



Laboratório de Programação | 21178

Nome: Carlos Miguel Faria Martins da Costa

N.º de Estudante: 2203524

Curso: Licenciatura em Engenharia Informática

Docentes: Nelson Russo/ Pedro Viegas Júnior

Data: 01.05.2024

Ano Letivo: 2023/24

Nota previa: Coloquei em anexo informações adicionais sobre as escolhas que fiz neste projeto.

Questão 1)

Eu escolhi para a realização da Atividade Formativa 2 o enunciado A que tinha como objetivo gerir o acervo de uma Biblioteca.

Questão 2a)

Começo por explicar a estratégia que utilizei para a parte de gestão dos livros. O módulo “book”:

Módulos e Interfaces:

book.h / book.c:

- Módulo que lida com a criação, manipulação e exclusão de livros.
- Fornece funções públicas para criar, aceder e modificar atributos de livros, bem como para emprestar e devolver livros.
- Utiliza a função **strman_copy()** para operações de cópia de strings.

strman.h / strman.c:

- Módulo utilitário para manipulação de strings.
- Oferece a função **strman_copy()** para copiar strings.

Interações entre os Módulos:

- O módulo book utiliza a função de ajuda **strman_copy()** para copiar strings ao definir o título, autor e gênero de um livro.
- Outros módulos podem interagir com o módulo “book” usando suas funções públicas para criar, aceder e modificar livros conforme necessário.

Agora temos o módulo “user” responsável pela criação, manipulação e exclusão de usuários da biblioteca.

Módulos e Interfaces:

user.h / user.c:

- Módulo que gere os usuários da biblioteca.
- Fornece funções públicas para criar, aceder e modificar atributos dos usuários.
- Interface para criar e excluir usuários, bem como para aceder e modificar seus dados.
- Usa a função **strman_copy()** : semelhante funcionamento ao modulo “book”.

strman.h / strman.c: semelhante ao modulo “book”.

Interações entre os Módulos:

- O módulo “user” utiliza a função de ajuda **strman_copy()** para copiar strings ao definir o nome e o endereço de um usuário.
- Outros módulos podem interagir com o módulo “user” usando suas funções públicas para criar, aceder e modificar usuários conforme necessário.

Passemos ao modulo “borrow” que lida com os empréstimos na biblioteca.

Módulos e Interfaces:

borrow.h / borrow.c:

- Módulo que gere os empréstimos na biblioteca.
- Fornece funções públicas para criar, aceder e modificar empréstimos.
- Interface para criar e excluir empréstimos, bem como para aceder e modificar seus dados.

common.h / common.c:

- Módulo utilitário que fornece funções comuns usadas em todo o programa.
- Funções públicas para verificar números de telefone, validar datas, comparar datas, analisar números, limpar o buffer de entrada, etc.

Interações entre os Módulos:

- O módulo “borrow” utiliza funções do módulo “common” para validar e comparar datas, além de analisar números.
- Outros módulos podem interagir com o módulo “borrow” usando suas funções públicas para criar, aceder e modificar empréstimos conforme necessário.

Passo a explicar a parte mais desafiante de todo o programa, “reports_menu” – os relatórios.

Módulos e Interfaces:

reports_menu.h / reports_menu.c:

- Módulo responsável por gerar e exibir relatórios da biblioteca.
- Fornece uma interface para mostrar relatórios sobre os livros mais emprestados, livros não devolvidos e usuários com mais empréstimos.
- Usa funções dos módulos book, user, e borrow para aceder dados sobre livros, usuários e empréstimos.
- Utiliza funções do módulo interface para interagir com o usuário, como exibir mensagens e solicitar entrada.

serializer.h / serializer.c:

- Módulo responsável por salvar e carregar dados da biblioteca em arquivos.
- Fornece funções para salvar e carregar usuários, livros e empréstimos.
- Usa funções dos módulos book, user, e borrow para aceder os dados necessários.

comparators.h / comparators.c:

