**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.

**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.

Apesar da idéia do trabalho ser simples, para se ter um bom aproveitamento dos recursos da linguagem, primeiro foi necessário pensar em estruturas de dados que iria atender, pois operações com números complexos são um pouco diferente das normais, então se fez necessário separar a parte real do numero da parte imaginaria, porem estas partes deveriam estar associadas ao mesmo numero, como poderemos ver a seguir no próximo item.

As operações de soma e multiplicações já com a estrutura montada foram feitas de maneira simplória. Para realizar uma soma de números complexos é necessário somar as partes reais dos números é separadamente somar as partes imaginarias do mesmo. Na multiplicação temos as seguintes operações que podemos demonstrar com um exemplo genérico, (a+bi)x(c+di), neste caso a parte real da solução seria (ac-bd) e a parte imaginaria (bc+ad), mesmo neste caso as operações são relativamente simples o que levou a implementar estas operações dentro da função que multiplica as matrizes, sem a necessidade de criar funções distintas.

**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 estão contidas no arquivo (matriz.h).

**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. O arquivo (main.c)contem procedimentos do fluxo principal e o (matriz.c) procedimentos relacionados ao TAD.

**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.

**INICIO**

**Captura quantidade de instancias e armazena em qt\_instancias.**

**SIM**

**NÃO**

**qt\_instancias>0?**

**Chama procedimento Aloca(dimensões,matriz\_genérica\_A)**

**Captura as dimensões o da matriz “A” a ser lida**

**FIM**

**Captura as dimensões da matriz ”B”a ser lida**

**Chama procedimento CapturaMatrizAquivo(matriz\_genérica\_A)**

**Chama procedimento Aloca(dimensões,matriz\_genérica\_B)**

**Chama procedimento CapturaMatrizAquivo(matriz\_genérica\_B)**

**Chama procedimento ImprimeResposta()**

**Chama procedimento MultiplicaMatriz() enviando as três variáveis das matrizes**

**qt\_instancia = qt\_instancias - 1**

- 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 ImprimeResposta(FILE\* out,MatrizComplexa \*matriz\_resposta,int qt\_coluna\_resposta,int qt\_linha\_resposta):O objetivo deste procedimento é imprimir a (matriz\_resposta) no arquivo apontado pelo ponteiro (out).Para realizar este procedimento são feitos dois loops referente as linhas e colunas da matriz ,são delimitados pelas variáveis (qt\_linha\_resposta) e (qt\_coluna\_resposta).Dentro do loop mais interno e feita a impressão no arquivo de saída no padrão especificado.

**Funções e procedimentos do arquivo (matriz.c) :**

-void MultiplicaMatriz(MatrizComplexa \*matriz\_generica\_a, MatrizComplexa \*matriz\_generica\_b, MatrizComplexa \*matriz\_resposta, int qt\_coluna\_resposta, int qt\_linha\_resposta ,int qt\_coluna\_a): O objetivo deste procedimento é multiplicar a (matriz\_generica\_a) pela (matriz\_generica\_b) é salvar o resultado na (matriz\_resposta). Para fazer está operação o procedimento realiza três loops internos, um percorre as linhas da matriz resposta o outro a coluna é loop mais interno a parte comum (qt\_coluna\_a ou qt\_linha\_b). Dentro deste loop mais interno ocorrem as operações de multiplicação dos elementos das matrizes para gerar o elemento resposta correspondente a posição dos contadores dos loops, vale lembrar que as operações são feitas separadamente, ou seja, os números reais são somados e multiplicados com a estrutura referente, o mesmo vale para a parte imaginaria dos números, assim gerando a matriz resposta.

- 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.

**3.Análise de Complexidade**

Procedimento Aloca: O procedimento tem uma complexidade de tempo O(n) onde n é o numero de linhas da matriz a ser alocada, esta complexidade se da devido ao loop para alocar as colunas da estrutura. A complexidade de espaço do programa é O(n\*m) onde m representa as colunas, pois esse procedimento tem como objetivo alocar uma estrutura em forma de matriz através da função malloc().

Procedimento CapturaMatrizAquivo: O procedimento tem uma complexidade de tempo O(n\*m), onde n e m se referem consecutivamente a quantidades de linhas e colunas da matriz a ser capturada. Esta complexidade de tempo e devido aos dois loops internos do programa, dentro do loop mais interno existe somente comparações e chamadas de custo fixo (1). A complexidade de espaço é O(1) pois são geradas neste procedimento somente variáveis de buffer.

Procedimento MultiplicaMatriz: O procedimento tem uma complexidade de tempo O(n\*m\*c), onde n e m se referem consecutivamente a quantidades de linhas e colunas da matriz resposta, c e referente a quantidade de linhas da matriz B. esta complexidade se da devido aos trez loops internos que percorrem a matriz resposta e mais interno realiza operações. Esta operações são todas com custo fixo (1). A complexidade de espaço é O(1) pois são geradas neste procedimento somente variáveis de buffer.

