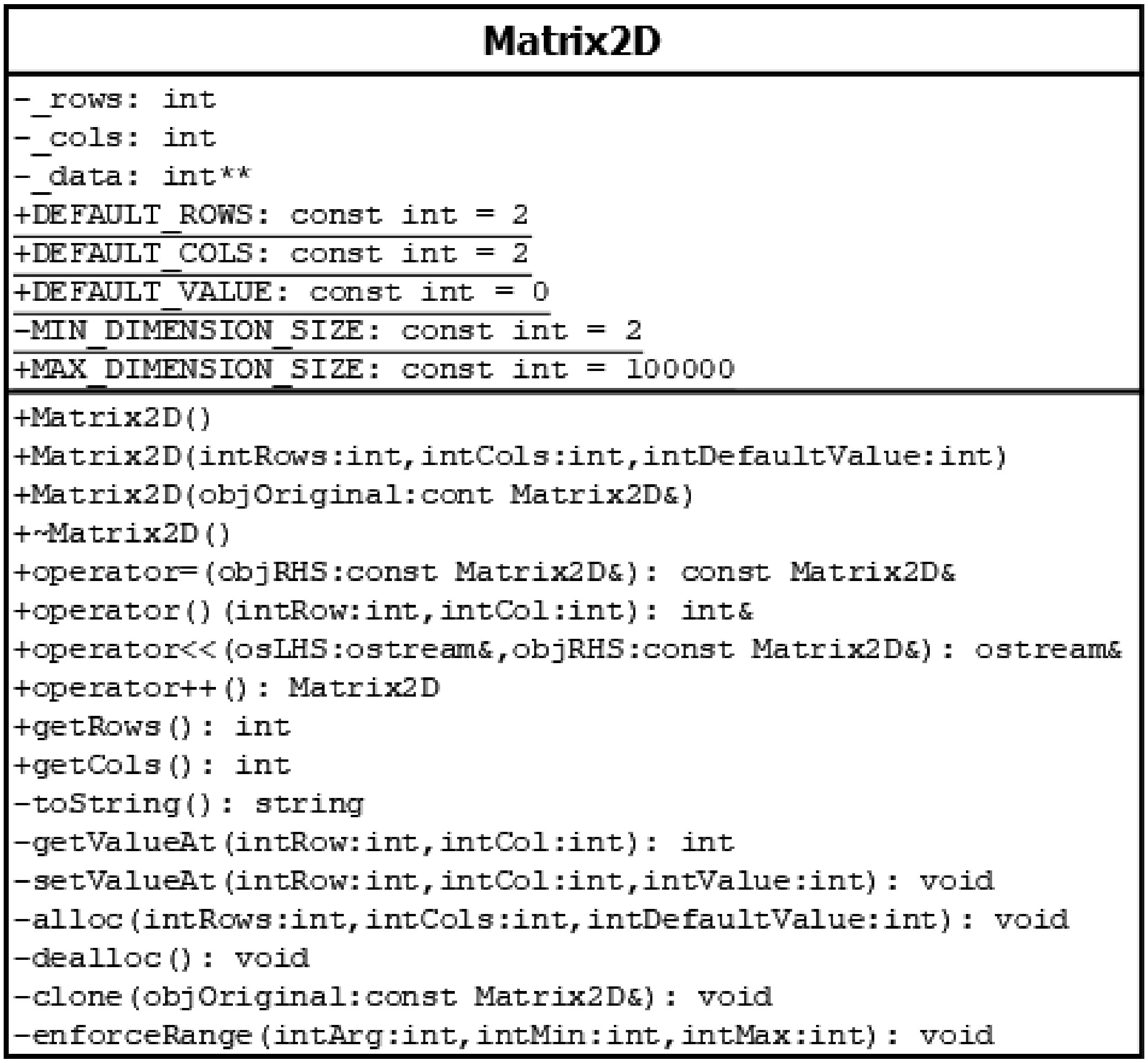
SP SITHUNGU

200000000

PRACTICAL 6 DESIGN



```
Matrix2D

-_rows: int
-_cols: int
-_data: int**
+DEFAULT_ROWS: const int = 2
+DEFAULT_COLS: const int = 2
+DEFAULT_VALUE: const int = 0
-MIN_DIMENSION_SIZE: const int = 2
+MAX_DIMENSION_SIZE: const int = 100000

+Matrix2D()
+Matrix2D(intRows:int,intCols:int,intDefaultValue:int)
+Matrix2D(objOriginal:cont Matrix2D&)
+~Matrix2D()
+operator=(objRHS:const Matrix2D&): const Matrix2D&
+operator()(intRow:int,intCol:int): int&
+operator<<(osLHS:ostream&,objRHS:const Matrix2D&): ostream&
+operator++(): Matrix2D
+getRows(): int
+getCols(): int
-toString(): string
-getValueAt(intRow:int,intCol:int): int
-setValueAt(intRow:int,intCol:int,intValue:int): void
-alloc(intRows:int,intCols:int,intDefaultValue:int): void
-dealloc(): void
-clone(objOriginal:const Matrix2D&): void
-enforceRange(intArg:int,intMin:int,intMax:int): void
```

```cpp
#ifndef MATRIX2D_H
#define MATRIX2D_H

#include <iostream>
#include <string>

enum ERROR_CODE{
    SUCCESS,
    ERROR_ARGS,
    ERROR_RANGE
};

class Matrix2D{
public:
    Matrix2D();
    Matrix2D(int intRows, int intCols, int intDefault);
    Matrix2D(const Matrix2D& objOriginal);

    const Matrix2D& operator=(const Matrix2D& objRHS);
    int& operator()(int intRow, int intCol);
    friend std::ostream& operator<<(std::ostream& osLHS, const Matrix2D& objRHS);
    Matrix2D operator++();

    int getRows() const;
    int getCols() const;

    static const int DEFAULT_ROWS = 2;
    static const int DEFAULT_COLS = 2;
    static const int DEFAULT_VALUE = 0;
    static const int MIN_DIMENSION_SIZE = 2;
    static const int MAX_DIMENSION_SIZE = 100000;

    ~Matrix2D();
private:
    std::string toString() const;
    int getValueAt(int intRow, int intCol) const;
    void setValueAt(int intRow, int intCol, int intValue);
    void alloc(int intRows, int intCols, int intDefaultValue);
    void dealloc();
    void clone(const Matrix2D& objOriginal);
    void enforceRange(int intArg, int intMin, int intMax) const;
    int** _data;
    int _rows;
    int _cols;
};

#endif // MATRIX2D_H
```

```cpp
#include "Matrix2D.h"

#include <cassert>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

Matrix2D::Matrix2D() : Matrix2D(DEFAULT_ROWS, DEFAULT_COLS, DEFAULT_VALUE){}

Matrix2D::Matrix2D(int intRows, int intCols, int intDefaultValue){
    alloc(intRows, intCols, intDefaultValue);
}

Matrix2D::Matrix2D(const Matrix2D& objOriginal) : Matrix2D(objOriginal._rows, objOriginal._cols, DEFAULT_VALUE){
    clone(objOriginal);
}

const Matrix2D& Matrix2D::operator=(const Matrix2D& objRHS){
    if(this != &objRHS){ // Check for self-assignment.
        dealloc();
        alloc(objRHS._rows, objRHS._cols, DEFAULT_VALUE);
        clone(objRHS);
    }
    return *this;
}

int& Matrix2D::operator()(int intRow, int intCol){
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    return _data[intRow][intCol];
}

ostream& operator<<(ostream& osLHS, const Matrix2D& objRHS){
    /*
     * We can use the already existing
