UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
Curso: Bacharelado em Engenharia da Computação

Relatório do sistema de gerenciamento da biblioteca

Disciplina: Laboratório de Algoritmos e Estrutura de Dados I

Professor: George Felipe Fernandes Vieira
Alunos: João Pedro Fernandes de Aquino;
Frederico Gregório Emídio Santos Ferreira;
Guilherme Silva Aguiar

Pau dos Ferros - RN
30/06/2025

Sumário

8. Arquivo "auxiliar.c" – Funções Utilitárias

8.1. ler_entrada_limitada

8.2. gerarNovoId

8.3. carregarLivros

8.4. buscarLivroPorISBN

8.5. carregarEmprestimos

1. Introdução

O sistema implementado permite o gerenciamento do sistema interno de uma biblioteca a partir de cadastros , atualizações, remoções de livros e empréstimos. Além disso, o sistema possibilita a geração de relatórios que possam ser do interesse da biblioteca a partir de parâmetros pré-estabelecidos. O sistema usa uma arquitetura modular dividida em:

- *tipos.h*: Que contém toda as estruturas de dados, e enums utilizados.
- *Main.c*: Menu inicial
- *emprestimos.c*: Gerenciamento dos empréstimos
- *livros.c*: Gerenciamento dos livros
- *relatorios.c*: Geração dos relatórios
- *tipos.c*: Funções de validação e recepção de dados
- *auxiliar.c*: Funções utilitárias ao código, incluindo: exibição de estruturas; verificação e criação de arquivos; recebimento e validação de entradas das estruturas; buscas e geração de id.

Arquivos (emprestimos*.txt, livros.txt*) no diretório *Arquivos.* Diretório *Arquivos* para armazenar os empréstimos e os livros.

2. Arquivo "tipos.h" – Estruturas de dados utilizadas

2.1. Livro

A estrutura *Livro* armazena informações sobre os livros que estão disponíveis na biblioteca. Cada livro possui um identificador único (*id*), um titulo (*titulo*), um autor(*autor*) e um identificador ISBN (*ISBN*). Além disso, o gênero do livro é definida por uma *enum* (*GENERO*), que pode ser um dos seguintes: *FICCAO, DIDATICO, BIOGRAFIA, COMEDIA, TERROR, ROMANCE*. Essa categorização permite que o sistema identifique rapidamente o gênero de cada livro.

2.2. Emprestimos

A estrutura *Emprestimos* armazena informações sobre os emprestimos que irão ser realizados na biblioteca. Cada empréstimo possui um identificador único (*id*), um leitor (*leitor*), um identificador ISBN (*ISBN*), a data do empréstimo são armazenadas em uma estrutura *Data*, que inclui o dia (*dia*), mês (*mes*), ano (*ano*). Além disso, o Status do empréstimo é definida por uma *enum* (*STATUS*), que pode ser um dos seguintes: *CONCLUIDO, ANDAMENTO, CANCELADO*. Essa categorização permite que o sistema identifique rapidamente o status de cada empréstimo.

2.3.Data

A estrutura *Data* é responsável por armazenar a data de um empréstimo. Ela inclui o dia (*dia*), mês (*mes*), ano (*ano*). Esses dados são utilizados dentro do *struct Emprestimos* (1.2).

3. Arquivo "Main.c" – O menu inicial

Inclui bibliotecas padrão do C, como <stdio.h> e <stdlib.h>, <string.h>, <locale.h> , além de cabeçalhos locais: "tipos.h", "livros.h", “emprestimos.h", "relatorios.h" e "auxiliar.h", que definem estruturas de dados e funções específicas para cada módulo do sistema.

A função única do arquivo main é implementar um menu interativo para o sistema de gerenciamento da biblioteca. Utiliza um loop *do-while* para manter o programa em execução até que o usuário selecione a opção de sair (0). A seleção é feita com *scanf()* e tratada por um *switch-case*, que direciona para os módulos: *menuLivros()*, *menuEmprestimos() e* menuRelatorios*()*. Cada função é responsável pela navegação e operações do respectivo módulo.

4. Arquivo "emprestimos.c" - Gerenciamento dos emprestimos

Os *includes* permitem as operações do módulo: <*stdio.h*> e <*stdlib.h*> habilitam entradas, saídas e gestão de memória. A biblioteca <*string.h*> fornece manipulação segura de strings, enquanto os cabeçalhos locais *"tipos.h"*, *"emprestimos.h"* e *"auxiliar.h"* definem estruturas de dados, declaram funções específicas e compartilham funções utilitárias.

