COSC1076 | Advanced Programming Techniques

Tutorial/Lab | Week 11

Overview

The Week 11 tutorial/lab is to revise content from the last few weeks.

- Inheritance
- Polymorphism
- Operator overloading

Tutorial Questions

The tutorial questions relate to the following C++ files:

A.h

```
class A {
public:
    A();
    virtual ~A();

    virtual int foo();
    virtual int foo(int x);
};
```

B.h

```
#include "A.h"

class B : public A {
public:
    B();
    virtual ~B();

    virtual int foo();
    virtual void bar();
    virtual void bar(double y);
};
```

main.cpp

```
#include "B.h"
#define EXIT_SUCCESS    0

int main (void) {
    A* a1 = new A();
    B* b1 = new B();

    a1->foo();
    a1->foo(7);

    b1->foo();
    b1->foo(10);
    b1->bar();
    b1->bar(-10);
```

```
15
16      A* a2 = b1;
17      a2->foo();
18      a2->foo(-7);
19
20      delete a1;
21      delete b1;
22      delete a2;
23
24      return EXIT_SUCCESS;
25  }
```

1. Why is the `virtual` keyword necessary?
2. For each line in `main.cpp`, precisely which method(s) get called on each class `A` and `B`. If multiple methods are called, specify the order in which they are called.
3. Show where the following concepts are used:
   - Overloading
   - Over-riding
   - Polymorphism
   - Implicit Typecasting
   - Generics

Assignment 2 Progress Update

Update your tutor on the progress of your group. You might consider such things like:

- ***What has each individual done in the past weeks?***
- What remains to be done *before* Wednesday (if relevant)
- What is your plan for the presentation?

The assignment is due WEDNESDAY!.

For Thursday and Friday labs, you will still have an update for what your group did to get the assignment completed.

Lab Questions

It is a good idea to attempt the lab questions before coming to class. The lab might also be longer than you can complete in 2 hours. It is a good to finish the lab at home.

You should demonstrate your work to your tutor.

Exercises

1. The following class represents a very simple, fixed-size Matrix of `double`'s.
   Recall that a Matrix, in the mathematical sense, is essentially a 2D array. For a brief revision, see here.

   Matrix.h

```
class Matrix {
public:
    Matrix();
    Matrix(Matrix& other);
    virtual ~Matrix();

    // Getter
    virtual double get(int row, int col) const;

    // Setter
    virtual void set(int row, int col, double value);

    // Add another matrix to this one - modifying this matrix
    virtual void add(const Matrix& other);

    // Compare this matrix against another for equality
    virtual bool operator==(const Matrix& other) const;

private:
    double mat[10][10];
};
```

   Implement the defined methods of the `Matrix`:
   (a) Constructor - initially all values should be 0.
   (b) Copy-constructor
   (c) Destructor
   (d) `get` - Get the value in the matrix at the given row/column.
   (e) `set` - Set the value in the matrix at the given row/column to the given value.
   (f) `add` - Add the given matrix (passed as a reference) to the current matrix. This modifies the current matrix.
   (g) Comparison operator, `operator==`

Lab questions continue on the next page

2. The following class extends the `Matrix` class that was implemented above. It defines an immutable matrix. It adds one method that when called makes the matrix immutable - that is, the values can no longer be changed. Implemented this class.
   The overridden versions of the methods of `Matrix` are commented-out. You must decide which method(s) need to be overridden and re-implemented in the `ImmutableMatrix` class.

ImmutableMatrix.h

```
class ImmutableMatrix : public Matrix {
public:
    ImmutableMatrix();
    ImmutableMatrix(ImmutableMatrix& other);
    virtual ~ImmutableMatrix();

    // Once called, this matrix may no longer be modified
    virtual void makeImmutable();

    //virtual double get(int row, int col) const;
    //virtual void set(int row, int col, double value);
    //virtual void add(const Matrix& other);
    //virtual bool operator==(const Matrix& other) const;
};
```

3. In a new separate file/project, modify the `Matrix` class to make it generic, using the header file set-up below.
   What assumption(s), if any, do you make about the generic type `T` in order for your implementation to work?

MatrixGeneric.h

```
template<typename T>
class Matrix {
public:
    Matrix();
    Matrix(Matrix<T>& other);
    virtual ~Matrix();

    // Getter
    virtual T get(int row, int col) const;

    // Setter
    virtual void set(int row, int col, T value);

    // Add another matrix to this one - modifying this matrix
    virtual void add(const Matrix<T>& other);

    // Compare this matrix against another for equality
    virtual bool operator==(const Matrix<T>& other) const;

private:
    T mat[10][10];
};
```

4. (Extension) Also make the `ImmutableMatrix` class immutable.