PROGRAMAÇÃO ORIENTADA À OBJETOS:
TRABALHO PRÁTICO 1:

INTRODUÇÃO:

O trabalho visa consolidar os conhecimentos de Programação Orientada a Objetos; dessa forma uma classe chamada Matriz deve ser implementada de tal forma que o usuário possa utilizá-la sem se preocupar com alocação de memória ou mesmo trabalhar posição por posição da matriz. Assim, pode-se comparar a implementação dessa classe com o tipo de programação *Matlab*.

COMO COMPILAR E EXECUTAR:

Compilação: *g++ Matriz.cpp Matriz.h main.cpp -o Matriz*
execução: *./Matriz*

IMPLEMENTAÇÃO DO CÓDIGO:

Header File:

O arquivo de cabeçalho foi nomeado como matriz.h, este arquivo foi desenvolvido com a finalidade de declarar todos os métodos, construtores e destrutor da classe Matriz. Desta forma, o código implementado simplesmente declara a classe da qual se refere o trabalho e suas atribuições, vale ressaltar que apenas o número de colunas e linhas da matriz e o seu ponteiro principal é declarado como privado, tendo em vista que o usuário não deve modificar esses valores diretamente. Assim o arquivo é apresentado pela figura abaixo:

```
#include <iostream>
using namespace std;
//criando a classe
class Matriz{
        private:
                int l,c;
        public:
                double **p;
                Matriz();
                Matriz(const Matriz &m);
                Matriz(int linhas, int colunas, const double &valor = 0);
                ~Matriz();
                int zeros();
                int getRows();
                int getCols();
                int ones();
                Matriz operator+(const Matriz &A)const;
                Matriz operator~();
                void operator+=(const Matriz &A);
                void operator*=(double const &right);
                Matriz operator*(double const &right);
                Matriz operator*(const Matriz &right);
                void operator*=(const Matriz &right);
                friend ostream& operator<< (ostream& os, const Matriz& m);
                Matriz operator-(const Matriz &A)const;
                void operator-=(const Matriz &A);
                friend istream& operator>>(istream& os, Matriz& m);
                void operator=(const Matriz &A);
                int operator==(const Matriz &A)const;
                int operator!=(const Matriz &A)const;
                double& operator()(int const &linhas, int const &colunas);
                void unit();
};
```

Construtores:

Os construtores foram implementados de acordo com o que foi recomendado, sempre que o usuário iniciar uma matriz sem argumentos uma matriz vazia deve ser criada. Existem mais duas configurações de inicialização de matriz sendo elas: a primeira, caso sejam usados 2 argumentos(linhas e colunas) a matriz deve ter o tamanho mencionado nos argumentos e seus valores devem ser padrões igual a 0, se a matriz for inicializada com 3 argumentos essa terá a dimensão dos dois primeiros argumentos(linhas e colunas) e os valores de cada posição será ditado pelo terceiro argumento. Por fim, um construtor de cópia também é implementado, recebendo uma matriz como argumento; tal construtor igual inicializa uma matriz com a mesma dimensão da matriz de argumento e atribui todas os elementos da matriz de argumento para a matriz criada.

Foi criado um destrutor que executa o comando *delete* para cada um dos vetores internos da matriz e por fim para o vetor de ponteiros:

```
//Destrutor
Matriz::~Matriz(){
        for(int i = 0; i < l ; i++){
                delete[] p[i];
        }
        delete[] p;
}
```

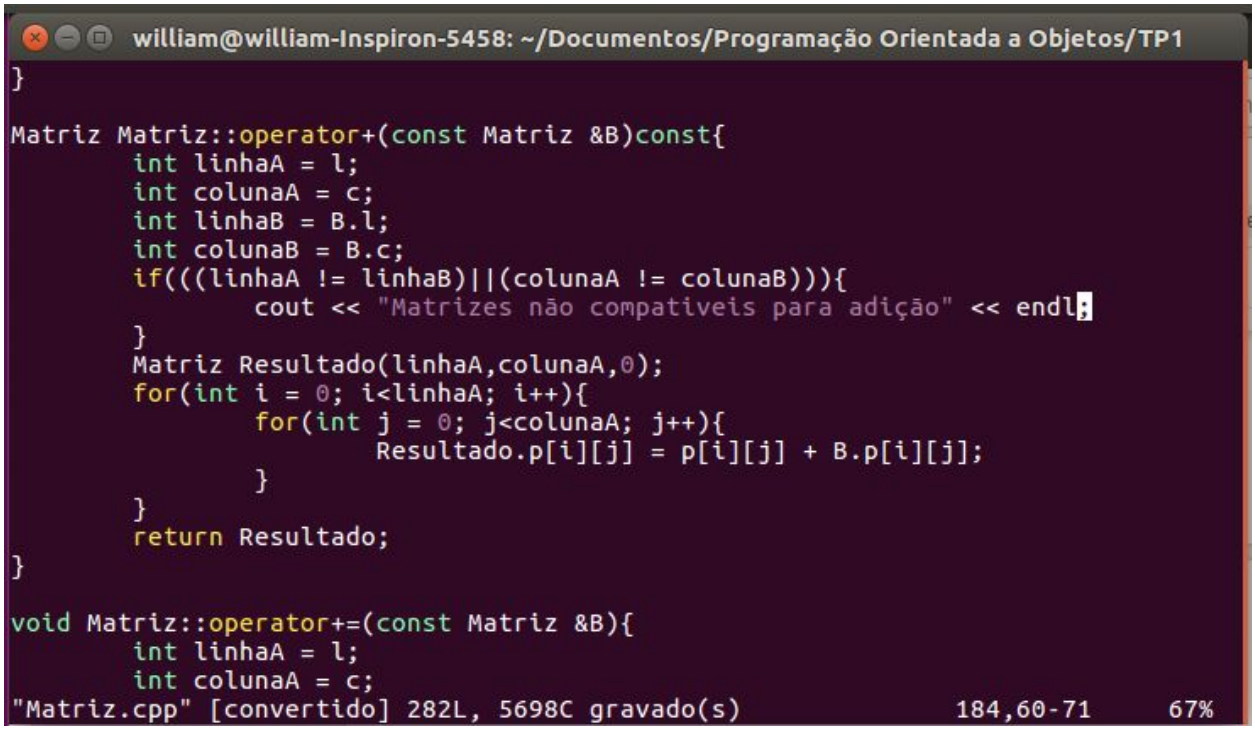
Sobrecarga de Operadores:

