = localizarLivro(venda.codlivro);
    if (livro.codigo==-1){
        printf("\nNenhum Livro localizado com este codigo!");
        return; //Retorna para o menu principal
    }
    printf("\Livro: %s",livro.titulo);
    printf("\nConfirma Livro(S/N)? ");
    fflush(stdin); scanf("%c",&opc);
    if ((opc=='n')||(opc=='N')){
        printf("\nLivro NAO Confirmado!");
        return; ////Retorna para o menu principal
    }

    //Pedir a Qtde
    printf("\nDigite a quantidade: ");
    fflush(stdin); scanf("%i",&venda.qtde);

    //Pedir o Desconto
    printf("\nDigite o Desconto: ");
    fflush(stdin); scanf("%f",&venda.desconto);

  //Confirmar Venda
    printf("\nConfirma Venda(S/N)? ");
    fflush(stdin); scanf("%c",&opc);
    if ((opc=='n')||(opc=='N')){
        printf("\nVenda NAO Confirmado!");
        return; ////Retorna para o menu principal
    }

  //Gravar os dados no arquivo de Vendas
    //Abrir o Arquivo
    fpvenda = fopen(VENDAS,"ab");
    //Gravar o registro da venda
    fwrite(&venda,sizeof(venda),1,fpvenda);
    //Fechar o Arquivo
    fclose(fpvenda);
    //Dar uma msg de feedback para o usuário
    printf("\n Venda Gravada com Sucesso.");
}//Fim efetuarVenda()
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE*fpvenda é um ponteiro para manipulação do arquivo de vendas, struct reg_venda venda, struct reg_cliente cliente, struct reg_livro livro são estruturas que representam os dados da venda, cliente e livro, respectivamente opc é uma variável de opção usada para confirmar as ações do usuário.

Entrada de dados: Solicitação do código da venda venda.codvenda, do código do cliente venda.codcliente e do código do livro venda.codlivro, bem como quantidade venda.qtde e desconto venda.desconto.

Localização e confirmação do cliente: A função chama localizarCliente para verificar a existência do cliente. Se o cliente não for encontrado cliente.codigo == -1, a função retorna ao menu principal. Se o cliente for encontrado, o nome do cliente é exibido e o programa solicita ao usuário que confirme os dados.

Localização e confirmação do livro: Similar ao cliente, a função chama localizarLivro para verificar a existência do livro. Se o livro não for encontrado livro.codigo == -1, a função retorna ao menu principal. Se o livro for encontrado, o título do livro é exibido e o programa solicita ao usuário que confirme os dados.

Confirmação da venda: O programa solicita ao usuário a confirmação da venda completa após a inserção de todos os dados.

Gravação no Arquivo de Vendas: Se a venda for confirmada, o arquivo VENDAS é aberto no modo "ab" (append binary), os dados da venda são gravados no arquivo usando fwrite, o arquivo é fechado e uma mensagem de sucesso é exibida.

Em resumo a função efetuarVenda() é responsável por registrar uma venda, verificando e confirmando dados de clientes e livros antes de armazená-los no arquivo de vendas.

4.1.12 Fechar pedido

A figura 4.12 apresenta a função fecharPedido() que tem como objetivo localizar um pedido de venda por meio do código fornecido pelo usuário, exibir os detalhes do pedido, permitir que o usuário confirme o fechamento do pedido e em seguida, gravar os detalhes do pedido em um arquivo de vendas. Essa função interage com o usuário para encontrar e confirmar o fechamento de um pedido específico.

Figura 4.12 - Fechar Pedido

```
454 void fecharPedido(){
455   FILE *fpvenda;
456   struct reg_venda venda;
457   struct reg_cliente cliente;
458   struct reg_livro livro;
459   char opc;
460   int cod, achou=0;
461
462   //Solicitar o codigo de venda a ser Localizado
463   printf("\nDigite o codigo de venda a ser localizado: ");
464   fflush(stdin); scanf("%i",&cod);
465
466   //Abrir o Arquivo
467   if ((fpvenda = fopen(VENDAS,"rb"))==NULL){//consistência-Abertura do Arquivo
468     printf("\nErro ao abrir o Arquivo %s",VENDAS);
469     return;  //Volta para o Programa Principal
470   }
471
472   //Ler registro por registro ate encontrar o codigo da venda
473   printf("\nProcessando Codigo de venda: %i",cod);
474   while ((achou==0) && (fread(&venda,sizeof(venda),1,fpvenda)==1)){
475     if (venda.codvenda==cod){
476     cliente = localizarCliente(venda.codcliente);
477     livro = localizarLivro(venda.codlivro);
478     printf("\nCodigo Venda: %i",venda.codvenda);
479     printf("\nNome Cliente: %s",cliente.nome);
480     printf("\nCodigo Livro: %i",venda.codlivro);
481     printf("\nTitulo: %s",livro.titulo);
482     printf("\nQtde.: %i",venda.qtde);
483     printf("\nPreco.: %5.2f",livro.preco);
484     printf("\nDesconto: %5.2f",venda.desconto);
485     printf("\nTotal: %5.2f",livro.preco - venda.desconto);
486     printf("\n-----------------------------------------");
487     achou = 1; //Sinaliza que encontrou o pedido
488      }
489   }
490
491   if (achou==0){
492     printf("\nNenhum pedido localizado com este codigo!");
493     fclose(fpvenda);
494     return; //Retorna ao Menu Principal
495   }
496
497       //Confirmar Fechar Pedido
498       printf("\nConfirma Fechamento do Pedido(S/N)? ");
499       fflush(stdin); scanf("%c",&opc);
500       if ((opc=='n')||(opc=='N')){
501           printf("\nFechamento NAO Confirmado!");
502           return; ////Retorna para o menu principal
503       }
504
505       //Gravar os dados no arquivo de Vendas
506       //Abrir o Arquivo
507       fpvenda = fopen(VENDAS,"ab");
508       //Gravar o registro da venda
509       fwrite(&venda,sizeof(venda),1,fpvenda);
510       //Fechar o Arquivo
511       fclose(fpvenda);
512       //Dar uma msg de feedback para o usuário
513       printf("\n Fechamento do Pedido Gravado com Sucesso.");
514 }// Fim fechamentoPedido
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fpvenda é um ponteiro para manipulação do arquivo de vendas, struct reg_venda venda, struct reg_cliente cliente, struct reg_livro livro são estruturas que representam os dados de venda, cliente e livro, respectivamente e as variáveis opc é usada para confirmar as ações do usuário e cod é o código de venda fornecido pelo usuário.

Solicitação do código de venda: O programa solicita ao usuário a fornecer o código do pedido que deseja fechar.

Abertura do arquivo de vendas: O arquivo de vendas é aberto no modo de leitura binária ("rb"). Se a abertura falhar, uma mensagem de erro é exibida e a função retorna ao menu principal.

Localização do Pedido: A função lê registros do arquivo um por um até encontrar o pedido com o código fornecido pelo usuário. Se o pedido for encontrado, os detalhes do pedido, incluindo o cliente e o livro associados, são exibidos na tela.

Confirmação do fechamento do pedido: O programa solicita ao usuário a confirmar o fechamento do pedido.

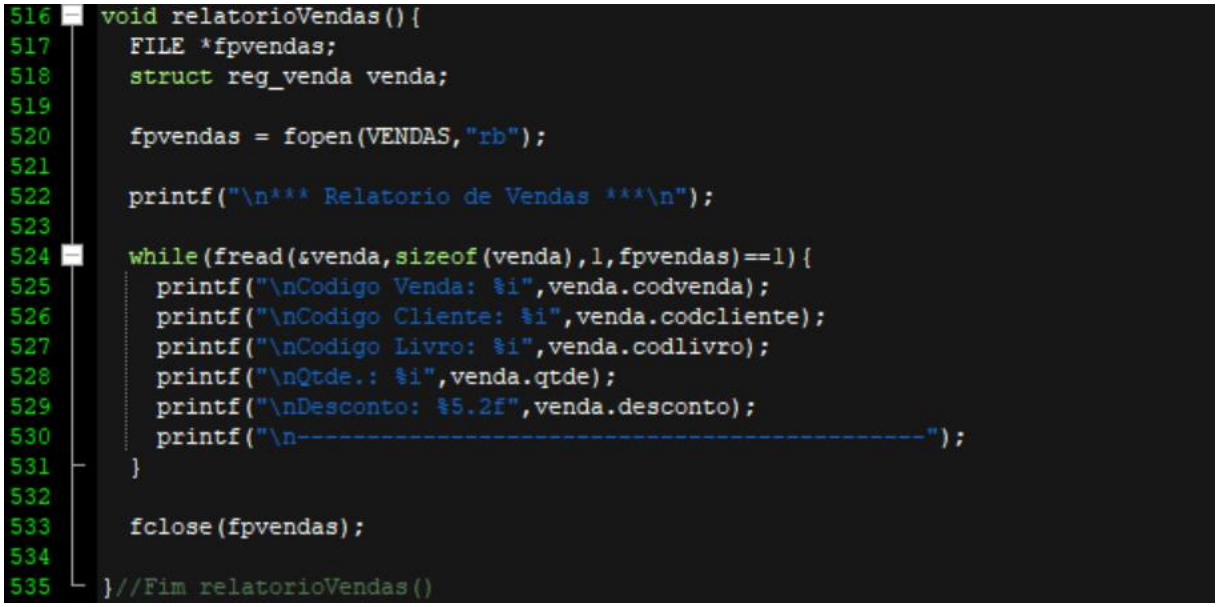
Gravação no arquivo de vendas: Se o usuário confirmar o fechamento do pedido, o arquivo de vendas é aberto no modo "ab" (append binary). Os detalhes do pedido são gravados no arquivo de vendas usando fwrite. O arquivo é fechado e uma mensagem de sucesso é exibida.

Em resumo a função fecharPedido() desempenha um papel importante no sistema de vendas, permitindo que os usuários localizem e confirmem o fechamento de pedidos.

4.1.13 Relatório de vendas

A figura 4.13 apresenta a função relatorioVendas() tem como objetivo gerar um relatório detalhado das vendas registradas no arquivo de vendas. Ela abre o arquivo de vendas, lê os registros um por um e exibe as informações importantes de cada venda, como o código da venda, o código do cliente, o código do livro, a quantidade vendida e o desconto aplicado.

Figura 4.13 - Relatório de vendas



```
516 void relatorioVendas(){
517   FILE *fpvendas;
518   struct reg_venda venda;
519
520   fpvendas = fopen(VENDAS,"rb");
521
522   printf("\n*** Relatorio de Vendas ***\n");
523
524   while(fread(&venda,sizeof(venda),1,fpvendas)==1){
525     printf("\nCodigo Venda: %i",venda.codvenda);
526     printf("\nCodigo Cliente: %i",venda.codcliente);
527     printf("\nCodigo Livro: %i",venda.codlivro);
528     printf("\nQtde.: %i",venda.qtde);
529     printf("\nDesconto: %5.2f",venda.desconto);
530     printf("\n-----------------------------------------");
531   }
532
533   fclose(fpvendas);
534
535 }//Fim relatorioVendas()
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fpvendas é um ponteiro para manipulação do arquivo de vendas e struct reg_venda venda representa os dados de cada venda.

Abertura do arquivo de vendas: O arquivo de vendas é aberto no modo de leitura binária ("rb"). Se a abertura falhar, a função termina sua execução.

Geração do relatório: A função entra em um loop enquanto houver registros para ler o arquivo de vendas. A cada iteração do loop, um registro de venda é lido no arquivo e suas informações são impressas na tela, incluindo o código da venda, o código do cliente, o código do livro, a quantidade vendida e o desconto aplicado.

Fechamento do arquivo: Após a leitura completa dos registros, o arquivo de vendas é encerrado para garantir que todas as alterações sejam salvas corretamente.

Em resumo a função relatorioVendas() é essencial para gerar os relatórios de vendas dentro do sistema, proporcionando uma visão detalhada das transações realizadas.

4.1.14 Relatório de vendas detalhado

A figura 4.14 ilustra a função relatorioDetalhadoVendas(), que tem a finalidade de criar um relatório abrangente das vendas armazenadas no arquivo correspondente. Esta função inicia abrindo o arquivo de vendas, percorre os registros individualmente

e apresenta detalhes sobre cada transação. Entre as informações exibidas estão o número do pedido, o cliente relacionado, e os itens vendidos, que incluem código, título, quantidade, preço unitário, desconto aplicado e valor total pago. Além disso, a função calcula e mostra o total geral das vendas.

Figura 4.14 - Relatório detalhado de vendas

```
537 void relatorioDetalhadoVendas() {
538     FILE *fpvendas;
539     struct reg_venda venda;
540     struct reg_livro livro;
541     struct reg_cliente cliente;
542     int Total = 0;
543 
544     // Abrir Arquivo de Vendas
545     fpvendas = fopen(VENDAS, "rb");
546     if (fpvendas == NULL) {
547         printf("\nErro ao abrir o Arquivo de Vendas!");
548         return;
549     }
550 
551     // Imprimir cabeçalho do relatório
552     printf("\n        ### Livraria TechInfo ###");
553 
554     // Ler registro por registro e imprimir na tela
555     while (fread(&venda, sizeof(venda), 1, fpvendas) == 1) {
556         cliente = localizarCliente(venda.codcliente);
557         livro = localizarLivro(venda.codlivro);
558 
559         printf("\nPedido nro: %i", venda.codvenda);
560         printf("\nCliente: %s", cliente.nome);
561         printf("\nCodigo - Titulo     Qtde.   Preco   Desconto    Valor Pago");
562         printf("\n %-7i %-24s %-7i %-9.2f %-11.2f %-5.2f",
563                livro.codigo, livro.titulo, venda.qtde, livro.preco,
564                venda.desconto * venda.qtde,
565                (livro.preco - venda.desconto) * venda.qtde);
566         printf("\n");
567         Total += (livro.preco - venda.desconto) * venda.qtde;
568     }
569 
570     // Imprimir o total geral das vendas
571     printf("\n                                    Total: %i", Total);
572 
573     // Fechar o arquivo
574     fclose(fpvendas);
575 }// Fim relatorioDetalhadoVendas()
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fpvendas é um ponteiro para manipulação do arquivo de vendas, struct reg_venda venda, struct reg_livro livro e struct reg_cliente cliente representam os dados de vendas, livros e clientes, respectivamente e a variável Total é uma variável para calcular o total geral das vendas.

Abertura do arquivo de vendas: O arquivo de vendas é aberto no modo de leitura binária (“rb”). Se ocorrer um erro durante a abertura do arquivo, uma mensagem de erro é exibida e a função é encerrada.

Geração do relatório: A função imprime o cabeçalho do relatório, exibindo o nome da livraria. Em seguida, entra em um loop que percorre todos os registros do arquivo de vendas. Para cada registro de venda lido, a função utiliza a função localizarCliente para obter informações sobre o cliente associado à venda e a função localizarLivro para obter informações sobre o livro vendido. Após obter os dados do cliente e do livro, a função imprime detalhes sobre o pedido, incluindo o número do pedido, o nome do cliente, o código e o título do livro, a quantidade vendida, o preço unitário, o desconto aplicado e o valor total pago pelo item. Durante a impressão, o valor total da venda é calculado e acumulado na variável Total.

Impressão do total geral das vendas: Após o término da iteração pelos registros de venda, a função imprime o total geral das vendas.

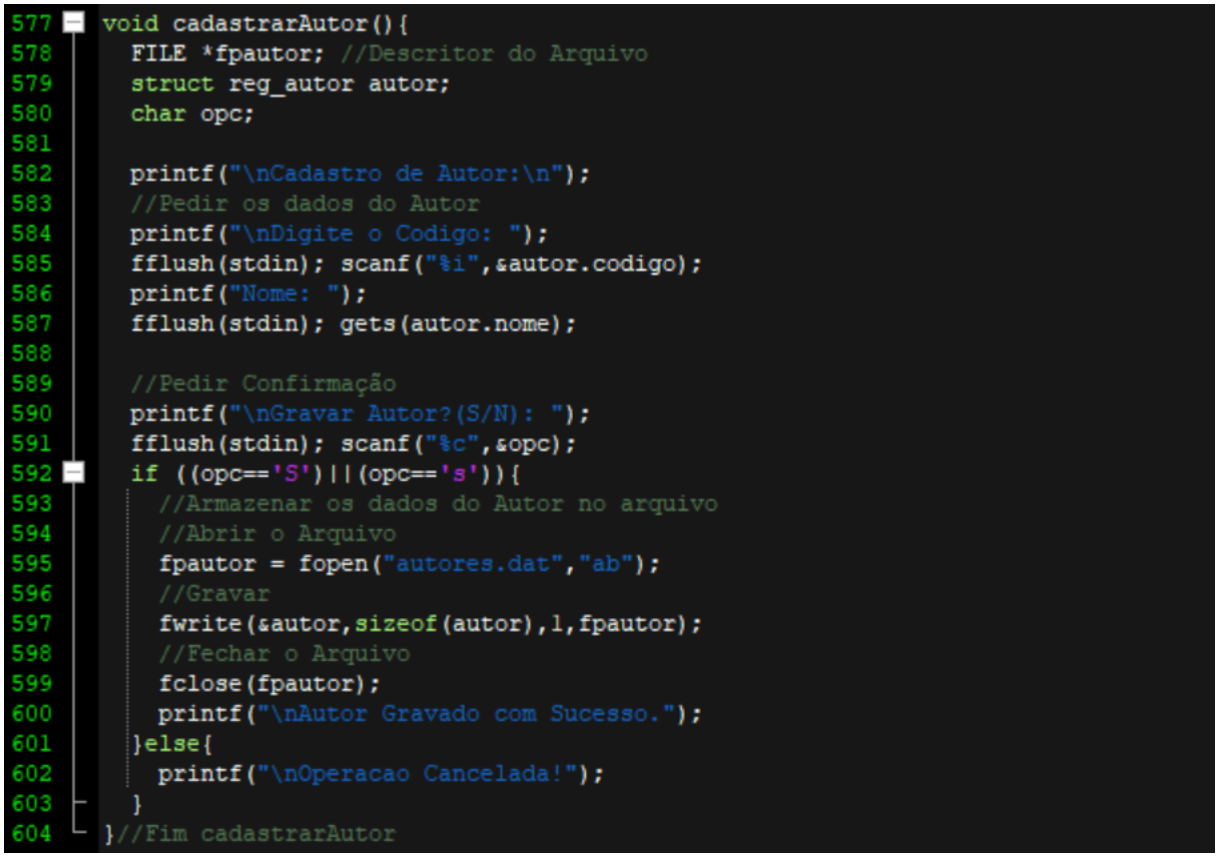
Fechamento do arquivo: o arquivo de vendas é fechado.

Em resumo a função relatorioDetalhadoVendas() desempenha um papel importante na geração de relatórios detalhados das vendas registradas.

4.1.15 Cadastrar autor

A figura 4.15 apresenta a função cadastrarAutor() que tem como objetivo permitir o cadastro de novos autores em um sistema de gerenciamento de informações. Ela solicita ao usuário que insira os dados do autor, incluindo código e nome, e oferece a opção de confirmar ou cancelar o cadastro. Caso a confirmação seja dada, as informações do autor são salvas em um arquivo de dados. Se cancelado, a operação é interrompida e uma mensagem indicando o cancelamento é exibida.

Figura 4.15 - Cadastrar autor



```
577 void cadastrarAutor(){
578   FILE *fpautor; //Descritor do Arquivo
579   struct reg_autor autor;
580   char opc;
581
582   printf("\nCadastro de Autor:\n");
583   //Pedir os dados do Autor
584   printf("\nDigite o Codigo: ");
585   fflush(stdin); scanf("%i",&autor.codigo);
586   printf("Nome: ");
587   fflush(stdin); gets(autor.nome);
588
589   //Pedir Confirmação
590   printf("\nGravar Autor?(S/N): ");
591   fflush(stdin); scanf("%c",&opc);
592   if ((opc=='S')||(opc=='s')){
593     //Armazenar os dados do Autor no arquivo
594     //Abrir o Arquivo
595     fpautor = fopen("autores.dat","ab");
596     //Gravar
597     fwrite(&autor,sizeof(autor),1,fpautor);
598     //Fechar o Arquivo
599     fclose(fpautor);
600     printf("\nAutor Gravado com Sucesso.");
601   }else{
602     printf("\nOperacao Cancelada!");
603   }
604 }//Fim cadastrarAutor
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fpautor é um ponteiro para manipulação do arquivo onde os dados dos autores serão armazenados. A struct reg_autor autor representa os dados do autor, incluindo código e nome. A variável opc armazena a opção de confirmação do usuário.

