Timothy Chu

George Ferguson

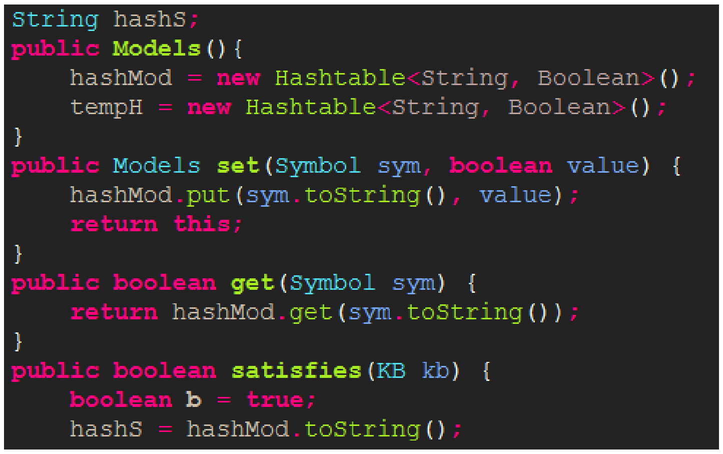
CSC 242

16 March 2016

Project 2 Write-up

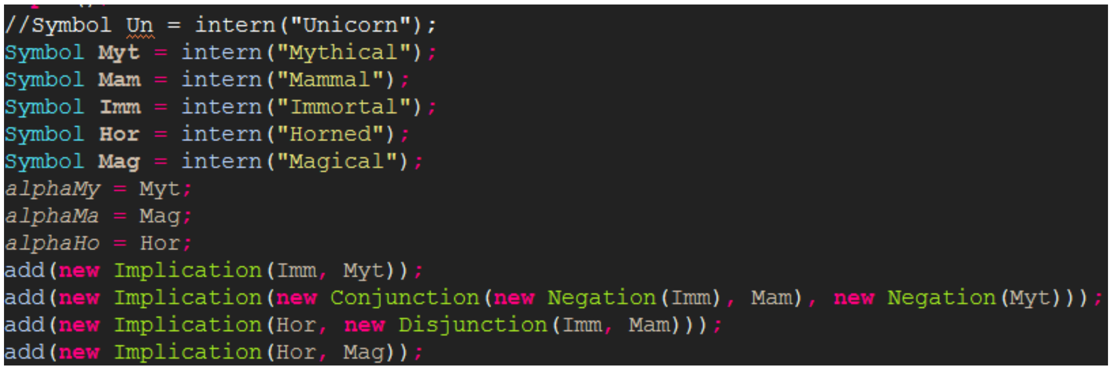
Note: I wasn't quite sure how to do a write-up for a computer science project, so I drew a lot of inspiration from this website [*http://www.cs.ucl.ac.uk/teaching_learning/msc_cgvi/projects/project_report_structure/*](http://www.cs.ucl.ac.uk/teaching_learning/msc_cgvi/projects/project_report_structure/)

Introduction/Background



```
String hashS;
public Models(){
    hashMod = new Hashtable<String, Boolean>();
    tempH = new Hashtable<String, Boolean>();
}
public Models set(Symbol sym, boolean value) {
    hashMod.put(sym.toString(), value);
    return this;
}
public boolean get(Symbol sym) {
    return hashMod.get(sym.toString());
}
public boolean satisfies(KB kb) {
    boolean b = true;
    hashS = hashMod.toString();
```

So this is a project that resolves propositional logic problems using Truth Table Enumeration and WalkSAT. I struggled a lot more with this project than I had initially anticipated. I had no clue how to implement the propositional logic statements. At first, I just wrote some code that would properly handle the Modus Ponens example case, however, the same code didn't work for the first statement, "P". I didn't quite understand how to use symbols as their own "sentence" so to speak. Even after I fixed that issue, I was unable to make that code work with the Wumpus case. I tweaked my code some more, and I again hit a wall. My code wouldn't work with the Truth Table Enumeration method I wrote. After asking around a little bit, I found out that we could use the code provided by Professor Ferguson.



```
//Symbol Un = intern("Unicorn");
Symbol Myt = intern("Mythical");
Symbol Mam = intern("Mammal");
Symbol Imm = intern("Immortal");
Symbol Hor = intern("Horned");
Symbol Mag = intern("Magical");
alphaMy = Myt;
alphaMa = Mag;
alphaHo = Hor;
add(new Implication(Imm, Myt));
add(new Implication(new Conjunction(new Negation(Imm), Mam), new Negation(Myt)));
add(new Implication(Hor, new Disjunction(Imm, Mam)));
add(new Implication(Hor, Mag));
```

I went ahead and implemented Truth Table Enumeration *again* using the code provided by Ferguson. I ran into more issues there. For whatever reason, I was having issues with changing values in my hashtable. I ended up making a local copy of the hashtable as a string. It sounds ridiculous, but it functioned the way I needed it to. Besides all of that, I would say the most difficult part of the project was figuring out how to properly write the propositional logic statements in code. For example, with the Unicorn problem, I initially had a unicorn symbol and I had used in all of the propositional logic sentences.