4.1 *cadastrarEmprestimo*

A função implementa uma busca linear sobre um vetor de estruturas, realizando filtragem por campo enumerado e exibindo os resultados de forma formatada, com tratamento de exceções e gerenciamento de memória dinâmica. Inicialmente, a função aloca dinamicamente um vetor de estruturas do tipo *Emprestimos* e carrega todos os registros presentes no arquivo persistente utilizando a função *carregarEmprestimos()*. O usuário é solicitado a informar o status desejado para filtragem (por exemplo: ANDAMENTO, CONCLUIDO, ATRASADO). A função *receberStatus()* é utilizada para garantir que o valor informado seja válido e compatível com o enum STATUS. A função percorre todo o vetor de empréstimos, comparando o campo status de cada registro com o status informado pelo usuário. Para cada empréstimo que satisfaz o critério, são exibidas na tela as informações relevantes: ID do empréstimo, nome do leitor, ISBN do livro e data do empréstimo. Ao final, a função libera a memória alocada dinamicamente para o vetor de empréstimos.

4.2. *registrarDevol*

A função implementa um fluxo de atualização de status de empréstimos, realizando busca linear por ID, validação de status, confirmação do usuário e persistência dos dados, com gerenciamento de memória dinâmica e tratamento de exceções. Aloca dinamicamente um vetor de estruturas do tipo Emprestimos e carrega todos os registros de empréstimos existentes utilizando a função *carregarEmprestimos()*. Percorre o vetor de empréstimos e exibe todos os registros cujo status é ANDAMENTO, mostrando ID, nome do leitor, ISBN do livro e data do empréstimo, se não houver nenhum empréstimo em andamento, libera a memória e encerra a função. Solicita ao usuário o ID do empréstimo que deseja devolver e procura no vetor o empréstimo com o ID informado e se não encontrar, exibe uma mensagem de erro. Exibe os dados do empréstimo selecionado e solicita confirmação do usuário para registrar a devolução e se confirmado, altera o status do empréstimo para CONCLUIDO e salva todos os empréstimos atualizados no arquivo persistente e se por acaso o empréstimo não estiver com status ANDAMENTO, informa o status atual e não permite a devolução

4.3. listarEmprestimoPorStatus

A função implementa uma busca linear sobre um vetor de estruturas, realizando filtragem por campo enumerado e exibindo os resultados de forma formatada, com tratamento de exceções e gerenciamento de memória dinâmica. Inicialmente, a função aloca dinamicamente um vetor de estruturas do tipo Emprestimos e carrega todos os registros presentes no arquivo persistente utilizando a função *carregarEmprestimos()*. O usuário é solicitado a informar o status desejado para filtragem (por exemplo: ANDAMENTO, CONCLUIDO, ATRASADO). A função *receberStatus()* é utilizada para garantir que o valor informado seja válido e compatível com o enum STATUS. A função percorre todo o vetor de empréstimos, comparando o campo status de cada registro com o status informado pelo usuário. Para cada empréstimo que satisfaz o critério, são exibidas na tela as informações relevantes: ID do empréstimo, nome do leitor, ISBN do livro e data do empréstimo.

4.3. *menuEmprestimos*

Essa função exibe um menu com *printf()* contendo opções numéricas. Usa *do-while* com *scanf()* para manter o menu ativo até seleção de saída (opção 0). Para opções válidas, chama cadastrarEmprestimo*()*, registrarDevol*()*, listarEmprestimoPorStatus*().*

5. Arquivo "livros.c" – Gerenciamento dos livros

Os *includes* permitem as operações do módulo: <*stdio.h*> e <*stdlib.h*> habilitam entradas, saídas e gestão de memória. A biblioteca <*string.h*> fornece manipulação segura de strings, enquanto os cabeçalhos locais *"tipos.h"* e *"auxiliar.h"* definem estruturas de dados dos livros, declaram funções específicas e compartilham funções utilitárias.