Solicitação de dados do autor: A função solicita ao usuário que insira o código e o nome do autor através de mensagens impressas na tela.

Confirmação do cadastro: Após a inserção dos dados, a função pede ao usuário para confirmar se deseja gravar os dados do autor no arquivo e a resposta é lida pelo programa.

Gravação dos dados no arquivo: Se o usuário confirmar o cadastro (resposta "S" ou "s"), o arquivo de autores é aberto no modo de escrita binária ("ab"). Os dados do autor são registrados no arquivo por meio da função fwrite. Depois que os dados são salvos, o arquivo é fechado.

Mensagem para o usuário: Com base na resposta fornecida pelo usuário, é apresentada uma mensagem confirmando a ação ou informando o cancelamento.

Em resumo a função cadastrarAutor() é responsável por permitir o cadastro de novos autores em um sistema.

4.1.16 Listar autores

A figura 4.16 apresenta a função listarAutores() tem como objetivo gerar um relatório listando todos os autores registrados em um arquivo de dados. Ela abre o arquivo de autores, lê os registros um por um e imprime na tela o código e o nome de cada autor. Após listar todos os autores, o arquivo é fechado.

Figura 4.16 - Listar autores

```
606 void listarAutores(){
607   FILE *fpautor;
608   struct reg_autor autor;
609
610   //Abrir o Arquivo
611   if ((fpautor = fopen(AUTORES,"rb"))==NULL){//consistência-Abertura do Arquivo
612     printf("\nErro ao abrir o Arquivo %s",AUTORES);
613     return;  //Volta para o Programa Principal
614   }
615
616   printf("\n####  Relatorio de Todos os Autores  ####");
617   printf("\nCodigo Autor");
618   //Ler registro por registro e mostrar na tela
619   while (fread(&autor,sizeof(autor),1,fpautor)==1){
620     printf("\n%-6i %-30s",autor.codigo,autor.nome);
621   }
622
623   //Fechar o Arquivo
624   fclose(fpautor);
625
626 }//Fim listarAutores()
627
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fpautor é um ponteiro para manipulação do arquivo de autores e a struct reg_autor autor representa os dados de cada autor, incluindo código e nome.

Abertura do arquivo de autores: O arquivo de autores é aberto no modo de leitura binária ("rb"). Caso haja um erro ao tentar abrir o arquivo, uma mensagem de erro será exibida, e a função será finalizada.

Geração do relatório: A função imprime um cabeçalho para o relatório indicando que se trata de um relatório de todos os autores. Em seguida, entra em um loop que percorre todos os registros do arquivo de autores. Para cada registro de autor lido, a função imprime na tela o código e o nome do autor.

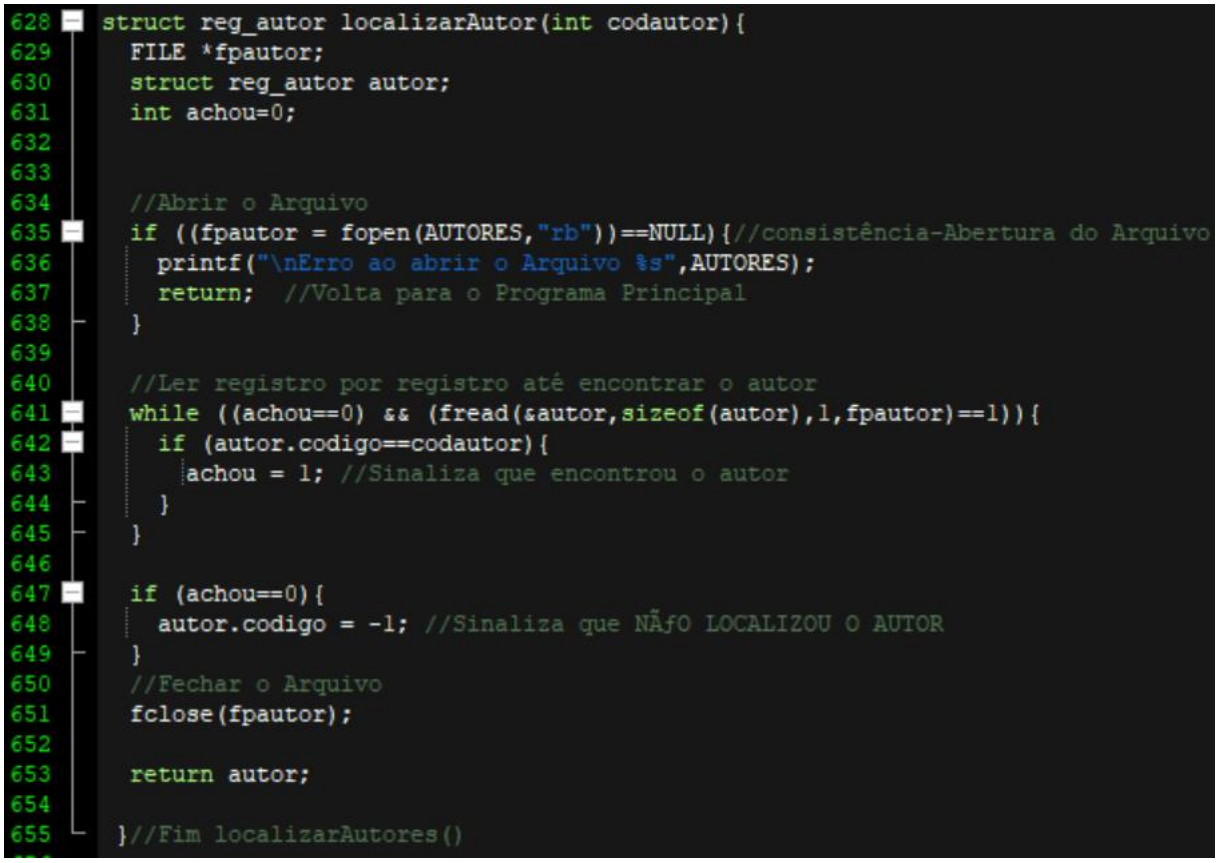
Fechamento do arquivo: Após listar todos os autores, o arquivo de autores é fechado.

Em resumo a função listarAutores() desempenha um papel importante na exibição de todos os autores registrados em um sistema.

4.1.17 Listar autores

A figura 4.17 apresenta a função localizarAutor() que busca um autor específico em um arquivo de dados com base no código fornecido codautor. Ela abre o arquivo de autores, lê os registros um por um até encontrar o autor com o código especificado e retorna a estrutura de dados do autor. Se o autor não for encontrado, a função retorna uma estrutura de autor com o código definido como -1, indicando que o autor não foi localizado.

Figura 4.17 - Listar autores



```
628 struct reg_autor localizarAutor(int codautor){
629   FILE *fpautor;
630   struct reg_autor autor;
631   int achou=0;
632
633
634   //Abrir o Arquivo
635   if ((fpautor = fopen(AUTORES,"rb"))==NULL){//consistência-Abertura do Arquivo
636     printf("\nErro ao abrir o Arquivo %s",AUTORES);
637     return;  //Volta para o Programa Principal
638   }
639
640   //Ler registro por registro até encontrar o autor
641   while ((achou==0) && (fread(&autor,sizeof(autor),1,fpautor)==1)){
642     if (autor.codigo==codautor){
643       achou = 1; //Sinaliza que encontrou o autor
644     }
645   }
646
647   if (achou==0){
648     autor.codigo = -1; //Sinaliza que NÃƒO LOCALIZOU O AUTOR
649   }
650   //Fechar o Arquivo
651   fclose(fpautor);
652
653   return autor;
654
655 }//Fim localizarAutores()
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fpautor é um ponteiro para manipulação do arquivo de autores. A struct reg_autor autor representa os dados do autor, incluindo código e nome. A variável achou é usada para sinalizar quando o autor é encontrado.

Abertura do arquivo de autores: O arquivo de autores é aberto no modo de leitura binária ("rb"). Se ocorrer um erro durante a abertura do arquivo, uma mensagem de erro é exibida e a função é encerrada retornando ao programa principal.

Procura pelo autor: A função inicia um loop que percorre os registros do arquivo de autores, lendo-os individualmente. Para cada registro, a função compara se o código do autor, representado por autor.codigo, corresponde ao código fornecido codautor. Se o autor é encontrado, a variável achou é definida como 1 para sair do loop.

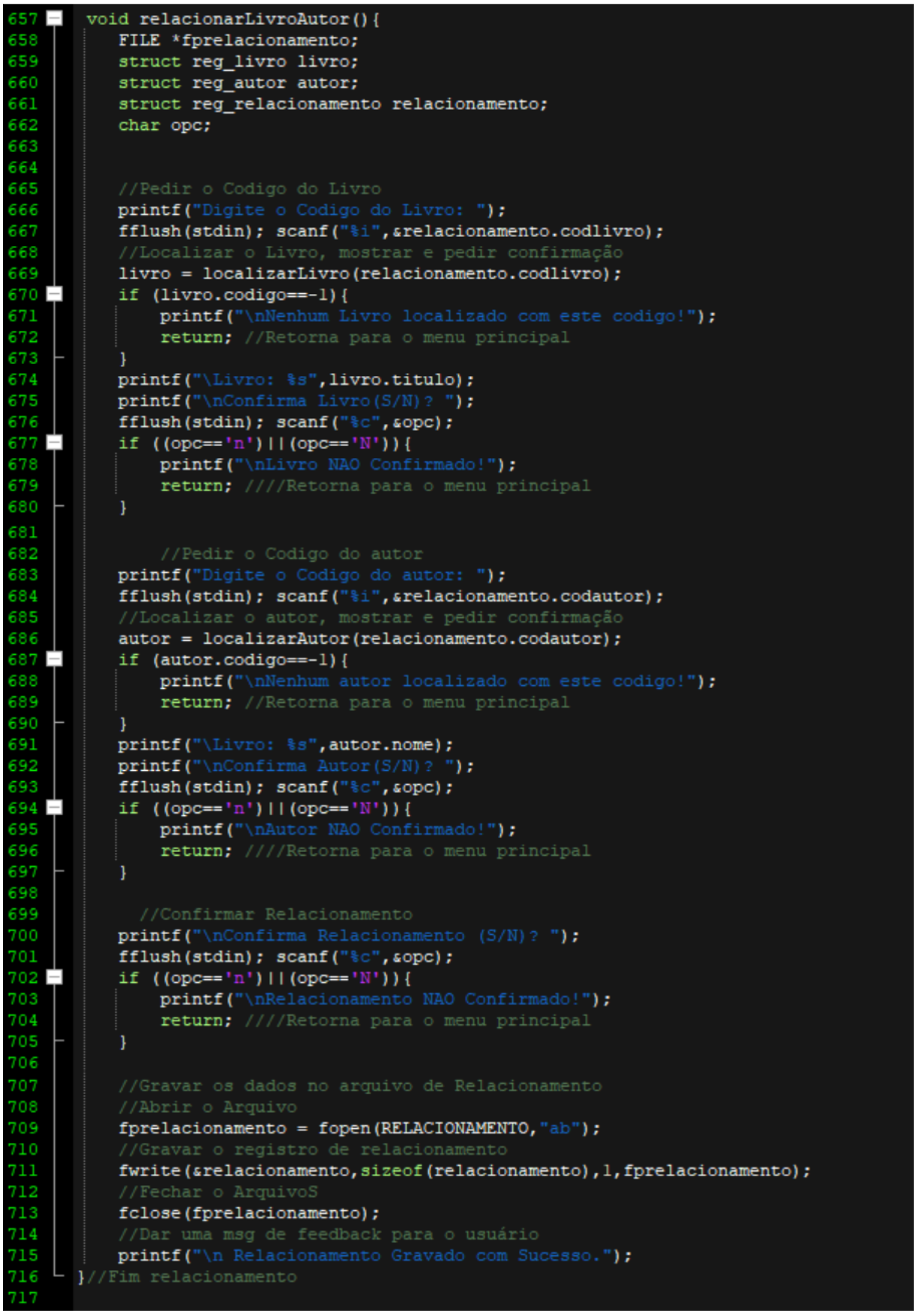
Verificação e retorno: Se a variável achou ainda for 0 após o loop, significa que o autor não foi encontrado. Nessa situação, o valor de autor.codigo é atribuído como -1 para sinalizar que o autor não foi encontrado. Após isso, o arquivo de autores é fechado e a função retorna a estrutura do autor.

Em resumo a função localizarAutor() desempenha um papel importante na busca e recuperação de registros de autores com base no código fornecido.

4.1.18 Relacionamento entre livro e autor

A figura 4.18 apresenta a função relacionarLivroAutor() que permite ao usuário associar um livro a um autor, salvando essa relação em um arquivo. Essa função é essencial para manter o vínculo entre livros e seus respectivos autores no sistema. O processo envolve a obtenção dos códigos do livro e do autor, confirmação por parte do usuário, e a gravação dos dados em um arquivo específico de relacionamentos.

Figura 4.18 - Relacionamento entre livro e autor



```
void relacionarLivroAutor(){
    FILE *fprelacionamento;
    struct reg_livro livro;
    struct reg_autor autor;
    struct reg_relacionamento relacionamento;
    char opc;


    //Pedir o Codigo do Livro
    printf("Digite o Codigo do Livro: ");
    fflush(stdin); scanf("%i",&relacionamento.codlivro);
    //Localizar o Livro, mostrar e pedir confirmação
    livro = localizarLivro(relacionamento.codlivro);
    if (livro.codigo==-1){
        printf("\nNenhum Livro localizado com este codigo!");
        return; //Retorna para o menu principal
    }
    printf("\Livro: %s",livro.titulo);
    printf("\nConfirma Livro(S/N)? ");
    fflush(stdin); scanf("%c",&opc);
    if ((opc=='n')||(opc=='N')){
        printf("\nLivro NAO Confirmado!");
        return; ////Retorna para o menu principal
    }

        //Pedir o Codigo do autor
    printf("Digite o Codigo do autor: ");
    fflush(stdin); scanf("%i",&relacionamento.codautor);
    //Localizar o autor, mostrar e pedir confirmação
    autor = localizarAutor(relacionamento.codautor);
    if (autor.codigo==-1){
        printf("\nNenhum autor localizado com este codigo!");
        return; //Retorna para o menu principal
    }
    printf("\Livro: %s",autor.nome);
    printf("\nConfirma Autor(S/N)? ");
    fflush(stdin); scanf("%c",&opc);
    if ((opc=='n')||(opc=='N')){
        printf("\nAutor NAO Confirmado!");
        return; ////Retorna para o menu principal
    }

      //Confirmar Relacionamento
    printf("\nConfirma Relacionamento (S/N)? ");
    fflush(stdin); scanf("%c",&opc);
    if ((opc=='n')||(opc=='N')){
        printf("\nRelacionamento NAO Confirmado!");
        return; ////Retorna para o menu principal
    }

    //Gravar os dados no arquivo de Relacionamento
    //Abrir o Arquivo
    fprelacionamento = fopen(RELACIONAMENTO,"ab");
    //Gravar o registro de relacionamento
    fwrite(&relacionamento,sizeof(relacionamento),1,fprelacionamento);
    //Fechar o ArquivoS
    fclose(fprelacionamento);
    //Dar uma msg de feedback para o usuário
    printf("\n Relacionamento Gravado com Sucesso.");
}//Fim relacionamento
```

Fonte: Elaborada pela autora, 2024

Declarações: ILE *fprelacionamento: ponteiro para manipulação do arquivo de relacionamentos, struct reg_livro livro: estrutura para armazenar dados do livro, struct reg_autor autor: estrutura para armazenar dados do autor e struct reg_relacionamento relacionamento: estrutura para armazenar o relacionamento entre livro e autor. A variável opc é usada para armazenar a confirmação do usuário em várias etapas do processo.

Entrada e confirmação dos dados: o programa solicita o código do livro ao usuário e o armazena em relacionamento.codlivro. A função localizarLivro é utilizada para buscar os dados do livro no arquivo. Se não encontrado, exibe uma mensagem de erro e retorna ao menu principal. Exibe os dados do livro e pede confirmação ao usuário. Se o usuário não confirmar, a função retorna ao menu principal. O código do autor também é solicitado ao usuário e o armazena em relacionamento.codautor. A função localizarAutor é utilizada para buscar os dados do autor no arquivo. Se não encontrado, exibe uma mensagem de erro e retorna ao menu principal. Exibe os dados do autor e pede confirmação ao usuário. Se o usuário não confirmar, a função retorna ao menu principal.