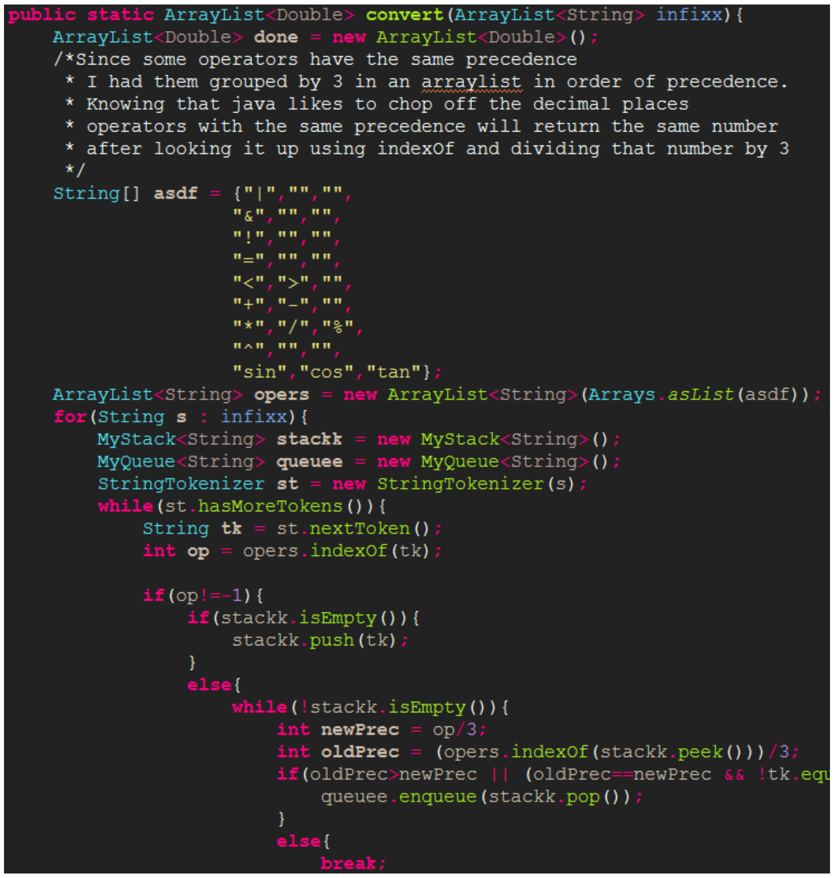
In my code, my "main" three problems I chose were Modus Ponens, Wumpus World (simple), and Horn Clauses. The two liar and truth teller problems were intended to be the additional problems mentioned in Ferguson's assignment.

Analysis and Design/Implementation

Professor Ferguson discussed propositional logic and the functions used a great deal during class. When I sat down to do this project, I knew exactly how my program *should* work, however, I was unable to get it to work (at first). I feel like implementing Truth Table Enumeration and WalkSAT was not that hard given some pseudocode and a description on how the algorithm is supposed to run. My initial implementation of logical operations and propositional logic statements did not work at all. Sure, I was able to get the sentences to print, but that did absolutely nothing for me. In order to actually process and test different models with my propositional logic sentences, I had to go back to the infix to postfix calculator that I made back in CSC 172. By using that and modifying the cases that I used, the infix to postfix calculator, it worked perfectly for this program. I feel as though there was probably a more efficient way of doing that, but for me, simply copying my old code was the most time efficient way of getting that part of the assignment done. At that point, I just need to make the algorithms to preform Truth Table Enumeration and WalkSAT.

