# MC202E - ESTRUTURAS DE DADOS

### Lab03: Clube do Livro

O clube do livro da sua cidade montou um novo sistema de empréstimos de livros, no qual se baseia na ideia de que para cada livro pego emprestado, o leitor deve deixar um de seus próprios livros para empréstimo. É bem simples, não eh!! No entanto, está sendo uma tarefa muito difćil para a administração do clube gerenciar o registro de quais leitores estão com a devolução atrasada, devido ao grande número de leitores. Para ajudá-los nesta tarefa, você deverá implementar um programa em C que utiliza conceitos de abstração de dados e, principalmente, **alocação dinâmica** e **passagem por referência** (i.e., uso de ponteiros).

## Tarefa

Para esta tarefa você receberá o arquivo principal `lab03.c` e os arquivos `data.*`, `leitor.*` e `livro.*`, os quais estão disponíveis em **Arquivos auxiliares** (compactados no arquivo Workspace.zip). Os **arquivos de cabeçalho não devem ser modificados e não serão submetidos no *SuSy***. Detalhes de implementação são encontrados nos arquivos de cabeçalho e devem ser seguidos.

## Entrada

A entrada é composta por várias linhas. A primeira linha da entrada contém dois inteiros positivos $m$ e $n$ que indicam respectivamente o número de livros registrados e o número de leitores ativos. Cada uma das $m$ linhas seguintes contém a data de devolução, no formato `dd-mm-aaaa`, e o título do livro (não será informado a quantidade de caracteres a serem lidos). Se o livro está disponível, então o campo data de devolução possui valor nulo, i.e., `00-00-0000`. Cada livro é identificado por um inteiro positivo, começando a partir de 1, na ordem em que aparecem na entrada. Cada uma das $n$ linhas seguintes contém apenas o primeiro nome de um leitor ativo, seguido de uma **lista codificada** pela administração do clube do livro. Nesta lista, o primeiro inteiro, indicado aqui por $e$, corresponde ao número de livros deixados para empréstimo mais o número de livros a serem devolvidos pelo leitor. Os $e$ inteiros seguintes da lista são sinalizados, onde $+i$ indica que o $i$-ésimo livro foi deixado para empréstimo; e $-i$ indica que o $i$-ésimo livro foi pego emprestado. Aqui, $i$ (sem a sinalização) corresponde a codificação dada ao livro durante a leitura. Por fim, **a última linha corresponde a data atual da consulta** que, em outras palavras, corresponde ao *threshold* (limiar de decisão) da saída a ser apresentada por seu programa. Para mais detalhes, veja o exemplo de entrada.

**Restrições**

- ❑ $3 \leq \boldsymbol{n} \leq \boldsymbol{m}$
- ❑ O título de cada livro possui no máximo 100 caracteres.
- ❑ O nome de cada leitor ativo possui no máximo 15 caracteres.

**Observações**

- ❑ Para a leitura do título, você pode usar **`scanf(" %[^\n]", titulo)`**;
  - ☞ O formato do `scanf` acima consiste em duas partes:
    - *i*) o caractere de espaço no começo do formato indica que enquanto os caracteres da entrada forem espaço em branco, nova linha, etc., os mesmos devem ignorados, ou seja, removidos da entrada (conhecido como limpar o *buffer* do teclado);
    - *ii*) a expressão `%[^\n]` diz que todos os caracteres não igual ao caractere de nova linha devem ser armazenados (mais um caractere `'\0'` de terminação) na string `titulo`.

  Ou você pode usar a função **`fgets(titulo, MAXCHAR_TITULO, stdin)`**, mas note que nesse caso ainda é necessário limpar o *buffer* do teclado antes da leitura.
- ❑ Para a leitura dos inteiros sinalizados, em particular +*i*, basta fazer um leitura normal com **`%d`** (e.g., **`scanf("%d", &i)`**).

# Saída

A saída do seu programa deverá ser compostas por $\boldsymbol{n}$ descrições. As descrições seguem a ordem de leitura dos leitores, onde a primeira linha de cada descrição corresponde ao nome do leitor. Para completar uma descrição, as demais linhas devem manter a ordem da leitura da codificação dada aos **livros deixados para empréstimos** (i.e., o clube do livro deve fazer a **restituição** do livro ao seu dono); seguido da codificação dada aos **livros a serem devolvido pelo leitor**, ou seja, uma **devolução**. Se for o caso de uma restituição, então a linha de conter inicialmente `"Restituição: "`, seguido do título do livro; caso contrário, i.e., uma devolução, então a linha deve conter inicialmente `"Devolução: "`, seguido do título do livro. Em todos os casos em que a data de devolução possui **valor nulo** ou **igual ou maior que** a data atual da consulta, uma frase para o título do livro **não deverá aparecer na descrição** apresentada por seu programa. Em particular ao valor nulo na data de devolução, observe que tal valor indica que o livro está disponível, logo não está em posse de um leitor com atraso na devolução do mesmo.

**Observações**

- ❑ Em relação aos casos de testes, você pode sempre assumir que cada leitor ativo possui pelo menos um livro a ser restituído ou um livro a ser devolvido.
- ❑ Pode existir mais devoluções do que restituições.
- ❑ A última linha da saída deve estar em branco, i.e., deve conter o caracter `'\n'`.

# Exemplo

A grafia da saída abaixo deve ser seguida rigorosamente por seu programa, inclusive a impressão da linha em branco no final da saída.

