Segundo Trabalho Prático de Estrutura de Dados I

**** Rafael de O. Cozer Matricula:2020101470**
DI/UFES**

Introdução

> O objetivo do projeto visa elaborar a criação de uma árvore binária para cálculo de expressões aritméticas. O código foi elaborado de forma relativamente simples, embora não fácil. Foram utilizadas apenas duas TAD's para o trabalho, sendo uma para a árvore binária e uma apenas para rodar o código, de modo a não "tumultuar" a main com trechos de código, visando um código mais limpo e organizado.

> Nos tópicos a seguir, um pouco de cada uma dessas funcionalidades implementadas, seguindo a linha de raciocínio utilizada.

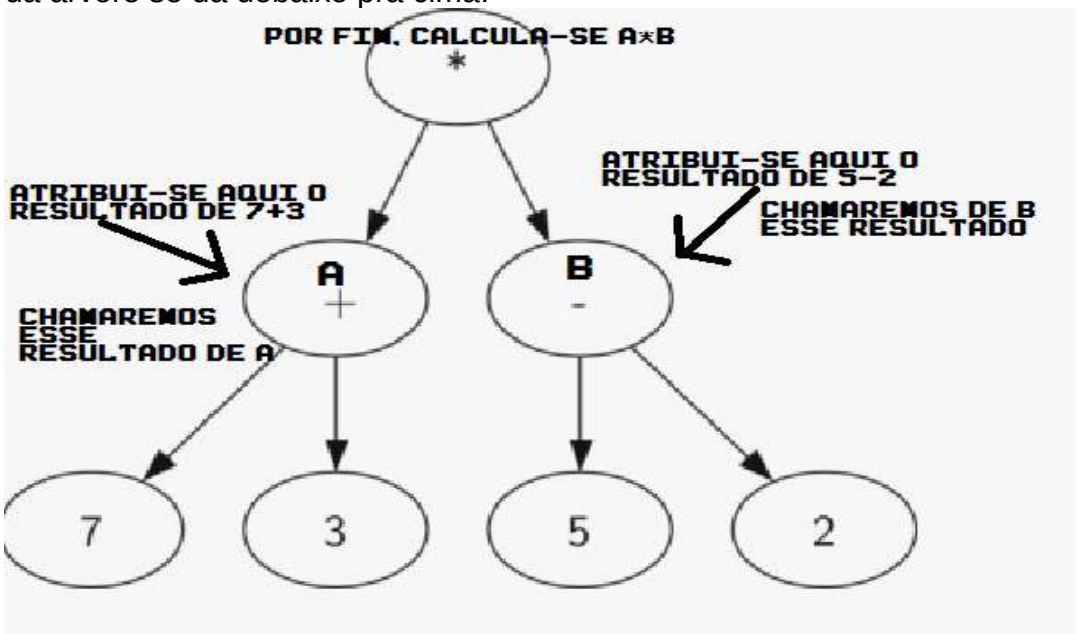
1. arvore.h

Principal TAD do trabalho, onde implementaram-se todas as funções relacionadas ao trabalho, ou seja, funções para criação e manipulação de árvores/nós.

Utilizou-se muito da **recursividade** nas funções dessa TAD, onde uma função chama a si mesma para dar continuidade ao código. Isso foi essencial para o fluxo do programa, uma vez que ao se trabalhar com árvores, a recursividade se torna necessária.

Sobre a estrutura de dados "árvore":

É uma estrutura que organiza seus elementos de forma hierárquica, onde existe um elemento que fica no topo da árvore, chamado de raiz e existem os elementos subordinados a ele, que são chamados de nós filhos. Cada nós filho pode conter zero, um ou mais de um nós filhos. No caso deste trabalho, cada nó pai poderia ter dois nós filhos. Em caso de nós sem nós filhos, estes então são chamados de nós folha e, aqui, nós folhas obrigatoriamente representam números/operandos. Enquanto isso, nós internos representam operadores e o nó raiz, representa o operador "final", que unirá as expressões da direita e esquerda da árvore, a fim de calcular o resultado final, uma vez que o cálculo da árvore se dá debaixo pra cima:



2. leArquivo.h

Esta TAD foi implementada exclusivamente para que o código pudesse ser rodado, através da função “leArquivo”, que deve ler o arquivo de entrada e trata-lo, utilizando as funções construídas no TAD arvore.h.
Em resumo, foi um TAD criado apenas para quesito organização de código.

