Universidade Federal de Minas Gerais

Escola de Engenharia

ELE078 - Programação Orientada a Objetos

Documentação TP1

Felipe Bartelt de Assis Pessoa 2016026841

23 de maio de 2022

Sumário

1 Implementação

1.1 Considerações

A interface da classe Matrix foi baseada na fornecida no escopo do trabalho. Assim, a classe tem três variáveis privadas: `rols, cols`, que são inteiros e representam o número de linhas e colunas da matriz, respectivamente; `m`, um ponteiro simples do tipo `Tvalor`, que armazena os elementos da matriz e é alocado de forma dinâmica. Ainda, apresenta dois operadores privados `operator>>` e `operator<<`. Os demais operadores e funções foram definidos como públicos.

Uma vez que todas as funções e operadores dependem de `template <class Tvalor>`, esses foram implementados dentro de `matrix.h`. As únicas bibliotecas utilizadas foram `iostream`, `fstream`, `algorithm` e `vector`.

1.2 Construtores e Destrutores

Foram definidos seis construtores, um construtor *default*, um construtor com valor fixo, um com elementos armazenados em ponteiros, um com elementos armazenados em vectors, um com elementos armazenados em um arquivo e um último construtor por cópia.

- `Matrix()`, o construtor *default*: não apresenta argumentos e foi definido para inicializar `rows = 0`, `cols = 0` e `m = new Tvalor[0]`. Assim inicializa uma matriz vazia.

- `Matrix(int t_rows, int t_cols, const Tvalor &val = 0.0)`, recebe o número de linhas `t_rows` e colunas `t_cols` desejados para a matriz. Também aceita um parâmetro opcional `val`, cujo valor *default* é 0, tal que todos os elementos da matriz sejam inicializados como esse valor.

- `Matrix(int t_rows, int t_cols, const Tvalor *vec)`, recebe o número de linhas `t_rows` e colunas `t_cols` desejados para a matriz, assim como um ponteiro `vec`, que armazena os elementos da matriz a ser criada. Esse construtor somente foi criado para que o funcionamento de alguns operadores fosse facilitado. A construção de matrizes com valores específicos é recomendada por meio de arquivos de texto ou pelo construtor via vectors.

- `Matrix(int t_rows, int t_cols, const vector<Tvalor> &vec)`, recebe o número de linhas `t_rows` e colunas `t_cols` desejados para a matriz, assim como um vetor `vec`, que armazena os elementos da matriz a ser criada. Permite assim a criação de uma matriz com todos os elementos especificados. Caso `vec` tenha menos elementos do que a matriz pode comportar, os demais elementos são preenchidos com 0s.

- `Matrix(const Matrix<Tvalor> &mat)`, o construtor por cópia: recebe uma matriz para ser copiada. O construtor copia as dimensões da matriz `mat`, assim como seu ponteiro `m`, sem que uma matriz aponte para a outra.

- `Matrix(ifstream &myFile)`, o construtor por arquivo: recebe um `ifstream` como referência. O arquivo utilizado deve conter os elementos de cada linha separada por espaços em branco e as linhas da matriz separadas por quebras de linha. Baseado no número de elementos percorridos por linha e o número de quebras de linha, o construtor determina as dimensões da matriz, além de alocar cada valor do arquivo em sua respectiva posição da matriz. Caso o número de elementos por linha não seja consistente (divergência entre o número de colunas desejado), uma `exception` é gerada.

- `~Matrix()`, o único destrutor. Deleta o ponteiro alocado dinamicamente por meio de `delete []m`, além de imprimir na tela que o destrutor foi chamado.

1.3 Getters e Setters

Foram implementados três *getters*: `getRows()`, `getCols()` e `get()`; e somente um *setter*, `reshape()`. Como todos são funções simples, foram implementadas como `inline`.

- `int getRows() const`: retorna o número de linhas da matriz, acessando `rows`.
- `int getCols() const`: retorna o número de colunas da matriz, acessando `cols`.
- `Tvalor get(int row, int col) const`: retorna o elemento de posição (row, col) da matriz. Os índices começam de 1, portanto uma `exception` é gerada caso qualquer um dos argumentos seja menor ou igual a 0. Caso se tente acessar elementos em posições maiores do que as comportadas pela matriz, também é gerada uma `exception`.
- `void reshape(int row, int col)`: função que permite o usuário alterar as dimensões atuais da matriz para uma dimensão `row`×`col`. Essa função não altera a ordem dos elementos, somente as dimensões da matriz, assim não é substituta de uma operação de transposição. Caso o produto `row`·`col` seja diferente de `rows`·`cols`, ou seja, o número de elementos comportados pela nova dimensão é diferente do número de elementos comportados atualmente pela matriz, uma `exception` é gerada. Também é gerada uma`exception` caso o usuário tente arbitrar alguma dimensão como 0.