5.1. salvarTodosLivros

A função abre o arquivo "arquivos/livros.txt" no modo escrita ("w"), o que faz com que o conteúdo anterior do arquivo seja apagado e um novo conteúdo seja escrito a partir do início. Antes de gravar os dados dos livros, a função escreve uma linha de cabeçalho no arquivo, contendo os nomes dos campos: id,ISBN,titulo,autor,genero. Isso facilita a leitura e manutenção do arquivo posteriormente. A função percorre o vetor de estruturas do tipo Livro recebido como parâmetro, utilizando um laço for. Para cada elemento do vetor, ela escreve uma linha no arquivo contendo os dados do livro, separados por vírgula, seguindo o padrão CSV (Comma-Separated Values). Os campos gravados são: ID, ISBN, título, autor e gênero. Após gravar todos os registros, a função fecha o arquivo utilizando fclose, garantindo que todos os dados sejam efetivamente salvos e liberando o recurso do sistema operacional.

5.2 *salvarNovoLivro*

A função abre o arquivo "arquivos/livros.txt" no modo de adição ("a"), garantindo que o novo registro será acrescentado ao final do arquivo, preservando os dados já existentes. Antes de gravar o novo livro, a função utiliza gerarNovoId("arquivos/livros.txt") para obter um identificador único para o novo registo, evitando duplicidade de IDs. O novo livro, representado por uma estrutura do tipo Livro, tem seus campos (ID, ISBN, título, autor e gênero) gravados em uma nova linha do arquivo.

5.3 *cadastrarLivro*

Aloca dinamicamente uma estrutura do tipo Livro para armazenar temporariamente os dados do novo livro a ser cadastrado. Solicita ao usuário o ISBN do livro, utilizando a função *receberISBN()*, e verifica se o ISBN já está cadastrado no sistema, garantindo unicidade. Solicita o título do livro com a função *receberTitulo()*. Solicita o nome do autor com a função *receberNome()*. Solicita o gênero do livro com a função *receberGenero()*. Exibe os dados coletados utilizando a função *exibirLivro()* para que o usuário possa revisar as informações antes de confirmar o cadastro. Pergunta ao usuário se deseja salvar os dados ou descartar o cadastro. Se o usuário optar por salvar, chama a função *salvarNovoLivro()*, que persiste o novo registro no arquivo de livros. Se o usuário optar por descartar, libera a memória e encerra a função sem salvar.

5.4 *removerLivro*

A função *removerLivro()* permite ao usuário excluir um livro do sistema informando o ISBN. Ela carrega todos os livros do arquivo usando *carregarLivros()*, solicita o ISBN com *receberISBN()*, busca o livro usando *buscarLivroPorISBN()*, exibe os dados do livro encontrado com *exibirLivro()* e pede confirmação da remoção. Se confirmado, cria um novo vetor sem o livro removido, salva esse novo vetor sobrescrevendo o arquivo original com *salvarTodosLivros()* e libera a memória utilizada. Se o livro não for encontrado ou a operação for cancelada, apenas exibe uma mensagem e encerra.

5.5 *atualizarLivro*

A função *atualizarLivro()* permite ao usuário modificar os dados de um livro já cadastrado no sistema. Ela carrega todos os livros do arquivo usando *carregarLivros()*, solicita o ISBN do livro a ser atualizado com *receberISBN()* e busca o livro usando *buscarLivroPorISBN()*. Se encontrado, exibe os dados atuais e permite ao usuário escolher qual campo deseja alterar (ISBN, título, autor ou gênero), utilizando funções como *receberISBN()*, *receberTitulo()*, *receberNome()* e *receberGenero()* para coletar os novos valores. Após as alterações, pergunta se o usuário deseja salvar as mudanças; se confirmado, salva o vetor atualizado com *salvarTodosLivros()*. Por fim, libera a memória utilizada.

5.6 *MenuLivros*

Essa função exibe menu com *printf()* contendo opções numéricas. Usa *do-while* com *scanf()* para manter o menu ativo até seleção de saída (opção 0). Para opções válidas, chama *cadastrarLivro()*, removerLivro*() ou* atualizarLivro*()*.

6. Arquivo “relatorios.c” - Geração de relatórios analíticos

O cabeçalho do código contém as inclusões essenciais para o funcionamento do módulo de relatórios do sistema. São incluídas as bibliotecas padrão stdio.h (entrada e saída), stdlib.h (alocação dinâmica), string.h (manipulação de strings). Além disso, são incluídos os arquivos personalizados tipos.h, relatorios.h, auxiliar.h, que definem estruturas, funções auxiliares e funcionalidades relacionadas ao gerenciamento de relatórios.

