Homework 8: Virtual Machine I

Requirements:

Build the Virtual Machine Translator (Part I), in Java, per the instructions and guidance covered in class.

Grading method:

As usual with programming assignments, we look for elegance, clarity, reasonable documentation, and neatness.

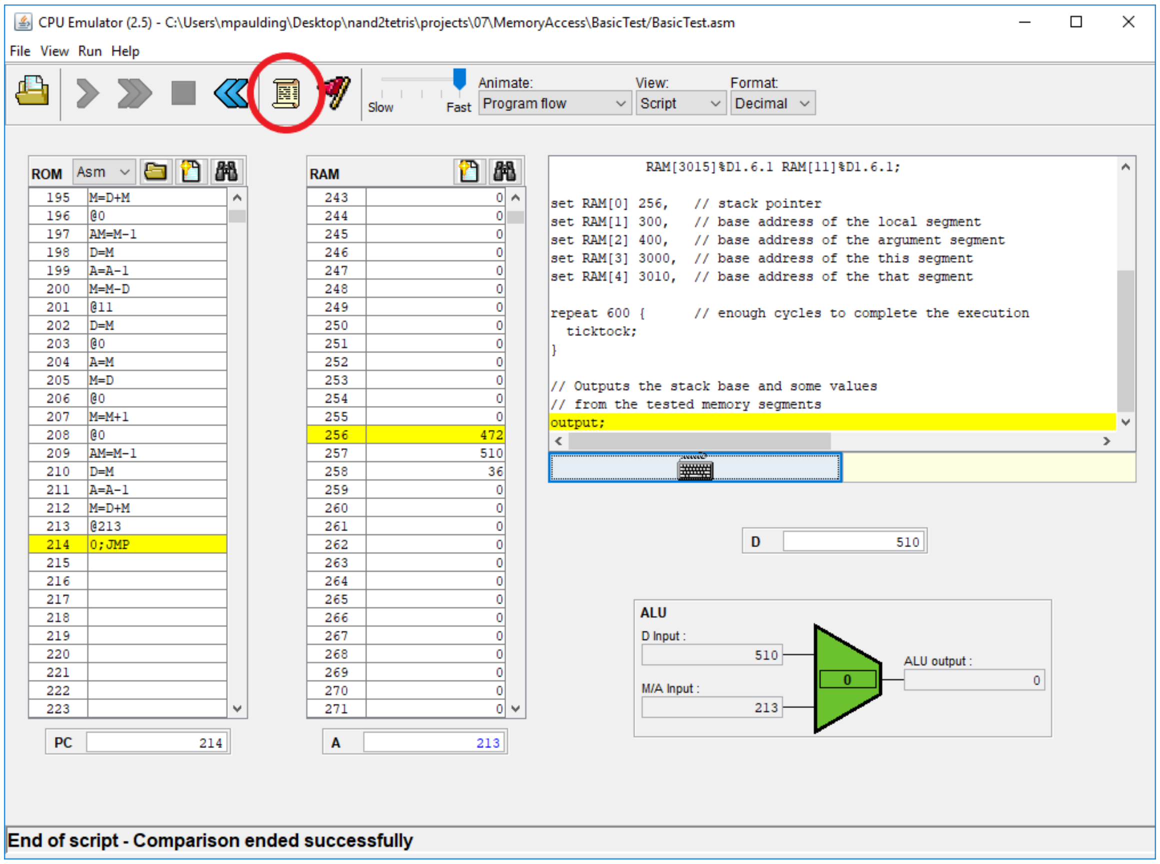
Follow the instructions in lecture as far the classes and methods to build, as well as allowing command-line arguments as instructed. Document each method (description, precondition, postcondition) and add author information on each file. Provide an algorithm for your main method that drives the Virtual Machine translation process.

What do you turn in?

Create one Word document (or PDF) with the following in order:

1. The **VMTranslator.java** source code (documented)
2. The **Parser.java** source code (documented)
3. The **CodeWriter.java** source code (documented)
4. Run your VMTranslator.java code on the 5 provided .vm files (e.g. BasicTest.vm, PointerTest.vm, StaticTest.vm, SimpleAdd.vm and StackTest.vm). This will produce 5 corresponding assembly files (e.g. the same names with file extension .asm)
5. Open the CPU Emulator and load each of the 5 test scripts (e.g. BasicTest.tst, PointerTest.tst, StaticTest.tst, SimpleAdd.tst and StackTest.tst)
6. Create a screen shot of each test (5 total) after it completes. Following is an example of a successful test after loading BasicTest.tst:

See http://nand2tetris.org/07.php for some tips/resources/tools (note that the assignment on the website may be substantially different from the assignment that is described above, if you need clarification email your instructor. You will be graded based on this documents requirements).


