INSTITUTO FEDERAL FLUMINENSE - IFF
Campus Itaperuna

Prof.: Leandro Fernandes dos Santos | ✉ leandro.f.santos@iff.edu.br
Disciplina: Estruturas de Dados

Atividade Avaliativa 2 - A2

Turma: Sistemas de Informação 3º Período

Objetivo: A presente atividade avaliativa tem como objetivo consolidar os conceitos vistos sobre Listas Encadeadas e sua implementação utilizando a linguagem C.

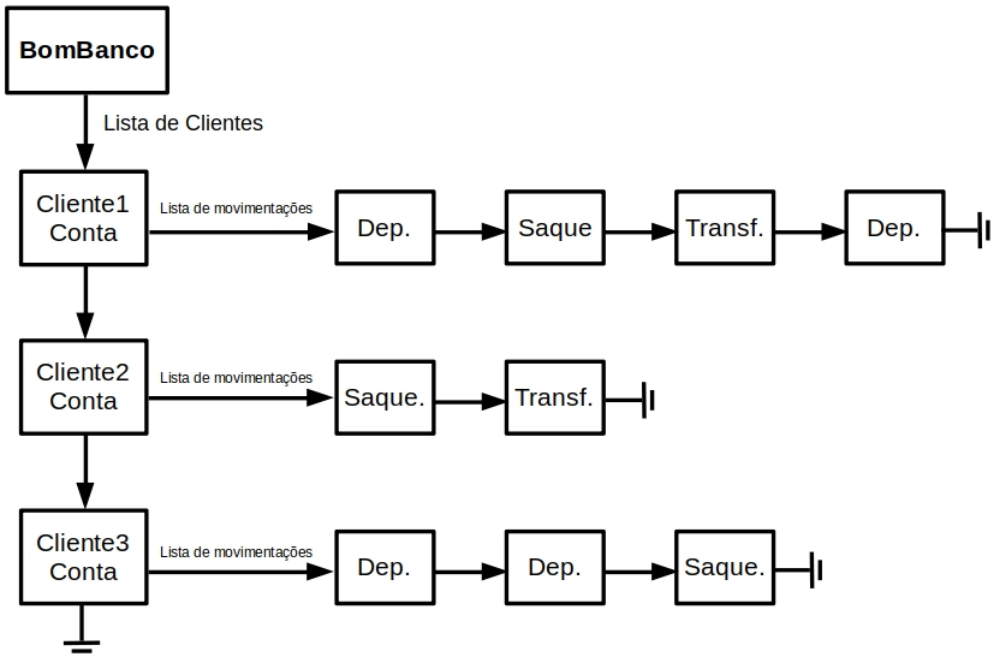
Observações e regras de entrega:

- Esta atividade poderá ser realizada em **dupla**.
- Terá o valor de 6 pontos na composição da (A2).
- A solução deverá ser enviada por meio da plataforma run.codes até as 23h59min do dia 19/03/2023.
- Pontos importantes a serem seguidos:
  - **Conformidade com os requisitos** aqui apresentados.
  - Uso dos conceitos de modularização.
  - Códigos bem estruturados e bem indentados.
- Caso seja detectada a ocorrência de plágio, será atribuída nota zero a **todos os envolvidos**.
- Submissões que apresentarem erros de compilação ou falhas de segmentação no run.codes **NÃO** serão corrigidas.
- O não atendimento dos **requisitos especificados neste documento** acarretará em perda de pontos.

Especificação

Sistema Bancário Simplificado

Considere um banco fictício chamado "BomBanco" no qual possui, clientes, contas e movimentações bancárias (saques, depósitos e transferências). Por simplificação, dentro deste banco, cada cliente possui uma única conta, sendo associada a esta conta todas as movimentações bancárias realizadas. Os clientes deste banco podem ser organizados por meio de listas encadeadas, e as respectivas movimentações de suas contas, também podem ser modeladas com listas encadeadas. Desta forma podería-se ter em um determinado momento a seguinte configuração:



Nesta atividade você deverá implementar em C este sistema bancário simplificado. Para tal você deverá seguir a estrutura básica inicial (bank.c) fornecida juntamente com esta especifição. Nesta estrutura se encontra implementada a leitura de uma única variável denominada path, sendo esta o caminho para o arquivo de entrada nomeado como bank.data e que também é fornecido com a especificação. Neste arquivo se encontram todos os dados referentes ao sistema bancário que são necessários para seu funcionamento.

No arquivo bank.c também há uma função denominada readFile. Esta função realiza a leitura dos dados que se encontram padronizados no arquivo bank.data. Para a realização desta atividade você deverá completar a implementação das estruturas e funções já predefinidas no arquivo.

Os tipos básicos (TAD's) que devem ser completados são:

- **Movimentacao**
  - tipo: Valor inteiro representando uma operação. Valores possíveis:
    - 0: Depósito
    - 1: Saque
    - 2: transferência
  - idClienteDest: Caso seja uma transferência, este campo deve conter o id do cliente para quem a transferência foi enviada. Para as demais operacoes, este valor pode ser 0.
  - idClienteOrig: Em todos os tipos de movimentações, este campo deve conter o id do cliente de origem.
  - valor: Valor em ponto flutuante representando o valor da movimentação.
- **Conta**
  - numero: número inteiro representando o número da conta
  - saldo: saldo da conta que é atualizado de acordo com as movimentações
  - movimentacoes: lista de movimentações (Lista Encadeada) realizadas na respectiva conta
- **Cliente**
  - id: número inteiro para identificação do cliente
  - conta: conta do respectivo cliente
- **Banco**
  - nome: nome do banco
  - clientes: lista de clientes (Lista Encadeada) do banco

Após o término da especificação dos tipos acima, você deverá terminar a implementação das seguintes funções:

- Banco* criarBanco(): Tem por objetivo criar um novo banco e retornar um ponteiro para o banco criado.
- Cliente* criarNovoCliente(int idCliente, int numConta, float saldo): Tem por objetivo criar um novo cliente e retornar um ponteiro para o cliente criado.
- Movimentacao* criarNovaMovimentacao(short int tipo, float valor, int idClienteOrig, int idClienteDest): Tem por objetivo criar uma nova movimentação e retornar um ponteiro para a movimentação criada. Caso o tipo da movimentação seja diferente de 2 (transferência) então o valor de idClienteDest deverá ser 0.