6.1. MenuRelatorios

Essa função exibe menu com *printf()* contendo opções numéricas. Usa *do-while* com *scanf()* para manter o menu ativo até seleção de saída (opção 0). Para opções válidas, chama *listarLivrosPorGenero() ou listarLivrosPorStatus().*

6.2. listarLivrosPorGenero

A função *listarLivrosPorGenero()* gera um relatório mostrando todos os livros cadastrados de um determinado gênero. Ela carrega todos os livros do arquivo usando *carregarLivros()*, solicita ao usuário o gênero desejado com *receberGenero()* e percorre o vetor de livros, exibindo com *exibirLivro()* apenas aqueles cujo campo genero corresponde ao gênero informado. Ao final, mostra o total de livros encontrados para o gênero escolhido e libera a memória alocada. Se nenhum livro do gênero for encontrado, exibe uma mensagem informando isso.

6.3. listarLivrosPorStatus

A função *listarLivrosPorStatus()* gera um relatório mostrando todos os livros cadastrados e o status de cada um (Disponível ou Emprestado). Ela carrega todos os livros do arquivo usando *carregarLivros()* e todos os empréstimos usando *carregarEmprestimos()*. Para cada livro, verifica se existe algum empréstimo em andamento (ANDAMENTO) com o mesmo ISBN; se houver, marca o livro como "Emprestado", caso contrário, marca como "Disponível". Exibe na tela o título, ISBN e status de cada livro. Ao final, libera a memória alocada para os vetores de livros e empréstimos.

7. Arquivo “tipos.c” – Funções de validação e recepção de dados

O cabeçalho do código contém as *inclusões* essenciais para o funcionamento do sistema de consultas médicas. São incluídas as bibliotecas padrão de entrada e saída (*stdio.h*), manipulação de strings (*string.h*) e alocação dinâmica de memória (*stdlib.h*). Além disso, há a inclusão dos arquivos de cabeçalho personalizados — *tipos.h* e *auxiliar.h* — que definem as estruturas, funções auxiliares necessárias para o correto funcionamento das operações.

7.1. generoParaTexto

A função *generoParaTexto(enum GENERO genero)* recebe um valor do tipo enumeração GENERO e retorna uma string correspondente ao nome textual desse gênero. Ela utiliza um switch para mapear cada valor do enum (como FICCAO, DIDATICO, BIOGRAFIA, etc.) para uma string descritiva, como "Ficção", "Didático", "Biografia" e assim por diante. Se o valor informado não corresponder a nenhum caso conhecido, retorna "DESCONHECIDO". Essa função facilita a exibição do nome do gênero em relatórios e telas para o usuário, convertendo o valor numérico do enum em texto legível.

7.2. receberGenero

A função *receberGenero()* exibe uma lista numerada de todos os gêneros disponíveis (usando a função *generoParaTexto()*), solicita ao usuário que digite o número correspondente ao gênero desejado, valida se a opção está dentro do intervalo permitido e, se válida, armazena o valor selecionado no ponteiro passado como argumento. O processo se repete até que o usuário informe uma opção válida, garantindo assim a correta seleção do gênero do livro.

7.3. validarISBN

A função *validarISBN(const char *isbn)* verifica se o ISBN informado é válido. Ela retorna 1 (verdadeiro) se o ISBN tiver exatamente 13 caracteres e todos forem dígitos numéricos; caso contrário, retorna 0 (falso). Isso garante que apenas ISBNs no formato correto sejam aceitos no sistema.

7.4. receberISBN

A função *receberISBN(char *ISBN)* solicita ao usuário que digite o ISBN do livro, lê a entrada usando *ler_entrada_limitada()*, e valida o valor com a função *validarISBN()*. O processo se repete até que o usuário informe um ISBN válido (ou seja, exatamente 13 dígitos numéricos). Quando um ISBN válido é digitado, ele é armazenado na variável passada por parâmetro. Isso garante que apenas ISBNs corretos sejam aceitos no sistema.

7.5. validarTitulo

A função *validarTitulo(const char *nome)* verifica se o título informado é válido. Ela retorna 1 (verdadeiro) se o título não estiver vazio, não exceder 199 caracteres e não for composto apenas por espaços. Caso contrário, retorna 0 (falso). Isso garante que apenas títulos preenchidos corretamente sejam aceitos no sistema.

7.6. receberTitulo

A função *receberTitulo(char *nome)* solicita ao usuário que digite o título do livro, lê a entrada usando *ler_entrada_limitada()*, e valida o valor com a função *validarTitulo()*. O processo se repete até que o usuário informe um título válido (não vazio, com até 199 caracteres e não composto apenas por espaços). Quando um título válido é digitado, ele é armazenado na variável passada por parâmetro. Isso garante que apenas títulos corretos sejam aceitos no sistema.