CPU Emulator (2.5) - C:\Users\mpaulding\Desktop\nand2tetris\projects\07\MemoryAccess\BasicTest\BasicTest.asm

Animate: Program flow View: Script Format: Decimal

ROM	Asm
195	M=D+M
196	@0
197	AM=M-1
198	D=M
199	A=A-1
200	M=M-D
201	@11
202	D=M
203	@0
204	A=M
205	M=D
206	@0
207	M=M+1
208	@0
209	AM=M-1
210	D=M
211	A=A-1
212	M=D+M
213	@213
214	0;JMP
215	
216	
217	
218	
219	
220	
221	
222	
223	

RAM	
243	0
244	0
245	0
246	0
247	0
248	0
249	0
250	0
251	0
252	0
253	0
254	0
255	0
256	472
257	510
258	36
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0
271	0

```
            RAM[3015]%D1.6.1 RAM[11]%D1.6.1;

set RAM[0] 256,   // stack pointer
set RAM[1] 300,   // base address of the local segment
set RAM[2] 400,   // base address of the argument segment
set RAM[3] 3000,  // base address of the this segment
set RAM[4] 3010,  // base address of the that segment

repeat 600 {      // enough cycles to complete the execution
  ticktock;
}

// Outputs the stack base and some values
// from the tested memory segments
output;
```

D 510

D Input: 510 M/A Input: 213 ALU output: 0

PC 214 A 213

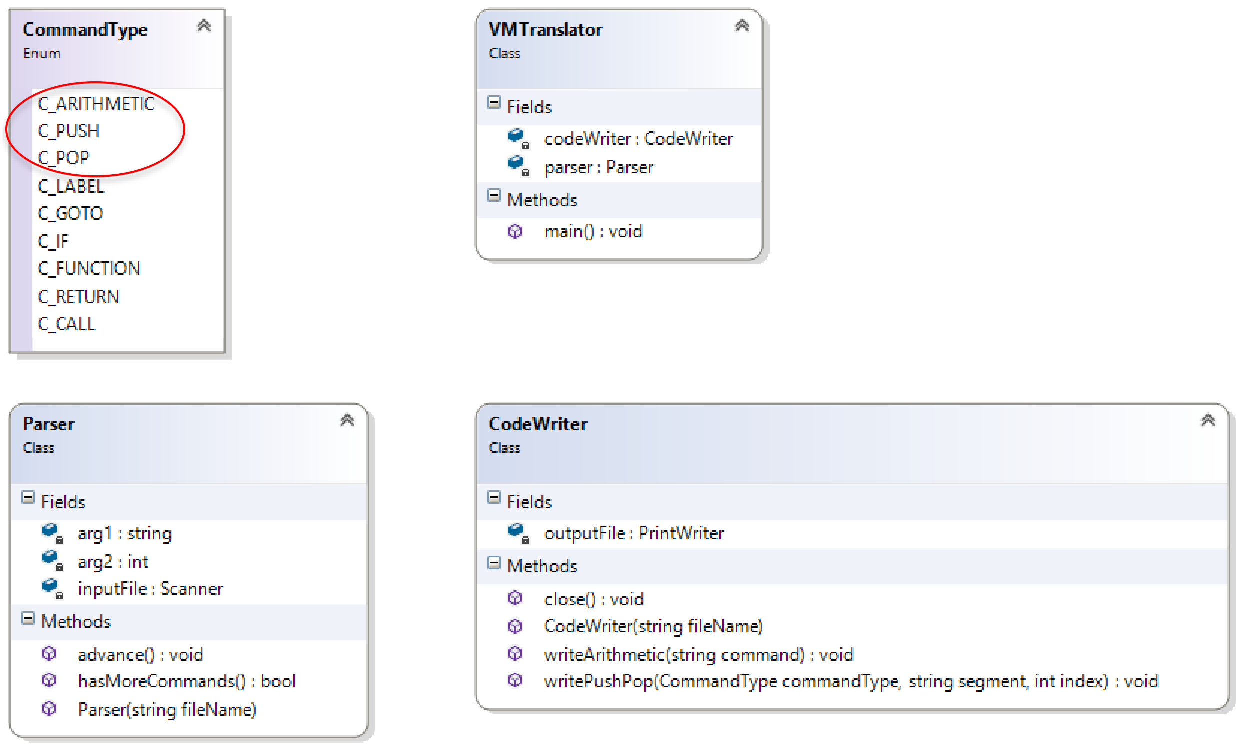
End of script - Comparison ended successfully

7. The **.java files.** The three Java files are: ***VMTranslator.java, Parser.java,*** and ***CodeWriter.java***. Please reuse the start of the main method from Assembler.java to accept command line/Scanner input to ask for the .vm file to translate.

