UFMG - UNIVERSIDADE FEDERAL

DE MINAS GERAIS

TRABALHO PRÁTICO 0

Belo Horizonte

2014

UFMG

Trabalho Pratico 0.

João Pedro Samarino Mat: 2013048933

1. Introdução

O trabalho teve como objetivo a implementação de um programa que multiplica matrizes complexas, para esta implementação foi necessário revisar e familiarizar com conceitos da linguagem C que já foram mais bem explicados em outros cursos, estes tem estrema importância para o curso de AEDS III, são eles: ambiente de programação Unix, alocação dinâmica e o utilitário make.

A multiplicação de matrizes complexas consiste em multiplicar uma matriz complexa A (NxM) por uma outra matriz complexa B (MxK) ,onde o numero de elementos colunas de "A" deve ser igual o numero de linhas de "B", caso contrario não se pode multiplicar. A ordem que as operações são realizadas e a mesma da multiplicação de matrizes normais, ou seja , cada elemento "C_{ij} " é obtido por meio da soma dos produtos dos elementos correspondentes da i-ésima linha de "A" pelos elementos da j-ésima coluna "B". O que difere de uma multiplicação normal são as operações de soma e multiplicação que estão ligadas a estrutura dos números complexos.

Um multiplicador de matrizes complexas realiza simplesmente operações com os números complexos de cada célula das matrizes é multiplicando linhas por colunas gerando assim uma matriz resposta, processo que é similar a uma simples multiplicação de matrizes mudando somente as operações de multiplicação e soma, pois são realizadas com números complexos.

2. Solução Proposta

A solução proposta constitui de simples algoritmos para processamento do texto, modularização das matrizes e operações com números complexos. Para facilitar o trabalho foram criados dois tipos abstratos de dados, e o programa foi separado em funções (modulos) elementares independentes, desta maneira foi possível criar um programa simples é menor.

2.1. Estrutura de Dados :

Para entendermos o funcionamento do programa primeiro temos que entender as estruturas de dados abstratas criadas para a solução, para então explorarmos os algoritmos de maneira completa.

Foram criadas duas estruturas básicas para armazenamento de registros de maneira mais abstrata, ou seja, para facilitar a manipulação pelas funções dos registros lá armazenados.

As estruturas de dados abstratos são:

- typedef struct NumeroComplexo_{double real; double imaginario; } NumeroComplexo;

A estrutura (NumeroComplexo) é responsável por armazenar um numero do tipo complexo, guardando separadamente a parte real da parte imaginaria através de duas variáveis do tipo double.

- typedef struct MatrizComplexa_{NumeroComplexo** numero;} MatrizComplexa;

