

CSE687 Midterm #2

Name: _____ **Instructor's Solution** _____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All examinations will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be easiest.

1. Write all the code for a Tracer class that behaves like a string and provides additional functions `log(const std::string& msg)` and `add(const std::string& msg)` where `log` builds a named, time-stamped¹ log string that contains `msg`, and the function `add` concatenates its message to the existing log string. Show how you will display the log with only one line of your code. Are there any issues with the use of your class?

Answer:

Issue: `std::string` doesn't have a virtual destructor.

```
class Tracer : public std::string {
public:
    Tracer(const std::string& name) : _name(name) {}
    void log(const std::string& msg);
    void add(const std::string& msg);
private:
    std::string _name;
};

std::string getTime()
{
    using namespace std::chrono;
    std::time_t tt = system_clock::to_time_t (system_clock::now());
    return ctime(&tt);
}

void Tracer::log(const std::string& msg)
{
    *this += _name + " : " + std::string(getTime()) + msg + "\n";
}

void Tracer::add(const std::string& msg)
{
    *this += msg + "\n";
}
```

Display: `std::cout << trace << "\n\n";`

¹ You may assume that a function, `std::string getTime()`, is available for use.

2. What coding errors adversely affect Liskov substitution.

Answer:

- a. Overloading across base and derived class scopes
- b. Overloading virtual base functions that are redefined in derived class
- c. Using different default parameters in virtual functions in base and derived
- d. Failing to declare base destructor as virtual
- e. Overriding non-virtual base functions in derived class
- f. Derived class should not make pre-conditions stronger or post-conditions weaker than those of the base class

3. Given the following code:

```
class X {
public:
    ...
private:
    std::ostream& out;
    std::vector<std::string> stuff;
};
```

Write an assignment operator for the class X, assuming that out and stuff are its only data members. Are there any issues with the behavior of your assignment?

Answer:

The `std::ostream&` member can't be reset, so the target of the assignment continues to use its original stream, even if the source's stream is different. That may or may not be appropriate for a given application. Problem if ostream goes out of scope before X instances are destroyed.

```
class X
{
public:
    X(const std::vector<std::string>& strs, std::ostream& out=std::cout)
        : stuff(strs), out_(out) {}
    // this is what compiler generated ctor does
    X(const X& x) : stuff(x.stuff), out_(x.out_) {}
    // compiler can't generate due to reference member
    X& operator=(const X& x)
    {
        if(this == &x) return *this;
        // can't assign reference member out_
        stuff = x.stuff;
        return *this;
    }
    void write()
    {
        for(auto character : stuff)
        {
            out_ << "\n    " << character;
        }
    }
private:
    std::ostream& out_;
    std::vector<std::string> stuff;
};
```

4. Does the Open/Closed Principle apply to any parts of your implementation of Project #2²? If so, discuss why it applies. If not, why not?

Answer:

The OCP requires that entities are not modified, but extended through Liskov substitution or template parameterization.

So yes. Many parts of the type analyzer and data structures are open for extension and do not need to be modified to meet requirements of this project. Specifically:

- a. Parser is extended by adding new rules derived from IRule.
- b. Rules are extended by adding new actions derived from IAction.
- c. Parser supports extension through the ITokenCollection interface which allows token collections to be created from any class deriving from that.
- d. Graph is extended by supplying types for template arguments for the Vertex type and Edge type.
- e. SymbolTable is open for extension by providing types for the key and value template parameters.

² The Project #2 statement is attached at the end of this exam package.

5. Write all the code for a function that formats several `std::vector<std::string>`s into a table with headings and vertical columns with fixed field widths where the widths may vary from column to column. Your function should be able to format its input without prior knowledge of the number of columns of data and their widths. Please specify column widths as part of the function's input. Your function may be a member of some class if you wish.

Answer:

```
typedef std::vector<int> Schema;           // column width
typedef std::vector<std::string> Row;     // row of data
typedef std::vector<Row> Table;           // table of rows

class Display {
public:
    Display(std::ostream& out) : _out(out)
    {
        out.setf(std::ios::right, std::ios::adjustfield);
    }

    void ShowRow(const Schema& schema, const Row& row)
    {
        for(size_t colCount=0; colCount<row.size(); ++colCount)
            ShowCell(schema[colCount], row[colCount]);
        _out << "\n ";
    }

    void ShowCell(int width, const std::string& data)
    {
        _out << std::setw(width) << data;
    }

    void ShowTable(const Schema& schema, const Table& table)
    {
        _out << "\n ";
        ShowRow(schema, table[0]);
        ShowBorder(schema); // not shown here
        for(size_t rowCount=1; rowCount<table.size(); ++rowCount)
        {
            ShowRow(schema, table[rowCount]);
        }
        _out << "\n";
    }
private:
    std::ostream& _out;
};
```

6. Given the class declaration:

```
template<typename U, typename V>
class X { ... };
```

Show how you add a template member function for the parameter W;

Answer:

```
template<typename U, typename V>
class X
{
public:
    // some class member functions
    template<typename W>
    void fun(W w);
};

template<typename U, typename V>
template<typename W>
void X<U,V>::fun(W w)
{
    std::cout << "\n " << typeid(w).name() << ": value is " << w;
}
```

7. Draw a class diagram for the type analysis in your design of Project #2. You don't need to show members of the classes.

Answer:

The most important rule and action are shown below. Others have been elided for clarity.