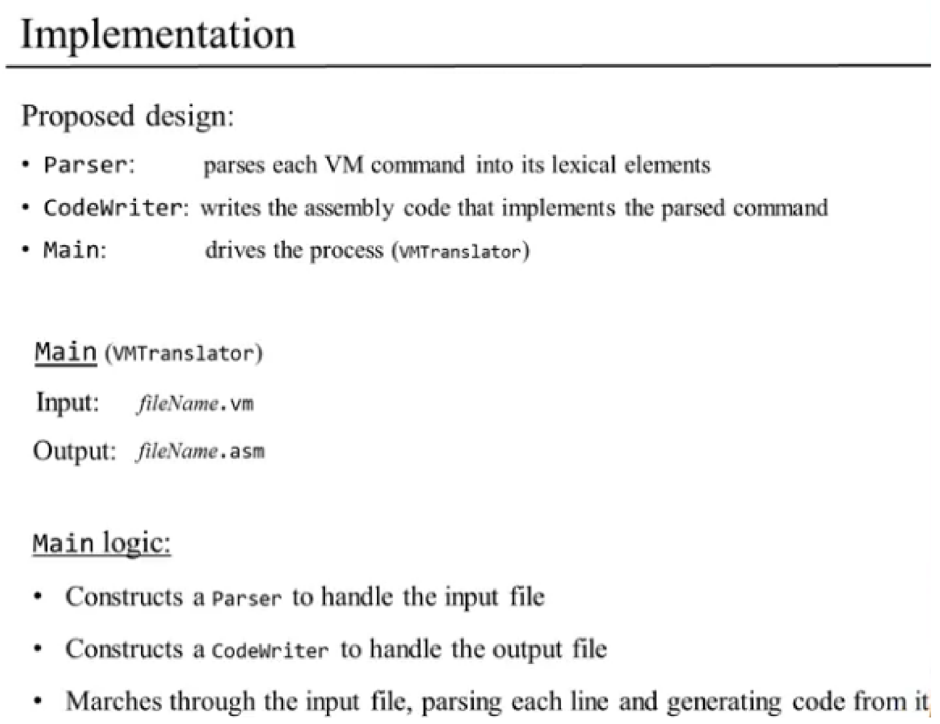
VM Part I	***Working?***
Working?	/ 60
Well built?	/ 30
Subtotal	/ 90
Documentation	/ 100

See http://nand2tetris.org/07.php for some tips/resources/tools (note that the assignment on the website may be substantially different from the assignment that is described above, if you need clarification email your instructor. You will be graded based on this documents requirements).

Below is a UML Diagram depicting the high-level details of the VM Translator software:



More details for each class:



Implementation

Proposed design:

- `Parser`: parses each VM command into its lexical elements
- `CodeWriter`: writes the assembly code that implements the parsed command
- `Main`: drives the process (`VMTranslator`)

<u>`Main`</u> (`VMTranslator`)