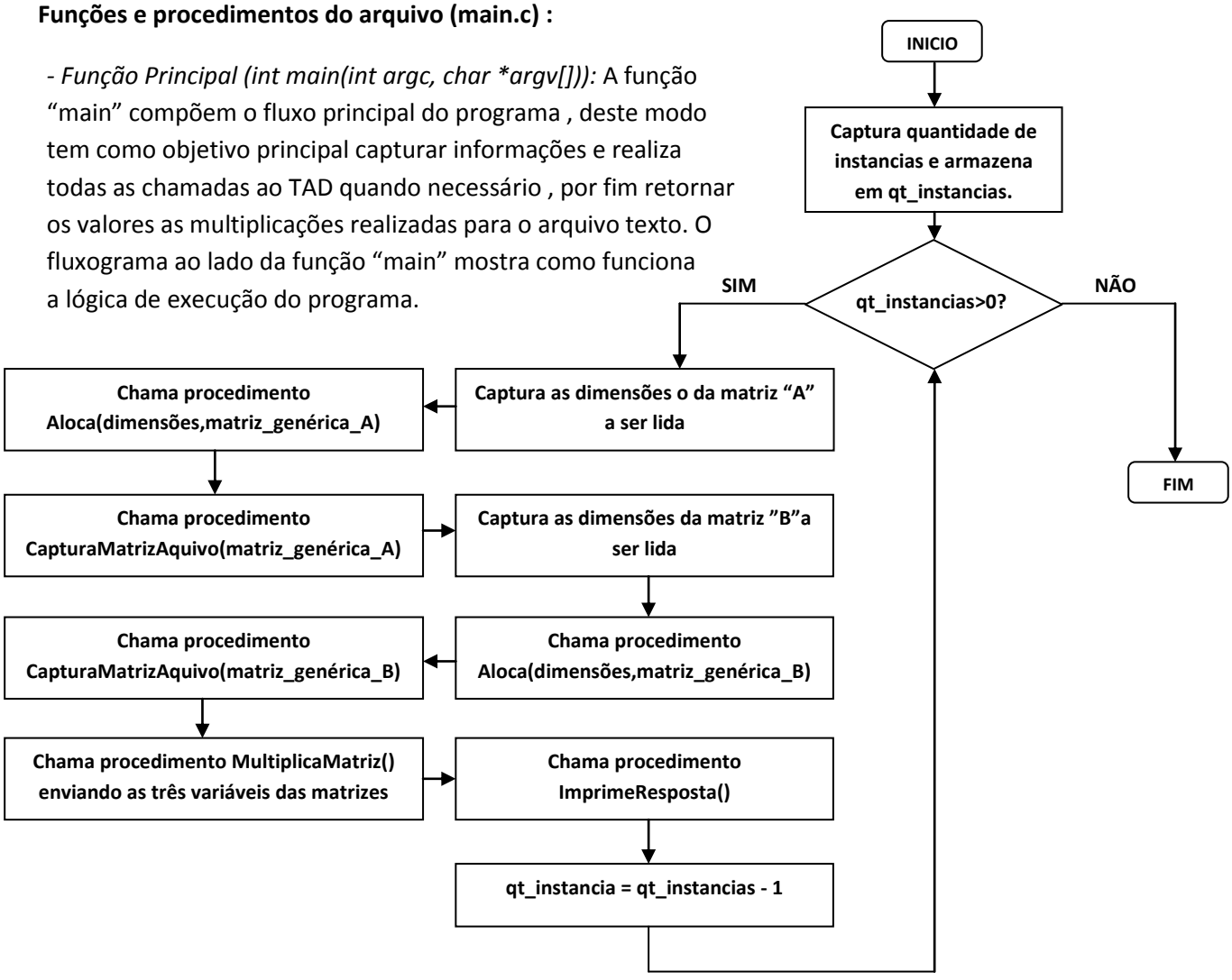
A estrutura (MatrizComplexa) é responsável por armazenar um ponteiro de ponteiros do tipo (NumeroComplexo), a estrutura foi constituída desta maneira pois a alocação da matriz e feita de maneira dinâmica assim o ponteiro de ponteiros pode ser alocado e desalocado quando for necessário em formato de uma matriz genérica (MxN).

2.2.Funções e Procedimentos

No programa foram criadas funções é procedimentos para manipulação de dados é execução da proposta de solução que tem como objetivo gerar o arquivo solução.

Funções e procedimentos do arquivo (main.c) :

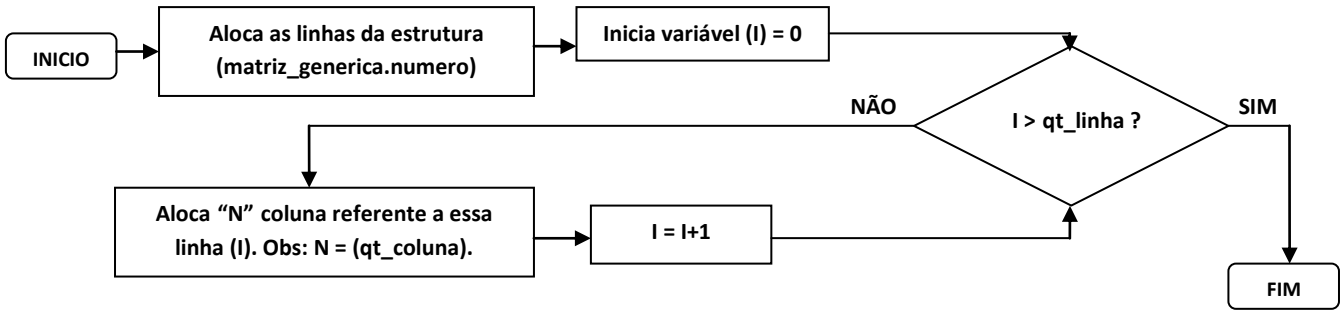
- *Função Principal (int main(int argc, char *argv[])):* A função "main" compõem o fluxo principal do programa , deste modo tem como objetivo principal capturar informações e realiza todas as chamadas ao TAD quando necessário , por fim retornar os valores as multiplicações realizadas para o arquivo texto. O fluxograma ao lado da função "main" mostra como funciona a lógica de execução do programa.



