Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

Professores: Gisele Pappa e Wagner Meira Jr.
Monitores: A definir
Estruturas de Dados 2021.2
Belo Horizonte, 20 de Outubro de 2021

Trabalho Prático 0: Operações com matrizes alocadas dinamicamente

---

Trabalho Individual. Valor: 10 pontos **Entrega: 11 de Novembro de 2021**

1 Introdução

Matrizes são, sem dúvida nenhuma, representações matemáticas largamente utilizadas em várias áreas do conhecimento. Operações sobre matrizes são de fundamental importância desde as origens da Computação. Exemplos de operações populares são a soma, a multiplicação e a transposição de matrizes.

Um aspecto desafiador é que os tamanhos das matrizes podem variar significativamente e, frequentemente, essas matrizes estão armazenadas em arquivos, sendo necessário lê-las antes de iniciar o processamento.

Este trabalho prático tem por objetivo recordar alguns conceitos fundamentais do desenvolvimento de programas em linguagem C/C++, além de introduzir alguns conceitos em termos de tipos abstratos de dados, em particular a sua abstração, desempenho e robustez.

2 Especificações

O programa a ser implementado deve realizar três operações sobre matrizes: soma, multiplicação e transposição. Soma e multiplicação recebem duas matrizes, enquanto a transposição recebe uma matriz. A execução do programa é definida por parâmetros da linha de comando, como descrito a seguir:

Soma: '`programa -s -1 m1.txt -2 m2.txt -o res.txt`'

Multiplicação: '`programa -m -1 m1.txt -2 m2.txt -o res.txt`'

Transposição: '`programa -t -1 m1.txt -o res.txt`'

Os parâmetros '`-s`', '`-m`' e '`-t`' indicam qual a operação a ser realizada, soma, multiplicação e transposição, respectivamente. Os parâmetros '`-1`' e '`-2`' indicam os arquivos de entrada das matrizes, com a ressalva que a transposição é realizada apenas sobre a matriz contida no arquivo especificado por '`-1`'. Finalmente, a opção '`-o`' indica o arquivo onde a matriz resultante das operações é armazenada.

Sem perda de generalidade, vamos considerar que todas as matrizes sobre as quais operadores devem ser implementados são bidimensionais. Essas matrizes estão armazenadas em um arquivo do tipo texto, onde a primeira linha do arquivo contem dois inteiros m e n que definem as dimensões da matriz. O arquivo então contem m linhas cada uma com n números reais. O seu programa deve tratar inconsistências no arquivo de entrada.

A matriz deve ser implementada como um vetor de vetores do tipo '`double`' (i.e., '`double **`' e deve ser alocada dinamicamente a partir da leitura das dimensões da matriz em questão. Não se esqueça de desalocar as matrizes ao fim da computação.

A avaliação de desempenho deve ser realizada utilizando a biblioteca '`memlog`', que faz parte do programa exemplo anexo a este enunciado. Para cada operação deve ser avaliado o seu custo computacional à medida que variamos as dimensões das matrizes e também o seu padrão de acesso à memória

e localidade de referência. É necessário ainda uma outra opção de linha de comando, '`-p`', que indica o arquivo onde será feito o registro de acesso e uma opção '`-l`' que indica se devem ser registrados os acessos à memória, ou apenas o tempo de execução. Uma observação pertinente é que o custo de registro de acesso a memória pode ser significativo e sugerimos que sejam feitas duas baterias de testes, uma para desempenho computacional e outra para o padrão de acesso à memória.

A robustez do código deve ser implementada utilizando as macros definidas em '`msgassert.h`' e devem tratar todos os tipos de erros associados ao tipo abstrato de dados definido.

Em suma a mensagem de uso do seu programa pode ser:

```
programa
    -s                  (soma matrizes)
    -m                  (multiplica matrizes)
    -t                  (transpõe matriz 1)
    -1  m1.txt          (matriz 1)
    -2  m2.txt          (matriz 2)
    -o  res.out         (matriz resultante)
    -p  log.out         (registro de desempenho)
    -l                  (padrão de acesso e localidade)
```

Está disponível no Moodle uma mesma versão deste trabalho implementada com matrizes estáticas, ou seja, sem a utilização de ponteiros. Um primeiro passo é você estudar esse código com alocação estática, e depois reescrevê-lo utilizando alocação dinâmica.

3 Entregáveis

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é **terminantemente vetado**. Caso seja necessário, use as estruturas que **você** implementou nos Trabalhos Práticos anteriores para criar **suas próprias implementações** para todas as classes, estruturas, e algoritmos.