Fluxo do Programa

O programa inicia-se na leArquivo, linha 6 de leArquivo.c. Então, o fluxo do programa acontece da seguinte forma:

- Abre-se o arquivo de entrada para leitura das expressões e cria-se os arquivos de saída, até então vazios;
- Dentro de um loop, que tem como condição rodar o trecho de código até o fim do arquivo de entrada, cria-se uma nova arvore, através da função “montaArvore”, onde passa-se o arquivo de entrada como parâmetro;
- Chama-se a função calculaExpressões, que irá calcular as expressões da árvore já montada. O resultado retornado dessa função, em float, é atribuído à uma variável “resultado”, criada dentro desse loop;
- Utiliza-se de fprintf’s para printar a estrutura do arquivo graphviz.txt e chama-se a função “preOrdem”, para organizar em preOrdem, dentro dessa estrutura do arquivo, os caracteres da árvore;
- Por fim, libera-se a memória alocada para a árvore e fecha-se os arquivos após o fim do loop, para que a memória alocada destes (ao serem abertos) também seja liberada.

Arquivos

É importante ressaltar a importância da localização dos arquivos de entrada, que **devem** estar localizadas dentro da pasta “data” para que o código funcione de forma apropriada.
Como arquivo de entrada, temos apenas um arquivo “entrada.txt”, onde estão as expressões aritméticas a serem atribuídas à arvores, como no exemplo a seguir:

((((5)-(3))*((4)/(1)))+(10))
((((10)*(3))+(5))/((10)-(5)))
((((47)-(7))*(2))-(3))

Como arquivos de saída, espera-se um “saída.txt”, onde devem estar, linha por linha, os respectivos resultados das expressões. Além disso, espera-se um arquivo “graphviz.txt“, para ser utilizado no GraphViz de modo a visualizar as árvores construídas.

Conclusão

O trabalho permitiu trabalhar bem o conceito da estrutura de dados “árvore”, uma vez que todo o projeto girou em torno dessa ideia. Embora simples, o projeto exigiu bastante raciocínio lógico uma vez que foi necessária a utilização de uma técnica chamada **“parsing”.** Tal técnica tem como objetivo realizar a análise sintática de uma string, de modo a encontrar sua estrutura lógica. Esta etapa foi essencial e indispensável para construção da árvore.
Foi um trabalho desafiador e houveram algumas dificuldades. Dessa forma, frente ao desafio, pôde-se aprimorar e fixar bem os conceitos vistos, de tal maneira a contribuir para evolução dos alunos enquanto cientistas

da computação.

Como principais dificuldades vistas, pôde-se apresentar:

- Dificuldade relacionada ao parsing
A maior dificuldade se encontrou ao tentar realizar o parsing das expressões matemáticas, o que atrasou o andamento do trabalho, uma vez que todo o resto dependia disso. Porém, uma vez encontrada a lógica para se fazer o parsing, o trabalho tornou-se mais simples, já que montar a árvore e realizar o cálculo das expressões foi relativamente tranquilo.
- A criação do graphviz.txt e enumeração de nós
Houve uma grande dificuldade em enumerar os nós da árvore, o que gerou transtornos na geração do arquivo graphviz.txt, uma vez que na estrutura do arquivo, os nós deveriam estar devidamente enumerados, para que as árvores pudessem ser plenamente vistas no Graphviz.
O arquivo foi gerado com sucesso, porém infelizmente o problema de enumeração dos nós não se conseguiu sanar.
- Problemas com \r\n do Windows
No Windows, sistema operacional utilizado durante o trabalho, ocorre que ao final das linhas das expressões, estava sendo considerado \r\n, o que ocasionava na criação de árvores fantasmas, já que o programa tratava apenas o \n. O problema foi descoberto printando-se o valor numérico do “caracter” vazio que estava construindo árvores vazias e procurando-se este valor na tabela ASCII, onde constatou-se se tratar de um \r.
Tal problema foi sanado utilizando um fgetc(arq), na linha 57 da arvore.c. O objetivo deste fgetc era “consumir” este caracter. Contudo, tal problema não ocorre no Linux, que utiliza apenas \n, e como os testes do trabalho serão realizados em ambiente Linux, essa linha foi comentada.

 # Bibliografia
https://pt.stackoverflow.com/
https://devdoc.net/
https://cboard.cprogramming.com/
https://www.geeksforgeeks.org/

https://www.programiz.com/