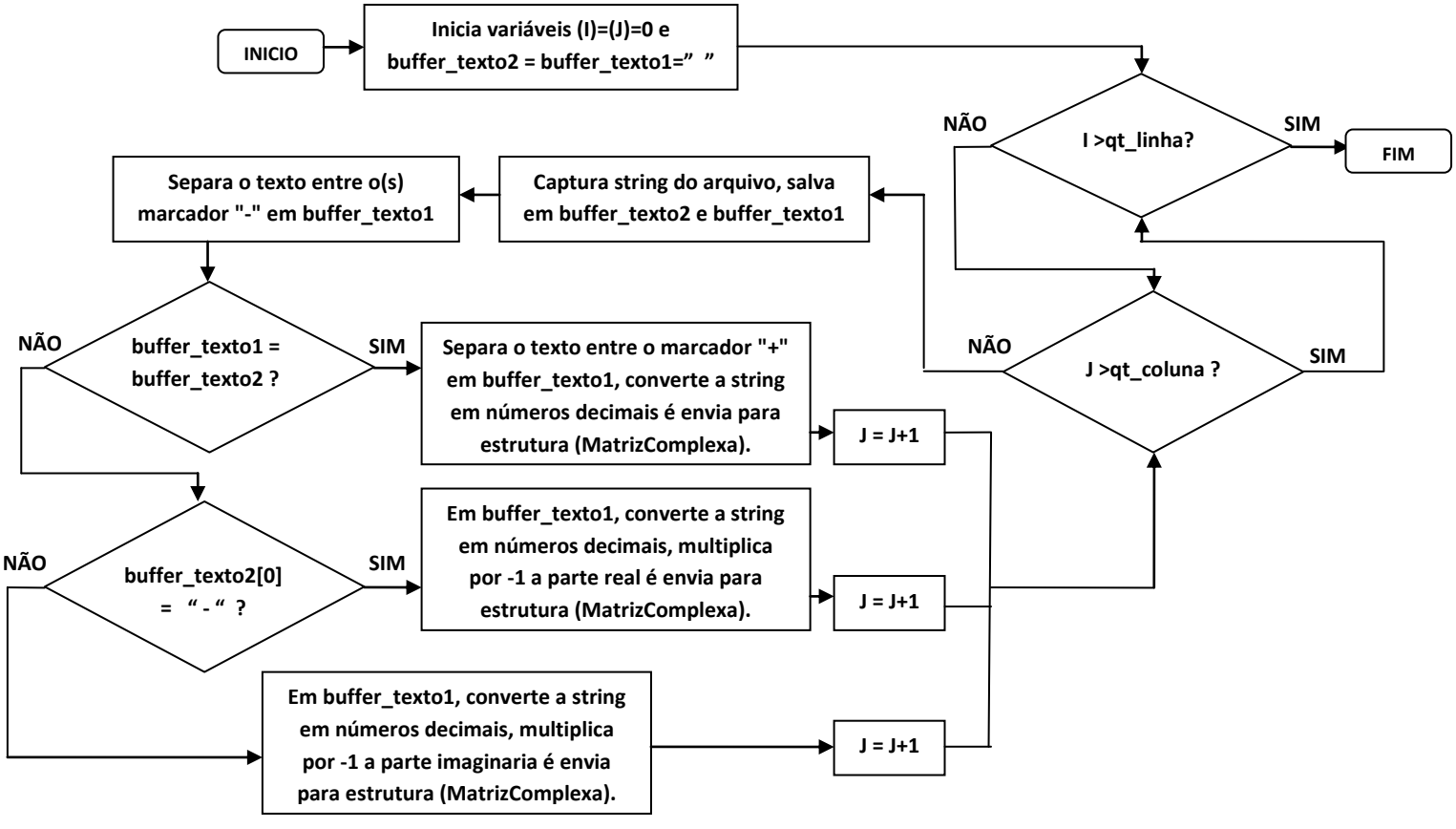
Funções e procedimentos do arquivo (matriz.c) :

- void Aloca (MatrizComplexa *matriz_generica , int qt_coluna , int qt_linha): O procedimento tem como objetivo alocar os números complexos dentro da estrutura (matriz_generica) recebida pelo procedimento, a quantidade de linhas e colunas e delimitada pelas variáveis

recebidas (qt_coluna) e (qt_linha). Para gerar a matriz na estrutura o programa primeiro aloca um vetor de ponteiros correspondente as linhas da matriz e depois através de um loop que e dado pela quantidade de linhas ele aloca as colunas. Através do fluxograma abaixo podemos ver a ordem de execução dos comandos nela realizados.