- void adicionarCliente(Banco *b, Cliente *c): Esta função deverá adicionar o cliente (*c) à lista de clientes do banco (*b). A adição deverá ser realizada de maneira ordenada (crescente) pelo número do cliente.

- void realizarDeposito(Cliente *c, Movimentacao* dep): Função que adiciona uma movimentação de depósito na lista de movimentações da conta do cliente (*c). Lembre-se que o saldo do cliente deverá ser atualizado após a movimentação. A inserção deverá ser realizada sempre no final da lista de movimentações.

- float realizarSaque(Cliente *c, Movimentacao* saque): Função que adiciona uma movimentação de saque na lista de movimentações da conta do cliente (*c). Lembre-se que o saldo do cliente deverá ser atualizado após a movimentação. A inserção deverá ser realizada sempre no final da lista de movimentações.

- void realizarTransferencia(Cliente *clienteOrig, Cliente *clienteDest, float valor): Função que adiciona uma movimentação de saque na lista de movimentações da conta do cliente de origem (*clienteOrig) e adiciona uma movimentação de depósito na lista de movimentações da conta do cliente de destino (*clienteDest). Lembre-se que o saldo de ambos os clientes deverão ser atualizados após a realização das movimentações. As inserções deverão ser realizadas sempre no final da lista de movimentações dos clientes.

- Cliente* buscarCliente(Banco *b, int idCliente): Esta função deverá retornar o endereço do cliente que possui o idCliente na lista de clientes do banco (*b). A função deverá retornar NULL caso o cliente não se encontre na lista.

- void liberarBanco(Banco *b): Função que tem por objetivo liberar toda e qualquer memória alocada dinamicamente para o banco.

- void imprimirDados(Banco *B): Função que tem por objetivo imprimir todos os dados do banco no padrão requerido da especificação.

Você **NÃO** deverá modificar o padrão de leitura dos dados no arquivo de entrada, pois caso o modifique, os testes poderão não funcionar corretamente, uma vez que estes estão baseados no formato proposto.

Você poderá criar outras funções auxiliares se achar necessário.

Após a inserção dos dados em todas as estruturas criadas, o programa deverá apresentar uma saída resumindo todas as movimentações dos clientes conforme o exemplo mostrado a seguir:
Obs.: A saída de exemplo é correspondente ao arquivo de entrada de teste fornecido.

==========
Id. Cliente : 101
Numero Conta : 120
Saldo inicial : 0.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Deposito | Valor: 500.00
Tipo: Saque | Valor: -100.00
Tipo: Transf. | Valor: -120.00 ===> Destinatario: 110
Saldo Final: 280.00
==========
Id. Cliente : 102
Numero Conta : 121
Saldo inicial : 100.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Saque | Valor: -30.00
Tipo: Transf. | Valor: 90.00 ===> Origem: 103
Saldo Final: 160.00
==========
Id. Cliente : 103
Numero Conta : 122
Saldo inicial : 20.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Deposito | Valor: 600.00
Tipo: Transf. | Valor: -90.00 ===> Destinatario: 102
Saldo Final: 530.00
==========
Id. Cliente : 104
Numero Conta : 123
Saldo inicial : 50.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Deposito | Valor: 750.00
Tipo: Saque | Valor: -50.00
Tipo: Transf. | Valor: -50.00 ===> Destinatario: 107
Saldo Final: 700.00
==========
Id. Cliente : 105
Numero Conta : 124
Saldo inicial : 0.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Deposito | Valor: 1000.00
Tipo: Saque | Valor: -300.00
Tipo: Transf. | Valor: -100.00 ===> Destinatario: 108
Saldo Final: 600.00
==========
Id. Cliente : 106

Numero Conta : 125
Saldo inicial : 50.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Transf. | Valor: 200.00 ===> Origem: 109
Saldo Final: 250.00
==========

Id. Cliente : 107
Numero Conta : 126
Saldo inicial : 0.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Deposito | Valor: 550.00
Tipo: Saque | Valor: -200.00
Tipo: Transf. | Valor: 50.00 ===> Origem: 104
Saldo Final: 400.00
==========

Id. Cliente : 108
Numero Conta : 127
Saldo inicial : 50.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Transf. | Valor: 100.00 ===> Origem: 105
Saldo Final: 150.00
==========

Id. Cliente : 109
Numero Conta : 128
Saldo inicial : 0.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Deposito | Valor: 1500.00
Tipo: Saque | Valor: -200.00
Tipo: Transf. | Valor: -200.00 ===> Destinatario: 106
Saldo Final: 1100.00
==========

Id. Cliente : 110
Numero Conta : 129
Saldo inicial : 50.00
- - - - - - - - - - - - - - - - - - - Movimentacoes - - - - - - - - - - - - - - - - - - -
Tipo: Transf. | Valor: 120.00 ===> Origem: 101
Saldo Final: 170.00
==========

Comece a tentar resolver a atividade o quanto antes, enquanto os assuntos tratados estão frescos na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.

Bom Trabalho!