

CSE687 Midterm #4

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 functions to convert an integer to a string containing it's hexadecimal representation and to convert the hexadecimal representation back to an integer.

Answer:

Production code should check stream state and throw an invalid argument exception if !good().

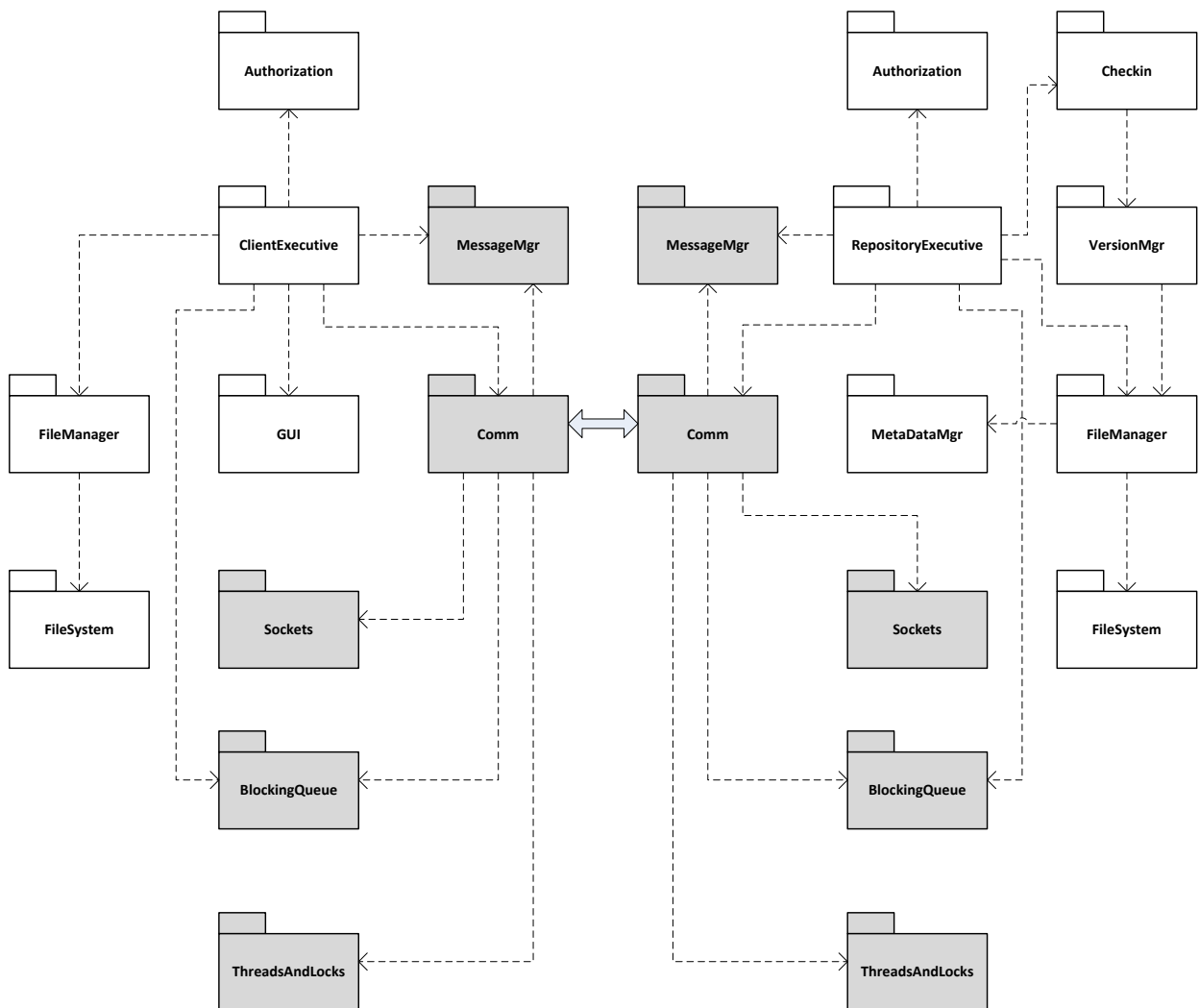
```
std::string ToHex(long int src)
{
    std::ostringstream conv;
    conv.setf(std::ios::showbase);
    conv << std::hex << src;
    return conv.str();
}

long int FromHex(const std::string& src)
{
    std::istringstream in(src);
    long int dest;
    in.setf(std::ios::hex, std::ios::basefield);
    in >> dest;
    return dest;
}
```

2. Draw a package diagram for your initial design of Project #4¹.

Answer:

Gray packages are from Project #3. MetaDataManager needs XmlReader and XmlWriter. Didn't have space for those. Client provides Authorization credentials, Repository compares them to stored values.



¹ The Project #4 statement is attached to the back of this exam package.