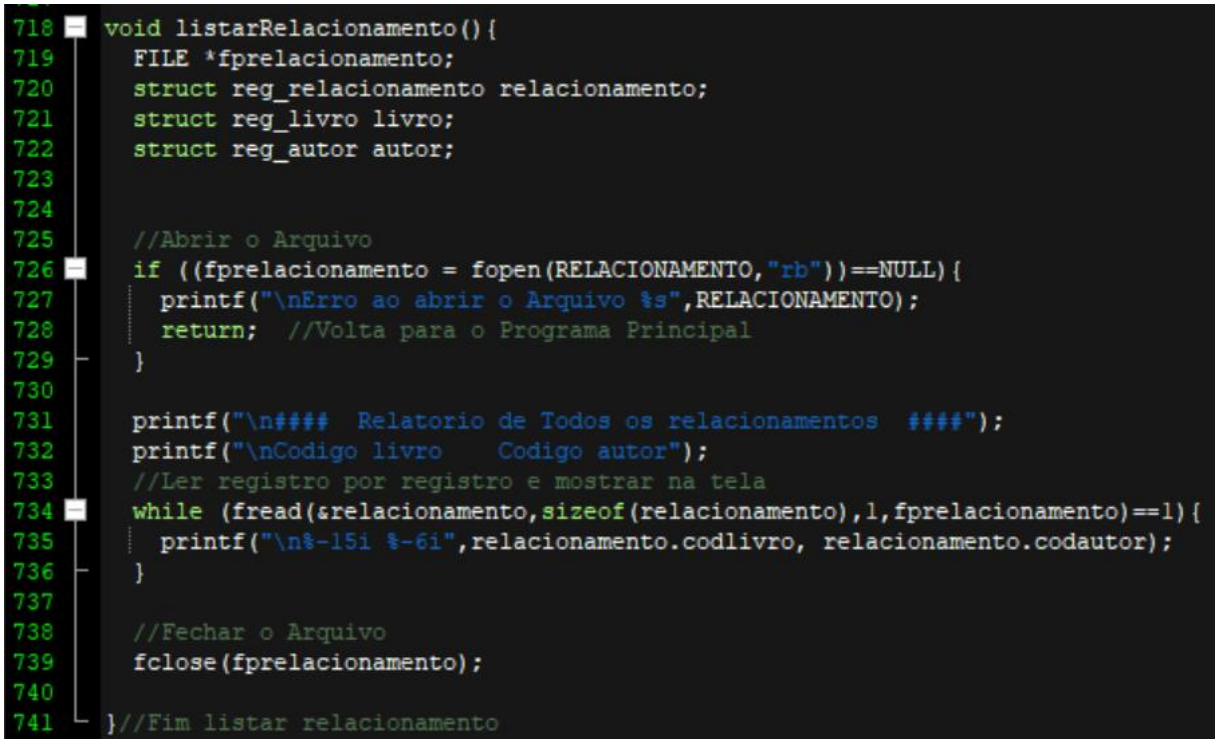
Confirmação final e gravação: Pede ao usuário uma confirmação final para salvar o relacionamento entre o livro e o autor. Se confirmado, abre o arquivo de relacionamentos no modo de adição binária ("ab"), grava o relacionamento e fecha o arquivo e exibe uma mensagem de sucesso ao usuário.

Em resumo a função relacionarLivroAutor() desempenha um papel essencial na manutenção dos vínculos entre livros e autores no sistema.

4.1.19 Listar relacionamento

A figura 4.19 apresenta a função listarRelacionamento() tem a finalidade de listar todos os relacionamentos entre livros e autores armazenados no sistema. Ela lê cada registro do arquivo de relacionamentos e exibe os códigos dos livros e autores associados.

Figura 4.19 - Listar relacionamento



```
718 void listarRelacionamento(){
719   FILE *fprelacionamento;
720   struct reg_relacionamento relacionamento;
721   struct reg_livro livro;
722   struct reg_autor autor;
723
724
725   //Abrir o Arquivo
726   if ((fprelacionamento = fopen(RELACIONAMENTO,"rb"))==NULL){
727     printf("\nErro ao abrir o Arquivo %s",RELACIONAMENTO);
728     return;  //Volta para o Programa Principal
729   }
730
731   printf("\n####  Relatorio de Todos os relacionamentos  ####");
732   printf("\nCodigo livro    Codigo autor");
733   //Ler registro por registro e mostrar na tela
734   while (fread(&relacionamento,sizeof(relacionamento),1,fprelacionamento)==1){
735     printf("\n%-15i %-6i",relacionamento.codlivro, relacionamento.codautor);
736   }
737
738   //Fechar o Arquivo
739   fclose(fprelacionamento);
740
741 }//Fim listar relacionamento
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fprelacionamento: ponteiro para manipulação do arquivo de relacionamentos, struct reg_relacionamento relacionamento: estrutura que armazena os dados de um relacionamento entre livro e autor, struct reg_livro livro: estrutura para dados de um livro e struct reg_autor autor: estrutura para dados de um autor.

Abertura do arquivo: O arquivo de relacionamentos, RELACIONAMENTO, é aberto no modo de leitura binária ("rb"). Se a abertura do arquivo falhar retornar NULL, uma mensagem de erro é exibida e a função retorna, saindo sem realizar qualquer operação adicional.

Impressão do cabeçalho: Exibe um cabeçalho informativo para a lista de relacionamentos, incluindo uma linha de título que descreve as colunas de código de livro e código de autor.

Leitura e exibição dos registros: Um loop while é utilizado para ler cada registro do arquivo de relacionamentos. Para cada registro lido com sucesso é verificado por fread retornando 1, o programa exibe os códigos do livro e do autor. A formatação usada (%-15i %-6i) alinha os códigos em colunas para melhor legibilidade.

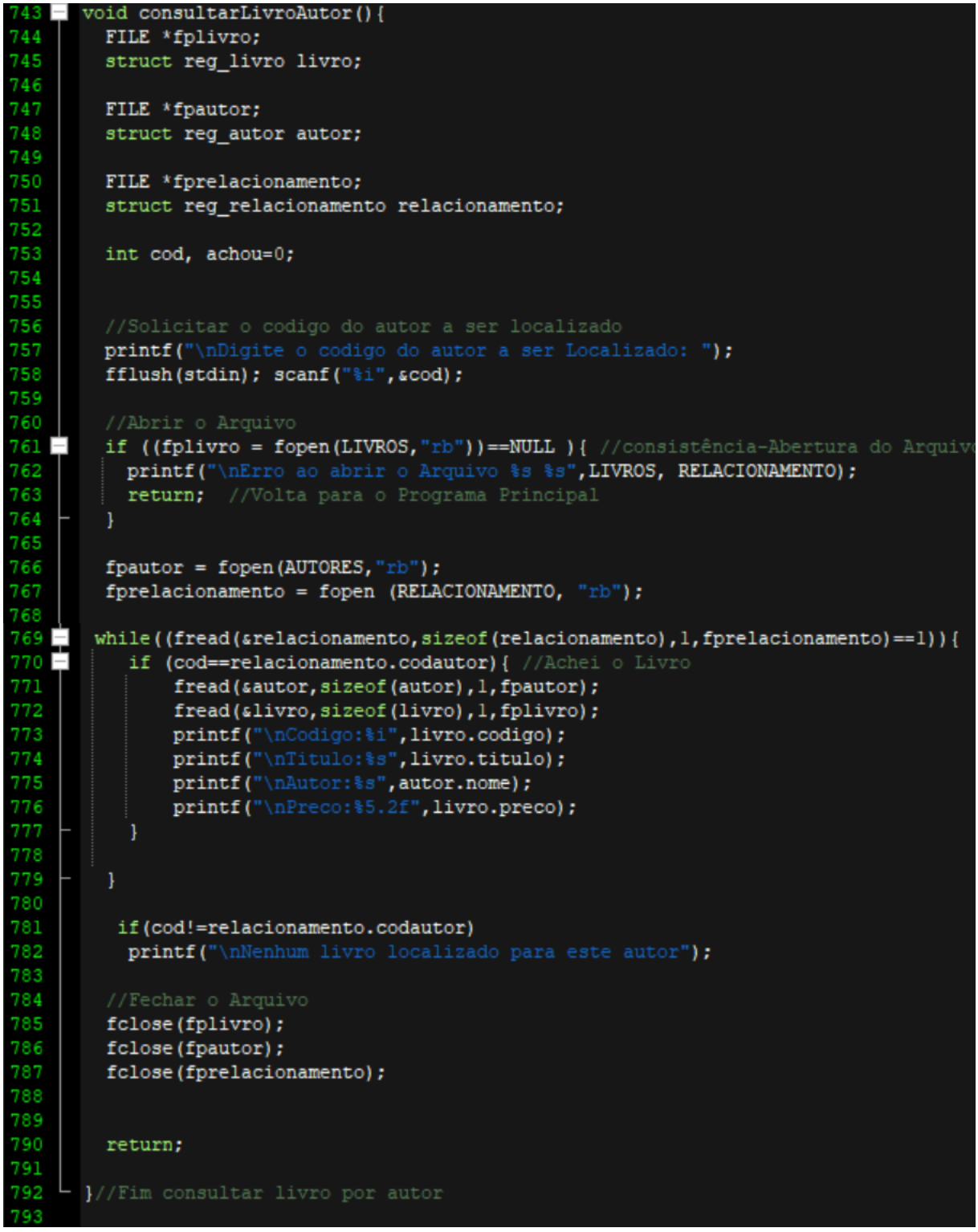
Fechamento do arquivo: Após a leitura completa dos registros, o arquivo é fechado.

Em resumo a função listarRelacionamento() é essencial para visualizar os vínculos entre livros e autores armazenados no sistema.

4.1.20 Consultar livro por autor

A figura 4.20 apresenta a função consultarLivroAutor() que tem como objetivo listar todos os livros associados a um determinado autor. O usuário fornece o código do autor, e a função busca e exibe os livros que ele escreveu, mostrando detalhes como código, título, nome do autor e preço do livro.

Figura 4.20 - Consultar livro por autor



```
743 void consultarLivroAutor(){
744   FILE *fplivro;
745   struct reg_livro livro;
746
747   FILE *fpautor;
748   struct reg_autor autor;
749
750   FILE *fprelacionamento;
751   struct reg_relacionamento relacionamento;
752
753   int cod, achou=0;
754
755
756   //Solicitar o codigo do autor a ser localizado
757   printf("\nDigite o codigo do autor a ser Localizado: ");
758   fflush(stdin); scanf("%i",&cod);
759
760   //Abrir o Arquivo
761   if ((fplivro = fopen(LIVROS,"rb"))==NULL ){ //consistência-Abertura do Arquiv
762     printf("\nErro ao abrir o Arquivo %s %s",LIVROS, RELACIONAMENTO);
763     return;  //Volta para o Programa Principal
764   }
765
766   fpautor = fopen(AUTORES,"rb");
767   fprelacionamento = fopen (RELACIONAMENTO, "rb");
768
769  while((fread(&relacionamento,sizeof(relacionamento),1,fprelacionamento)==1)){
770     if (cod==relacionamento.codautor){ //Achei o Livro
771         fread(&autor,sizeof(autor),1,fpautor);
772         fread(&livro,sizeof(livro),1,fplivro);
773         printf("\nCodigo:%i",livro.codigo);
774         printf("\nTitulo:%s",livro.titulo);
775         printf("\nAutor:%s",autor.nome);
776         printf("\nPreco:%5.2f",livro.preco);
777     }
778
779   }
780
781    if(cod!=relacionamento.codautor)
782     printf("\nNenhum livro localizado para este autor");
783
784   //Fechar o Arquivo
785   fclose(fplivro);
786   fclose(fpautor);
787   fclose(fprelacionamento);
788
789
790   return;
791
792 }//Fim consultar livro por autor
793
```

Fonte: Elaborada pela autora, 2024

Declarações: FILE *fplivro, FILE *fpautor, FILE *fprelacionamento: são ponteiros para manipulação dos arquivos de livros, autores e relacionamentos, respectivamente. As struct reg_livro livro, struct reg_autor autor, struct

reg_relacionamento relacionamento: são estruturas que armazenam os dados de livros, autores e relacionamentos.

Entrada de dados: O programa pede ao usuário que insira o código do autor que deseja localizar.

Abertura dos arquivos: Tenta abrir o arquivo de livros, LIVROS, no modo de leitura binária ("rb"). Caso a abertura falhe, uma mensagem de erro é exibida, e o controle retorna ao programa principal. Abre os arquivos de autores, AUTORES, e relacionamentos, RELACIONAMENTO, também no modo de leitura binária ("rb").

Leitura e exibição dos registros: Um loop while é utilizado para ler cada registro do arquivo de relacionamentos. Para cada relacionamento encontrado, verifica se o código do autor no registro de relacionamento corresponde ao código fornecido pelo usuário. Se uma correspondência é encontrada, lê os registros de autor e livro e exibe os detalhes na tela. Caso nenhum livro seja encontrado para o autor fornecido, exibe uma mensagem informando que nenhum livro foi localizado para aquele autor.

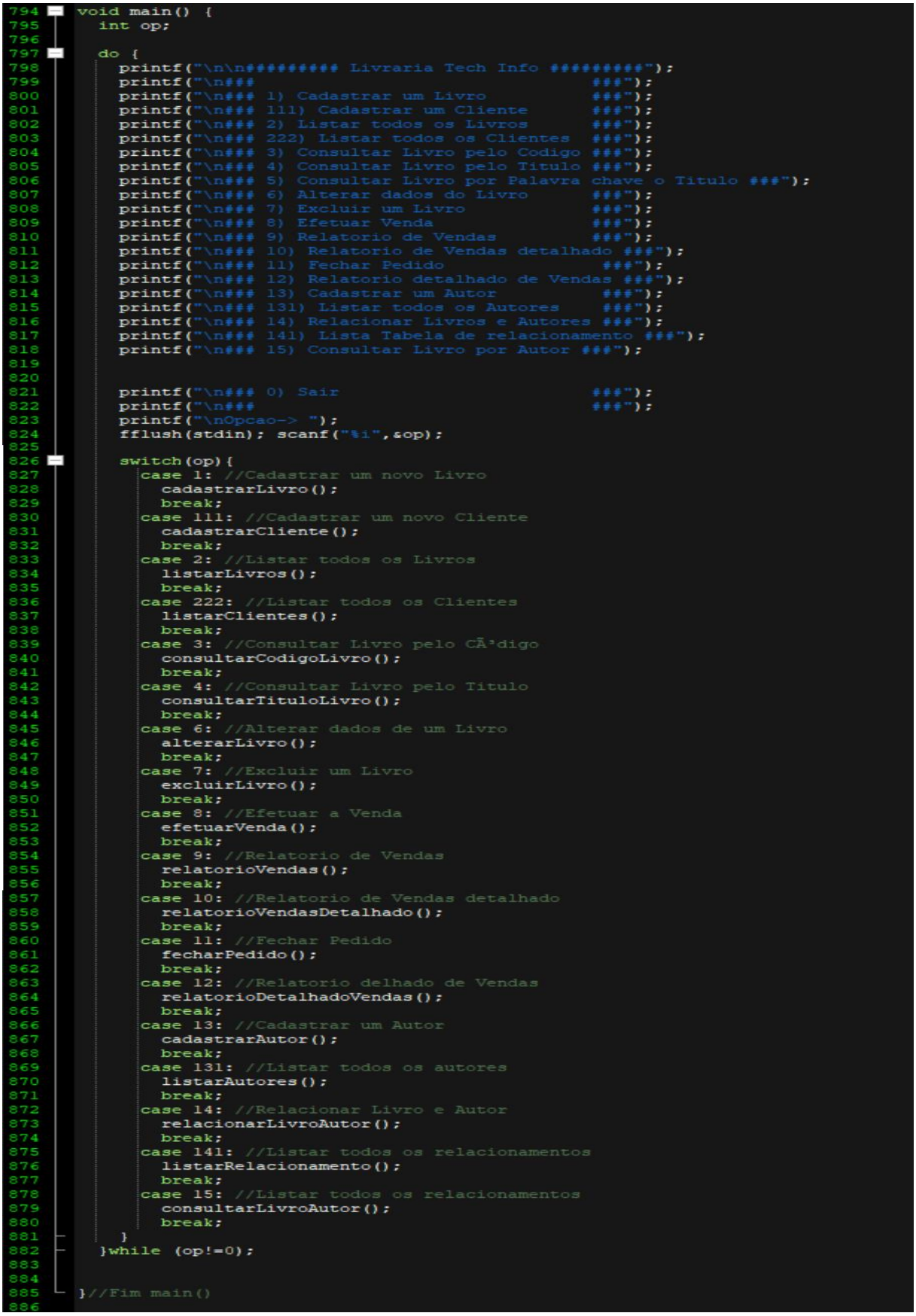
Fechamento dos arquivos: Após processar todos os registros, fecha todos os arquivos abertos.

Em resumo a função consultarLivroAutor() desempenha um papel importante na busca de livros por autor e facilita a visualização dos livros escritos por um autor específico.

4.1.21 Menu principal

A figura 4.21 apresenta a função main() que serve como o ponto de entrada para o programa e implementa o menu principal para uma aplicação de livraria. Esta função exibe opções ao usuário, recebe a escolha do usuário e chama a função correspondente com base na escolha feita. O menu é iterado até que o usuário escolha a opção de sair.

Figura 4.21 - Menu principal



