CS 3450 – Design Patterns
Command (Chapter 6)
Due: See Syllabus

Write a program that processes batch requests for updating simple databases. (Imagine that commands have been saved during the day for an overnight run that actually updates the databases.) The databases can be simulated using a *map* data structure, using strings for both keys and values. Here is the Database class interface in C++:

```
class Database {
    map<string, string> data;
    etc.
public:
    Database(const string& id);
    string getID() const;
    void add(const string& key, const string& value);
    string get(const string& key) const;
    void update(const string& key, const string& value);
    void remove(const string& key);
    void display(ostream& dest) const;  // Writes key|value lines
};
```

(Note: the above interface may be modified if necessary. And you don't have to use a map. It's just easiest.)

Each database has a string **id** that identifies it. You will read a file of commands to update the database. The file has the following layout grammar:

```
<command> <database id> <key> [<value>] | B | E
```

Here is a sample file:

```
A one key1 value1
B
A one key2 value two
A one key3 value3
E
A two key4 value four
R one key1
U two key4 value4
```

Keys contain no spaces but values can.

The command are A (for add), U (for update), and R (for remove). The delimiters B and E begin and end a transaction, respectively, which means their intervening contents form a *macro command*. Macro commands should *nest* – i.e., macros can hold other macros. Keep a list of databases and create a database the first time you encounter a command that references its **id**. Do not allow an Add command to modify a record whose key already exists. Update should do nothing if the key in its request does not exist. Make all of your commands undoable, including macro commands.

The name of the command file should be a command-line argument.

After you have read sample commands from a file into a sequence of command objects, process them (i.e., call their execute() methods) and then print all databases (by calling **display** with a file stream or standard output as argument). Then *undo* all commands in reverse order, displaying snapshots of the affected database after each undo. Finally, verify that your databases are empty by displaying them once more. If you are using C++, watch for memory leaks.

If you are using Java, feel free to just use the **toString** method for your map container – don't fret about the formatting below.

Macros execute invisibly, so print a note when a macro begins execution, and another note when it ends execution. Likewise, print a note when undoing a command.

The sample output for the command file above is:

```
Beginning a Macro
Ending a Macro

Contents of Databases:
Database one:
key2| value two
key3| value3

Database two:
key4| value4

Undid UpdateCommand
key4| value four

Undid RemoveCommand
key1| value1
key2| value two
key3| value3

Undid AddCommand

Begin Undoing Macro

Undid AddCommand
key1| value1
key2| value two

Undid AddCommand
key1| value1

End Undoing Macro
```

```
Undid AddCommand

Contents of Databases:
Database one:

Database two:
```

Please submit a UML class diagram along with your source code and executable.

Note: You need to handle these error conditions. Report them, and don't do anything.

- If you try to add something that is already there.
- If you try to remove or change something that isn't there.
- If you try to undo a command that didn't work.