Procedimento ImprimeResposta: O procedimento tem uma complexidade de tempo O(n\*m), onde n e m se referem consecutivamente a quantidades de linhas e colunas da matriz a ser impressa no arquivo . A complexidade é devido aos loops necessários para percorrer toda a matriz, dentro do loop mais interno existem funções de impressão que tem um custo fixo (1), gerando assim esta complexidade. ). A complexidade de espaço é O(1) pois são geradas neste procedimento somente variáveis de buffer.

Função main – função principal: a função principal faz 1 varreduras que podem varia de 0 a X , onde X é a quantidade de instancias informada no arquivo de entrada, no final de cada ciclo o programa chama as funções da TAD , porem ele pode chamar ate X vezes uma função O(n\*m\*c) alem de executar comando O(1) , considerando que se pode ser feitos indefinidas multiplicações. Temos no pior caso O(x)\* O(n\*m\*c) +O(1)=O(c\*m\*n\*x) . A complexidade de espaço é O(n\*m), o programa pode chamar X vezes uma função de complexidade O(n\*m) porem a cada novo ciclo o espaço alocado e limpado para poder ser novamente alocado com um novo tamanho.

# 4.Implementação

Para desenvolver este trabalho, o mesmo foi dividido em três arquivos com o intuito de melhorar a organização e normalizar as funções e estruturas de acordo com os seus objetivos.

**4.1.Arquivos utilizados**

(main.c): Arquivo principal, responsável pelo fluxo do programa, contem funções relacionadas a entrada e saída de dados dos arquivos, contem a função (main()), que controla todas as chamadas ao TAD.

(matriz.c): Arquivo responsável pelos procedimentos relacionados ao TAD, ou seja, aqueles que iram fazer operações diretamente com a estrutura de dados relacionada.

(matriz.h): Este arquivo contem o cabeçalho das funções relacionadas ao TAD que estão contidas no arquivo (matriz.c), Também possui as estruturas de dados ligadas ao programa e suas definições.

**4.2. Compilação**

O programa deve ser compilado no compilador GCC através de um makeﬁle ou do seguinte comando no terminal:

*gcc principal.c simulador.c fila.c lista.c -o tp0*

**4.3. Execução**

A execução do programa tem como parâmetros:

• Arquivo de entrada.

• Arquivo de saída.

O comando no terminal para a execução do programa e da forma:

*./tp0 < arquivo de entrada > < Arquivo de saída >*

**4.3.1. formato dos arquivos**

Arquivo de entrada: O arquivo de entrada deve conter as especificações básicas dos procedimentos que iram ser realizados pelo programa. Este ira conter na seguinte ordem formatada, (Instancias), (dimensão da matriz 1), (elementos da matriz 1),(dimensão da matriz 2), (elementos da matriz 2) , e assim sucessivamente , pois a quantidade de matrizes a serem multiplicada é igual a quantidade de instancias vezes dois. Um exemplo de um arquivo de entrada seria:

**2**

**2 3**

**3+0i 4+0i 5+0i**

**6+0i 7+0i 8+0i**

**3 2**

**3+0i 4+0i**

**6+0i 7+0i**

**8+0i 9+0i**

**2 2**

**0+1i -1.5+0i**

**1+0i 0+0i**

**2 2**

**1+1i 2+0i 3+0i 4+1i**

Arquivo de saída: Este arquivo ira conter a solução da multiplicação das matrizes, este resultado e formatado na seguinte ordem, (quebra de linha), (elementos da matriz 1), (quebra de linha), (elementos da matriz 2) e assim sucessivamente, pois a quantidade de matrizes resposta e igual a quantidade de instancias informadas no arquivo de entrada.O exemplo do arquivo de saída associado a entrada a cima seria:

**73+0i 85+0i**

**124+0i 145+0i**

**-5.5+1i -6+0.5i**

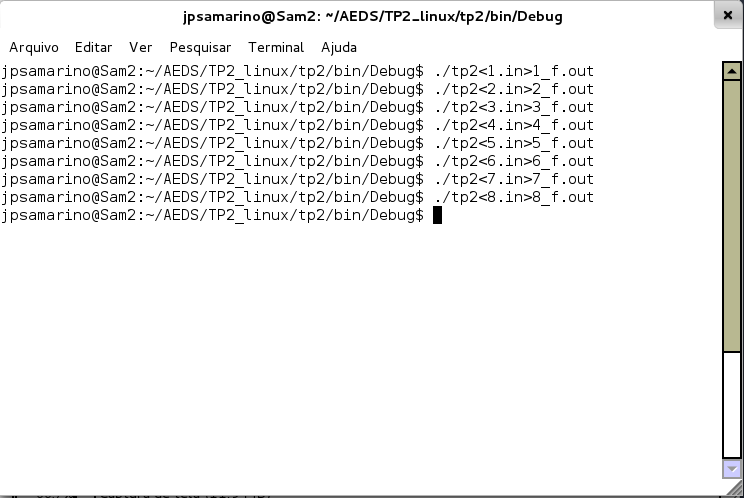
**1+1i 2+0i**

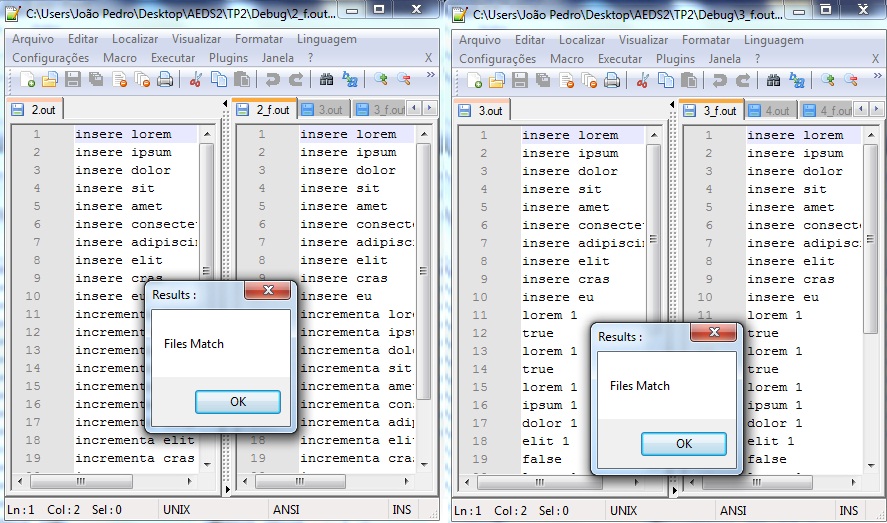
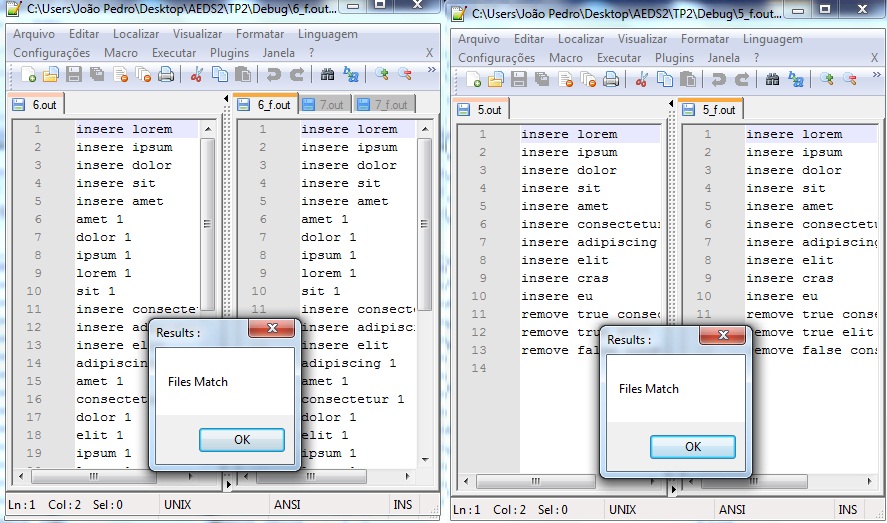
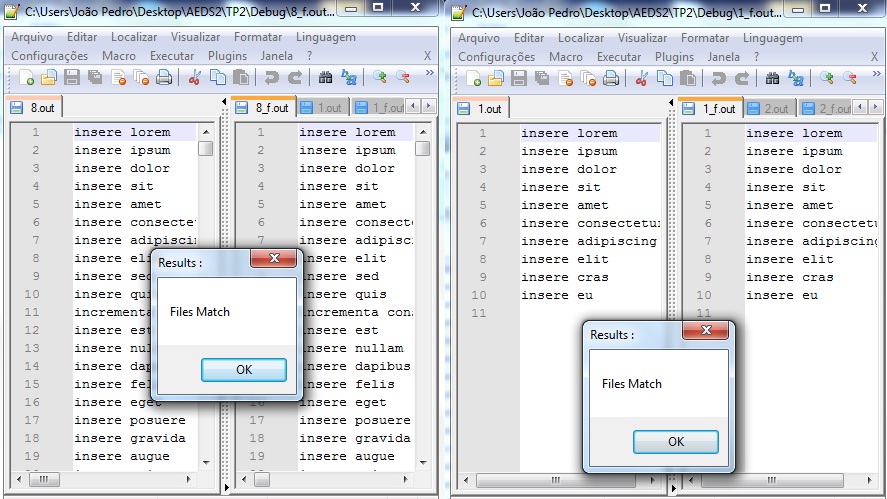
**5. Experimentos**

Os experimentos realizados tiveram como objetivo verificar o funcionamento do programa é o tempo gasto para execução do mesmo, verificando assim a qualidade dos algoritmos implementados no mesmo.

**5.1. Maquina utilizada**

Os testes e a compilação do programa foram realizadas em um Pentium core 2 duo, com 3 Gb de memória principal, o sistema operacional utilizado foi o Ubuntu Linux para arquitetura AMD64 (64 bits por ciclo de maquina) .





# 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