```
void main() {
  int op;

  do {
    printf("\n\n######### Livraria Tech Info #########");
    printf("\n###                                 ###");
    printf("\n### 1) Cadastrar um Livro           ###");
    printf("\n### 111) Cadastrar um Cliente       ###");
    printf("\n### 2) Listar todos os Livros       ###");
    printf("\n### 222) Listar todos os Clientes   ###");
    printf("\n### 3) Consultar Livro pelo Codigo ###");
    printf("\n### 4) Consultar Livro pelo Titulo ###");
    printf("\n### 5) Consultar Livro por Palavra chave o Titulo ###");
    printf("\n### 6) Alterar dados do Livro       ###");
    printf("\n### 7) Excluir um Livro             ###");
    printf("\n### 8) Efetuar Venda                ###");
    printf("\n### 9) Relatorio de Vendas          ###");
    printf("\n### 10) Relatorio de Vendas detalhado ###");
    printf("\n### 11) Fechar Pedido               ###");
    printf("\n### 12) Relatorio detalhado de Vendas ###");
    printf("\n### 13) Cadastrar um Autor          ###");
    printf("\n### 131) Listar todos os Autores    ###");
    printf("\n### 14) Relacionar Livros e Autores ###");
    printf("\n### 141) Lista Tabela de relacionamento ###");
    printf("\n### 15) Consultar Livro por Autor ###");


    printf("\n### 0) Sair                         ###");
    printf("\n###                                 ###");
    printf("\nOpcao-> ");
    fflush(stdin); scanf("%i",&op);

    switch(op){
      case 1: //Cadastrar um novo Livro
        cadastrarLivro();
        break;
      case 111: //Cadastrar um novo Cliente
        cadastrarCliente();
        break;
      case 2: //Listar todos os Livros
        listarLivros();
        break;
      case 222: //Listar todos os Clientes
        listarClientes();
        break;
      case 3: //Consultar Livro pelo CÃ³digo
        consultarCodigoLivro();
        break;
      case 4: //Consultar Livro pelo Titulo
        consultarTituloLivro();
        break;
      case 6: //Alterar dados de um Livro
        alterarLivro();
        break;
      case 7: //Excluir um Livro
        excluirLivro();
        break;
      case 8: //Efetuar a Venda
        efetuarVenda();
        break;
      case 9: //Relatorio de Vendas
        relatorioVendas();
        break;
      case 10: //Relatorio de Vendas detalhado
        relatorioVendasDetalhado();
        break;
      case 11: //Fechar Pedido
        fecharPedido();
        break;
      case 12: //Relatorio delhado de Vendas
        relatorioDetalhadoVendas();
        break;
      case 13: //Cadastrar um Autor
        cadastrarAutor();
        break;
      case 131: //Listar todos os autores
        listarAutores();
        break;
      case 14: //Relacionar Livro e Autor
        relacionarLivroAutor();
        break;
      case 141: //Listar todos os relacionamentos
        listarRelacionamento();
        break;
      case 15: //Listar todos os relacionamentos
        consultarLivroAutor();
        break;
    }
  }while (op!=0);