- Módulo que contém funções para comparar objetos, como livros, usuários e empréstimos, com base nos seus IDs. *(Tomei esta decisão de utilizar os IDs pois pensei na logica de que o livro tem um QRCode e o usuário um cartão aonde são ‘scanados’ e entram os IDs no programa.)*

Interações entre os Módulos:

- O módulo “reports_menu” utiliza funções dos módulos book, user e borrow para acessar aos dados necessários para gerar os relatórios.
- Os relatórios são gerados usando iterações sobre listas de livros, usuários e empréstimos, e as informações são exibidas ao usuário utilizando a interface fornecida pelo módulo interface.
- O módulo “serializer” é usado para salvar e carregar dados da biblioteca em arquivos, garantindo que as informações necessárias estejam disponíveis para os relatórios.

Questão 2b)

As estruturas de dados usadas na implementação são:

Livro (Book):

Campos: ID, título, autor, gênero, quantidade, disponibilidade.

Funcionalidades:

- Criação de um novo livro.
- Acesso e atualização de informações.
- Empréstimo e retorno de exemplares.
- Verificação de disponibilidade.

Usuário (User):

Campos: ID, nome, endereço, telefone.

Funcionalidades:

- Criação de um novo usuário.
- Acesso e atualização de informações.

Empréstimo (Borrow):

Campos: ID, ID do usuário, ID do livro, data de empréstimo, data de retorno.

Funcionalidades:

- Criação de um novo empréstimo.
- Acesso e atualização de informações.
- Verificação de retorno.

Relatório (Reports):

Estrutura report t: Contém ID e contagem associada.

Funcionalidades:

- Relatório dos livros mais emprestados.
- Relatório dos livros não devolvidos.
- Relatório dos usuários com mais empréstimos.

Questão 2c)

O arquivo **main.c** é o ponto de entrada do programa que gere o acervo da biblioteca. Ele inicializa listas para usuários, livros e empréstimos, carrega dados dos arquivos, exibe o menu principal e executa as operações escolhidas pelo usuário. Vamos por partes:

Módulos e Interfaces:

main.c:

- Função principal **main()**: Inicia o programa, carrega os dados do banco de dados, exibe o menu principal e salva os dados ao sair.
- Função privada **app_show_main_menu()**: Exibe o menu principal e direciona para as diferentes funcionalidades do programa.

Headers:

- **common.h**: Declara funções comuns utilizadas em todo o programa.
- **list.h**: Declara funções relacionadas à manipulação de listas.
- **user.h, book.h, borrow.h**: Declarações de estruturas e funções relacionadas a usuários, livros e empréstimos.
- **interface.h**: Declara funções para exibição de mensagens e interação com o usuário.
- **serializer.h**: Declara funções para carregar e salvar dados do banco de dados.
- **books_menu.h, users_menu.h, borrows_menu.h, reports_menu.h**: Declarações de funções para os diferentes menus do programa.

Implementações:

- **main.c**: Implementa a função **main()** e a função privada **app_show_main_menu()**.
- Implementações dos módulos **user.c, book.c, borrow.c, reports.c** que contêm a lógica relacionada a usuários, livros, empréstimos e relatórios.
- Implementações dos módulos **interface.c, serializer.c, books_menu.c, users_menu.c, borrows_menu.c, reports_menu.c**.

Cada módulo encapsula uma parte específica da funcionalidade do programa, o que facilita a manutenção e o desenvolvimento modular.

Gostaria de salientar que na elaboração deste programa escolhi utilizar **lista duplamente encadeada circular** pois é estrutura relativamente simples de entender e implementar e oferece eficiência em operações de inserção e remoção no final da lista (**push_back** e **pop_back**), já que assim não é necessário percorrer toda a lista para encontrar o último elemento.

No que diz respeito ao **Teste de Integridade** elaborei um código para verificar se as funções de obtenção de informações sobre o livro estão funcionando conforme o esperado, imprimindo os valores dos atributos do livro na saída padrão. (**book_get_id, book_get_title, book_get_author, book_get_genre, book_get_quantity** e **book_get_available**).

