Projeto – Interpretador de Comandos para manipulação de Árvores Binárias
Versão 1.1 – 03/09/2013

1. Apresentação

O objetivo deste projeto é planejar e criar um interpretador de comandos para manipulação de árvores binárias. A ideia é permitir que o usuário tenha acesso às opções que realizam operações em árvores binárias, de modo que possa testar todas as operações através de linhas de comando.

2. A Equipe

O projeto pode ser feito em equipe de no máximo 2 (dois) alunos, sendo necessária a defesa do mesmo. A nota do projeto é **individualizada,** visto que serão levados em conta o conhecimento do código, segurança na lógica adotada e e capacidade de efetuar rapidamente possíveis modificações.

3. Requisitos

O projeto fundamenta-se em um **controlador de árvores binárias** que disponibiliza acesso, por meio de comandos de linha, a todas as operações de árvores binárias descritas na Tabela 1. Grande parte destas operações serão apresentadas em sala de aula, cabendo ao aluno desenvolver as novas operações. O interpretador deve ser flexível o possível para permitir ao usuário criar/operar sob uma árvore qualquer.

Tabela 1. Lista de Comandos do Interpretador

id	Comando	Objetivo
1	croot	Cria raiz da árvore
2	cleft	Cria filho esquerdo
3	cright	Cria filho direito
4	goleft	Caminha para o filho esquerdo
5	goright	Caminha para o filho direito
6	back	Volta para o nó ancestral
7	goroot	Volta para a raiz da árvore
8	search	Pesquisa pela existência de um nó
9	tree	Exibe a sequência de elementos da árvore em pré-ordem, pos-ordem ou in-ordem
10	deltree	Remove todos os nós da árvore ou uma sub-árvore de um nó
11	height	exibe a altura relativa da árvore
12	size	Exibe a quantidade de nós da árvore
13	cnextleft	Criar próximo filho disponível à esquerda
14	cnextright	Criar próximo filho disponível à direita
15	leafs	Exibe a quantidade de nós folhas
16	quit	Sair do interpretador de comandos

17	help	Exibe a lista de todos os comandos disponível
18	whereami	Exibe o local (conteúdo) onde se encontra o nó relativo

Cada comando deve ter um argumento de ajuda padronizado por "/?". A execução de **comando /?** Deve produzir um texto com uma ajuda sobre o comando e sua respectiva sintaxe.

As opções de 1 a 3 já foram (serão) discutidas em sala de aula. As opções de 4 a 7 fazem a "navegação" na árvore em construção. Conforme Figura 1, ao escolher a opção 4, por exemplo, o usuário "entra" na sub-árvore esquerda (se existir) da "raiz atual", e este passa a ser seu nó (raiz) atual (tal como numa árvore de diretórios de um sistema de arquivos).

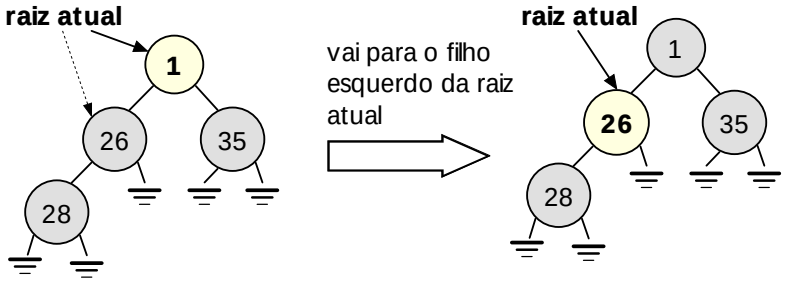

Figura 1. Caminhando para o filho esquerdo

Se, estando nesta sub-árvore esquerda, o usuário escolher a opção 3, então um filho direito será criado para **esta** sub-árvore esquerda, onde o ponteiro que representa a raiz atual está localizado atualmente. A opção 6 faz o usuário voltar para a raiz do nó atual (nó anterior) e a opção 7 faz o usuário voltar para a raiz absoluta da árvore, ou seja, o nó que originou toda a estrutura.

A opção 8 verifica a existência de um determinado nó. A opção 9 deve exibir a árvore de acordo com um dos três tipos de caminhamento. A opção 10 remove a sub-árvore de um nó onde se encontra atualmente o "cursor" (ponteiro relativo). Da mesma forma, pode ser utilizada para remover todos os nós da árvore, inclusive o nó raiz.

A opção 11 determina altura da árvore em relação à posição atual do "cursor". As opções 12 e 15 são auto-explicativas, pois só devolvem um número inteiro que representa a quantidade de nós da árvore e a quantidade de nós folhas, respectivamente.

As opções 13 e 14 criam um novo filho à esquerda ou à direita da árvore assim que encontrar algum nó com disponibilidade. Por exemplo, na figura 1, ao chamar a operação **cria próximo filho à esquerda,** o novo nó será criado como filho esquerdo do nó com valor 28. Caso deseje executar a operação **cria próximo filho à direita,** o novo nó será criado como filho direito do nó 35.

A opção 16 finaliza a execução do programa. A opção 17 exibe a lista de todos os comandos disponíveis. A opção 18 informa onde se encontra atualmente o "cursor" da árvore.

