UEE1303 S18: Object-Oriented Programming

LAB #13: Polymorphism

What you will learn from Lab 13

Exercise 13-1:

✓ Mtx is an abstract class which defines the interface of derived class. FullMatrix and SymmetricMatrix are two derived class and inherit from abstract class Mtx. Moreover, UpperTriMatrix and LowTriMatrix inherit from SymmetricMatrix. Please finish this program and show correct results. Note that you can add any member if necessary.

```
#include <iostream>
using namespace std;
class Mtx
{
public:
	virtual int &operator()(int i, int j) = 0;
	virtual const int &operator() (int i, int j) const = 0;
	virtual void showMatrix() const = 0;
	virtual ~Mtx(){}
protected:
	int dim;
};
class FullMatrix: public Mtx
{
public:
	FullMatrix(int n)
	{
		dim = n;
		matrix = new int *[dim];
		for (int i = 0; i < dim; i++)
matrix[i] = new int [dim];
		for (int i = 0; i < dim; i++)
			for (int j = 0; j < dim; j++)
matrix[i][j] = 0;
	}
	int &operator()(int i, int j)
	{
		// you may provide boundary checking
		return matrix[i][j];
	}
	// …
private:
	int **matrix;
};
class SymmetricMatrix: public Mtx
{
public:
	SymmetricMatrix(int n)
```

```
    {
        dim = n;
        matrix = new int *[dim];
        for (int i = 0; i < dim; i++)
            matrix[i] = new int [i+1];
        for (int i = 0; i < dim; i++)
            for (int j = 0;j <= i; j++)
                matrix[i][j] = 0;
    }
    int &operator()(int i, int j)
    {
        // you need provide boundary checking
        if (i >= j)
            return matrix[i][j];
        else
            return matrix[j][i];
    }
    // …
private:
    int **matrix;
};
class LowTriMatrix: public SymmetricMatrix
{
public:
    LowTriMatrix(int n): SymmetricMatrix(n){}
// …
};
class UpperTriMatrix : public SymmetricMatrix
{
public:
    UpperTriMatrix(int n):SymmetricMatrix(n){}
// …
};
int main()
{
    FullMatrix A(2);
    A(0,0) = 5; A(0,1) = 4; A(1,0) = 3; A(1,1) = 6; A(100,100) = 10;
    SymmetricMatrix B(2);
    B(0,0) = 5; B(1,0) = 3; B(1,1) = 6; B(100,100) = 10;
    UpperTriMatrix C(2);
    C(0,0) = 5; C(0,1) = 3; C(1,1) = 6; C(100,100) = 10;
    LowTriMatrix D(2);
    D(0,0) = 5; D(1,0) = 3; D(1,1) = 6; D(100,100) = 10;
    // you should not assign A(100,100), B(100,100), C(100,100) and D(100,100)
    UpperTriMatrix E(2);
    E(0,0) = 5; E(1,0) = 3; E(1,1) = 6; // you should not assign E(1,0)
    LowTriMatrix F(2);
    F(0,0) = 5; F(0,1) = 3; F(1,1) = 6; // you should not assign F(0,1)
    Mtx *vec[6];
    vec[0] = &A; vec[1] = &B;
    vec[2] = &C; vec[3] = &D;
    vec[4] = &E; vec[5] = &F;
    for (int i = 0; i < 6; i++)
    {
        vec[i]->showMatrix(); cout << endl;
```

```
    }
    return 0;
}
```

✓ The results are

```
5 4
3 6

5 3
3 6

5 3
0 6

5 0
3 6

5 0
0 6
```