- void CapturaMatrizAquivo(FILE* in, MatrizComplexa* matriz_generica, int qt_coluna, int qt_linha) : O objetivo deste procedimento é capturar os números complexos da matriz do arquivo que esta em formato texto apontado por (in) e transformá-los em números da estrutura (NumeroComplexo), assim preenchendo a estrutura (matriz_generica) com eles. Para realizar a captura dos números primeiro se faz um loop em relação as linhas da matriz e outro loop interno percorrendo as colunas do mesmo, dentro do loop mais interno a função captura uma string que é referente a um numero complexo, com a string capturada e feita uma separação da parte imaginaria da parte real através da função (strtok()), usando primeiro o marcador (-) e depois o (+) em busca de quais foram usadas no numero , depois acontece uma serie de comparações para saber quais marcadores foram utilizados, com os valores separados em duas strings converte são convertidos para numeros fracionário pela função (atof()) é a parte imaginaria e a real são gravadas na estrutura (MatrizComplexa).



void Imprime(Apontador no) : O objetivo desta função é imprimir todos os itens presentes na arvore em ordem lexicográfica , esta impressão é feita na tela, formata da seguinte forma ”<palavra> <quantidade>”. Para realiza a impressão esta função usa o método recursivo de caminhamento central de arvore, percorrendo na ordem certa as palavras é as imprimindo.

int Busca(TipoChave c, Apontador no) : O objetivo desta função é buscar uma determinada palavra na arvore é imprimir o caminho até o elemento . Se o elemento está contido na arvore ao final da função e impresso “true” é se não “false”, para fazer essa busca a função faz um encaminhamento recursivo pela arvore em busca do elemento.

void Remove(TipoChave c, Apontador *no) : O objetivo desta função é remover um determinado item da arvore , se o item existe este será removido é a função ira imprimir formatado : “ remove true <palavra > ” se não existir o elemento, será impresso “remove false <palavra-chave>”. Para fazer esta exclusão, primeiro a função faz um encaminhamento recursivo até o elemento, quando o encontra a mesma verifica quantos filhos este nó possui, se o nó possuir apenas um filho , a função transfere o nó filho para a posição do item a ser excluído é limpa a memória que o mesmo ocupava. Se o nó que será excluído não possuir filhos ele o exclui é coloca o valor de (NULL) em sua posição na arvore, Porem se o nó possuir dois filhos e chamado a função (dois_filhos()) para tratar este caso.

void dois_filhos(Apontador *no, Apontador *no_t): O objetivo desta função é tratar um caso especial da função (Remove()) , quando o registro a ser excluído possui dois filhos , a função exclui esse registro é o troca com o elemento mais a esquerda da sub-arvore a direita do nó a ser excluído , para achar esse elemento ele usa um encaminhamento recursivo por isso a necessidade de separar em duas funções a tarefa de excluir um item.

Organização do Código , Decisões de Implementação e Detalhes Técnicos :

O código está dividido em três arquivos principais: main.c , texto.h , texto.c.

A estrutura de dados principal esta totalmente implementada dentro dos arquivos texto.h é texto.c, onde estão as funções organizadas e formatadas no padrão UTF8 (LINUX).

A entrada é saída de dados do programa foi feita como foi especificada no trabalho, através do ponteiro de arquivos stdin é stout.

O compilador utilizado foi o “GNU GCC Compiler” é a ide Code Blocks 12.11 para a programação no sistema operacional Debian Linux.

Observação : A função que tira acento espera um arquivo na codificação UTF8 , codificação padrão LINUX.

Para executá-lo basta copilar os arquivos e fornecer uma entrada adequada.