Para a árvore em questão, considere que serão armazenados valores do tipo **char.**

4. Simulação

O exemplo descrito a seguir serve apenas como guia para o planejamento e execução do projeto. A equipe está livre para decidir a forma como os comandos serão definidos. Entretanto, requisitos de **encapsulamento**, **abstração, modularização** e **atendimento aos requisitos** não podem ser

desprezados. O exemplo abaixo constrói a árvore ilustrada na Figura 1, bem como provê uma visão geral do que se espera para a interpretação dos comandos.

```
TreeBuilder App v1.0
Author: Alex (alex@ifpb.edu.br)
========================================================================
> help
    croot            Cria raiz da árvore
    cleft            Cria filho esquerdo
    cright           Cria filho direito
    goleft           Caminha para o filho esquerdo
    goright          Caminha para o filho direito
    back             Volta para o nó ancestral
    goroot           Volta para a raiz da árvore
    search           Pesquisa pela existência de um nó
    tree             Exibe a sequência de elementos da árvore
                     em pré-ordem, pos-ordem ou in-ordem
    deltree          Remove todos os nós da árvore ou uma sub-
                     árvore de um nó
    height           exibe a altura relativa da árvore
    size             Exibe a quantidade de nós da árvore
    cnextleft        Criar próximo filho disponível à esquerda
    cnextright       Criar próximo filho disponível à direita
    leafs            Exibe a quantidade de nós folhas
    quit             Sair do interpretador de comandos
    help             Exibe a lista de todos os comandos
                     disponível
    whereami         Exibe o local (conteúdo) onde se encontra
                     o nó relativo

> croot /?
  Cria o no raiz da árvore.
  Sintaxe:
  croot <valor>
> croot 10
  [msg] raiz com valor 10 criado com sucesso
> croot 20
  [msg] A arvore já possuiu uma raiz. Operação invalida!
> cleft 26
  [msg] Filho esquerdo com valor 26 criado com sucesso.
> cright 35
  [msg] Filho direito com valor 35 criado com sucesso.
> goleft
  [msg] A raiz relativa agora esta no no 26
> cleft 28
  [msg] Filho esquerdo com valor 28 criado com sucesso.
> tree /pre
  [ 1 26 18 35 ]
> tree /pos
  [ 28 26 35 1 ]
> treeee /pos
```

```
  Comando desconhecido
> tree /pos /pre
  Erro de sintaxe na execução do comando tree
  tree – lista os nós da árvore em caminhamento pré-ordem, pos-ordem ou in-ordem

  Sintaxe:
  tree <argumento>

  Argumentos
  /pre : caminhamento em pré-ordem
  /pos : caminhamento em pos-ordem
  /in  : caminhamento em in-ordem
> whereami
  Você esta localizado no 26
> quit
```

É importante ressaltar que cada equipe deve verificar as situações incompatíveis no uso das operações e tratá-las convenientemente. Por exemplo, Se o cursor já está posicionado no **nó raiz,** a operação **back** não tem como voltar ao nó ancestral, portanto, deve permanecer no mesmo local. Outro exemplo: se o cursor se encontra em um **nó folha,** não será possível executar um **goleft** ou **goright.** E assim por diante. O programa deve interagir com o usuário informando todos os passos das operações requisitadas.

Em caso de dúvidas no entendimento das operações, ou como deve ser realizado um determinado procedimento, a equipe deve obrigatoriamente consultar o professor. Qualquer decisão sem a devida consulta ao professor **é por conta e risco da equipe.**

5. Opcional

Se a equipe implementar a funcionalidade de "buffer de comandos", terá **até 2,0 pontos adicionais na prova escrita.** O buffer de comandos tem o objetivo de guardar a ordem dos comandos executados desde o início do programa, do comando mais recente para o comando mais antigo. Isto deve ser feito com o uso das teclas "seta pra cima" e "seta pra baixo". Cada pressionamento de "Seta pra cima" exibe o comando **n, n-1, n-2,...,n-m.** Cada pressionamento de "seta para baixo" caminha para o último comando executado. Dica: digite comandos no prompt do DOS e verifique o funcionamento das teclas "para cima" e "para baixo".

6. Entrega do Projeto

O código fonte do projeto deve ser empacotado em formato .zip ou .7z e enviado por email até as **10h00** do dia **30/09/2013**. Projetos enviados fora do prazo **NÃO SERÃO ACEITOS**. No arquivo compactado, deve conter apenas os arquivos necessários para a reprodução da aplicação, por exemplo, .dev, .c, .cpp, .h. Não envie o executável (**.exe**) e arquivos **.a**.

7. Observações

- Lógica e/ou operações copiadas serão anuladas em **todas** as cópias, independente de quem fez e quem copiou.
- Importante: uma má estrutura do programa, logo no início, prejudicará potencialmente o desenvolvimento do código. Utilize apontadores de forma correta logo na fase inicial, pois fica mais difícil encontrar erros lógicos quando o programa já está em avançado estágio de codificação e, principalmente, sair corrigindo em cascata todos os erros de programação que possivelmente venham a ser cometidos.
- Contate o professor para direcionar adequadamente a codificação do seu projeto.