CSCI-410 Python Assignment - PY-02

The primary goal of this project is to familiarize you with simple finite state machines and basic XML document syntax. At the end of this assignment, you should have a file that

1) Opens a .hack file and exports the content into an XML document.

2) Reads the XML document and produces the same file that the first assignment did.

The Hack Scanner

Scanning the .hack file and producing the .xml file will utilize a finite state machine known as a finite automaton, FA, which reads in one symbol at a time and, based upon the symbol and the present state, performs an appropriate action. The state must capture all of the relevant information about the entire history of the machine. Wee are going to use an address counter along with our state variable; however this is still a true FA because we could simply call our "state" the combined contents of the counter and the state variable. The concepts used here will be the same ones around which the Hack Assembler will be constructed a later in the course.

Given the extreme simplicity of the structure of our input file, our basic FA state can consist of little, if anything, more than knowing how many bits of the present instruction have been read in. With such an approach, it becomes easy to see how we can add some leniency to the strict input file format requirement we had in PY01. In our new scanner we will therefore allow any number of tabs or spaces to appear before or between bits of the instruction and we will allow any number of characters of any type to follow an instruction prior to the newline character. Furthermore, we will permit any number of lines before, between, or after instructions provided the first character on the line, not counting spaces and tabs, is not a '0' or a '1'.

The XML Generator

While the second process could probably be done using a finite automaton in this particular case, the processing of something like an XML file is more properly done with a finite state machine known as a pushdown automaton, or PDA. A pushdown automaton is a machine that, like a finite automaton, reads one symbol of input at a time and takes appropriate action, but in addition to the present state of the machine, it also has access to a memory stack into which it can push information or pop information. Thus, it can 'remember' information about the history of the machine that cannot be captured in any finite number of states.

Instead of implementing a classic stack data structure, we will effectively use Python's function call stack as our PDA's stack. As we identify tags we will call a function specifically designed to process that type of tag. We will continue doing this until we identify the closing tag at which point we will return. We thus have a depth-first -- and potentially recursive – processing engine. The concepts used here will form the basis of the recursive decent parser that will be the foundation of the Jack compiler later in the course.

CSCI-410 Python Assignment - PY-02

The XML Syntax

As for XML, the basic syntax, which is completely sufficient for this assignment, is very straightforward and is quite HTML-like. The most basic XML document consists of properly nested pairs of tags with the tag's "element" between them. The format is **`<tagname> element </tagname>`**.

The `<hack>` tag

This tag is the root tag, of which there will be exactly one in the file. The elements of this tag will be a sequence of instruction tags that each be one of two possible types: **`<A-Instruction>`** and **`<C-Instruction>`**.

The `<A-Instruction>` tag

This tag's element will consist of two tags, an **`<address>`** tag and a **`<constant>`** tag.

The `<C-Instruction>` tag

This tag's element will consist of four tags, an **`<address>`** tag, a **`<comp>`** tag, a **`<dest>`** tag, and a **`<jump>`** tag.

The `<constant>` tag

This tag's element consists of the decimal value represented by the parent instruction's lower fifteen bits.

The `<comp>` tag

This tag's element consists of the one byte hexadecimal representation of decimal value represented by the seven bits in the parent instruction starting with the fourth (from the left) and ending with the tenth. The format is **`0xNN`** where '**`0x`**' is the numeral **`0`** followed by the lowercase letter '**`x`**', and **`NN`** is a zero-padded two-digit hexadecimal value represented the symbols {**`0`**-**`9`**, **`A`**-**`F`**} (note no lowercase).

The `<dest>` tag

This tag's element consists of the three bits in the parent instruction starting with the eleventh and ending with the thirteenth.

The `<jump>` tag

This tag's element consists of the final three bits in the parent instruction, starting with the fourteenth and ending with the sixteenth.

CSCI-410 Python Assignment - PY-02

Example

Using the example input from PY01, namely,

```
0000000000000010
1110110000010000
0000000000000011
1110000010010000
0000000000000000
1110001100001000
```

The output should be:

```
<hack>
<A-Instruction>
<address>0</address>
<constant>2</constant>
</A-Instruction>
<C-Instruction>
<address>1</address>
<comp>0x30</comp>
<dest>010</dest>
<jump>000</jump>
</C-Instruction>
<A-Instruction>
<address>2</address>
<constant>3</constant>
</A-Instruction>
<C-Instruction>
<address>3</address>
<comp>0x02</comp>
<dest>010</dest>
<jump>000</jump>
</C-Instruction>
<A-Instruction>
<address>4</address>
<constant>0</constant>
</A-Instruction>
<C-Instruction>
<address>5</address>
<comp>0x0C</comp>
<dest>001</dest>
<jump>000</jump>
</C-Instruction>
</hack>
```

CSCI-410 Python Assignment - PY-02

I/O Filenames

As with PY01, the input file will be hardcoded as “machine.hack” and assembly output file will be “assembly.asm”. The XML output file will be hardcoded as “machine.xml”.

A few files will be made available for you to test your code against.

Submission

In a manner similar to the ECS project, place all files needed for this assignment into a directory named PY02 and zip up the entire directory into a zip file of the form:

CS410_UserID_PY_02.zip

GRADING RUBRIC – 20 pts

5 pts	**Effort**
1 pt	**Processes non-instruction text properly**
1 pt	**For each tag type that is properly generated (.hack -> .xml).**
1 pt	**For each tag type that is properly processed (.xml -> .asm).**
-1 pt	**Incorrect header comments in Python script.**
-1 pt	**Incorrect submission (filename, etc).**