CS 3450 – Design Patterns
(Proxy)

Implement the Database class as described in lesson 6 to just read databases from files. Here's the C++ version of the interface it should implement:

```
class IDatabase {
public:
    virtual string getID()= 0;
    virtual bool exists(const string& key) = 0;
    virtual string get(const string& key) = 0;
};
```

This time, however, your concrete database class will actually retrieve values from a file. The format will be as before:

<key> <value>

on each line. Keys do not contain spaces but values can. Whenever a user calls **get(key)**, the value for that key is returned (throw an exception if it's not in the file). The database **id** is the file name. Use it to open the file, and keep the file open for the life of the object.

Each time you try to access the data via get() or exists(), go to the file (don't read in all the data at first and store it in the object).

Since retrieving values is now expensive, write a proxy named CacheDB that wraps an `IDatabase*` and keeps a most-recently-used list of up to **SIZE** elements (use 5). When users call **get( ) or exists()** on such a database, retrieve from the cache if it's previously been fetched and is still in the cache. Otherwise retrieve it from the file and add it to the cache. Don't let the cache exceed **SIZE** entries.

Note that when you retrieve anything from the cache, you must update the most-recently-used order.

Write a method of CacheDB called **inspect(),** which returns the keys in the cache in order, so we can see what is in the cache at any time. Call it after using the cache. Note that this is not part of a typical proxy, but is only there to show that you implemented the cache correctly. The inspect() method will not show up in the IDatabase interface or any other proxy (see below.)

Finally, write a SecureDB proxy that automatically requires users to login before allowing them to call the IDatabase methods. Have a special Database of users and passwords that authenticates on the first access attempt. Throw an exception if authentication fails.
Note that both get() and exists() are accesses and must be secured.

You should be able to mix and match the proxies at will.

Here is a sample driver in C++:

```
void test(IDatabase* db) {
    try {
        cout << db->get("one") << endl;
        cout << db->get("two") << endl;
        cout << db->exists("two") << endl;           // Call to exists
        cout << db->get("three") << endl;
        cout << db->get("four") << endl;
        cout << db->get("four") << endl;
        cout << db->get("five") << endl;
        cout << db->get("six") << endl;
        cout << db->get("one") << endl;
        cout << db->get("seven") << endl;
    }
    catch (runtime_error& x) {
        cout << x.what() << endl;
    }
```

```
    cout << endl;
}

int main() {
    Database db("db.dat");                      // Hard code this exact name
    test(&db);

    Database userdb("userdb.dat");              // Hard code this exact name
    SecureDB sdb(&db, &userdb);
    test(&sdb);

    CacheDB cdb(&sdb);
    test(&cdb);
    cout << "Cache contents: " << cdb.inspect() << endl;

    try {
        Database db2("noname.dat");
    }
    catch (runtime_error& x) {
        cout << x.what() << endl;
    }
}
```

Here's the input file, `db.dat`:

```
one the first value
two the second value
three the third value
four the fourth value
five the fifth value
six the sixth value
```

Here's the expected output:

```
the first value
the second value
true                                // or 1 (depending how "true" is printed)
the third value
the fourth value
the fourth value
the fifth value
the sixth value
the first value
No such record: seven

Enter username: chuck
Enter password: foo
the first value
the second value
true
the third value
the fourth value
the fourth value
the fifth value
the sixth value
the first value
No such record: seven

the first value
the second value
found key "two" in cache
true
the third value
the fourth value
found key "four" in cache
the fourth value
the fifth value
the sixth value
the first value
```

```
No such record: seven

Cache contents: [one], [six], [five], [four], [three],

noname.dat does not exist
```

Note the trace statement when accessing the cache, and the exceptions for non-existent records and files.

Note: this program is quite easy, but there are several important details that people miss if they are sloppy. Read the program specification carefully to make sure you get the details right.

Implementation notes:

1. Hard-code the following input files: “db.dat” for the database contents, and “userdb.dat” for the database of usernames and passwords. Do NOT use other names.
2. The SecureDB uses lazy authentication: It does NOT authenticate until the first request to get() or exists().
3. In CacheDB, note the way exists() works:
    a. First, it checks the cache, (and reorders the cache if it is found).
    b. If it isn’t found in the cache, then go to the database itself. However, since a call to the database exists() function doesn’t return the value, you will not be updating the cache.
4. Note that if you call get() and the key isn’t in the database, do NOT return a value like “NULL” or even “”.