**Entrada**

```
8 4
04-09-2019 A Menina do Narizinho Arrebitado
20-08-2019 O Pequeno Príncipe
00-00-0000 O Escaravelho do Diabo
13-10-2019 O Menino Maluquinho
27-08-2019 Malala, a Menina que Queria Ir Para a Escola
01-10-2019 O Diário de um Banana
10-10-2019 O Pequeno Príncipe
11-08-2019 O Jardim Secreto
Adriana 4 +2 +8 -4 -7
Daniel 2 +3 -8
Duda 5 -2 +1 +5 +7 -6
Jairo 4 +4 +6 -1 -5
01-10-2019
```

**Saída**

```
Adriana
Restituição: O Pequeno Príncipe
Restituição: O Jardim Secreto
Daniel
Devolução: O Jardim Secreto
Duda
Restituição: A Menina do Narizinho Arrebitado
Restituição: Malala, a Menina que Queria Ir Para a Escola
Devolução: O Pequeno Príncipe
Jairo
Devolução: A Menina do Narizinho Arrebitado
Devolução: Malala, a Menina que Queria Ir Para a Escola

```

Note que, no exemplo acima, para a data da consulta `01-10-2019`, temos que para a descrição do leitor `Jairo`, o livro 6, i.e., `O Diário de um Banana`, não aparece como a ser restituído.

# Critérios específicos

Os seguintes critérios específicos sobre o envio, implementação, compilação e execução devem ser satisfeitos.

***i***. Submeter no *SuSy* os arquivos:
- ⇒ **`lab03.c`**: Deverá conter o programa principal, no qual terá de chamar as funções descritas nos arquivos de cabeçalho (pois não é permitido acessar diretamente os campos de um TAD). Aliás, é necessário alocar dinamicamente os vetores `livros` e `leitores`.
- ⇒ **`data.c`**, **`leitor.c`** e **`livro.c`**: Deverão conter a implementação correta das funções, conforme descrito no arquivo de cabeçalho dos mesmos. Em particular a implementação do TAD Leitor, é necessário alocar dinamicamente os vetores `restituicoes` e `devolucoes` de tamanho igual a ***e***.

***ii***. **É obrigatório** seguir as especificações dos arquivos de cabeçalho, bem como verificar se ocorreu erro de alocação de memória e desalocar toda a memória alocada antes de encerrar o programa.

***iii***. Flags de compilação:
**`-std=c99 -Wall -Werror -g -lm`**

***iv***. Tempo máximo de execução: 1 segundo.

# Observações gerais

No decorrer do semestre haverá 3 tipos de tarefas no *SuSy* (descritas logo abaixo). As tarefas possuirão os mesmos casos de testes abertos e fechados, no entanto o número de submissões permitidas e prazos são diferentes. As seguintes tarefas estão disponíveis no *SuSy*:

- ❏ **Lab03-AmbienteDeTeste**: Esta tarefa serve para testar seu programa no *SuSy* antes de submeter a versão final. Nessa tarefa, tanto o prazo quanto o número de submissões são ilimitados, porém os arquivos submetidos aqui **não serão corrigidos**.
- ❏ **Lab03-Entrega**: Esta tarefa tem limite de uma **única** submissão e serve para entregar a **versão final** dentro do prazo estabelecido para o laboratório. Não use essa tarefa para testar o seu programa e submeta aqui apenas quando não for mais fazer alterações no seu programa.
- ❏ **Lab03-ForaDoPrazo**: Esta tarefa tem limite de uma **única** submissão e serve para entregar a versão final fora prazo estabelecido para o laboratório. Esta tarefa irá substituir a nota obtida na tarefa **Lab03-Entrega** apenas se o aluno tiver realizado as correções sugeridas no *feedback* ou caso não tenha enviado anteriormente na tarefa **Lab03-Entrega**.

**Avaliação**

Este laboratório será avaliado da seguinte maneira: se o seu programa apresentar resposta correta para todos os casos de teste do *SuSy*, então é nota 10; caso contrário, é nota 0 (i.e, seu programa apresentou resposta incorreta para pelo menos um caso de teste aberto ou fechado).

**Testando seu programa**

Para compilar usando o `Makefile` fornecido e testar se a solução seu programa está correta, basta seguir o exemplo abaixo.

```
make
./lab03 < testes_abertos/arq01.in > testes_abertos/arq01.out
diff testes_abertos/arq01.out testes_abertos/arq01.res
```

O `arq01.in` é a entrada (caso de teste disponível no *SuSy*) e `arq01.out` é a saída do seu programa. O `Makefile` também contém regras para baixar e testar todos os testes de uma única vez; nesse caso, basta digitar conforme o exemplo a seguir.

```
make baixar_abertos
```

```
make testar_abertos
```

A primeira instrução irá baixar os casos de teste abertos para a pasta **testes_abertos**, criada automaticamente, e a segunda instrução irá testar o seu programa com os casos de teste abertos. Após o prazo, os casos de teste fechados serão liberados e podem ser baixados e testados da mesma forma que os testes abertos. Para isso, basta trocar `"_abertos"` por `"_fechados"` (e.g., `make baixar_fechados`).

**Checando seu programa**

Para checar se seu programa está desalocando toda a memória alocada dinamicamente antes de finalizar a execução, basta usar o `Makefile` conforme o exemplo a seguir.

```
make checar_abertos
```

A instrução irá checar o seu programa para os casos de teste abertos usando o *software valgrind* com as seguintes opções: `-q --leak-check=full` **`--error-exitcode=1`**. A última opção corresponde a uma **nova atualização realizada no arquivo `Makefile`**.

Caso exista algum *memory leak*, i.e., memória que não foi desalocado, ou erros de acesso a posições inválidas de memória, então será impresso uma descrição do erro (ou erros) seguindo da interrupção da verificação.

Caso preferir, você também checar manualmente, para cada teste, escrevendo no terminal conforme o exemplo a seguir.

```
valgrind --leak-check=full ./lab03 < testes_abertos/arq01.in
```