1.4 Operadores

Todos os operadores especificados foram implementados, com o acréscimo do operador de multiplicação por constante com atribuição `operator*=` e o operador de atribuição `operator=`. Para a implementação dos operadores `operator>>` e `operator<<`, tornou-se necessário a pré definição dos mesmos no arquivo `matrix.h`, de forma a se tornar possível realizar o *overloading* com *template*.

- `Tvalor &operator()(int row, int col)`: Esse operador permite a obtenção de uma referência para o elemetno de posição (`row`, `col`) da matriz, sendo possível alterar seu valor por uma simples atribuição. Os índices começam de 1, portanto uma `exception` é gerada caso qualquer um dos argumentos seja menor ou igual a 0. Caso se tente acessar elementos em posições maiores do que as comportadas pela matriz, também é gerada uma `exception`.
- `Matrix &operator=(const Matrix &mat)`: Tornou-se necessário alterar o operador de atribuição para que a manipulação das cópias dos ponteiros `*m` fosse realizada corretamente. Assim, caso `this` aponte para o mesmo lugar de `&mat`, retorna-se uma referência para si mesmo. Caso contrário, realiza-se uma cópia do ponteiro por meio da função `std::copy()`, que é atribuída ao ponteiro `this->m`. A função retorna uma referência para a própria matriz `*this`.
- `Matrix operator+(const Matrix &mat) const`: O operador de soma de matrizes. Retorna uma nova matriz resultante da soma de `this` e `mat`, sem alterar nenhuma das duas. Caso as dimensões de ambas as matrizes sejam diferentes, uma `exception` é gerada.
- `Matrix operator+=(const Matrix &mat)`: O operador de soma de matrizes com atribuição. Altera os elementos de `this` para o resultante da soma entre `this` e `mat`, sem alterar essa última. Caso as dimensões de ambas as matrizes sejam diferentes, uma `exception` é gerada.

- `Matrix operator-(const Matrix &mat) const`: O operador de subtração de matrizes. Retorna uma nova matriz resultante da subtração de `this` e `mat`, sem alterar nenhuma das duas. Caso as dimensões de ambas as matrizes sejam diferentes, uma `exception` é gerada.

- `Matrix operator-=(const Matrix &mat)`: O operador de subtração de matrizes com atribuição. Altera os elementos de `this` para o resultante da subtração entre `this` e `mat`, sem alterar essa última. Caso as dimensões de ambas as matrizes sejam diferentes, uma `exception` é gerada.

- `Matrix operator*(const Tvalor &val) const`: O operador de multiplicação por constante. Retorna uma nova matriz resultante da multiplicação entre `this` e a constante `val`, sem alterar a matriz original.

- `Matrix operator*=(const Tvalor &val)`: O operador de multiplicação por constante com atribuição. Altera o valor dos elementos de `this` para o resultante da multiplicação entre seus elementos e a constante `val`.

- `Matrix operator*(const Matrix &mat) const`: O operador de multiplicação de matrizes. Retorna uma nova matriz resultante da multiplicação entre `this` e `mat`, sem alterar nenhuma das duas. Caso o número de colunas de `this` seja diferente do número de linhas de `mat`, uma `exception` é gerada.

- `Matrix operator*=(const Matrix &mat)`: O operador de multiplicação de matrizes com atribuição. Altera os elementos de `this` para o resultante da multiplicação entre `this` e `mat`, sem alterar essa última. Caso o número de colunas de `this` seja diferente do número de linhas de `mat`, uma `exception` é gerada.

- `Matrix operator~() const`: O operador de transposição. Essa função altera a ordenação dos elementos de uma cópia de `*m`, implementando a operação de transposição por meio de um ponteiro simples. Retorna-se uma nova matriz, com os elementos reordenados e dimensões invertidas `rows = this->cols` e `cols = this->rows`, dessa forma não se altera a matriz original.

- `bool operator==(const Matrix &mat) const`: Operador de igualdade. Confere se as dimensões das duas matrizes são iguais e também se todos os elementos, em ordem, são iguais. Caso ambas as condições sejam verdadeiras, o operador retorna uma variável booleana `true`, caso qualquer uma seja falsa, retorna `false`.