Testing/Results

I was able to fix many issues in my code simply by running it. While I was still working on solving the plethora of issues I had, I tossed System.out.println's pretty much everywhere. Just as I stated before, I ran into an issue where adding a value to a temporary hashtable also added it to the "real" copy of the temporary hashtable. I struggled with that for a few hours, and I still wasn't able to fix it. That's why I converted the "original" hashtable before any modifications into a String. As I started working on more problems that required more types of logical operations, I had to add more functions.



```java
public static ArrayList<Double> convert(ArrayList<String> infixx){
    ArrayList<Double> done = new ArrayList<Double>();
    /*Since some operators have the same precedence
     * I had them grouped by 3 in an arraylist in order of precedence.
     * Knowing that java likes to chop off the decimal places
     * operators with the same precedence will return the same number
     * after looking it up using indexOf and dividing that number by 3
     */
    String[] asdf = {"|","","",
                     "&","","",
                     "!","","",
                     "=","","",
                     "<",">","",
                     "+","-","",
                     "*","/","%",
                     "^","","",
                     "sin","cos","tan"};
    ArrayList<String> opers = new ArrayList<String>(Arrays.asList(asdf));
    for(String s : infixx){
        MyStack<String> stackk = new MyStack<String>();
        MyQueue<String> queuee = new MyQueue<String>();
        StringTokenizer st = new StringTokenizer(s);
        while(st.hasMoreTokens()){
            String tk = st.nextToken();
            int op = opers.indexOf(tk);

            if(op!=-1){
                if(stackk.isEmpty()){
                    stackk.push(tk);
                }
                else{
                    while(!stackk.isEmpty()){
                        int newPrec = op/3;
                        int oldPrec = (opers.indexOf(stackk.peek()))/3;
                        if(oldPrec>newPrec || (oldPrec==newPrec && !tk.equ
                            queuee.enqueue(stackk.pop());
                        }
                        else{
                            break;
```

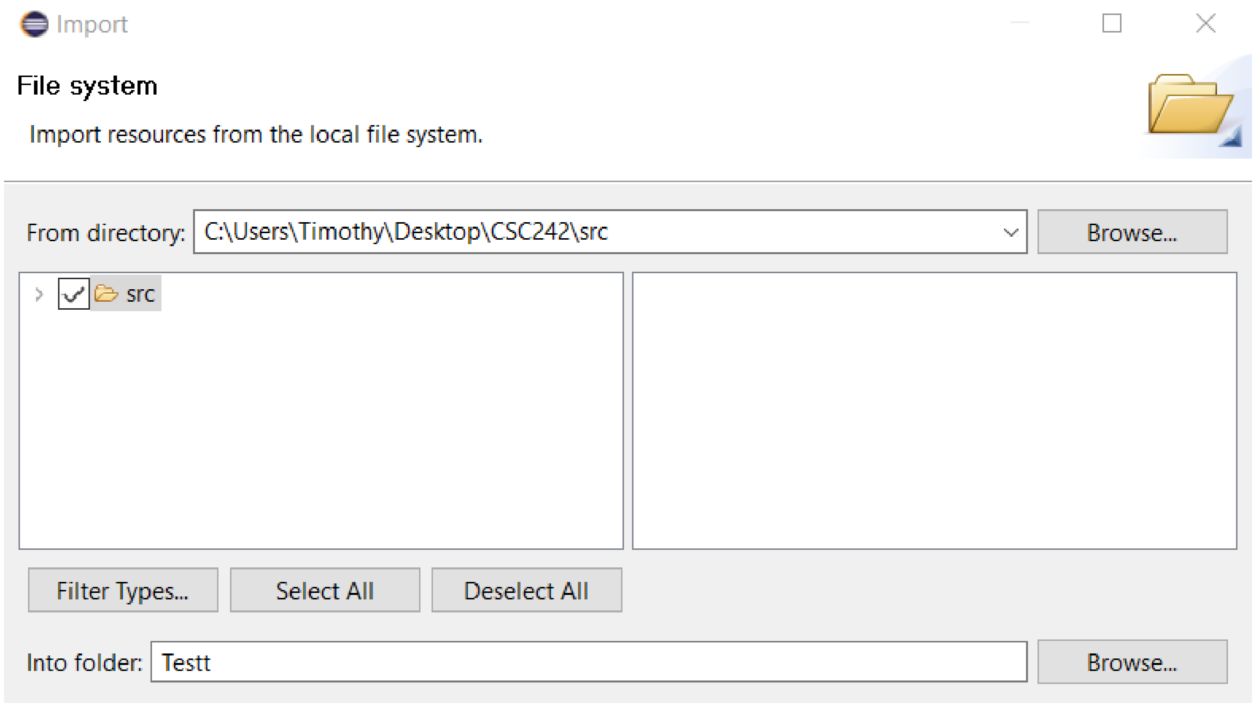
Conclusion

There is always room for improvement with *any* project. My code was very sloppy. I could have probably optimized my code much more. I should have started sooner. In addition, I felt that I would have learned the material and done the project better if I had worked with a partner. There were many times where I was asking people without partners for advice on how to do some parts of the project. Working with a partner would've helped so much more.

To compile/run

I was unable to get my code to compile though the command line. It does, however, work with eclipse.

Make a new project, left click and click on “import”. Choose the option to import a “file system” navigate to the “src” folder located inside of the .zip file and import as shown below. From there, just run any of the files inside of the “Examples” package.



Import

File system

Import resources from the local file system.

From directory: C:\Users\Timothy\Desktop\CSC242\src

✓ src

Into folder: Testt