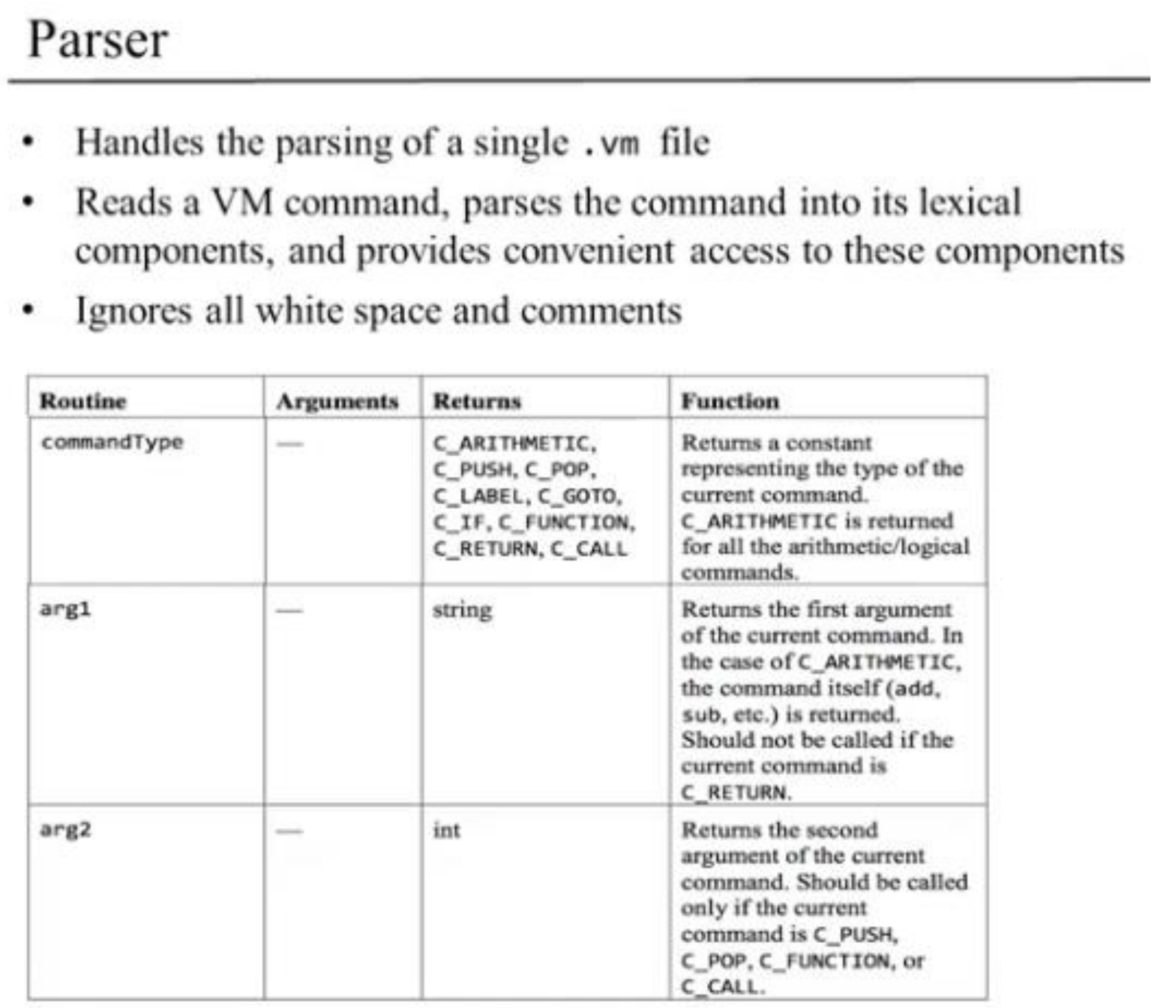
Input: *fileName*`.vm`

Output: *fileName*`.asm`

<u>Main logic:</u>

- Constructs a `Parser` to handle the input file
- Constructs a `CodeWriter` to handle the output file
- Marches through the input file, parsing each line and generating code from it

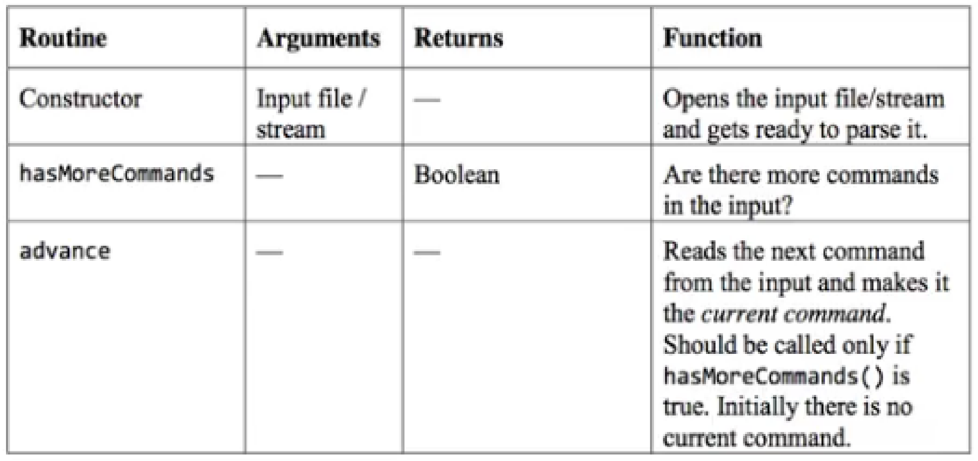
See http://nand2tetris.org/07.php for some tips/resources/tools (note that the assignment on the website may be substantially different from the assignment that is described above, if you need clarification email your instructor. You will be graded based on this documents requirements).