- `bool operator!=(const Matrix &mat) const`: Operador de desiigualdade. Confere se as dimensões das duas matrizes são diferentes e também se algum dos elementos, em ordem, é diferente. Caso alguma as condições sejam verdadeiras, o operador retorna uma variável booleana `true`, caso qualquer uma seja falsa, retorna `false`.

- `friend ostream &operator<<<>(ostream &os, const Matrix<Tvalor> &mat)`: Operador de inserção. É definido como `friend` para permitir o controle das variáveis privadas por funções públicas. O operador retorna um `ostream &os`, que contém strings necessárias para a impressão correta da matriz delimitada por barras verticais, os elementos são separados por espaços em branco e as linhas separadas por quebras de linha.

- `friend istream &operator>><>(istream &is, Matrix<Tvalor> &mat)`: Operador de extração. É definido como `friend` para permitir o controle das variáveis privadas por funções públicas. O operador espera elementos separados por espaços brancos. Caso a entrada seja dada interativamente, isso é, via `cin`, qualquer carácter não numérico ou

a tecla Enter ser pressionada, indica que o usuário terminou de preencher os elementos desejados. Os elementos são armazenados em um ponteiro temporário, alocado dinamicamente. Caso as dimensões da matriz original sejam nulas, um *Warning* é impresso indicando que será criado um vetor linha. Se as dimensões da matriz original não forem nulas, mas o número de elementos fornecidos for diferente da capacidade da matriz atual, uma `exception` é gerada. Se o número de elementos estiver correto, então o ponteiro `*m` é alterado para ser igual ao ponteiro temporário. As dimensões da matrizes não são alteradas, portanto os elementos inseridos devem estar em ordem.

2 Especificações Utilizadas

As versões do *vscode*, g++, gcc, gdb e as especificações do computador utiizado são mostradas abaixo, conforme as saídas dos comandos utilizados no terminal.

```
>> g++ --version
g++ (GCC) 12.1.0

>> gcc --version
gcc (GCC) 12.1.0

>> gdb -v
GNU gdb (GDB) 12.1

>> code --version
1.67.1
da15b6fd3ef856477bf6f4fb29ba1b7af717770d
x64

>> neofetch
                    /-                      fbartelt@fbartelt-nitroan51555
                   ooo:                     ------------------------------
                  yoooo/                    OS: ArcoLinux
                 yooooooo                   Kernel: 5.17.7-arch1-2
                yoooooooooo                 Uptime: 9 hours, 10 mins
              yooooooooooooo                Packages: 1179 (pacman)
            .yooooooooooooo                 Shell: zsh 5.8.1
           .ooooooooooooooooo               Resolution: 1920x1080
          .ooooooarcooooooooo               WM: i3
         .oooooooooo-oooooooooo             Theme: Arc-Dark [GTK2/3]
        .oooooooooo-   ooooooooooo          Icons: DarK-svg [GTK2/3]
       :ooooooooo.      :ooooooooo          Terminal: kitty
      :ooooooooo.       :ooooooooo          CPU: Intel i5-10300H (8) @ 4.500GHz
     :oooarcooo           .oooarcooo        GPU: Intel CometLake-H GT2 [UHD Graphics]
    :ooooooooy             .ooooooooo       GPU: NVIDIA GeForce GTX 1650 Mobile / Max-Q
   :ooooooooo    /ooooooooooooooooooo       Memory: 4995MiB / 7779MiB (64%)
  :ooooooooo       .-oooooooooooooooo.
 ooooooooo-              -ooooooooooooo.
oooooooo-                   .-ooooooooooo.
ooooooooo.                      -ooooooooo
```

3 Testes

Complementou-se o programa teste presente no escopo do trabalho para que fossem testadas todas as operações definidas. Acompanhou-se pelo *debugger* do vscode o comportamento de cada objeto, confirmando-se que as operações e variáveis`const` realmente atendiam ao proposto, além de se confirmar a alocação dinâmica correta dos ponteiros e o funcionamento dos destrutores, junto da desalocação de ponteiros. Também verificou-se, por meio de cláusulas `try` e `catch` o funcionamento de todas as exceções criadas.

Todos esses testes estão presentes no arquivo `main.cpp` e devem ser acompanhados dos arquivos `myMatrix.txt` e `myThrowMatrix1.txt`, para as devidas checagens de entrada via arquivo. Com os resultados obtidos por meio desse programa, permite-se concluir que todas as funções desejadas para a classe Matrix foram executadas e atendem ao funcionamento esperado para as mesmas. Todas as exceções geradas são factíveis e funcionam como proposto. Dessa forma, a classe proposta foi implementada por inteira com funcionamento coerente.