

B552 HW1- Tips and Guidelines (Java flavored)

If you choose to code Hw 1 in Java, the following tips from Vahid should be helpful. These are based on initial questions we have received.

1- Be sure to capture the rules and the facts in an appropriate way:

Let's say you are going to capture the rule "If a patient has a very high fever, the patient has a high fever." in Java, one way to do this would be as a String, e.g.:

```
String rule1 = "If a patient has a very high fever, the patient has a high fever."
```

But there are at least three major issues with this representation:

1. Variables are not distinguished from non-variables.
2. Antecedent(s) and consequent(s) are not clearly separated.
3. Representing rules as String would be troublesome for rule application, especially for rules with multiple/nested antecedents/consequents.
4. The representation is too close to natural language, resulting in the inclusion of ambiguous terms ("has") and unnecessary parts ("a", "the"). We will be discussing this in class.

Recommended approach: Use a data structure such as "List" for storing rules and facts. In Scheme/Lisp, list manipulation is extremely handy and in fact, is part of the language syntax. But, if you are using another PL such as Java, you can use a data structure like "List" (e.g. "Linked List" or "ArrayList") or any similar data structure to implement the needed behavior. You could code the above- mentioned rule as follows (make sure to import java.util in your code):

```
ArrayList rules = new ArrayList();
ArrayList rule1 = new ArrayList();
rule1.add("1");
rule1.add(new ArrayList(Arrays.asList("fever ?x very-high")));
rule1.add(new ArrayList(Arrays.asList("fever ?x high")));
rules.add(rule1);
```

Do Not:

Do not hard code the rules. As is explicitly mentioned in the instructions, rules, facts and the code should be separate (actually, you are supposed to submit separate files for each, for more information read [Submission](#) guidelines for hw1. For convenience, you may include another copy of the rule-related code in the same file as your program, but the rules should be separate data.)

2- Where should I start?

I'd say start by implementing the *substitute* function. Make sure that you choose the right signature for your functions. If I'm coding substitute, I'd use the signature below:

```
private ArrayList substitute(ArrayList substitution, ArrayList pattern)
{
```

```

        //to be implemented
        return pattern;
    }

```

Having implemented substitute, continue by implementing other functions as it's listed in the instructions.

Q: Could I implement my own algorithm rather than following the instructions?

As long as you are implementing each and every step of the forward chaining process, that's OK, though you don't need to. You're provided with the detailed instructions to help on your design for this assignment. Personally, I'd stick to the instructions for avoiding extra complexity and uncertainty.

Do:

Do Unit test your code. After implementing each function, it would be a very good idea to test it with several inputs to make sure it's working as expected.

Q: I'm still confused how to start coding. Can you give me a concrete example?

Ok, As an example, let's implement the var? function in Java:

```

private Boolean isVariable(String input)
{
    //I implemented isVariable so that it takes input of type String,
    //though depending on your implementation you might want to
    //define input of type Object
    if (input.startsWith("?"))
        return true;
    else
        return false;
}

```

Q: But this was an easy one, what about something more challenging?

I'd say as long as you are familiar with recursive function calls and basic control flows (e.g. while loop) in any programming language, you won't have a difficult time implementing any of the suggested functions. It doesn't mean that they are extremely easy but with the detailed comments in the instructions it's fair enough. However, I'm going to provide you with a template for a possible implementation of the substitute function here:

```

private ArrayList substitute(ArrayList substitution, ArrayList pattern)
{
    //not necessarily the best implementation but will give you some hints
    Iterator<ArrayList> substituteIterator = substitution.iterator();
    while (substituteIterator.hasNext())
    {
        ArrayList singleSubstitution = substituteIterator.next();
        Iterator<ArrayList> patternIterator = pattern.iterator();
        while (patternIterator.hasNext())
        {

```

```
        /*if patternIterator.next is a list, call
        substitute(singleSubstitution, patternIterator.next)
        and replace patternIterator.next in pattern with the return value
        of substitute function
        else if patternIterator.next is a variable and
        singleSubstitution.get(0) is equal to patternIterator.next
        replace patternIterator.next with singleSubstitution.get(1) */
    }
    }
    return pattern;
}
```