A sobrecarga de operadores foi implementada justamente para que o usuário utilizasse a classe como se utiliza o *Matlab*, sem se preocupar em atribuir valores ou realizar operações posição por posição em uma matriz.

Não ocorreu muitas complicações na implementação dos operadores aritméticos, pois a construção do código foi bem intuitiva e direta. Cada operador foi implementado com o comando "*operator*", na implementação de cada operador alguns casos foram verificados, com o intuito de não acessar posições de matrizes inexistentes ou realizar operações com matrizes incompatíveis.

Como exemplo, pode-se citar o operador "+" nesse comando era necessário verificar se as duas matrizes submetidas a operação tinham o mesmo tamanho, caso contrário a operação não poderia ser executada. Tal situação pode ser notada pela imagem abaixo no "*if*" dentro da função do operador "+":

Foi criado um overload do operador "=" de acordo com a orientação do professor, ou seja, que a matriz *this* fosse destruída, e depois fosse construída uma nova matriz de acordo com a matriz argumento. Nos operadores onde ocorria atribuição (**=, -=, +=,* =) era retornado a matriz *this* por referência, também por recomendação do professor, para tratar de situações como "*A = A = A;*".



Getters e Setters básicos:

No código há duas funções gets, tais funções retornam o valor das linhas e das colunas da Matriz, *GetRows() e GetCols()* respectivamente. Há também uma função que serve tanto como *get* quanto *set* de elementos (posições) de matrizes, já que retorna o endereço desse *double*, esta função é a sobrecarga *operator()(int const &linhas, int const &colunas)*, por exemplo, "*A(1,2) = 10*" ou "*double x = A(2,3)*".

Tratamento de exceções:

Nos operadores de soma e subtração, as matrizes devem ter a mesma dimensão, se não forem, o código não altera a matriz que iria ser modificada (caso += e -=), e não retorna nada (caso + e -). Além disso, lança uma mensagem de erro. O mesmo ocorre nos operadores associados a multiplicação de matrizes se o número de colunas da primeira matriz não for igual ao número de linhas da segunda.

Também há um erro quando o usuário tenta acessar um local da Matriz para modificá-lo por meio de uma atribuição e ele não existe. Isso ocorre no operador parênteses.

No construtor com argumentos para o número de linhas e colunas, também é lançado um erro caso sejam inseridos valores negativos para as dimensões.

Função main():

A função main tem como objetivo demonstrar a funcionalidade das funções mais complicadas implementadas ao criar o programa.

```cpp
#include <iostream>
#include "Matriz.h"
/* run this program using the console pauser or add your own getch, system("pause") or input loop */

int main(){
     cout << "Cria 3 Matrizes A, B e C"<< endl;
     Matriz A(3,3,1), B(3,3,3), C(3,3,2);
     cout << "A = " << endl << A << "B = " << endl << B << "C = " << endl << C << endl;
     cout << "C(1,1) = 3"<< endl;
     C(1,1) = 3;
     cout << C<< endl;
     ~C;
     cout << "Trasposta de C = " <<endl << C << endl;
     C*= B;
     cout << " C*=B ==>" << endl << C << endl;
     A+=C;
     cout << "A+=C ==>" << endl << A << endl;
     A-=C;
     cout << "A-=C ==>" << endl << A << endl;
     A*=2;
     cout << "A*=2 ==>" << endl << A << endl;
     A = B*C;
     cout << "A = B*C ==>" << endl << A << endl << "B =" << B<< endl;

      if (A!=B) {
          cout << "A é diferente de B" << endl;
      }

      A = B;

      cout << "A=B" << endl << A << endl;

           if (A==B) {
          cout << "A é igual a B" << endl;
      }
}
```

CONCLUSÃO:

O trabalho prático foi um sucesso, todas as funções e funcionalidades requeridas foram executadas, da forma orientada pelo professor e monitor. Isto é, foi feito quando possível a utilização de passagens por referência ou a palavra chave *const*, para que o código fosse otimizado. Foi também implementado o tratamento de exceções de uma forma elegante, utilizando o comando *throw std::invalid_argument(...)* para que um programador utilizando o nosso pacote pudesse debugar o seu programa de forma adequada.

É claro que estas soluções apresentaram a sua dificuldade, isto é, no aprendizado e entendimento do que foi estudado em sala de aula, mas este TP nos fez compreender melhor objetos e principalmente a sobrecarga de operadores.