}//Fim main()
```

Fonte: Elaborada pela autora, 2024

Declaração: op, variável para armazenar a escolha do usuário.

Laço principal (do-while): Exibe um menu com várias opções para o usuário. As opções cobrem uma ampla gama de funcionalidades da livraria, incluindo cadastrar livros, clientes e autores, listar registros, consultar livros por diversos critérios, efetuar vendas, gerar relatórios e relacionar livros a autores.

Menu e opções: O menu é impresso na tela usando printf. O usuário insere uma opção, que é lida usando scanf.

Estrutura de controle (switch-case): A estrutura de controle switch-case do programa permite a execução de diferentes funções com base na opção fornecida pelo usuário. Dependendo da escolha do usuário, a função correspondente é chamada para executar a ação desejada, facilitando a gestão e operações da livraria de forma organizada e eficiente.

Repetição do menu: O menu se repete enquanto a opção selecionada não for 0, opção de saída.

Em resumo o menu principal do programa serve como um ponto central de controle, permitindo ao usuário interagir com várias funcionalidades da livraria de forma intuitiva. Usando uma estrutura de controle switch-case, o programa oferece uma ampla gama de opções que incluem operações como cadastrar livros e clientes, listar registros, consultar informações específicas, alterar e excluir dados, efetuar vendas, e gerar relatórios detalhados. Cada opção é mapeada para uma função específica que executa a operação desejada, garantindo que todas as funcionalidades necessárias para a gestão da livraria estejam acessíveis a partir de um único menu. Este design modular facilita a manutenção do código e melhora a experiência do usuário, permitindo uma navegação clara e eficiente através das diversas tarefas administrativas.

4.2 CONCLUSÃO

O desenvolvimento do sistema de gerenciamento de livraria durante o curso de Estruturas de Dados evidenciou a importância de selecionar e utilizar adequadamente diversas estruturas de dados para resolver problemas específicos. Através da aplicação prática de conceitos como listas ligadas, ponteiros, manipulação de strings e gerenciamento de arquivos, os estudantes foram capazes de criar um sistema funcional e eficiente que atende às necessidades de uma livraria real. A escolha de

listas para gerenciar livros, clientes e autores, por exemplo, facilitou operações frequentes de inserção e remoção, enquanto o uso de ponteiros permitiu uma gestão dinâmica e eficiente da memória. A manipulação de strings foi importante para o tratamento das informações textuais dos livros, e a persistência de dados foi garantida através do gerenciamento eficaz de arquivos.

Este projeto não apenas consolidou o conhecimento teórico dos alunos, mas também os preparou para enfrentar desafios reais na área de desenvolvimento de sistemas. A compreensão e a aplicação correta das estruturas de dados são fundamentais para a criação de algoritmos eficientes e soluções robustas. Assim, a disciplina de Estruturas de Dados proporcionou uma base sólida para o desenvolvimento de habilidades críticas na área de tecnologia da informação, preparando os alunos para futuros projetos e desafios profissionais.

5 BANCO DE DADOS

Na disciplina de Banco de Dados, ministrada no quarto semestre do Curso de ADS, foi possível aprender os conceitos do Modelo Entidade-Relacionamento (MER) e realizar o mapeamento de modelos de dados.

Elmasri e Navathe (2011, p.3) definem: “Um banco de dados é uma coleção de dados relacionados. Com dados, queremos dizer fatos conhecidos que podem ser registrados e possuem significado implícito.”.

Na sociedade moderna, os bancos de dados são elementos fundamentais, já que muitas das nossas atividades diárias dependem da interação com eles. Nomes, números de telefones, datas de aniversários e endereços podem estar registrados em uma agenda. Segundo Elmasri e Navathe (2011, p.3): “Essa coleção de dados relacionados, com um significado implícito, é um banco de dados.”.

5.1 MODELO ENTIDADE-RELACIONAMENTO E MAPEAMENTO DE MODELOS DE DADOS

A modelagem de banco de dados é uma etapa crucial para garantir o sucesso de um banco de dados.

> Modelagem de dados é o estudo das informações existentes em um contexto sob observação para a construção de um modelo de representação e entendimento de tal contexto. A modelagem de dados minera as informações que representam um contexto, estruturando-as em um conjunto que denominamos modelo lógico de dados.
> Uma das principais características da abordagem de modelagem de banco de dados é que ela fornece níveis de abstração de dados que omitem do usuário final detalhes sobre o armazenamento dos dados. Não existe a preocupação com um banco de dados tecnologicamente falando (Machado, 2014, p. 15).

De acordo com Machado (2014, p. 16): “O modelo de Entidade-Relacionamento (ER) é o método mais comum para construção de modelos de dados para bancos de dados relacionais.”

No Modelo ER, as entidades representam objetos ou conceitos importantes que são modelados no banco de dados. Cada entidade do MER é traduzida em uma tabela no banco de dados relacional.

Cada tabela no banco de dados corresponde a uma entidade no MER. Por exemplo, uma entidade chamada Clientes seria convertida em uma tabela com o

mesmo nome. As tabelas armazenam os dados reais e são compostas por linhas e colunas.

Os atributos das entidades no MER se transformam em colunas nas tabelas. Por exemplo, se uma entidade Cliente possui atributos como nome, endereço e telefone, essas características se tornam colunas na tabela Clientes.

Relacionamentos entre entidades no MER são representados por chaves estrangeiras e tabelas associativas no banco de dados relacional. Isso garante que as tabelas estejam corretamente interligadas e que as integridades referencial e relacional sejam mantidas.

Para construir um modelo de dados de forma eficaz, é fundamental entender e aplicar o conceito de abstração.

> Uma abstração ou a capacidade de abstração é um processo, com certeza mental, que usamos quando selecionamos várias características e propriedades de um conjunto de objetos ou fatos, e excluímos outras que não são relevantes em um contexto. Em outras palavras, aplicamos abstração quando nos concentramos nas propriedades de um conjunto de objetos ou coisas que consideramos essenciais, e desprezamos outras que não consideramos importantes, sempre dentro de um determinado contexto sob observação ou análise. (Machado, 2014, p. 17).

5.1.1 Atividades práticas do Modelo Entidade-Relacionamento (MER)

Para aplicar os conceitos de MER, realizamos diversas atividades práticas que auxiliaram na consolidação do conteúdo.

Exemplo de atividade com MER: *Uma empresa deseja elaborar um cadastro completo de seus empregados e suas atividades.*

Para cada funcionário, é necessário registrar seu nome, número funcional, telefone (com DDD e número), além de seus dependentes (nem todos possuem dependentes). Também deve-se armazenar o departamento em que o funcionário trabalha (todo funcionário está associado a um departamento), o departamento que ele gerencia (caso ele seja gerente) e os projetos nos quais está envolvido (nem todos participam de projetos).

Para cada departamento, é preciso registrar seu nome, número, os funcionários que pertencem a ele (todo departamento tem um gerente), o gerente responsável

(todo departamento tem um gerente) e os projetos sob sua supervisão (nem todo departamento supervisiona projetos).

Para cada projeto, deve-se armazenar seu nome, número, as cidades onde está sendo desenvolvido, os funcionários envolvidos (todo projeto tem funcionários) e o departamento responsável por sua supervisão (todo projeto é supervisionado por um departamento).

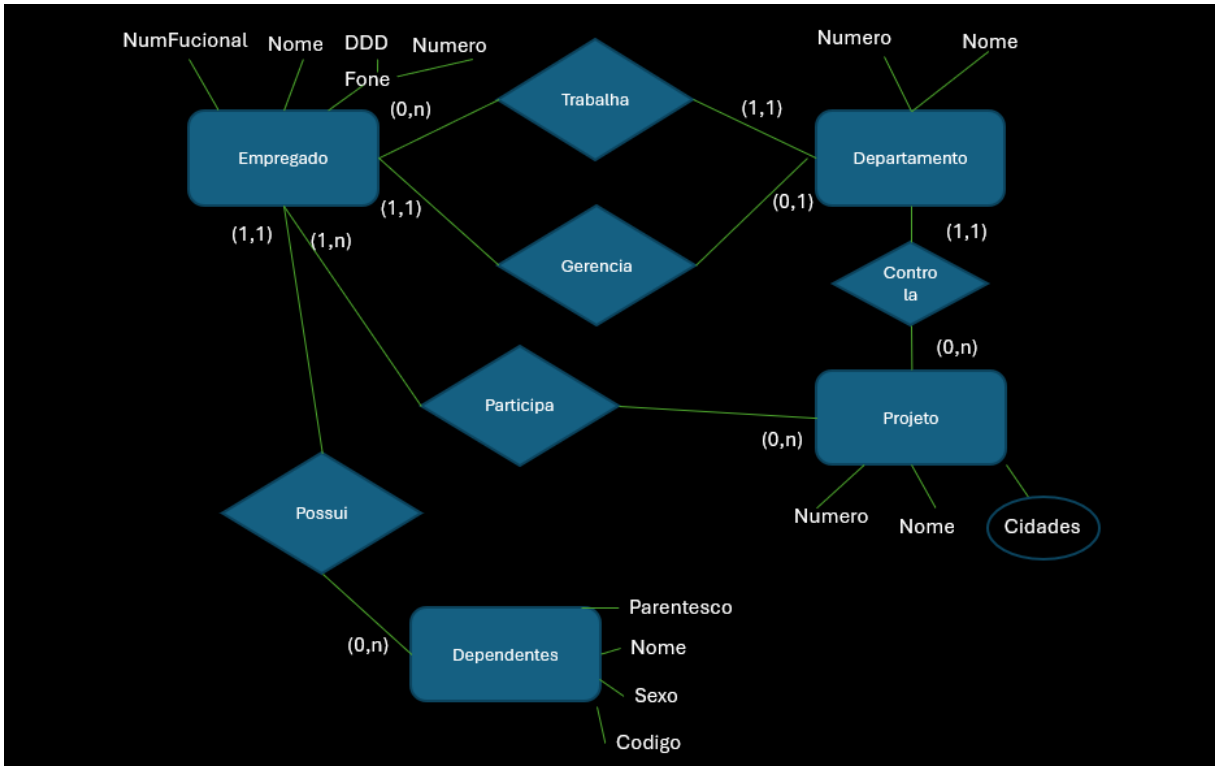
Resolução da atividade:

Entidades:

- Empregados (nome, numero funcional, fone, dependentes, departamento trabalha, departamento gerencia, projeto participa).
- Departamentos (nome, numero, empregados, empregado gerente, projetos).
- Projetos (nome, numero, cidades, funcionários, departamento).
- Dependentes (nome, sexo, parentesco, empregado).

A figura 5.1. apresenta a resolução do exercício anterior.

Figura 5.1 - Resolução atividade



Fonte: Elaborada pela autora, 2024

No mapeamento de modelos de dados há sete regras a se seguir.

Toda entidade se torna uma tabela: cada entidade no modelo Entidade-Relacionamento (ER) é mapeada para uma tabela no banco de dados relacional.

Entidade fraca: quando há uma entidade fraca, como nas tabelas A e B, a tabela correspondente à entidade fraca (B) recebe a chave primária da tabela correspondente à entidade forte (A) como chave estrangeira.

Relacionamento (0,1) e (1,1): em relacionamentos com cardinalidade (0,1) e (1,1), ambas as tabelas podem receber a chave primária da outra como chave estrangeira.

Relacionamento (0,N), (1,1) e (1,N): para relacionamentos com cardinalidade (0,N), (1,1) e (1,N), a entidade que possui a cardinalidade N recebe a chave primária da outra tabela como chave estrangeira. Toda chave estrangeira deve estar conectada à chave primária da tabela de origem.

Relacionamento (1,N) e (0,N): pode haver atributos associados ao relacionamento. Para relacionamentos com cardinalidade (N,N), é criada uma nova

tabela. Esta nova tabela recebe as chaves primárias de ambas as tabelas envolvidas no relacionamento, além de quaisquer atributos de relacionamento.

Atributo multivalorado: atributos multivalorados são transformados em tabelas separadas. A nova tabela inclui a chave primária da tabela de origem e o atributo multivalorado.

Relacionamento (N:N:N): relacionamentos ternários com cardinalidade (N:N:N) são muito raros. Quando ocorrem, eles geralmente exigem a criação de uma nova tabela que inclui as chaves primárias de todas as entidades envolvidas.

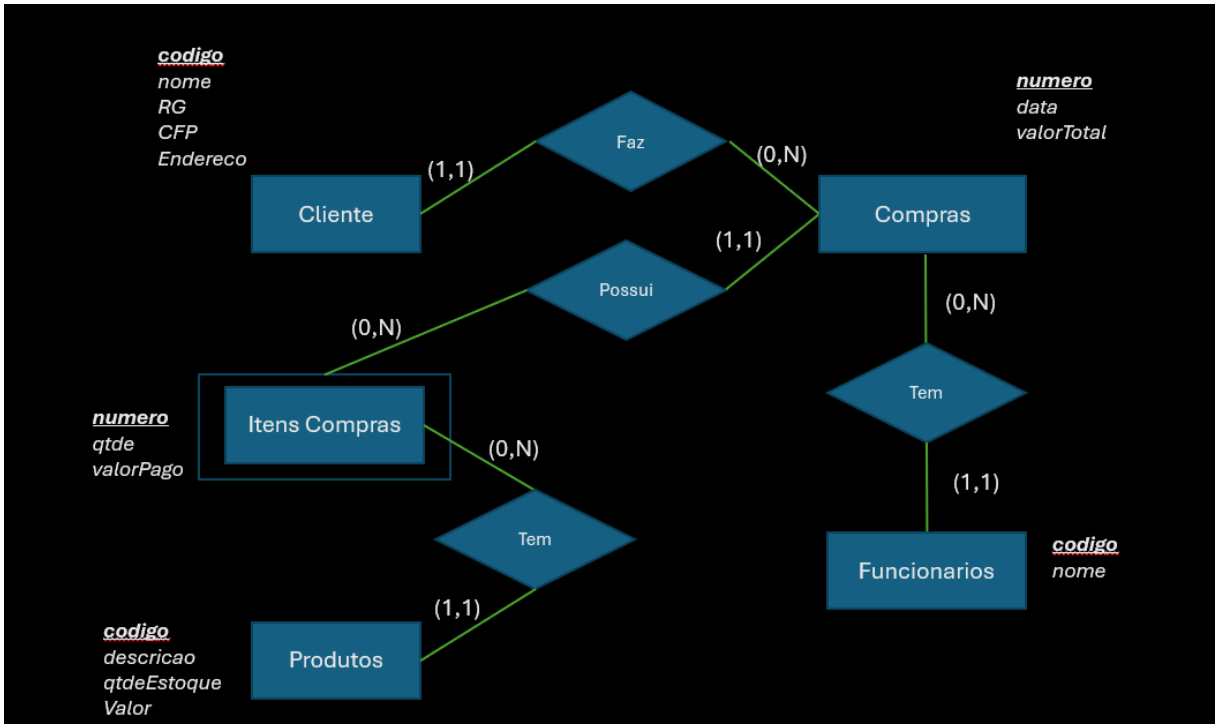
Exemplo de atividade de mapeamento de modelos de dados: Criar um diagrama ER baseado nas tabelas fornecidas, mostrando todas as entidades, atributos e os relacionamentos entre as entidades.

Tabelas:

Cliente: (codigo, nome, rg,cfp,rua, numero, bairro,cep, cidade)
Telefones: (codCliente, fone)
Compras: (numero, data, valorTotal, codCliente, codFuncinario)
Funcionarios: (codigo, nome)
Itens compras: (numero, qtde, valorPago, codCompras, codProduto)
Produtos: (codigo, descricao, qtdeEstoque, valor)