Parser

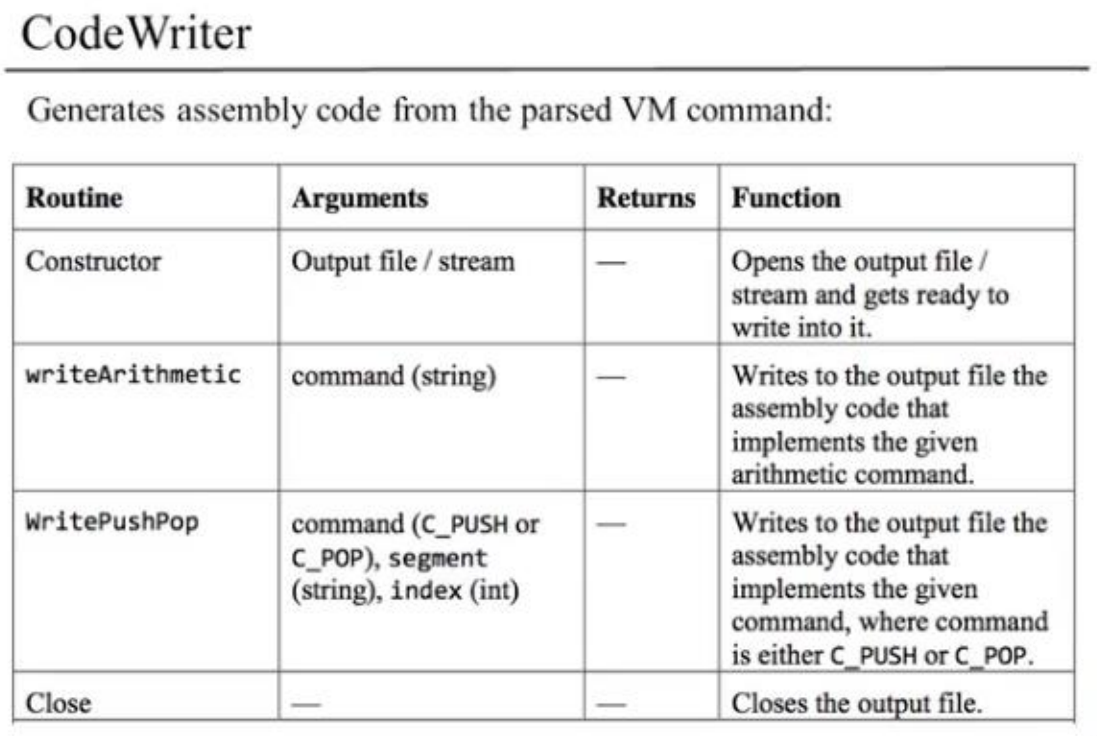
- Handles the parsing of a single `.vm` file
- Reads a VM command, parses the command into its lexical components, and provides convenient access to these components
- Ignores all white space and comments

Routine	Arguments	Returns	Function
`commandType`	—	`C_ARITHMETIC`, `C_PUSH`, `C_POP`, `C_LABEL`, `C_GOTO`, `C_IF`, `C_FUNCTION`, `C_RETURN`, `C_CALL`	Returns a constant representing the type of the current command. `C_ARITHMETIC` is returned for all the arithmetic/logical commands.
`arg1`	—	string	Returns the first argument of the current command. In the case of `C_ARITHMETIC`, the command itself (`add`, `sub`, etc.) is returned. Should not be called if the current command is `C_RETURN`.
`arg2`	—	int	Returns the second argument of the current command. Should be called only if the current command is `C_PUSH`, `C_POP`, `C_FUNCTION`, or `C_CALL`.



Routine	Arguments	Returns	Function
Constructor	Input file / stream	—	Opens the input file/stream and gets ready to parse it.
`hasMoreCommands`	—	Boolean	Are there more commands in the input?
`advance`	—	—	Reads the next command from the input and makes it the *current command*. Should be called only if `hasMoreCommands()` is true. Initially there is no current command.

See http://nand2tetris.org/07.php for some tips/resources/tools (note that the assignment on the website may be substantially different from the assignment that is described above, if you need clarification email your instructor. You will be graded based on this documents requirements).



CodeWriter

Generates assembly code from the parsed VM command:

Routine	Arguments	Returns	Function
Constructor	Output file / stream	—	Opens the output file / stream and gets ready to write into it.
`writeArithmetic`	command (string)	—	Writes to the output file the assembly code that implements the given arithmetic command.
`WritePushPop`	command (`C_PUSH` or `C_POP`), `segment` (string), `index` (int)	—	Writes to the output file the assembly code that implements the given command, where command is either `C_PUSH` or `C_POP`.
Close	—	—	Closes the output file.

See http://nand2tetris.org/07.php for some tips/resources/tools (note that the assignment on the website may be substantially different from the assignment that is described above, if you need clarification email your instructor. You will be graded based on this documents requirements).