No que concerne ao **Teste de Unidade** idealizei um código que verifica se a estrutura de dados “User” funciona corretamente, especialmente em relação à criação de usuários, obtenção de informações e definição de números de telefone. Neste conjunto de testes, é esperado que haja um número de telefone inválido, logo uma mensagem detalhada sobre o teste em causa será exibida.

Questão 3a)

Módulos/funções que poderiam ser reutilizadas:

Lista circular duplamente encadeada: A estrutura de dados para armazenar os pratos e os pedidos pode ser reutilizada, pois permite operações eficientes de inserção, remoção e busca.

Escrita e leitura de dados em diferentes formatos: As funções utilizadas para salvar e carregar dados de um arquivo CSV podem ser reutilizadas com pequenas modificações para lidar com os dados dos pratos e pedidos do restaurante.

Validação de dados: Funções que validam informações, como datas, números de telefone ou formatos de texto, podem ser úteis para garantir a integridade dos dados inseridos pelo usuário.

Questão 3b)

Código que pode ser adaptado e esforço estimado de adaptação:

Manipulação de pratos: A lógica para adicionar, remover, editar e pesquisar pratos pode ser adaptada do código existente para manipular os pratos do restaurante. Seria necessário ajustar os tipos de dados e os campos dos pratos, bem como as operações específicas de cada função.

Registro de pedidos e controle de mesas: A estrutura de dados e as operações para registrar pedidos e gerenciar mesas podem ser adaptadas da lista circular duplamente encadeada existente. Seria necessário adicionar campos adicionais para controlar o estado dos pedidos e a disponibilidade das mesas.

Sistema de relatórios: Um sistema de relatórios semelhante ao descrito pode ser implementado utilizando os dados dos pedidos e das mesas. Seria necessário criar funções para calcular os pratos mais pedidos, as mesas com maior taxa de ocupação e o valor médio do total da conta por mesa.

Questão 3c)

Módulos/funções que teriam de ser desenvolvidos:

Leitura de dados do arquivo CSV: Funções para ler os dados dos pratos e dos pedidos de arquivos CSV precisariam ser desenvolvidas para inicializar o programa com os dados do restaurante.

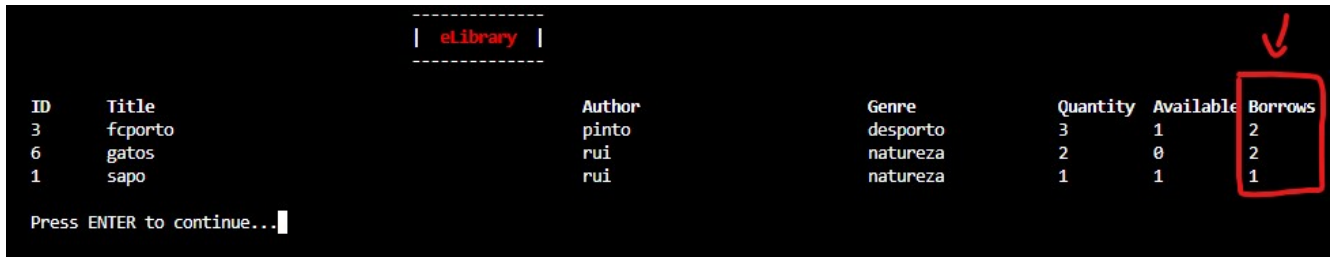
Escrita de dados no arquivo CSV: Funções para gravar os dados dos pratos e dos pedidos em arquivos CSV seriam necessárias para salvar os dados antes de fechar o programa.

Interface do usuário: Um sistema de interface do usuário seria necessário para permitir que o usuário interaja com o programa, adicionando pratos, registrando pedidos, gerindo mesas e visualizando relatórios.

Em resumo, o esforço para adaptar o código existente para gerir as operações de um restaurante seria moderado. A maior parte da lógica de manipulação de dados e operações de lista poderia ser reutilizada, mas algumas adaptações seriam necessárias para lidar com os requisitos específicos do restaurante, como a leitura de dados do arquivo CSV, a manipulação de pedidos e mesas, e a implementação da interface do usuário.