     * toString member function from P5.
     */
    osLHS << objRHS.toString() << endl;
    return osLHS;
}

Matrix2D Matrix2D::operator++(){
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
            // increment every value from the array by 1.
            _data[r][c] += 1;
        }
    }
    return *this;
}

string Matrix2D::toString() const{
    stringstream ssReturn;
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
            ssReturn << _data[r][c] << ' ';
        }
        ssReturn << endl;
    }
    return ssReturn.str();
}

int Matrix2D::getRows() const{
    return _rows;
}

int Matrix2D::getCols() const{
    return _cols;
}

int Matrix2D::getValueAt(int intRow, int intCol) const{
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    return _data[intRow][intCol];
}

void Matrix2D::setValueAt(int intRow, int intCol, int intValue){
    enforceRange(intRow, 0, _rows - 1);
    enforceRange(intCol, 0, _cols - 1);
    _data[intRow][intCol] = intValue;
}

void Matrix2D::alloc(int intRows, int intCols, int intDefaultValue){
    enforceRange(intRows, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
    enforceRange(intCols, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
    _rows = intRows;
    _cols = intCols;
    _data = new int*[_rows];
    for(int r = 0; r < _rows; r++){
        _data[r] = new int[_cols];
        for(int c = 0; c < _cols; c++){
            _data[r][c] = intDefaultValue;
        }
    }
}

void Matrix2D::dealloc(){
    assert(_data != nullptr);
    for(int r = 0; r < _rows; r++){
        delete [] _data[r];
    }
    delete [] _data;
}

void Matrix2D::clone(const Matrix2D& objOriginal){
    for(int r = 0; r < _rows; r++){
        for(int c = 0; c < _cols; c++){
            _data[r][c] = objOriginal._data[r][c];
        }
    }
}

void Matrix2D::enforceRange(int intArg, int intMin, int intMax) const{
        if(intArg < intMin || intArg >intMax){
            cerr << intArg << " must be in [" << intMin
                 << ", " << intMax << "]" << endl;
            exit(ERROR_RANGE);
        }
}

Matrix2D::~Matrix2D(){
    dealloc();
}
```

```cpp
#include "Matrix2D.h"

#include <iostream>

using namespace std;

int rangedRandom(int intLB, int intUB);

int main()
{
    // Testing the fully parameterised constructor.
    Matrix2D objMatrix(10, 10, 1);
    // Testing the copy constructor.
    Matrix2D objCopy = objMatrix;
    //Testing the indexing operator.
    for(int r = 0; r < objCopy.getRows(); r++){
        for(int c = 0; c < objCopy.getCols(); c++){
            objCopy(r, c) = rangedRandom(0, 8);
        }
    }
    // Testing the overloaded assignment operator.
    Matrix2D objSmallerSizedMatrix(2, 2, 5);
    objSmallerSizedMatrix = objCopy;
    // Testing the pre-increment operator.
    ++objSmallerSizedMatrix;
    // Testing the stream insertion operator.
    cout << objSmallerSizedMatrix << "Done!" << endl;
    return 0;
}

int rangedRandom(int intLB, int intUB){
    return rand() % (intUB - intLB + 1) + intLB;
}
```