A figura 5.2 apresenta a resolução da atividade.

Figura 5.2 - Resolução atividade



Fonte: Elaborada pela autora, 2024

5.2 CONCLUSÃO

Através de atividades práticas, conseguimos consolidar os conhecimentos adquiridos, compreendendo a importância da modelagem e do mapeamento correto das entidades e seus relacionamentos para garantir a integridade e a eficiência do banco de dados. A habilidade de abstrair e mapear dados de forma precisa é essencial para o desenvolvimento de sistemas, refletindo diretamente na qualidade e na usabilidade dos sistemas implementados.

6 REDES DE COMPUTADORES

Durante o quinto semestre do curso de ADS, a disciplina de Redes de Computadores se mostrou fundamental para o desenvolvimento de competências essenciais na área de tecnologia da informação. Durante o semestre foi abordado os tipos de comunicação, tipos de redes, topologias, Switch, Dynamic Host Configuration Protocol (DHCP), Web Server, Sistema de Nomes de Domínio (DNS), Router e Bridge.

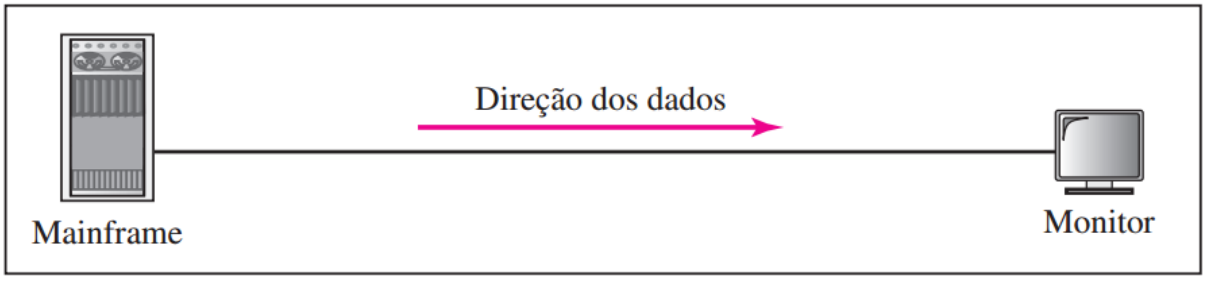
Para consolidar o aprendizado teórico, utilizamos o Cisco Packet Tracer, uma ferramenta de simulação de redes. Com ele, pudemos criar e testar configurações de redes, experimentar com diferentes topologias e dispositivos, e resolver problemas práticos. A prática com o Packet Tracer permitiu a simulação de redes, testes de diferentes configurações sem risco de danos reais e facilitou a compreensão dos conceitos abordados.

6.1 TIPOS DE COMUINICAÇÃO

Simplex: A comunicação simplex é a forma mais básica de transmissão de dados, onde a comunicação ocorre em apenas uma direção. Um exemplo clássico é a transmissão de rádio, onde a estação de rádio transmite o sinal e os receptores apenas recebem. Em redes de computadores, esse tipo de comunicação é menos comum, pois a maioria das aplicações exige comunicação bidirecional. No entanto, ele pode ser adequado para situações em que apenas uma direção de comunicação é necessária, como em sistemas de transmissão de TV.

A figura 6.1 exemplifica a comunicação simplex.

Figura 6.1 - Comunicação Simplex

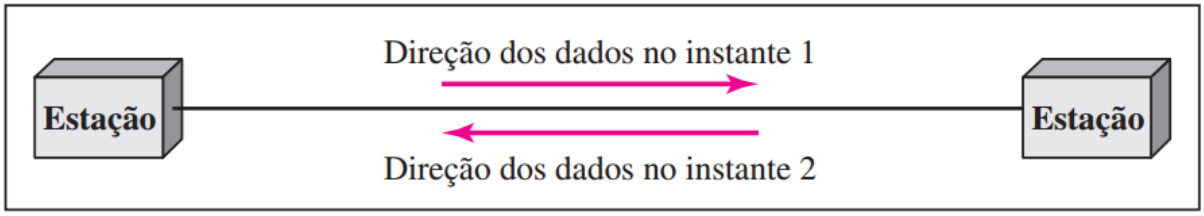


Fonte: Forouzan, 2010, p. 39

Half-Duplex: A comunicação half-duplex permite a transmissão de dados em ambas as direções, mas não ao mesmo tempo. Isso significa que enquanto um dispositivo está enviando dados, o outro precisa aguardar antes de responder. Exemplos típicos incluem walkie-talkies, onde os usuários precisam alternar entre falar e ouvir. Em redes de computadores, o half-duplex é utilizado em conexões Ethernet mais antigas ou em redes que requerem controle de acesso ao meio para evitar colisões de dados. Essa forma de comunicação é mais eficiente do que o simplex, pois permite a troca bidirecional de informações, embora não simultaneamente.

A figura 6.2 ilustra a comunicação half-duplex.

Figura 6.2 - Comunicação Half-duplex

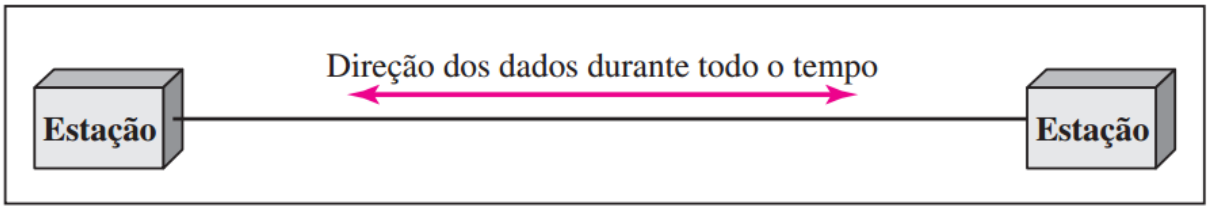


Fonte: Forouzan, 2010, p. 39

Full-Duplex: A comunicação full-duplex possibilita a transmissão simultânea de dados em ambas as direções. Um exemplo cotidiano é a comunicação telefônica, onde ambas as partes podem falar e ouvir ao mesmo tempo. Em redes de computadores modernas, o full-duplex é amplamente utilizado, especialmente em conexões Ethernet e Wi-Fi, permitindo uma comunicação mais rápida e eficiente. Essa forma de transmissão maximiza a largura de banda disponível, reduzindo o tempo de espera e aumentando o desempenho geral da rede. O full-duplex é essencial para aplicações que requerem alta taxa de transferência de dados e baixa latência, como videoconferências e jogos online.

A figura 6.3 exemplifica a comunicação full-duplex.

Figura 6.3 - Comunicação Full-duplex



Fonte: Forouzan, 2010, p. 39

6.2 TIPOS DE REDES

LAN (Local Area Network): Rede local, geralmente confinada a um único edifício ou campus. Utilizada para conectar computadores e dispositivos em um espaço geográfico restrito.

MAN (Metropolitan Area Network): Rede metropolitana que cobre uma área geográfica maior que a LAN, como uma cidade ou um campus universitário.

WAN (Wide Area Network): Rede de longa distância que abrange uma área geográfica extensa, conectando cidades, países ou até continentes. A Internet é o exemplo mais notável de uma WAN.

PAN (Personal Area Network): Rede de pequena escala projetada para comunicação entre dispositivos pessoais próximos, geralmente dentro de um raio de alguns metros. Exemplos típicos de PAN incluem a conexão entre um smartphone e um fone de ouvido Bluetooth, ou a sincronização de dados entre um laptop e um smartwatch.

WLAN (Wireless Local Area Network): Rede local que utiliza tecnologia sem **fio para permitir a comunicação entre dispositivos dentro de um espaço limitado, como** uma casa, escritório ou campus universitário. As WLANs costumam seguir os padrões IEEE 802.11, amplamente conhecidos como Wi-Fi. Esses padrões permitem que dispositivos como laptops, smartphones e tablets se conectem a um ponto de acesso sem fio (AP), que por sua vez está conectado a uma rede com fio ou à internet. As WLANs oferecem flexibilidade e facilidade de acesso, eliminando a necessidade de cabos e permitindo a mobilidade dos usuários dentro da área de cobertura.

WMAN (Wireless Metropolitan Area Network): Rede que cobre uma área geográfica mais extensa do que uma WLAN, como uma cidade ou uma região

metropolitana. As WMANs são empregadas para interligar diversos edifícios, escritórios e até bairros inteiros.

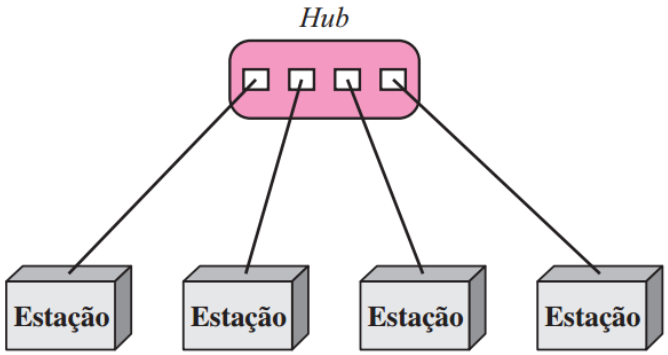
6.3 TOPOLOGIAS DE REDES

Segundo Forouzan, (2010, p. 41): “O termo topologia física se refere à maneira pela qual uma rede é organizada fisicamente.”. Há quatro topologias básicas possíveis: estrela, barramento, anel e mesh.

Topologia em Estrela: Topologia em Estrela: Nesta configuração, todos os dispositivos são ligados a um dispositivo central, geralmente um switch. Esta topologia é fácil de gerenciar e diagnosticar falhas.

A figura 6.4 exemplifica a topologia estrela.

Figura 6.4 - Topologia estrela conectando 4 estações

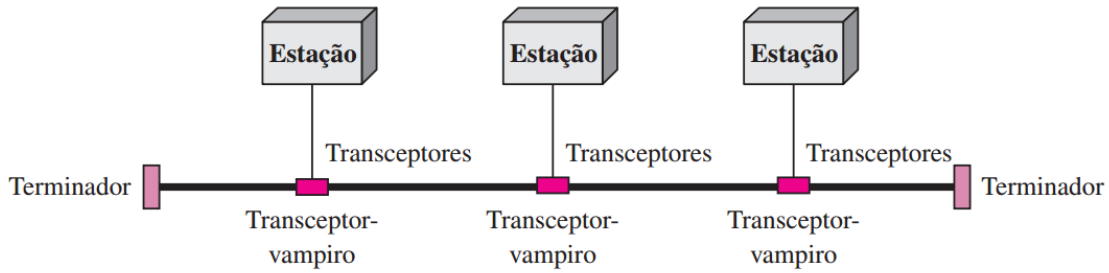


Fonte: Forouzan, 2010, p. 44

Topologia em Barramento: Todos os dispositivos compartilham um único meio de comunicação. É econômica, mas tem limitações de desempenho e dificuldades de manutenção.

A figura 6.5 exemplifica a topologia barramento.

Figura 6.5 - Topologia barramento conectando 3 estações

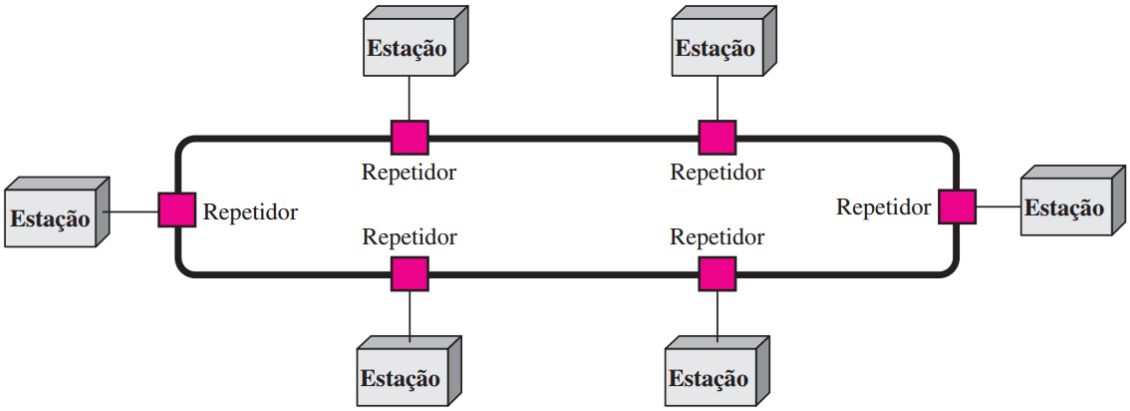


Fonte: Forouzan, 2010, p. 44

Topologia em Anel: Nesta estrutura, cada dispositivo está ligado ao seguinte, formando um circuito fechado. Os dados são enviados através do anel até alcançarem o destinatário. Apesar de sua eficiência, uma falha em qualquer parte pode afetar toda a rede.

A figura 6.6 ilustra a topologia anel.

Figura 6.6 - Topologia anel conectando 6 estações

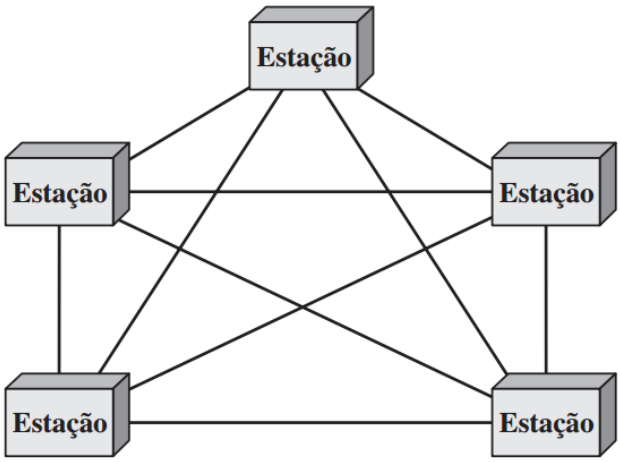


Fonte: Forouzan, 2010, p. 45

Topologia Mesh: Nesta configuração, cada dispositivo se conecta a vários outros, garantindo redundância e alta disponibilidade. No entanto, é uma topologia complexa e dispendiosa para implementar.

A figura 6.7 exemplifica a topologia mesh.

Figura 6.7 - Topologia mesh completamente conectada (5 dispositivos)



Fonte: Forouzan, 2010, p. 43

6.4 DISPOSITIVOS DE REDES

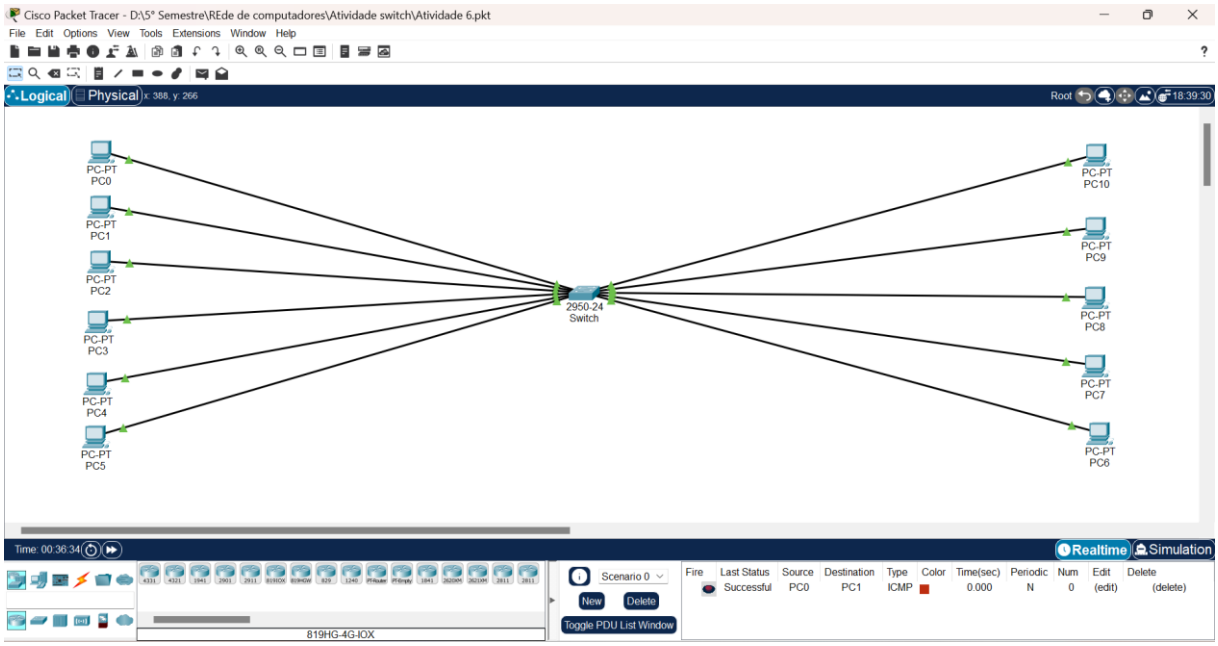
Os dispositivos de rede desempenham um papel crucial na comunicação entre computadores e outros aparelhos dentro de uma rede. Eles ajudam a gerenciar, controlar e direcionar o fluxo de dados, garantindo que as informações cheguem aos seus destinos corretamente. Dispositivos como switches, roteadores, bridges e pontos de acesso desempenham papéis específicos que são vitais para o bom funcionamento da rede.

6.4.1 Switch

Um switch é um dispositivo que conecta múltiplos dispositivos em uma rede local (LAN) e encaminha pacotes de dados entre eles com base no endereço MAC de destino. Utilizar switches em uma rede melhora a eficiência e o desempenho ao permitir a comunicação simultânea entre múltiplos pares de dispositivos.

A figura 6.8 ilustra um exemplo prático da utilização de um switch.

Figura 6.8 - Exemplo prático de Switch



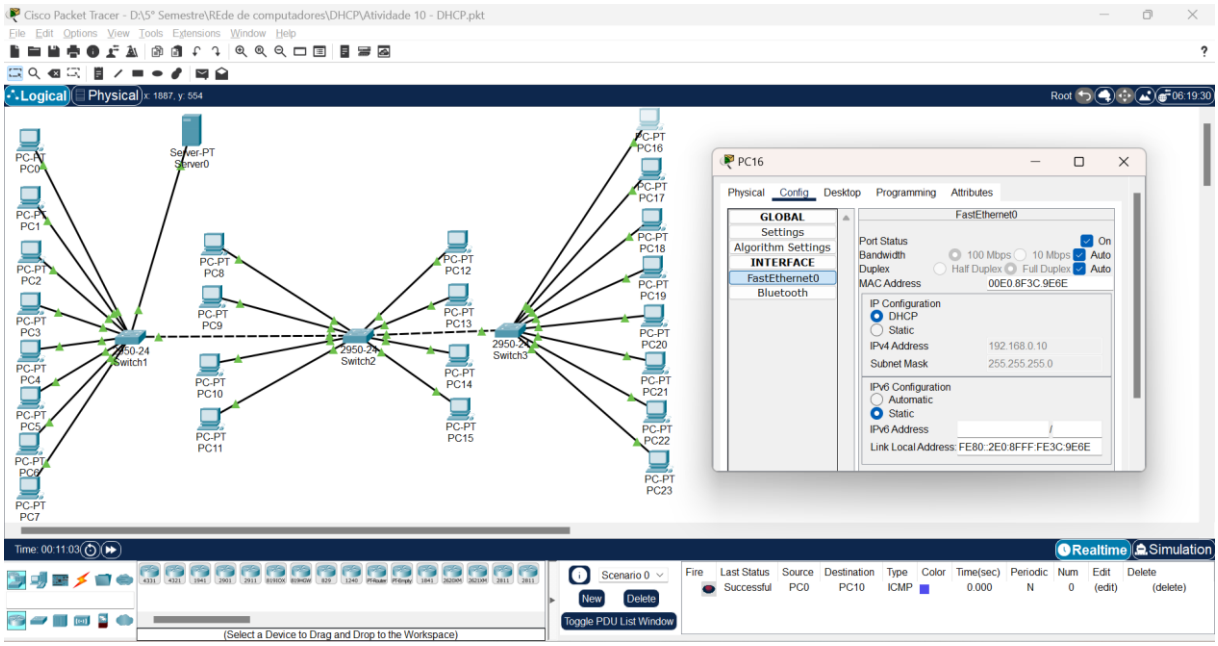
Fonte: Elaborada pela autora, 2024

6.4.2 DHCP (Dynamic Host Configuration Protocol)

O DHCP é um protocolo que permite a configuração automática de dispositivos em uma rede, atribuindo endereços IP dinamicamente. Isso simplifica a administração de redes, evitando a necessidade de configuração manual de endereços IP.

A figura 6.9 ilustra um exemplo prático da utilização de um DHCP.

Figura 6.9 - Exemplo prático de DHCP



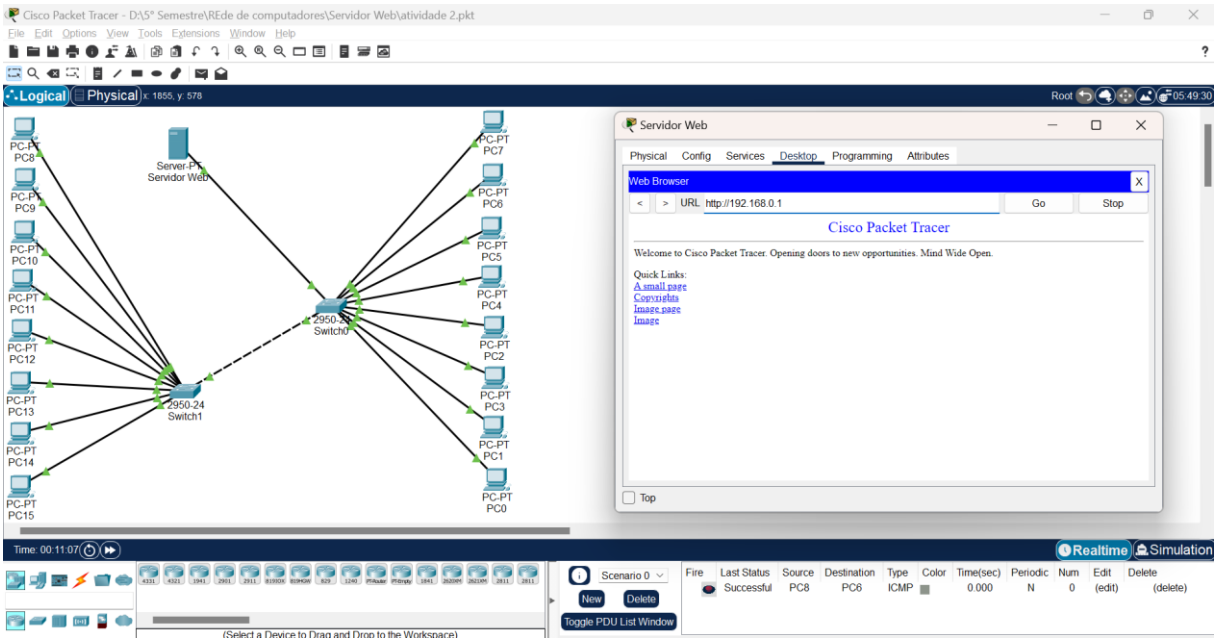
Fonte: Elaborada pela autora, 2024

6.4.3 Web Server

Um servidor web é um software responsável por fornecer páginas web aos usuários. Ele responde a solicitações HTTP enviadas por navegadores e entrega os recursos solicitados, como HTML, CSS e imagens. Compreender a configuração e o gerenciamento de servidores web é crucial para qualquer profissional de TI.

A figura 6.10 ilustra um exemplo prático da utilização de um Web Server.

Figura 6.10 - Exemplo prático de Web Server



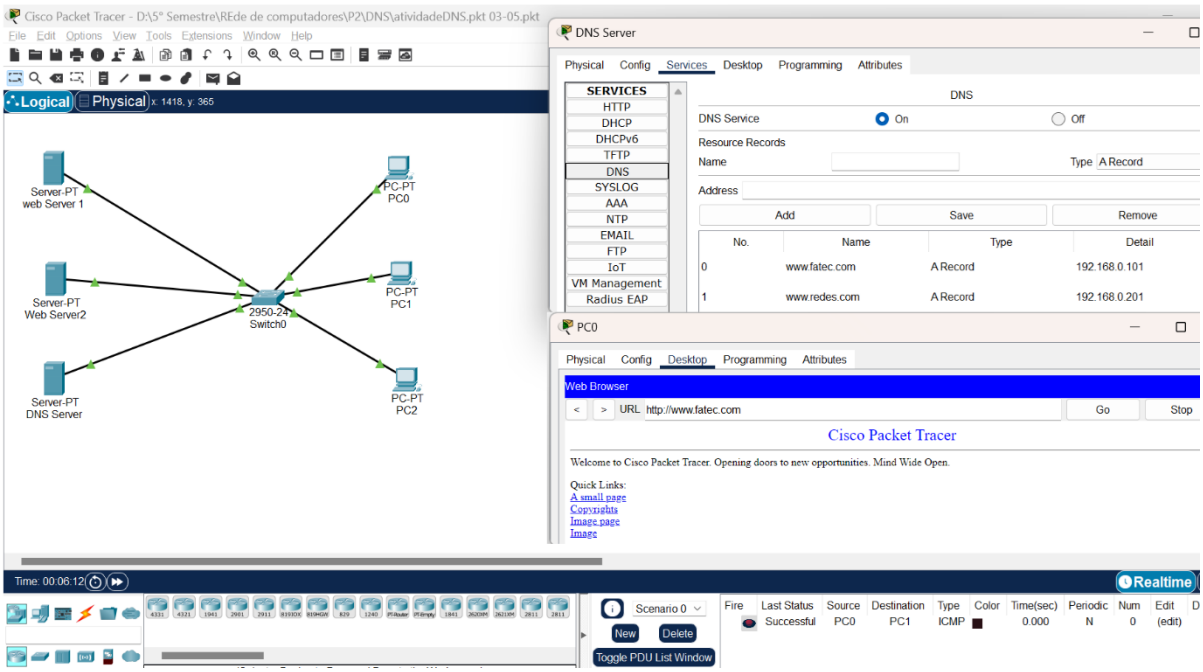
Fonte: Elaborada pela autora, 2024

6.4.4 Sistema de Nomes de Domínio (DNS)

O Sistema de Nomes de Domínio (DNS) é fundamental para a navegação na internet, funcionando como um tradutor que converte nomes de domínio legíveis por humanos como www.fatec.com em endereços IP legíveis por máquinas como 192.168.0.101. Na prática, quando um usuário digita um URL em seu navegador, o DNS entra em ação para localizar o servidor correspondente ao endereço digitado. Por exemplo, ao acessar www.fatec.com, o DNS consulta seu banco de dados para encontrar o endereço IP do servidor do Google, permitindo que a conexão seja estabelecida.

A figura 6.11 ilustra um exemplo prático da utilização de um DNS.

Figura 6.11 - Exemplo prático de DNS



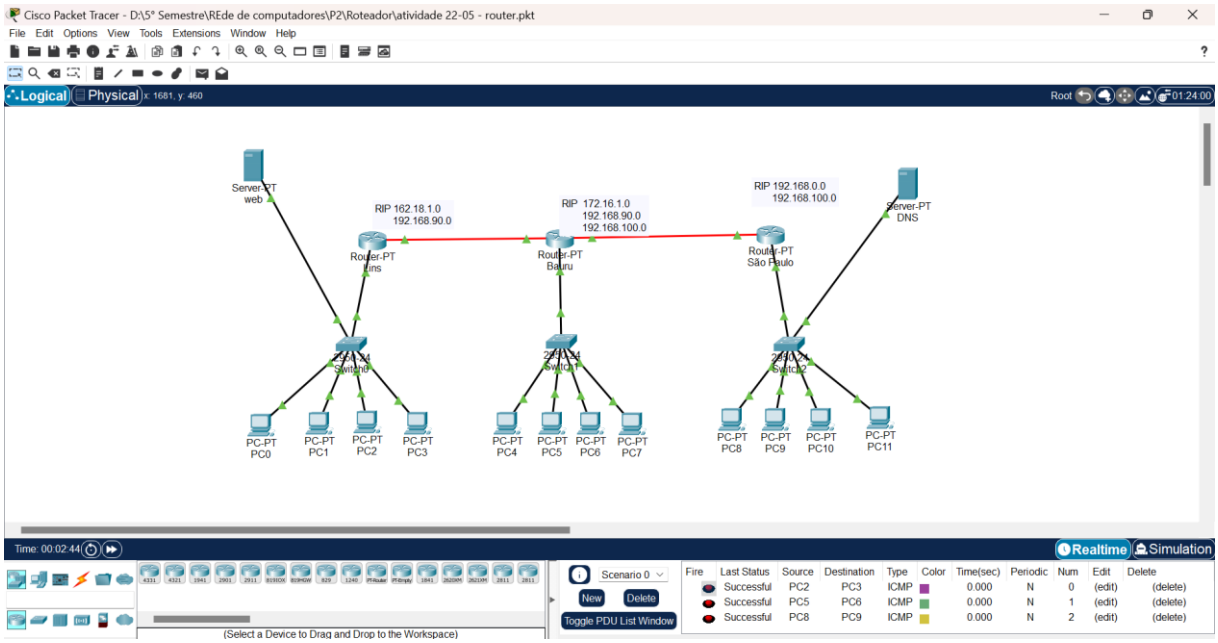
Fonte: Elaborada pela autora, 2024

6.4.5 Router

Um roteador é um dispositivo que encaminha pacotes de dados entre redes diferentes, usando endereços IP para determinar o melhor caminho. Roteadores são fundamentais para a interconexão de redes locais e redes remotas.

A figura 6.12 ilustra um exemplo prático da utilização de Router.

Figura 6.12 - Exemplo prático de Router



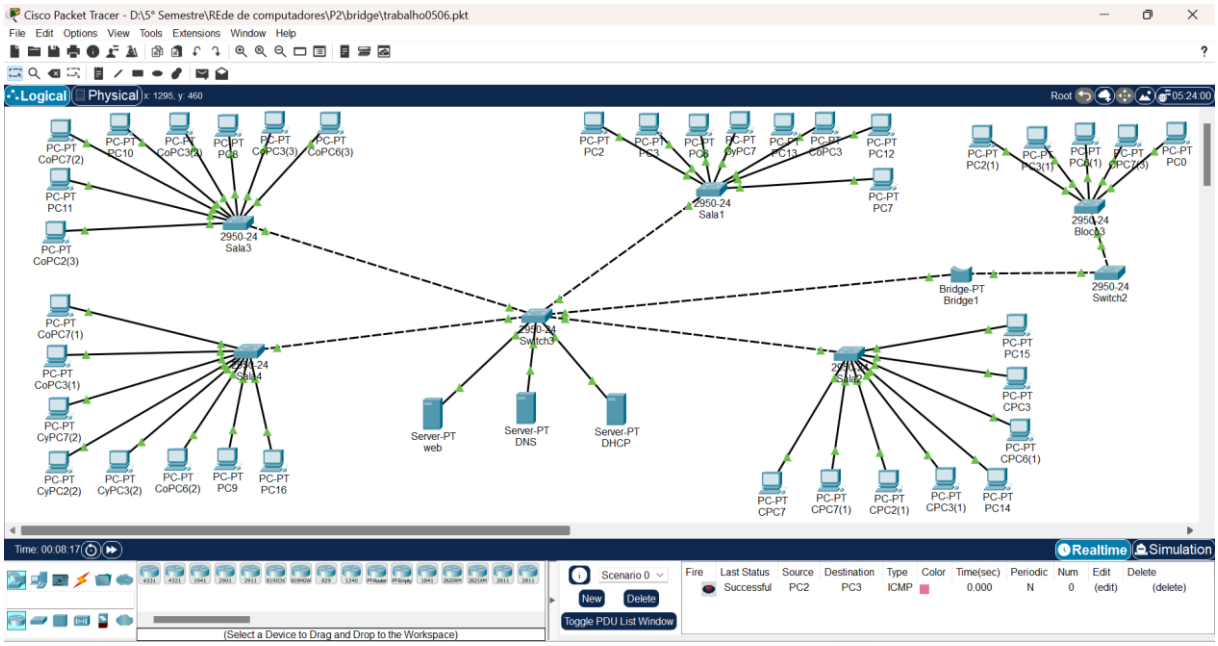
Fonte: Elaborada pela autora, 2024