Anexos

1. Quando pretendo eliminar um livro ou um usuário ele vai apagar do sistema o IDs respetivo para sempre, não podendo mais ser utilizado;
2. No que diz respeito às datas que devemos introduzir nos empréstimos, pensei colocar este processo automático (acrescentar 14 dias automaticamente a data da reserva), mas por falta de tempo não coloquei;
3. No que diz respeito aos empréstimos, “upadte borrow”, resolvi colocar a logica de primeiro ser perguntado se pretende devolver o livro – y/n? - e caso fosse “não” aparece a opção renovar data de empréstimo;
4. Relativamente aos Relatórios:
 - Livros mais emprestados (Most borrowed books)



ID	Title	Author	Genre	Quantity	Available	Borrows
3	fcporto	pinto	desporto	3	1	2
6	gatos	rui	natureza	2	0	2
1	sapo	rui	natureza	1	1	1

Press ENTER to continue...

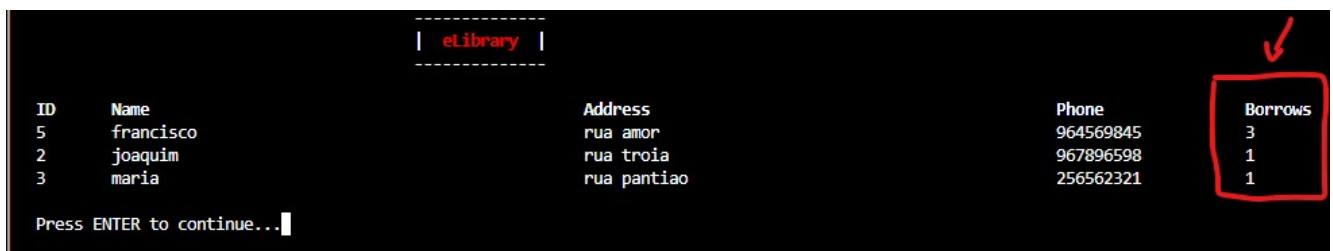
- Livros não devolvidos (Non-returned books)



ID	Title	Author	Genre	Quantity	Borrowed
3	fcporto	pinto	desporto	3	2
6	gatos	rui	natureza	2	2

Press ENTER to continue...

- Usuários com mais empréstimos (Users with more borrows)

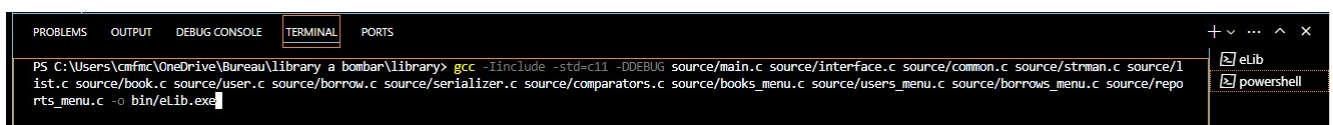


ID	Name	Address	Phone	Borrows
5	francisco	rua amor	964569845	3
2	joaquim	rua troia	967896598	1
3	maria	rua pantiao	256562321	1

Press ENTER to continue...

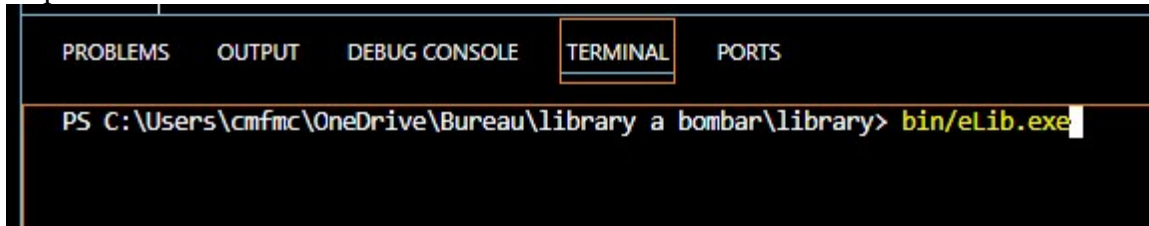
5. No que concerne a parte de compilar e executar o programa (página seguinte tem a ilustração):
 - Para aparecer o executável exe de todo o programa (exceção aos testes) - **eLib.exe** - (*1) utilizar o comando:

```
gcc -Iinclude -std=c11 -DDEBUG source/main.c source/interface.c source/common.c
source/strman.c source/list.c source/book.c source/user.c source/borrow.c source/serializer.c
source/comparators.c source/books_menu.c source/users_menu.c source/borrows_menu.c
source/reports_menu.c -o bin/eLib.exe
```