Você **DEVE utilizar** a estrutura de projeto abaixo junto ao '`Makefile`' disponibilizado no *Moodle*:

```
- TP
    |- src
    |- bin
    |- obj
    |- include
    Makefile
```

A pasta '`TP`' é a raiz do projeto; a pasta '`bin`' deve estar vazia; '`src`' deve armazenar arquivos de código ('`*.c`', '`*.cpp`', ou '`*.cc`'); a pasta '`include`', os cabeçalhos (*headers*) do projeto, com extensão '`*.h`', por fim a pasta '`obj`' deve estar vazia. O **Makefile** disponibilizado no **Moodle** deve estar na **raiz do projeto**. A execução do **Makefile** deve gerar os códigos objeto '`*.o`' no diretório '`obj`' e o executável do TP no diretório '`bin`'.

Existe no Moodle um vídeo onde um dos monitores do semestre passado discute sobre a função e uso de makefiles.

3.1 Documentação

A documentação do trabalho deve ser entregue em formato **pdf** e também **DEVE** seguir o modelo de relatório que será postado no Moodle. Além disso, a documentação deve conter **TODOS** os itens descritos a seguir **NA ORDEM** em que são descritos,

- Título, nome, e matrícula.
- **Introdução:** Contém a apresentação do contexto, problema, e qual solução será empregada.
- **Método:** Descrição da implementação que detalhe as estruturas de dados, tipos abstratos de dados (ou classes) e funções (ou métodos) implementados.
- **Análise de Complexidade:** Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
- **Estratégias de Robustez:** Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
- **Análise Experimental:** Apresenta os experimentos realizados em termos de desempenho computacional (avaliação do impacto da variação de parâmetros e da entrada) e eficiência de acesso à memória (padrão de acesso, localidade de referência e conjunto de trabalho).
- **Conclusões:** A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.
- **Bibliografia:** Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.
- **Instruções para compilação e execução**: Esta seção deve ser colocada em um apêndice ao fim do documento e em uma página exclusiva (não divide página com outras seções).

3.2 Submissão

Todos os arquivos relacionados ao trabalho devem ser submetidos na atividade designada para tal no *Moodle*. A entrega deve ser feita **em um único arquivo** com extensão **.zip**, com nomenclatura '`nome_sobrenome_matricula.zip`', onde '`nome`', '`sobrenome`', e '`matricula`' devem ser substituídos por suas informações pessoais. O arquivo **.zip** deve conter a sua documentação no formato **.pdf** e a estrutura de projeto descrita na Seção 3.

4 Avaliação

O trabalho será avaliado em 5 pontos com a seguinte distribuição:

1. Corretude na execução dos casos de teste - **50% da nota total**
2. Apresentação da análise de complexidade das implementações - **25% da nota total**
3. Estrutura e conteúdo exigidos para a documentação - **10% da nota total**
4. Indentação do código fonte - **5% da nota total**
5. Comentários no código fonte em forma de documentação em cima da assinatura de **TODAS** as funções criadas - **5% da nota total**
6. Cumprimento total da especificação - **5% da nota total**

Se o programa submetido não compilar[1], seu trabalho não será avaliado e sua nota será 0. Trabalhos entregues com atrasos sofrerão penalização de $2^d - 1$ pontos, com $d =$ dias de atraso.

5 Considerações Finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega estará tão distante quanto jamais estará.
2. Leia **atentamente** o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
4. **Plágio é CRIME**. Trabalho onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas. Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na sessão de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos**.

[1]Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.

6 FaQ (Frequently asked Questions)

1. Posso utilizar o tipo String? **SIM.**
2. Posso utilizar o tipo String para simular minhas estruturas de dados? **NÃO.**
3. Posso utilizar alguma biblioteca para tratar exceções? **SIM.**
4. Posso utilizar alguma biblioteca para gerenciar memória? **SIM.**
5. As análises e apresentação dos resultados são importantes na documentação? **SIM.**
6. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? **NÃO.**
7. Com relação ao código fonte, será avaliado apenas o que foi explicitado na Seção 4, incluindo indentação de código e comentário para documentação das funções? **SIM.**
8. Posso utilizar alguma estrutura de dados do tipo Queue, Stack, Vector, List, e **etc**..., do C++? **NÃO.**
9. Posso fazer o trabalho em dupla ou em grupo? **NÃO.**
10. Posso trocar informações com os colegas sobre a teoria? **SIM.**
11. Posso fazer o trabalho no Windows, Linux, ou MacOS? **SIM.**
12. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? **SIM.**