7.7. exibirLivro

A função *exibirLivro(Livro livro)* exibe na tela as informações de um livro formatadas para o usuário. Ela mostra o ISBN, o título, o autor e o gênero do livro (convertendo o valor do enum para texto usando a função *generoParaTexto()*). Isso facilita a visualização dos dados completos de um livro cadastrado no sistema.

7.8. statusParaTexto

A função *statusParaTexto(enum STATUS status)* recebe um valor do tipo enumeração STATUS e retorna uma string correspondente ao nome textual desse status. Ela utiliza um switch para mapear cada valor do enum (como CANCELADO, ANDAMENTO, CONCLUIDO) para uma string descritiva ("CANCELADO", "EM ANDAMENTO", "CONCLUÍDO"). Se o valor informado não corresponder a nenhum caso conhecido, retorna "DESCONHECIDO". Essa função facilita a exibição do status do empréstimo de forma legível para o usuário.

7.9. receberStatus

A função *receberStatus(enum STATUS *status)* exibe uma lista numerada com todos os status possíveis (usando a função *statusParaTexto()*), solicita ao usuário que digite o número correspondente ao status desejado, valida se a opção está dentro do intervalo permitido e, se válida, armazena o valor selecionado no ponteiro passado como argumento. O processo se repete até que o usuário informe uma opção válida, garantindo assim a correta seleção do status do empréstimo.

7.10. validarHora

A função *validarHora(int hora, int minuto)* verifica se os valores informados para hora e minuto representam um horário válido. Ela retorna 1 (verdadeiro) se a hora estiver entre 0 e 23 e o minuto entre 0 e 59; caso contrário, retorna 0 (falso). Isso garante que apenas horários válidos sejam aceitos no sistema.

7.11. validarData

A função *validarData(int dia, int mes, int ano)* verifica se uma data informada é válida. Ela retorna 1 (verdadeiro) se o ano estiver entre 1900 e 2100, o mês entre 1 e 12, e o dia for compatível com o mês e o ano (considerando anos bissextos para fevereiro). Caso contrário, retorna 0 (falso). Isso garante que apenas datas reais e possíveis sejam aceitas no sistema.

7.12. receberData

A função *receberData(Data *dh)* solicita ao usuário que digite uma data no formato DD/MM/AAAA, lê a entrada como string, faz o parsing para dia, mês e ano usando sscanf, e valida a data com a função *validarData()*. O processo se repete até que o usuário informe uma data válida. Quando uma data correta é digitada, ela é armazenada na estrutura Data passada por parâmetro, garantindo que apenas datas válidas sejam aceitas no sistema.

7.13. validarNome

A função *validarNome(const char *nome)* verifica se o nome informado é válido. Ela retorna 1 (verdadeiro) se o nome não estiver vazio, não exceder 199 caracteres, não for composto apenas por espaços e contiver apenas letras e espaços. Caso contrário, retorna 0 (falso). Isso garante que apenas nomes corretos sejam aceitos no sistema.

7.14. receberNome

A função *receberNome(char *nome)* solicita ao usuário que digite um nome (por exemplo, de autor ou leitor), lê a entrada usando *ler_entrada_limitada()*, e valida o valor com a função *validarNome()*. O processo se repete até que o usuário informe um nome válido (não vazio, com até 199 caracteres, apenas letras e espaços). Quando um nome válido é digitado, ele é armazenado na variável passada por parâmetro, garantindo que apenas nomes corretos sejam aceitos no sistema.

8. Arquivo “auxiliar.c" - As funções utilitárias do código

Este arquivo funciona como uma caixa de ferramentas que sustenta e da suporte a todo sistema. Nele estão concentradas as funções que garantem o funcionamento do programa.

8.1. ler_entrada_limitada

A função *ler_entrada_limitada* serve para ler uma string do usuário com um tamanho máximo definido, evitando estouro de buffer e problemas de leitura. Ela lê a entrada com fgets, remove o caractere de nova linha (\n) ao final, e se o usuário digitar mais caracteres do que o permitido, limpa o restante do buffer para evitar problemas em leituras futuras. Retorna 1 se a leitura foi bem-sucedida e 0 em caso de erro ou estouro do limite.