3. Análise de Complexidade

Função Maior: A função executa três comando O (1) depois entra em um loop que é executado (m) vezes onde (m) é a quantidade de letras da menor palavra. Dentro desse loop, são feitos um comando O(1) (comparação). Portanto temos: 3.O(1) + m.O(1) = O(m).

Função Insere: Esta função realiza um procedimento recursivo log(n) vezes no pior caso, onde n é a quantidade de nós da arvore que a mesma faz uma busca. Em cada chamada é feito uma chamada a uma função de O(m) onde m é a quantidade de letras da palavra q se percorre no nó atual. Portanto temos no pior caso O (m*log(n)) é no melhor O (m*1).

Procedimento Imprime: O procedimento varre toda á arvore usando encaminhamento central, é a cada chamada recursiva da mesma realiza um comando O(1) (impressão). Portanto temos: n*O(1) = O(n).

Função Busca: A função realiza um procedimento recursivo log(n) vezes no pior caso, onde n é a quantidade de nós da arvore, no melhor caso ele não realiza nenhuma chamada recursiva. Em cada chamada e feito um requerimento de outra função O(m) é outras operações O(1). Então temos que no pior caso O (m*log(n)) é no melhor O (m*1).

Função Remove: A função realiza um procedimento recursivo, em cada ciclo de execução , dentro do ciclo e chamado uma função O(m), a função pode terminar executando a chamada de uma função O(log(y)*m) onde y é a sub-arvore a partir do ponto de parada do nó a ser excluído , ela pode também executar comandos O(1) caso ela não entre nesta função. Como o procedimento recursivo desta função é executado no pior caso log(n) vezes é no melhor nenhuma vez, então temos que no pior caso O (m*(log(n)+log(y))) é no melhor O (m*1).

Função dois_filhos: a função realiza um procedimento log(y) vezes no pior caso , onde y é a sub-arvore a direita do nó enviado , sendo N sempre maior que Y , dentro de cada ciclo de execução do algoritmo temos comandos O(1) é chamada de uma função O(m) . Então temos que no pior caso O (m*log(y)) é no melhor O (m*1).

Procedimento minusculo: O procedimento executa um loop que vai de 0 a quantidade de letras da palavra a ser transformada que podemos chamar de (m) , dentro deste laço é executado somente um comando O(1). Então temos m*O(1) = O(m).

Função especial: a função realiza uma simples comparação O(1) é uma outra atribuição que também é O(1). Então temos 2*O(1) = O(1).

