

SCC0502 Algoritmos e Estruturas de Dados 1

Exercício 6 - Conta bancária

Objetivo

O objetivo deste exercício prático é estimular os estudantes a se familiarizarem com o comportamento e a lógica associados à estrutura de dados **árvore de busca binária**.

Queremos que os alunos se habituem com a implementação dessa estrutura e consigam entender as funções básicas, que garantem a execução adequada desse *TAD*, em diferentes contextos.

Para acostumar os alunos com conceitos de modularização e boas práticas de escrita de código, será exigido o uso de múltiplos arquivos *.c* e *.h* no projeto, bem como a construção de um arquivo **Makefile**, responsável por gerenciar a execução do programa.

Descrição

Continuando o desenvolvimento do seu projeto de sistema bancário, agora sua tarefa é lidar com a gestão de registros de clientes (inserção, remoção e busca). Lembre-se sempre do ponto principal, para garantir a satisfação de seus clientes o banco não pode deixar as pessoas zangadas esperando na hora de realizarem suas operações.

Entrada

Seu algoritmo deve receber como entrada um número n , que representa a quantidade de clientes cadastrados no nosso banco. Após isso ele receberá um sequência de n registros, representando os **CPFs**, **Nomes**, **Idades** e **Saldos** dos clientes.

O primeiro indivíduo deve ser considerado como a raiz da sua árvore, os demais cadastros serão inseridos na árvore por uma regra que no futuro facilitará a busca por pessoas:

Caso o valor numérico do CPF do novo registro de cliente seja **maior** que o valor do cpf da raiz, ele deve ser inserido na sua sub-árvore **direita**, caso contrário, na sua sub-árvore **esquerda** e assim sucessivamente até encontrar a posição adequada.

Lembre que um mesmo cpf **não** pode estar associado a mais e uma conta dentro do banco.

Após os n valores, seu programa deve receber um *char* **c** representando uma operação que deverá ser realizada ['I' : Inserção, 'R' : Remoção, 'B' : Busca]. Receberá então um **cpf** como parâmetro para todas as operações. Para a inserção de um novo registro, serão passadas as informações da pessoa como foi feito anteriormente na entrada.

```
118.469.561-02,Maria Cecília Gonçalves,50,3318.74
498.228.938-76,Júlia Lorena Isadora Ribeiro,68,-20278.86
250.627.718-89,Gabriel Maite Helena Martins,41,15079.44
904.621.899-61,Livia Isadora Joaquim Santos,47,9113.10
025.801.967-06,Samuel Marques,23,-3041.86
539.696.899-02,Joaquim Samuel Sophia Jesus,0,6868.53
782.819.678-46,Gabriel Jesus,47,12237.87
370.986.568-95,Maria Júlia Moreira,60,-14561.91
550.067.540-01,Arthur Heloísa Lorena Almeida,19,-8101.50
992.090.680-84,Isaac Anthony Fernandes,50,11898.13
B
904.621.899-61
```

Saída

Como saída, seu programa deve apresentar os dados da conta:

```
Conta :: Livia Isadora Joaquim Santos
CPF :: 904.621.899-61
Idade :: 47
Saldo atual :: R$ 9113.10
```

Caso o **cpf** em questão não esteja mais na base de dados, deve imprimir a mensagem:

```
Pessoa nao encontrada.
```

Para os casos de Inserção, e remoção deverá novamente imprimir a árvore na lógica de **preorder**.

Observações da implementação

Como é descrito na sessão objetivos desse documento, não queremos apenas que os alunos resolvam o problema, mas que utilizem métodos que serão comuns no decorrer da disciplina.

Devido a esse objetivo, será exigido que vocês desenvolvam seu projeto representando ambas as entidades citadas na descrição como **TAD completos e fechados sobre si mesmos**, isto é, com funções auxiliares que permitam o acesso e a manipulação de seus atributos em diferentes contextos(arquivos .c separados). Os dados devem ser armazenados como um **TAD cliente** e dentro de uma **Árvore Binária** implementada com nós de ponteiros duplos.

Para remoção, substitua o nó excluído, pelo menor entre seus maiores filhos, isto é, o nó de menor valor da sua sub-árvore direita.

A memória deve ser alocada **dinamicamente** e ser devidamente liberada ao fim da execução.

Utilizar múltiplos arquivos .c e .h para separar a implementação e a responsabilidade dos métodos de cada objeto.

Construir funções para realizar operações repetitivas, ou seja, modularizar adequadamente seu código.

Escrever um arquivo **Makefile** que será responsável por gerenciar a execução do projeto dentro da plataforma *run.codes*.

Observações da avaliação

A avaliação do seu programa será feita além do resultado da plataforma *run.codes*. Portanto, ter um bom resultado com os casos de teste, não será suficiente para garantir a **nota máxima** e nem a **aprovação do exercício**.

Caso seu projeto não satisfaça os pontos exigidos nos **objetivos** e explicitados nas **observações de implementação**, sua nota poderá ser reduzida ou ser desconsiderada.

Cópias de código entre alunos, acusadas pela plataforma, resultarão imediatamente em **zero** aos dois ou mais alunos envolvidos.