8.2. gerarNovoId

A função *gerarNovoId(const char *arquivoPath)* serve para gerar automaticamente um novo ID único para registros em arquivos (como livros ou empréstimos). Ela abre o arquivo indicado, lê todas as linhas ignorando o cabeçalho, extrai o primeiro campo (ID) de cada linha e mantém o maior valor encontrado. Ao final, retorna esse maior valor incrementado de 1, garantindo que o novo ID não será duplicado. Se o arquivo não existir ou não houver IDs válidos, retorna 1 como primeiro ID. Isso garante que cada novo registro terá um identificador único e sequencial no sistema.

8.3. carregarLivros

A função *carregarLivros(Livro **livros, int *total)* lê todos os registros de livros do arquivo "arquivos/livros.txt" e carrega esses dados em um vetor dinâmico de estruturas Livro. Ela abre o arquivo, ignora a primeira linha (cabeçalho), lê cada linha seguinte extraindo os campos (id, ISBN, título, autor e gênero), converte o gênero para o tipo enum correspondente e armazena cada livro no vetor, realocando memória conforme necessário. Ao final, atualiza o ponteiro do vetor e o total de livros lidos, permitindo que outras funções acessem todos os livros cadastrados em memória. Se o arquivo não existir, define o vetor como NULL e o total como zero.

8.3. carregarLivros

A função *carregarLivros(Livro **livros, int *total)* lê todos os registros de livros do arquivo "arquivos/livros.txt" e carrega esses dados em um vetor dinâmico de estruturas Livro. Ela abre o arquivo, ignora a primeira linha (cabeçalho), lê cada linha seguinte extraindo os campos (id, ISBN, título, autor e gênero), converte o gênero para o tipo enum correspondente e armazena cada livro no vetor, realocando memória conforme necessário. Ao final, atualiza o ponteiro do vetor e o total de livros lidos, permitindo que outras funções acessem todos os livros cadastrados em memória. Se o arquivo não existir, define o vetor como NULL e o total como zero.

8.3. carregarLivros

A função *carregarLivros(Livro **livros, int *total)* lê todos os registros de livros do arquivo "arquivos/livros.txt" e carrega esses dados em um vetor dinâmico de estruturas Livro. Ela abre o arquivo, ignora a primeira linha (cabeçalho), lê cada linha seguinte extraindo os campos (id, ISBN, título, autor e gênero), converte o gênero para o tipo enum correspondente e armazena cada livro no vetor, realocando memória conforme necessário. Ao final, atualiza o ponteiro do vetor e o total de livros lidos, permitindo que outras funções acessem todos os livros cadastrados em memória. Se o arquivo não existir, define o vetor como NULL e o total como zero.

8.3. carregarLivros

A função *carregarLivros(Livro **livros, int *total)* lê todos os registros de livros do arquivo "arquivos/livros.txt" e carrega esses dados em um vetor dinâmico de estruturas Livro. Ela abre o arquivo, ignora a primeira linha (cabeçalho), lê cada linha seguinte extraindo os campos (id, ISBN, título, autor e gênero), converte o gênero para o tipo enum correspondente e armazena cada livro no vetor, realocando memória conforme necessário. Ao final, atualiza o ponteiro do vetor e o total de livros lidos, permitindo que outras funções acessem todos os livros cadastrados em memória. Se o arquivo não existir, define o vetor como NULL e o total como zero.

8.4. buscarLivroPorISBN

A função *buscarLivroPorISBN(Livro *livros, int total, const char *isbnBusca)* realiza uma busca linear no vetor de livros carregados em memória. Ela compara o ISBN de cada livro com o ISBN informado pelo usuário (isbnBusca). Se encontrar um livro com o ISBN correspondente, retorna o índice desse livro no vetor; caso contrário, retorna -1. Isso permite localizar rapidamente a posição de um livro específico para operações como exibição, atualização ou remoção.

8.5. carregarEmprestimos

A função *carregarEmprestimos(Emprestimos **emprestimos, int *total)* lê todos os registros de empréstimos do arquivo "arquivos/emprestimos.txt" e carrega esses dados em um vetor dinâmico de estruturas Emprestimos. Ela abre o arquivo, ignora a primeira linha (cabeçalho), lê cada linha seguinte extraindo os campos (id, leitor, ISBN, data e status), converte o status para o tipo enum correspondente e armazena cada empréstimo no vetor, realocando memória conforme necessário. Ao final, atualiza o ponteiro do vetor e o total de empréstimos lidos, permitindo que outras funções acessem todos os empréstimos cadastrados em memória. Se o arquivo não existir, define o vetor como NULL e o total como zero.