3. Given the following code:

```
class B { ... private: C c; };  
class D : public B { ... private: A* pA; };
```

Write a copy constructor for the class D. You may assume that code, not disclosed to you, ensures that either $pA = 0$ or it points to a valid instance of A on the heap and that c and pA are the only data members of these classes. Please state any assumptions you need to make in order to write this code.

Answer:

No assumptions are necessary other than the proviso that pA is either zero or points to a valid object on the C++ heap. If your design ensures that pA is always initialized, by doing that in all constructors and deleting only in the destructor, then you can initialize pA with an initializer, as shown in the code example for this question.

```
D(const D& d) : B(d)  
{  
    if(!d.pA)  
        pA = 0;  
    else  
        pA = new A(*d.pA);  
}
```

4. Write all the code for a Tree class where each node may contain an arbitrary finite number of child nodes. Each node should hold an instance of some unspecified type T and refers to its children through pointers². To support this code you will need to provide the declaration of a Node class that provides operations needed by Tree.

Answer:

```
template <typename T>
class Node {
public:
    typedef std::vector<Node<T>*> Children;
    Node(const T& t) : _t(t) {}
    void add(Node<T>* pChild) { _children.push_back(pChild); }
    Children& children() { return _children; }
    T& value() { return _t; }
private:
    Children _children;
    T _t;
};

template <typename T> // this tree doesn't own its nodes
class Tree {
public:
    typedef void (*NodeProc)(Node<T>* pNode);
    Tree() {}
    void makeRoot(Node<T>* pRoot) { _nodes.push_back(pRoot); }
    bool addNode(Node<T>* pParent, Node<T>* pChild);
    void dfs(Node<T>* node, NodeProc proc);
private:
    Tree(const Tree<T>&);
    Tree<T>& operator=(const Tree<T>&);
    std::vector<Node<T>*> _nodes;
};

template<typename T>
bool Tree<T>::addNode(Node<T>* pParent, Node<T>* pChild)
{
    std::vector<Node<T>*>::iterator iter
        = std::find(_nodes.begin(), _nodes.end(), pParent);
    if(iter == _nodes.end())
    {
        std::cout << " can't find parent";
        return false;
    }
    (*iter)->add(pChild);
    _nodes.push_back(pChild);
    return true;
}
```

² Note that there is no notion of balancing associated with this Tree class.

5. What can you say about virtual function pointer tables defined by the code in Question 3? What role do they play in Object Oriented Designs using these classes?

Answer:

If B is a good base class citizen it will provide a virtual destructor. If B defines any virtual functions both B and D will have virtual function pointer tables. B's table will have one function pointer bound to each of its virtual functions.

D's table will have one function pointer for each of B's virtual functions to which will be added function pointers to each new virtual function defined in D, not found in B. D's Vtbl will have pointers bound to each virtual function it defines. If it does not override one or more of B's virtual functions, then D's Vtbl will have pointers bound to those B function definitions.

Virtual function pointer tables are the core implementation for dynamic binding and are therefore the basis for Liskov Substitution and Polymorphism.

See MT3Q5 for further discussion.

6. Write all the code for a class that stores and displays file dependency information without using a Graph class.

Answer:

```
typedef std::string FileName;
typedef std::list<std::string> DepList;
typedef std::unordered_map<FileName, DepList> Dependencies;

class FileDependencies {
public:
    void Add(const FileName& src, const FileName& dst);
    void Show(std::ostream& out);
private:
    Dependencies deps;
};

void FileDependencies::Add(const FileName& src, const FileName& dst)
{
    Dependencies::iterator iter = deps.find(src);
    if(iter == deps.end())
    {
        DepList newDeps;
        newDeps.push_back(dst);
        deps[src] = newDeps;
    }
    else
    {
        iter->second.push_back(dst);
    }
}

void FileDependencies::Show(std::ostream& out)
{
    for(auto pair : deps)
    {
        std::cout << "\n " << pair.first << "\n ";
        for(auto file : pair.second)
            std::cout << file << " ";
        std::cout << std::endl;
    }
}
```

7. What is the order of complexity³ of depth first search for the Tree class in Question 4. Explain how you would write an efficient function to find all vertices in an instance of Tree that have no children. You don't need to write code to answer this question.

Answer:

DFS has order $O(N)$ where N is the number of nodes in the tree, since there is only one parent edge for each node and each node is visited exactly once.

To find all the nodes that have no children, e.g., leaves of the tree, you need to create a functor that holds a `std::vector<Node>` to hold the nodes to return. Then pass the functor by reference to DFS and store each node encountered that has no children. When DFS returns from the initial call the functor holds a collection of all nodes with no children. Note that this works whether your tree holds a vector of nodes or just the root of a linked tree.

³ Order of complexity describes how the computation time of this operation grows with the size of the tree.