Exercício 05 - Conta Bancária

Disciplina: SCC0122 - Estruturas de Dados
Prazo de Entrega: 11/12/2022 23:55:55 Fechado

Exercício 05 - 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

Bancos são instituições conhecidas por possuírem uma extensa base de usuários, afinal ninguém mais guarda o dinheiro no colchão não é mesmo?

Para poder garantir a satisfação de seus clientes o banco não podem deixar as pessoas zangadas esperando na hora de realizarem transações. Antes de se realizar uma operação, é necessário primeiro identificar quem é a parte que está transferindo e a parte que está recebendo a quantia em questão.

Não se pode implementar qualquer algoritmo de para se identificar as partes, no contexto de transações monetárias, performance é algo inegociável.

Eis que você é contratado pelo banco Jorge&Lemans para resolver esse dilema, desenvolver uma ferramenta que seja capaz de atender às demandas do banco.

Como um estudante da USP você se lembrou que as árvores são estruturas de dados que oferecem um boa complexidade para diversas operações de acesso a dados e resolveu construir uma para solucionar o problema.

Sua tarefa é criar essa estrutura de dados e 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, sa 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,Lívia 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 :: Lívia 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**.

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

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.

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.

Esconder Descrição

Arquivos:

README.md (/ExerciseFiles/fileDownload/12016)

Este exercício aceita os seguintes tipos de arquivos:
ZIP Makefile

Atenção: Para ser corretamente corrigido, seu código, se entregue em um único arquivo Zip/Makefile:

- deve obrigatoriamente ter um arquivo Makefile na raíz do arquivo zip;
- deve conter **apenas** o comando de **compilação** na diretiva "all" (ou seja, será executado o comando "make all" para compilar o seu código);
- deve conter **apenas** o comando de **execução** na diretiva "run" (ou seja, será executado o comando "make run" para execução do seu código);

Caso seu arquivo Makefile não siga as instruções acima, seu código não será corrigido. Se você tem alguma dúvida, entre em contato com support@run.codes. Baixar Casos de Teste (/Exercises/downloadCases/27220)

Novo Envio

(/Exercises/exportExerciseToGoogleCalendar/27220)

O exercício está fechado

11/12/2022 23:55:55

Fechado

Meu Último Envio

Download (/Commits/download/1958620)

status

Incompleto

compilado