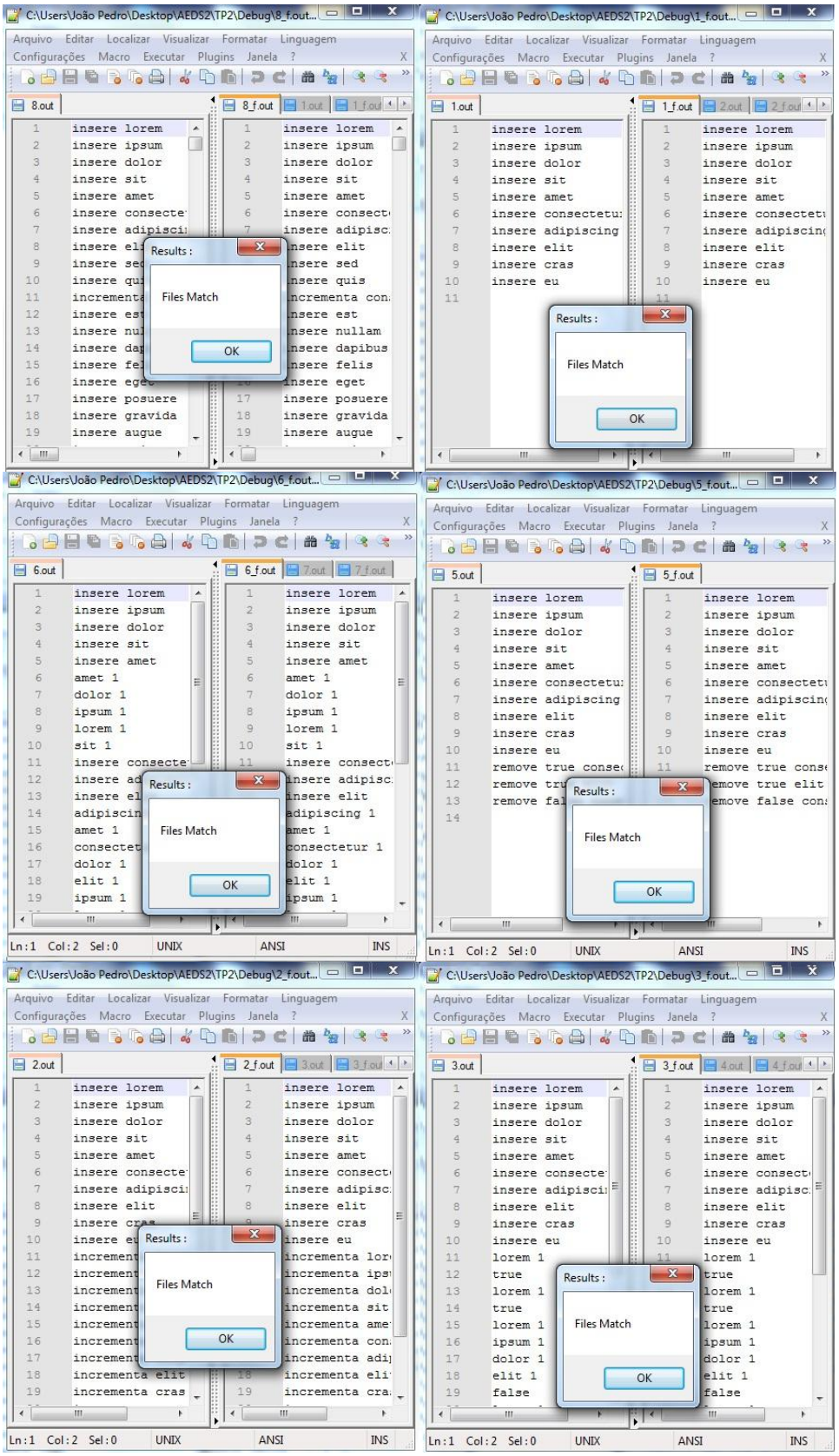
Função acento: a função realiza comparações O(1) é retorna o valor característico, sendo assim temos O(1) como ordem de complexidade.

Função comando: a função realiza no máximo três comparações O(1) é retorna um valor característico, sendo assim temos O(1) como ordem de complexidade.

Função main – função principal: a função principal faz 1 varreduras que podem varia de 0 a X , onde X é a quantidade de comandos, no final de cada ciclo o programa chama as funções da TAD , porem ele pode chamar ate X vezes uma função O(n) alem de fazer chamadas de funções internas O(1) , considerando que se pode ser feitos indefinidos testes consecutivos. Temos no pior caso O(x)*O(n)+O(1)=O(n*x) é no melhor O(1).

4. Testes

Vários testes foram realizados com o programa de forma a verificar o seu funcionamento. Os testes foram realizados em um Pentium core 2 duo, com 3 Gb de memória. A figura abaixo mostra os testes realizados, as saídas foram comparadas usando o programa NotePad++.


C:\Users\João Pedro\Desktop\AEDS2\TP2\Debug\8_f.out...
C:\Users\João Pedro\Desktop\AEDS2\TP2\Debug\1_f.out...
C:\Users\João Pedro\Desktop\AEDS2\TP2\Debug\6_f.out...
C:\Users\João Pedro\Desktop\AEDS2\TP2\Debug\5_f.out...
C:\Users\João Pedro\Desktop\AEDS2\TP2\Debug\2_f.out...
C:\Users\João Pedro\Desktop\AEDS2\TP2\Debug\3_f.out...
Results :
Files Match
OK

5. Conclusão

Foi possível com este trabalho aprender mais sobre arvores binárias é como implementá-las, também foi útil para o conhecimento em relação á processamento de strings, a principal dificuldade para este projeto foi testar a eficácia do algoritmo , pois existiam um numero muito grande de casos é exceções , o trabalho em geral também foi útil para melhor entendimento do funcionamento de ponteiros em C. Apesar de muito tempo gasto o trabalho me ajudou a melhorar meu processo de elaboração de programas é foi útil a meu aprendizado.

Referências

- Slides do professor disponibilizados no moodle.

Anexos

Listagem dos programas: main.c ; texto.h; texto.c