THU Vasiliy Klyosov

A5: Generic Programming

Matrix Class Definition + Constructors and Operators Definitions

```
template<typename T>
class Imatrix {
private:
    std::vector<std::vector<T>> data;

public:
 // look at the implementation in the code for more intriguing details
};

Imatrix() {}

Imatrix(size_t rows, size_t cols)
    : data(rows, std::vector<T>(cols, T{})) {}

T& operator()(size_t x, size_t y) { /* stuff */ }
const T& operator()(size_t x, size_t y) const { /* things */ }

Imatrix& operator=(const Imatrix& other) {}
Imatrix& operator=(Imatrix&& other) noexcept {}
Imatrix(const Imatrix& other) : data(other.data) {}
Imatrix(Imatrix&& other) noexcept : data(std::move(other.data)) {}
```

Imatrix class template, which is a generic class that allows to create matrices of different data types ('T') with constructors and overloaded function call operators for accessing individual elements of the matrix + assignment and move constructors and assignment operators for the class, which allow to copy and move matrix data from one instance to another

Arithmetic Operations

```
Imatrix operator+(const Imatrix& other) const {}
Imatrix operator-(const Imatrix& other) const {}
Imatrix operator*(const Imatrix& other) const {}
Imatrix operator/(const T& scalar) const {}
```

Overloaded operators that perform basic matrix operations like addition, substraction, multiplication, division using scalar

```
std::vector<T> Row(size_t n) const {}
std::vector<T> Column(size_t n) const {}
void Move(size_t x, size_t y) {}
```

Functions to extract rows and columns from the matrix, as well as move the data in rows

Chess_piece

```
enum class PieceType {
    EMPTY,
    PAWN,
    KNIGHT,
    BISHOP,
    ROOK,
    QUEEN,
    KING
};

class Chess_piece {
private:
    PieceType type;
    bool is_white;

public:
    Chess_piece() : type(PieceType::EMPTY), is_white(true) {}
    Chess_piece(PieceType t, bool white) : type(t), is_white(white) {}

    PieceType get_type() const {
        return type;
    }

    bool is_white_piece() const {
        return is_white;
    }
};
```

Enum for types of chess pieces The class 'Chess_piece' - represents a chess piece and includes info about its type

Main

```
int main() {
    try {
        // Testing Imatrix<int>
        Imatrix<int> int_matrix(2, 2);

        Imatrix<std::string> str_matrix(2, 2);
```

```
        Chess_piece empty_piece(PieceType::EMPTY, true);
        Imatrix<Chess_piece> chess_matrix(2, 2);

    } catch (const std::exception& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

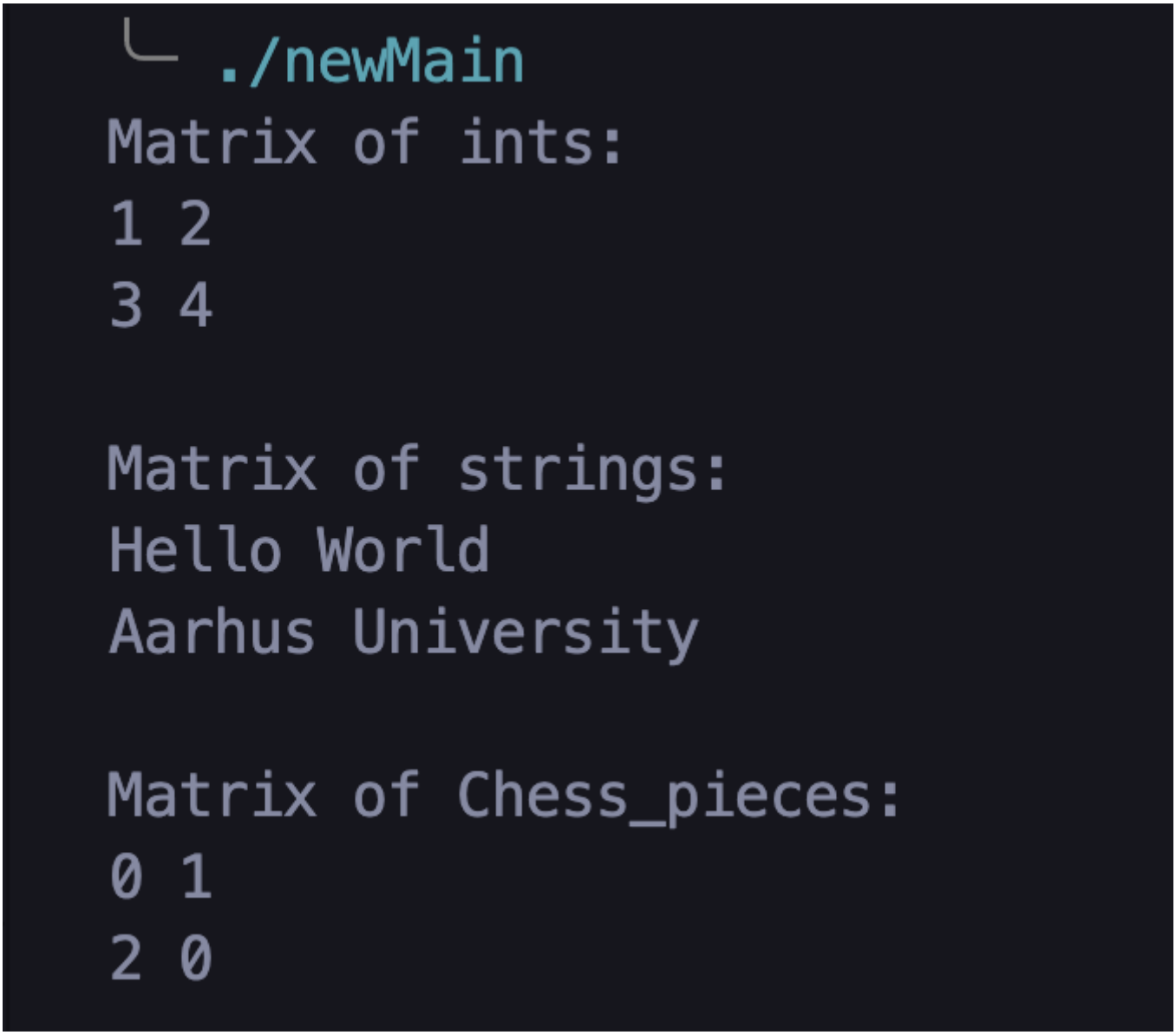
instances of matrices and operations that demostrate the usage of Imatrix class with different data types

Results:

To compile my code and create an executable file I did:

```
c++ -std=c++20 -o newMain newMain.cpp -lstdc++
```

Results I achieved:



And in case of an error like code:

```
        Imatrix<int> int_matrix(2, 2);
        int_matrix(0, 0) = 1;
        int_matrix(0, 1) = 2;
        int_matrix(1, 0) = 3;
        int_matrix(1, 1) = 4;

        // Access an element out of range to trigger an exception
        int value = int_matrix(2, 1);  // This will throw an exception

    } catch (const std::exception& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }
```

We will get