6.4.6 Bridge

Uma bridge é um dispositivo que conecta duas ou mais redes diferentes dentro de uma mesma rede maior, fazendo com que elas funcionem como se fossem uma só. Analogicamente, pode-se pensar na bridge como uma ponte que une diferentes áreas de uma mesma rede, permitindo que elas se comuniquem de forma mais eficaz. É útil para segmentar redes grandes em menores, melhorando o desempenho e a organização.

A figura 6.13 ilustra um exemplo prático da utilização de um Bridge.

Figura 6.13 - Exemplo prático de Bridge



Fonte: Elaborada pela autora, 2024

6.5 CONCLUSÃO

O quinto semestre de Redes de Computadores proporcionou uma base sólida em conceitos e práticas essenciais. Através de uma combinação de teoria e prática, foi possível desenvolver as competências necessárias para projetar, implementar e gerenciar redes de computadores eficazes.

7 GERENCIAMENTO DE PROJETOS

Durante o sexto semestre do curso de ADS, a disciplina de Gestão de Projetos se mostrou fundamental para o desenvolvimento de profissionais na área de tecnologia. O conhecimento nesta área é essencial para coordenar atividades, alocar recursos e garantir o cumprimento de prazos e objetivos, promovendo um diferencial competitivo no desenvolvimento de soluções tecnológicas.

> O gerenciamento de projetos é um conjunto de ferramentas gerenciais que permitem que a empresa desenvolva um conjunto de habilidades, incluindo conhecimento e capacidades individuais, destinados ao controle de eventos não repetitivos, únicos e complexos, dentro de um cenário de tempo, custo e qualidade predeterminados. (Vargas, 2014, p.5).

Essa disciplina proporcionou uma visão estruturada de como planejar, monitorar e concluir projetos, aplicando técnicas como a gestão de escopo, cronograma, partes interessadas e comunicação. Neste capítulo, exploramos a importância de cada uma dessas áreas, enfatizando seu papel no desenvolvimento de projetos.

7.1 GESTÃO DE ESCOPO

A gestão de escopo é uma das bases de qualquer projeto bem-sucedido, uma vez que define os limites e os objetivos do projeto, especificando o que será e o que não será entregue. Sem uma gestão de escopo precisa, o projeto está sujeito a sofrer com "escopo flutuante", onde requisitos adicionais podem surgir, comprometendo o cronograma e o orçamento.

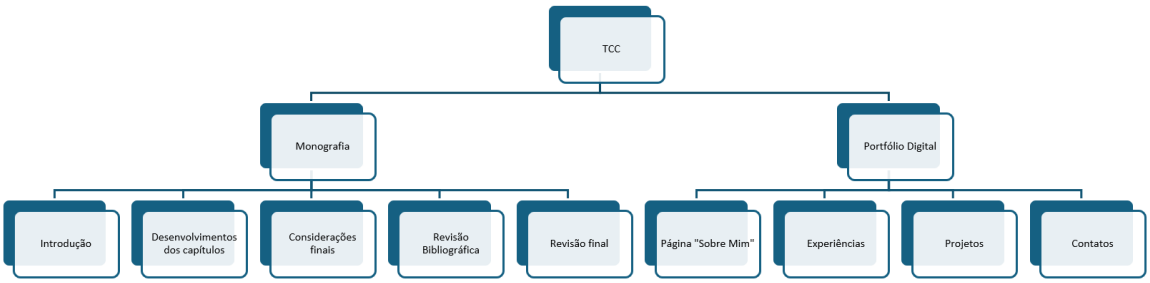
Durante o curso, foi abordado como estruturar e documentar o escopo, garantindo que todos os envolvidos tenham uma visão clara das entregas esperadas. Aprender a delimitar escopos específicos, alinhar as expectativas do cliente e da equipe técnica, proporciona uma base sólida para a execução do projeto. Dessa forma, os alunos desenvolvem uma capacidade crítica de identificar e documentar os requisitos essenciais, minimizando riscos e orientando as atividades de desenvolvimento.

Para garantir que o escopo seja cumprido de maneira organizada e eficiente, é fundamental dividir o projeto em partes menores e mais gerenciáveis, o que é viabilizado pelo uso da Estrutura Analítica do Projeto (EAP).

A **EAP** é uma ferramenta que traduz o escopo do projeto em uma hierarquia organizada de tarefas e sub-tarefas. Ela permite que o escopo seja desmembrado em componentes específicos, facilitando a alocação de recursos e o acompanhamento do progresso de cada etapa.

A figura 7.1 ilustra um exemplo da EAP desenvolvido para o projeto do Trabalho de conclusão de curso.

Figura 7.1 - EAP



Fonte: Elaborada pela autora, 2024

7.2 GESTÃO DAS PARTES INTERESSADAS

A gestão das partes interessadas é outra competência fundamental para o desenvolvimento de projetos, pois há uma diversidade de stakeholders, que incluem clientes, usuários finais, equipe técnica e patrocinadores. Cada um possui expectativas e interesses específicos, e a habilidade de gerenciá-los é muito importante para o sucesso do projeto.

Ao longo da disciplina, foram exploradas técnicas para identificar e analisar stakeholders, avaliando seu nível de poder e interesse. Dessa forma, é possível desenvolver estratégias de engajamento adequadas, que garantam o alinhamento entre as expectativas das partes interessadas e os objetivos do projeto. Esse conhecimento auxilia o profissional de ADS a evitar conflitos, mantendo o foco nas metas e promovendo um ambiente de colaboração.

A figura 7.2 ilustra o gerenciamento das partes interessadas.

Figura 7.2 - Gerenciamento das partes interessadas

		Identificação					Avaliação							
Cód.	Importância	Parte interessada	Área	Função	e-mail	Celular	Requisitos de Comunicação	Principais responsabilidades	Principais expectativas	Potencial impacto	Poder no projeto	Interesse no projeto	interna/ externa	Nível de engajamento	Comentários
1	25	Orientador	Computação	Docente	orientador@gmail.com	14999999	Cronograma, apresentação do desenvolvimento do trabalho escrito, apresentação do portifólio em desenvolvimento. Apresentação do portifólio digital.	Orientar quanto as normas de TCC, auxiliar nas dúvidas técnicas, ideiais e sugestões relacionadas ao projeto	Trabalho em conformidade com as normas da instituição de ensino, garantindo conteúdo bem estruturado, claro e entregue dentro dos prazos estabelecidos.	ALTO	5-Muito Alto	5-Muito Alto	Interna	Apoiador	
2	25	Banca Examinadora	Diversos	Docente	bancatcc@gmail.com	14999999874	Disponibilização de materiais para análise antes da apresentação.	Avaliar o trabalho com base nos critérios de TCC, oferecer feedback detalhado.	Entendimento claro dos conceitos e aplicação prática.	ALTO	5-Muito Alto	5-Muito Alto	Interna	Neutro	Importante se preparar bem para a defesa.
3	9	Professores de outras disciplinas	Diversos	Docente	professores@gmail.com	14988662254	Informações gerais sobre o andamento do TCC.	Contribuir com insights e sugestões relacionados às suas áreas de especialidade.	Integração dos conhecimentos de outras disciplinas no TCC.	MÉDIO	3-Médio	3-Médio	Interna	Neutro	
4	25	Aluno	Computação	Aluno	aluno@gmail.com	14999865236	Manter cronograma atualizado, registrar o progresso e revisões.	Conduzir o desenvolvimento do TCC com base nos requisitos estabelecidos.	Finalização do projeto com êxito.	ALTO	5-Muito Alto	5-Muito Alto	Interna	Lidera	
5	9	Família	Diversos	Apoio emocional			Acompanhamento periódico sobre o andamento do projeto.	Oferecer apoio moral e incentivo durante o desenvolvimento do TCC.	Conclusão do TCC e sucesso acadêmico.	MÉDIO	3-Médio	3-Médio	Externa	Apoiador	

Fonte: Elaborada pela autora, 2024

A gestão de interesse, juntamente com a análise dos stakeholders, permite que o profissional de ADS identifique pontos de atenção e alinhe os objetivos dos stakeholders com as entregas do projeto. Assim, o curso preparou os alunos para mediar conflitos, gerenciar expectativas e manter um relacionamento saudável com todos os envolvidos.

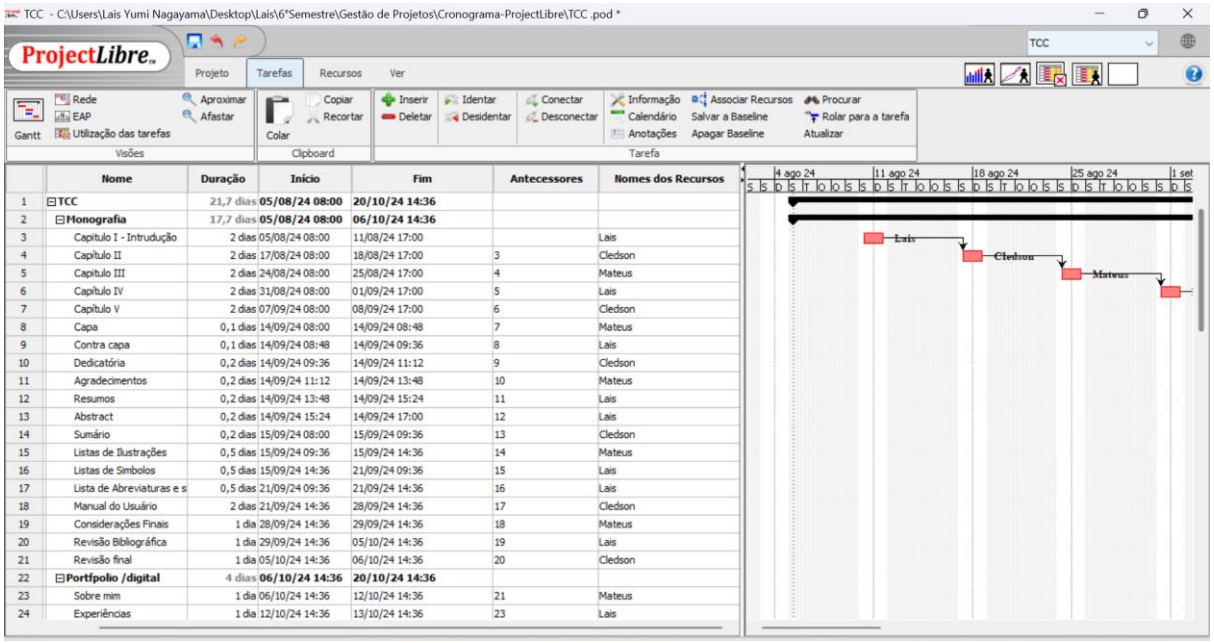
7.3 GESTÃO DE CRONOGRAMA

O cronograma é um dos elementos centrais de qualquer projeto. Durante o curso, foi introduzido o ProjectLibre, uma ferramenta de gestão de cronogramas que auxilia no planejamento e acompanhamento das atividades do projeto. Com essa ferramenta, os alunos puderam criar, monitorar e ajustar cronogramas de forma prática, simulando situações reais de gestão de projetos.

A utilização do ProjectLibre oferece uma visão de como as atividades se inter-relacionam e como um atraso em uma tarefa pode impactar o projeto como um todo. Ao aprender a manipular essa ferramenta, os alunos de ADS desenvolvem habilidades essenciais para manter o controle do cronograma.

A figura 7.3 ilustra o cronograma realizado com o ProjectLibre.

Figura 7.3 - Cronograma



Fonte: Elaborada pela autora, 2024

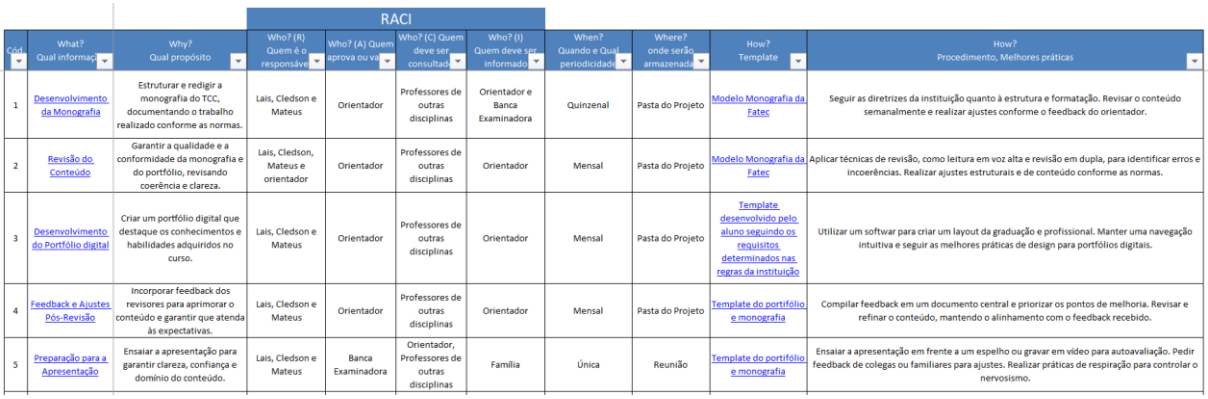
7.4 GESTÃO DA COMUNICAÇÃO

Em projetos complexos, a comunicação eficaz é um dos maiores desafios. Para facilitar o fluxo de informações entre as partes envolvidas, foi apresentada a Matriz RACI (Responsável, Aprovador, Consultado e Informado). A matriz RACI ajuda a definir as responsabilidades de cada membro da equipe, promovendo clareza e transparência.

A aplicação da matriz RACI é especialmente útil em equipes multidisciplinares, onde a responsabilidade de cada tarefa deve estar claramente estabelecida para evitar retrabalhos e atrasos. Com o uso desse modelo, os alunos aprendem a documentar e formalizar papéis dentro do projeto, permitindo um alinhamento eficaz entre os membros da equipe e facilitando a coordenação das atividades.

A figura 7.4 ilustra um exemplo de matriz RACI.

Figura 7.4 - Matriz RACI



			RACI							
Cód	What? Qual informaç	Why? Qual propósito	Who? (R) Quem é o responsáve	Who? (A) Quem aprova ou va	Who? (C) Quem deve ser consultad	Who? (I) Quem deve ser informado	When? Quando e Qual periodicidad	Where? onde serão armazenada	How? Template	How? Procedimento, Melhores práticas
1	Desenvolvimento da Monografia	Estruturar e redigir a monografia do TCC, documentando o trabalho realizado conforme as normas.	Lais, Cledson e Mateus	Orientador	Professores de outras disciplinas	Orientador e Banca Examinadora	Quinzenal	Pasta do Projeto	Modelo Monografia da Fatec	Seguir as diretrizes da instituição quanto à estrutura e formatação. Revisar o conteúdo semanalmente e realizar ajustes conforme o feedback do orientador.
2	Revisão do Conteúdo	Garantir a qualidade e a conformidade da monografia e do portfólio, revisando coerência e clareza.	Lais, Cledson, Mateus e orientador	Orientador	Professores de outras disciplinas	Orientador	Mensal	Pasta do Projeto	Modelo Monografia da Fatec	Aplicar técnicas de revisão, como leitura em voz alta e revisão em dupla, para identificar erros e incoerências. Realizar ajustes estruturais e de conteúdo conforme as normas.
3	Desenvolvimento do Portfólio digital	Criar um portfólio digital que destaque os conhecimentos e habilidades adquiridos no curso.	Lais, Cledson e Mateus	Orientador	Professores de outras disciplinas	Orientador	Mensal	Pasta do Projeto	Template desenvolvido pelo aluno seguindo os requisitos determinados nas regras da instituição	Utilizar um softwar para criar um layout da graduação e profissional. Manter uma navegação intuitiva e seguir as melhores práticas de design para portfólios digitais.
4	Feedback e Ajustes Pós-Revisão	Incorporar feedback dos revisores para aprimorar o conteúdo e garantir que atenda às expectativas.	Lais, Cledson e Mateus	Orientador	Professores de outras disciplinas	Orientador	Mensal	Pasta do Projeto	Template do portifólio e monografia	Compilar feedback em um documento central e priorizar os pontos de melhoria. Revisar e refinar o conteúdo, mantendo o alinhamento com o feedback recebido.