Vai aparecer o executável - eLib.exe - (*1) ver página seguinte aonde se situa.

Depois de termos este executável será somente necessário colocar - **bin/eLib.exe**



E assim acedemos ao menu:



6. Nos Testes de Integridade para obter o executável - **book_list_itest.exe** - (*2) utilizamos o comando:

```
gcc -linclude -std=c11 -DDEBUG test/integration/book_list_itest.c source/common.c
source/strman.c source/list.c source/book.c source/user.c source/borrow.c source/serializer.c -o
bin/book_list_itest.exe
```

E uma vez criado o executável será somente necessário colocar - **bin/book_list_itest.exe** - para executar os testes de integridade;

7. Nos Testes de Unidade para obter o executável - **user_utest.exe** - (*3) utilizamos o comando:

```
gcc -linclude -std=c11 -DDEBUG test/unit/user_utest.c source/common.c source/strman.c
source/list.c source/book.c source/user.c source/borrow.c source/serializer.c -o bin/user_utest.exe
```

E uma vez criado o executável será somente necessário colocar - **bin/user_utest.exe** - para executar os testes de unidade.

File Edit Selection View Go Run Terminal Help

library

EXPLORER

LIBRARY

- .vscode
- bin
 - book_list_itest
 - book_list_itest.exe (2*)
 - eLib.exe (1*)
 - user_utest.exe (3*)
- include
 - book.h
 - books_menu.h
 - borrow.h
 - borrows_menu.h
 - common.h
 - comparators.h
 - interface.h
 - list.h
 - logger.h
 - reports_menu.h
 - serializer.h
 - strman.h
 - user.h
 - users_menu.h
- source
 - book.c
 - books_menu.c
 - borrow.c
 - borrows_menu.c
 - common.c
 - comparators.c
 - interface.c
 - list.c
 - main.c
 - reports_menu.c
 - serializer.c
 - strman.c
 - user.c
 - users_menu.c
- test
 - integration
 - book_list_itest.c
 - unit
 - user_utest.c
 - books.csv
 - borrows.csv
 - users.csv

main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "common.h"
5 #include "list.h"
6 #include "user.h"
7 #include "book.h"
8 #include "borrow.h"
9 #include "interface.h"
10 #include "serializer.h"
11 #include "books_menu.h"
12 #include "users_menu.h"
13 #include "borrows_menu.h"
14 #include "reports_menu.h"
15
16 /* Private Functions */
17
18 void app_show_main_menu(List user_list, List book_list, List borrow_list, size_t *user_last_id, size_t *book_last_id, size_t *borrow_last_id);
19
20 /* Main Function */
21
22 int main(void) {
23     List user_list = list_new();
24     List book_list = list_new();
25     List borrow_list = list_new();
26
27     size_t user_last_id = serializer_load_users(USERS_FILE, user_list);
28     size_t book_last_id = serializer_load_books(BOOKS_FILE, book_list);
29     size_t borrow_last_id = serializer_load_borrows(BORROWS_FILE, borrow_list);
30
31     if(!user_last_id || !book_last_id || !borrow_last_id) {
32         ui_show_error(
33             "There was an error loading the database files.\n"
34             "Contact the system administrator.",
35             SYSTEM_ERROR_DATABASE_LOAD
36         );
37     }
38 }

```

TERMINAL

PS C:\Users\cmfmc\OneDrive\Bureau\library a bombar\library>

Ln 59, Col 1 Spaces: 4 UTF-8 LF () C linux-gcc-x64

0 0 0 0