5	Preparação para a Apresentação	Ensaiar a apresentação para garantir clareza, confiança e domínio do conteúdo.	Lais, Cledson e Mateus	Banca Examinadora	Orientador, Professores de outras disciplinas	Família	Única	Reunião	Template do portifólio e monografia	Ensaiar a apresentação em frente a um espelho ou gravar em vídeo para autoavaliação. Pedir feedback de colegas ou familiares para ajustes. Realizar práticas de respiração para controlar o nervosismo.

Fonte: Elaborada pela autora, 2024

7.5 CONCLUSÃO

A disciplina de Gerenciamento de Projetos no curso de ADS oferece um conjunto de habilidades que vai além do domínio técnico. Ela capacita os futuros profissionais para planejar, monitorar e gerenciar os diferentes aspectos de um projeto de desenvolvimento de software, promovendo entregas eficientes e alinhadas às expectativas das partes interessadas. Com a aplicação de ferramentas e técnicas como o ProjectLibre, a matriz RACI e a EAP, os alunos são preparados para enfrentar os desafios do mercado de trabalho, destacando-se pela capacidade de gerenciar projetos de forma organizada, colaborativa e focada em resultados.

Ao concluir essa disciplina, os estudantes de ADS não apenas adquirem um conhecimento técnico de gestão de projetos, mas também uma visão estratégica que os torna aptos a contribuir para o sucesso das organizações onde atuarão.

8 MANUAL DO USUÁRIO

Este manual tem como objetivo orientar os usuários na navegação e utilização das principais funcionalidades do portfólio digital desenvolvido para demonstrar as informações profissionais e os conhecimentos adquiridos ao longo do curso de Análise e Desenvolvimento de Sistemas.

Estruturado em HTML, CSS e JavaScript, o portfólio é responsivo e utiliza uma interface amigável.

A estrutura principal é definida no arquivo index.html, que organiza cada seção e carrega os recursos de estilo e interatividade.

Os estilos são distribuídos em arquivos de CSS para modularidade e flexibilidade, como style.css (estilo geral), responsive.css (responsividade), font-awesome.min.css e flaticon.css (ícones).

Scripts de navegação e animação, como bootstrap.min.js, custom.js, e owl.carousel.min.js, contribuem para a funcionalidade e experiência interativa do usuário. A biblioteca jQuery auxilia em funcionalidades dinâmicas, como o efeito de rolagem suave e transições animadas.

8.1 MENU PORTFÓLIO

O menu de navegação permite que o usuário acesse diretamente qualquer seção do portfólio sem precisar rolar manualmente a página. Basta clicar em um item do menu para que a página role suavemente até a seção correspondente.

O jquerysticky.js permite que a barra de navegação permaneça visível mesmo ao rolar a página, facilitando a troca rápida entre as seções.

A figura 8.1 ilustra a barra de navegação do portfólio.

Figura 8.1 – Barra de navegação

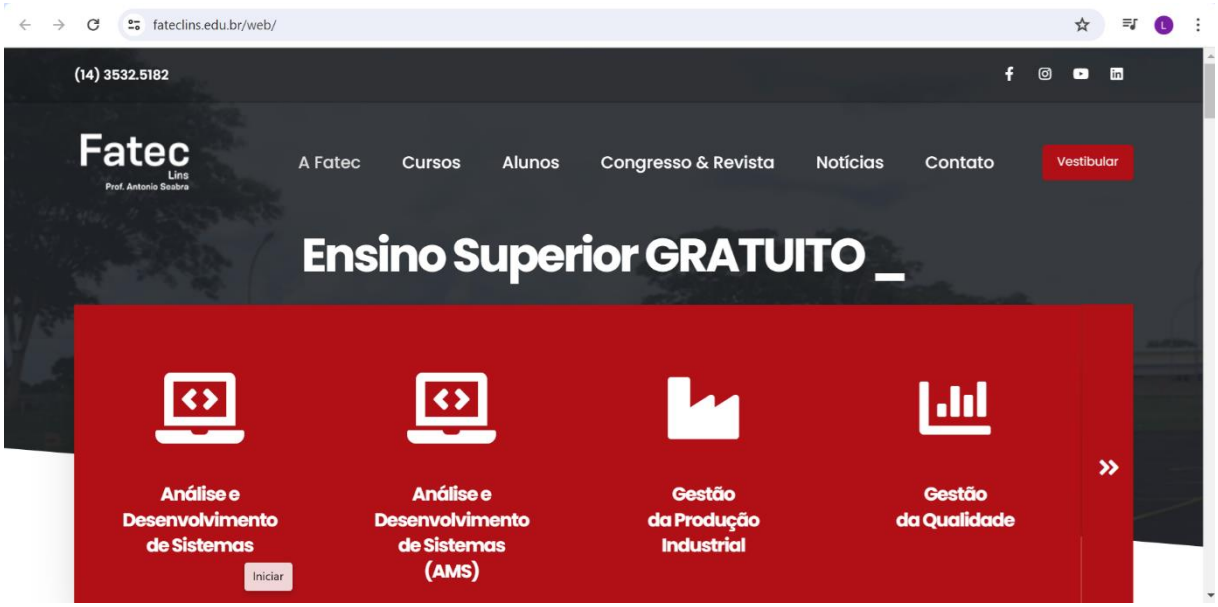
Fonte: Elaborada pela autora, 2024

A barra de navegação inclui um link direto para o site oficial da Fatec, acessível ao clicar no logótipo da instituição. Esse recurso facilita o acesso dos usuários a

informações sobre a Fatec, como detalhes institucionais, programas de ensino e cursos oferecidos, proporcionando uma experiência de navegação prática e intuitiva.

A Figura 8.2 ilustra o site da Fatec, acessada por meio do redirecionamento presente no portfólio.

Figura 8.2 – Site Fatec



Fonte: Elaborada pela autora, 2024

8.2 PÁGINA INICIAL

A página inicial contém uma breve introdução que inclui uma saudação e uma frase de apresentação. Nesta seção há um botão que permite baixar o currículo em PDF de forma simples. Basta clicar em "Download Currículo" para iniciar o download.

A figura 8.3 ilustra a página inicial.

Figura 8.3 – Página Inicial

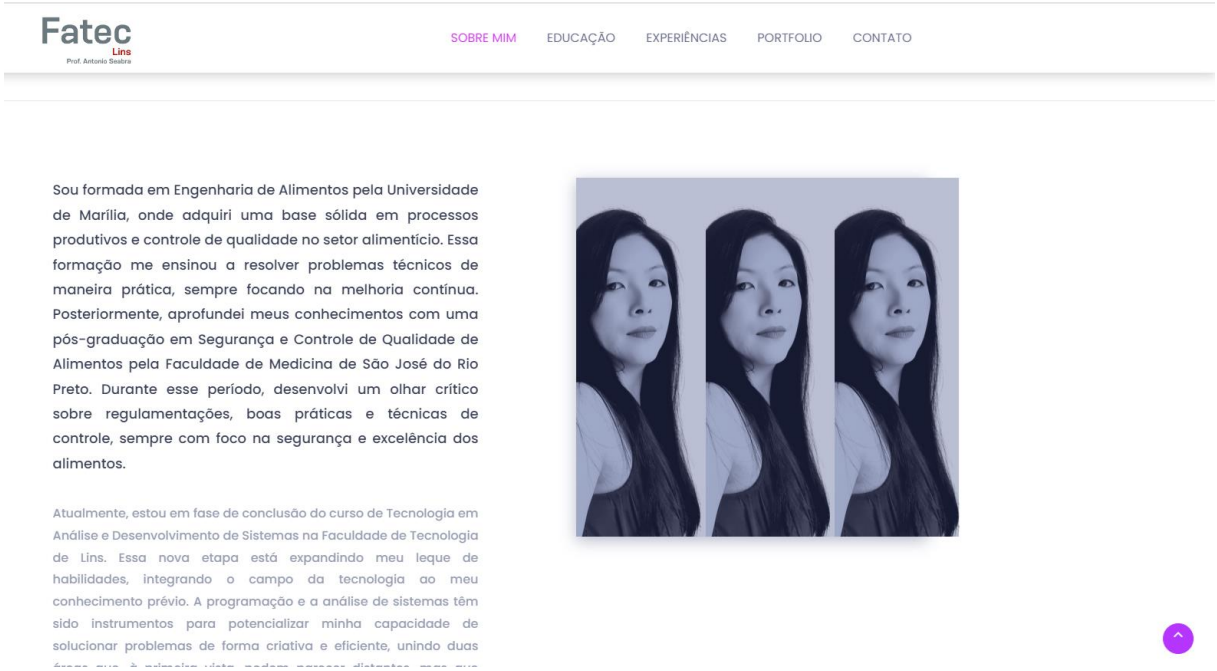
Fonte: Elaborada pela autora, 2024

8.3 SEÇÃO SOBRE MIM

A seção “Sobre mim” contém um resumo sobre a trajetória acadêmica e profissional, destacando habilidades e interesses.

A figura 8.4 ilustra a seção “Sobre mim”.

Figura 8.4 – Seção Sobre mim



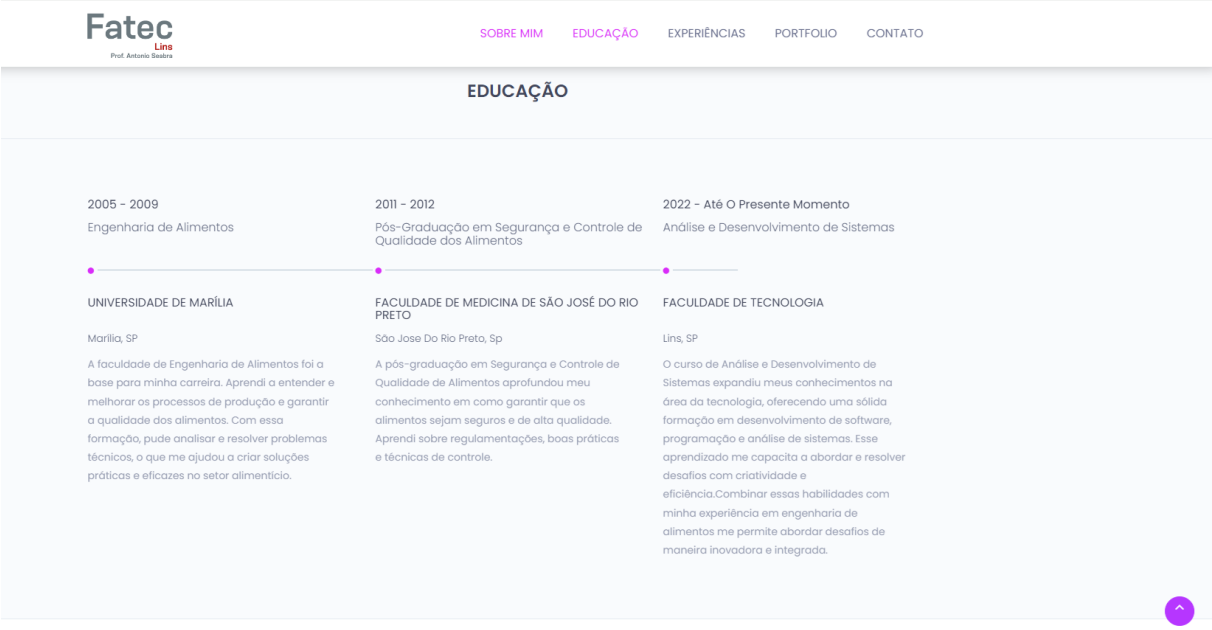
Fonte: Elaborada pela autora, 2024

8.4 SEÇÃO EDUCAÇÃO

A seção “Educação” contém detalhes sobre a formação acadêmica, incluindo cursos de graduação e pós-graduação.

A figura 8.5 ilustra a seção “Educação”.

Figura 8.5 – Seção Educação



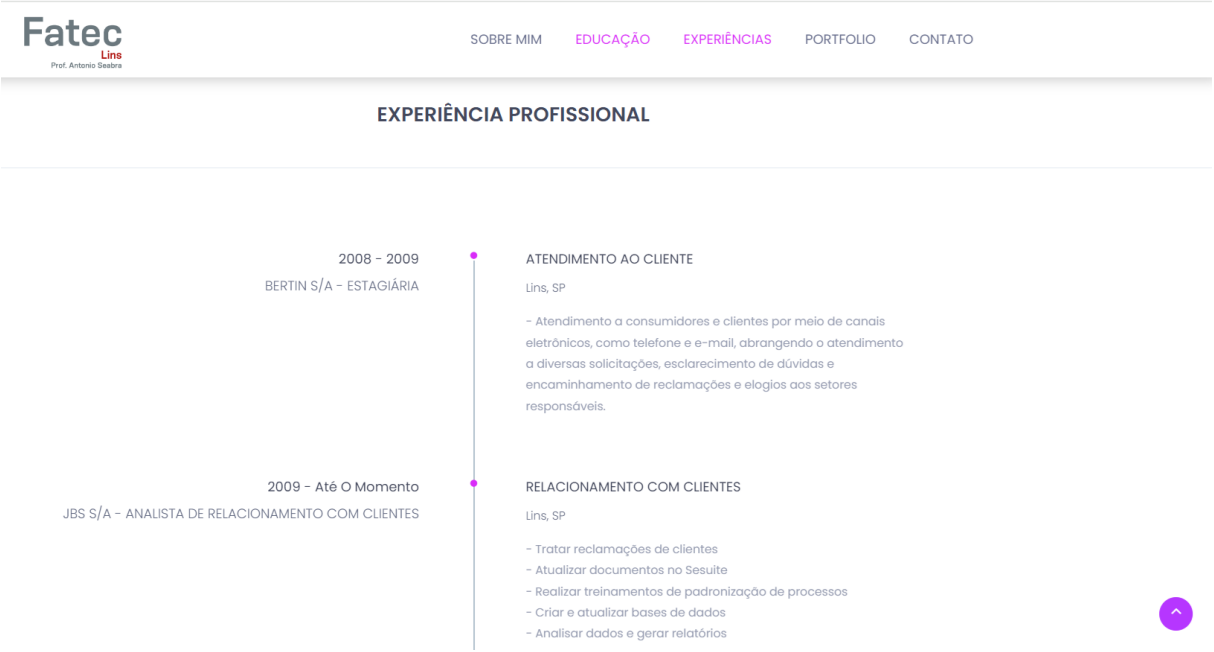
Fonte: Elaborada pela autora, 2024

8.5 SEÇÃO EXPERIÊNCIA PROFISSIONAL

A seção “Experiência” contém uma linha do tempo com os principais cargos e funções desempenhadas ao longo da carreira.

A figura 8.6 ilustra a seção “Experiência”.

Figura 8.6 – Seção Experiência



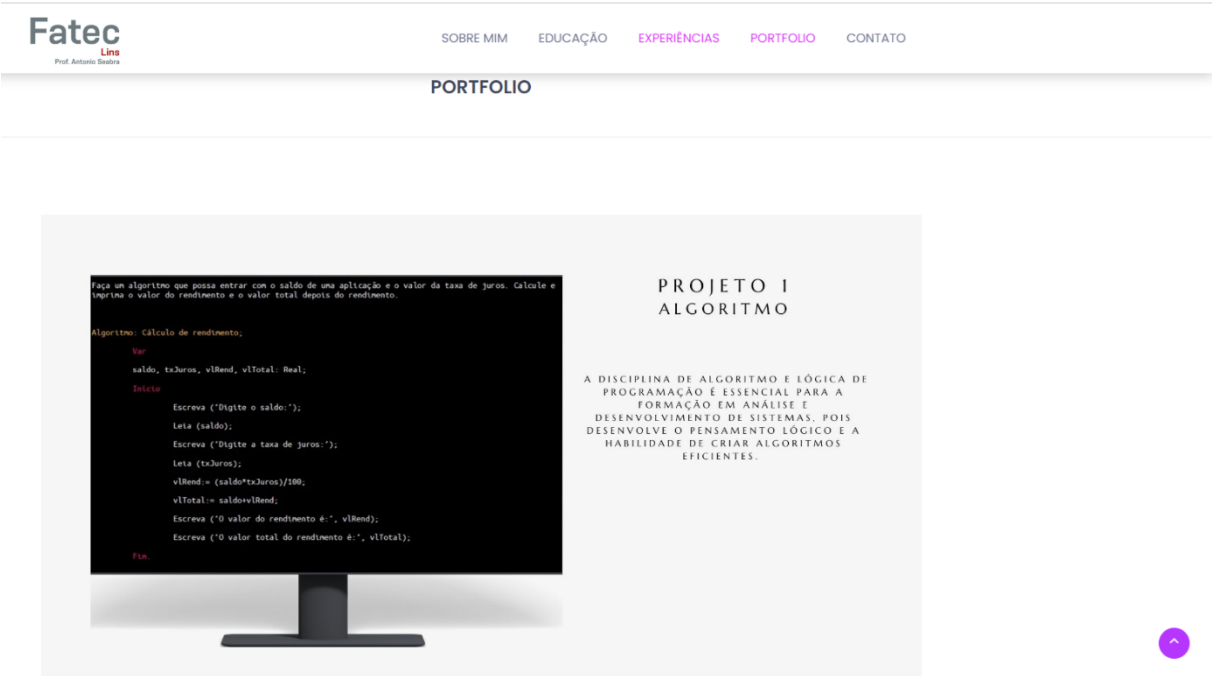
Fonte: Elaborada pela autora, 2024

8.6 SEÇÃO PORTFÓLIO

A seção “Portfólio” apresenta uma galeria dos projetos realizados ao longo do curso de ADS, abrangendo disciplinas como Algoritmo, Linguagem de Programação, Estrutura de Dados, Banco de Dados, Redes de Computadores e Gestão de Projetos. Nela, o visitante pode explorar cada projeto através de imagens representativas e descrições breves que destacam os principais aspectos desenvolvidos.

A figura 8.7 ilustra um dos projetos da seção “Portfólio”.

Figura 8.7 – Seção Portfólio



Fonte: Elaborada pela autora, 2024

A sessão Portfólio também conta um botão que permite o download deste portfólio em PDF. A figura 8.8 ilustra a funcionalidade de download do portfólio em PDF.

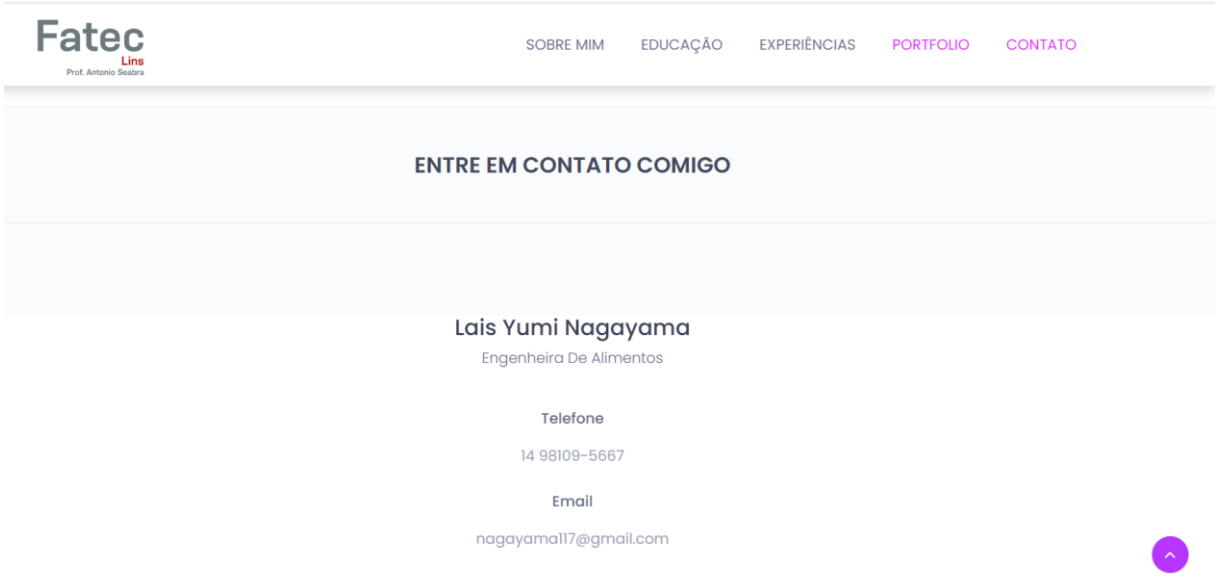
Figura 8.8 – Botão de download do portfólio

Fonte: Elaborada pela autora, 2024

8.7 SEÇÃO CONTATO

A seção “Contato” disponibiliza os dados de contato como telefone e e-mail. A figura 8.9 ilustra a seção “Contato”.

Figura 8.9 – Seção Contato



Fonte: Elaborada pela autora, 2024

9 CONCLUSÃO GERAL

Este Trabalho de Conclusão de Curso (TCC) apresenta uma visão abrangente dos conhecimentos adquiridos ao longo do curso de Análise e Desenvolvimento de Sistemas, cobrindo disciplinas como Algoritmo e Lógica de Programação, Linguagem de Programação, Estruturas de Dados, Banco de Dados, Redes de Computadores e Gestão de Projetos. Desde o início, este curso foi encarado como uma extensão dos conhecimentos adquiridos em minha formação em Engenharia de Alimentos, com o propósito claro de trabalhar em projetos estratégicos na empresa onde atuo.

O estudo de Algoritmo e Lógica de Programação estabeleceu uma base sólida para a resolução eficiente de problemas computacionais, essencial para estruturar e implementar soluções eficazes. Em Linguagem de Programação, foi possível traduzir esses algoritmos em código real, aprimorando habilidades na escrita e depuração de programas para sistemas funcionais.

Em Estruturas de Dados, a compreensão sobre a organização e manipulação de dados demonstrou como escolhas adequadas podem otimizar algoritmos e melhorar o desempenho do sistema. Já em Banco de Dados, exploramos o Modelo Entidade-Relacionamento (MER) e técnicas de mapeamento, garantindo que os bancos de dados fossem assertivos e eficientes, atendendo a requisitos de integridade.

O estudo de Redes de Computadores proporcionou uma visão prática e teórica sobre redes, topologias e dispositivos como switches, roteadores e servidores. A prática com ferramentas como o Cisco Packet Tracer consolidou o aprendizado e tornou possível configurar e gerenciar redes com eficiência. Por fim, Gestão de Projetos destacou a importância de ferramentas como a matriz RACI, a Estrutura Analítica de Projetos (EAP) e o uso de softwares como o ProjectLibre para organização e planejamento de projetos, habilidades essenciais para lidar com demandas de stakeholders e gerenciar recursos de maneira eficiente.

A criação de um portfólio digital consolidou a integração dos conhecimentos adquiridos, representando um marco na trajetória acadêmica. Este portfólio, com interface responsiva e amigável, expõe de maneira organizada os projetos desenvolvidos ao longo do curso, evidenciando não apenas as competências técnicas, mas também a capacidade de apresentar soluções de forma acessível e funcional.

Além disso, foi incluído um manual do usuário detalhado, garantindo uma experiência de navegação simplificada e refletindo o compromisso em oferecer uma solução completa e prática. Essa abordagem reforça o propósito que me guiou desde o início do curso: ampliar meus conhecimentos para atuar de forma estratégica em projetos, especialmente no contexto organizacional em que atuo.

Ao final desta jornada, posso afirmar com convicção que o curso de ADS cumpriu plenamente seu papel em permitir que eu alcançasse meus objetivos, fortalecendo minhas competências técnicas e ampliando minha visão profissional. Esta conquista não representa apenas o encerramento de uma etapa, mas a confirmação de que os conhecimentos adquiridos foram aplicados de maneira eficaz para transformar um propósito inicial em resultados concretos.

REFERÊNCIAS

CORMEN, T. H. et al. **Algoritmos: Teoria e Prática**. 3. ed. Rio de Janeiro: Elsevier, 2012.

ELMASRI, RAMEZ; NAVATHE, SHAMCANT B. **Sistemas de banco de dados.** 6 ed. São Paulo: Pearson Addison Wesley, 2011.

FORBELLONE, A.L.V.;EBERSPACHER, H.F. **Lógica de Programação**. 3 ed. São Paulo: Pearson Prentice Hall, 2005.

FOROUZAN, BEHROUZ A. **Comunicação de Dados e Redes de Computadores. 4 ed.** Porto Alegre: AMGH, 2010.

JUNIOR, P. J. **Curso Básico da Linguagem C**. 1 ed. São Paulo: Novaste Editora Ltda, 2019.

MACHADO, FELIPE NERY RODRIGUES. **Banco de dados: *Projetos e Implementação.* 3 ed.** São Paulo: Érica, 2014.

MISRAHI, VIVIANE VICTORINE. **Treinamento em Linguagem C.** 2 ed. São Paulo: Pearson Prentice Hall, 2008.

UFRN.. **Algoritmo e Lógica de Programação - Parte 1**. Apostila de Curso DCA 800 – Eng. Química. 2004. Disponível em: https://www.dca.ufrn.br/~affonso/DCA800/pdf/algoritmos_parte1.pdf . Acesso em: 03/03/2024

SCHILDT HERBERT. C Completo e Total. 3 ed. São Paulo: Pearson Education do Brasil, 1996.

SZWARCFITER, JAYME LUIZ; MARKENZON, LILIAN. **Estruturas de Dados e Seus Algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.

VARGAS, RICARDO VIANA. ***Gerenciamento de Projetos:*** *Estabelecendo Diferenciais Competitivos*. 7. ed. Rio de Janeiro: Brasport, 2014.