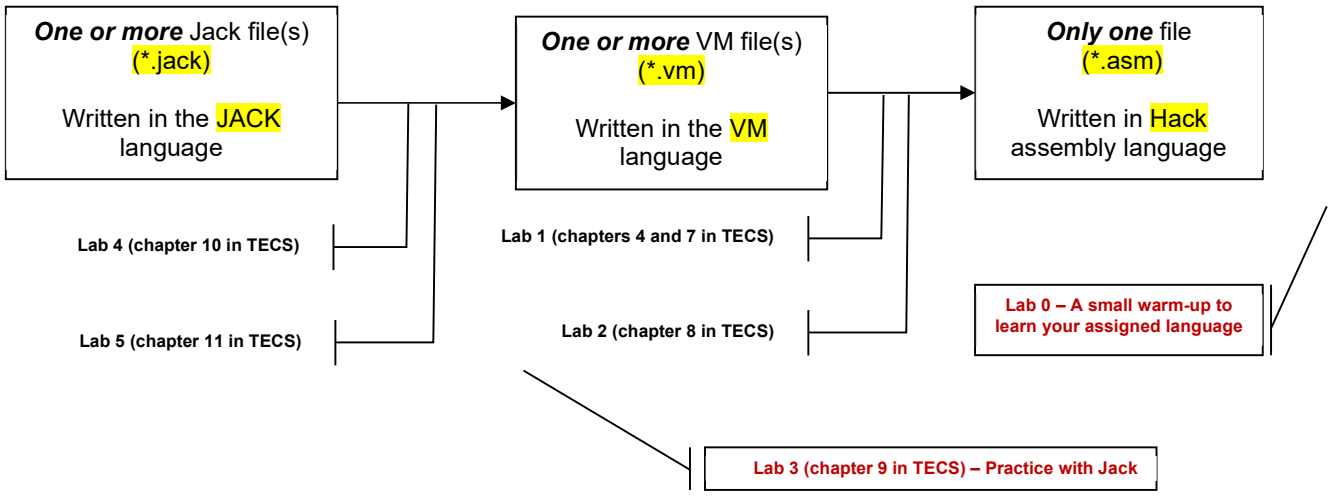
בס"ד

Principles of Programming Languages

Lab 0

Introduction to the lab:

- During the course, you will build a compiler that translates code written in a higher-level language, Jack, to the assembly language of pseudo-computer, Hack.
  - The translation is indirect, through an intermediate language, the language of a virtual machine (VM).
  - It is a 6 stage process.
- Every **pair** of students will build their version of the compiler using a different programming language.
  - The source and target languages, Jack and Hack, are the same for all, just the translation\compilation tool is different for each team.
- Each team is responsible to learn and setup their assigned language.
  - The source, intermediate and target languages (Jack, VM and Hack) will be taught as part of the course material.
- The following diagram illustrates the translation process we will follow:



Language assignment:

- Languages will be **randomly** assigned during the first lab meeting.
- Each ***pair*** of students will be assigned a language that (*hopefully*) they have not met before.

Lab 0 – Warm up

Purpose:

- Environment setup – download the tools you need to be able to program in your language
- Learn the basics of you language and it's programming environment
- Test the language's text processing capabilities.

Requirements for Lab 0

You must submit Lab 0 on time.

It will not be graded (no points will be awarded) but it is very important for you to do it in order to be sure that that language is appropriate to all what you will need to do during all the labs. .

Setup

Once you receive your language you should do the following:

1. Download and install the **compiler\interpreter** that you found for you language.
2. Download and install the **programming environment** for you language.
   a. Optimally, this should be a complete IDE (including an editor, debugger and compiler), or a plug-in into a common IDE (VS, VC, Eclipse, etc.)
   b. If no IDE (or plug-in) are available, you should find out how to integrate the code yourself (using a common editor like VIM and command-line tools for compiling and debugging)
   c. Read-up on what environment is suggested and optimal to your language. This will make your life easier later on.

Post setup

In the course site, you can find 2 **text files** whose names have the suffix **.vm** which will be the input to your program.

1. **Copy** the vm files into a directory on your computer (e.g. C:\temp\Tar0 or your home_dir\Tar0, etc.)
2. The provided files are written in VM, and you can assume that they are valid. **You don't need to check them.**
3. Every VM command (placed on separate lines) either 1 or 3 parts (you don't need to understand the commands yet – see sample at the end of this document).
    a. The first word is the command in VM
    b. The second word (if there is one) is the memory segment on which the command operates
    c. The third value (if there is one) is the offset within the memory segment
4. Write a program, using your assigned language, that
    a. Receives **from the user** the path to the folder where the vm files are stored (e.g. C:\temp\Tar0)
    b. Create an output file files whose name have the suffix **.asm** (also a **text file**).
        i. The file name should be the same as the folder that you are processing (eg. **Tar0.asm** for the above example)
    c. **Open** the file in (text) write mode.
    d. Traverse **all** .vm files in the provided folder.
        i. The number of .vm files in the folder are known in advance (they may be one file or more).
        ii. The processing order is irrelevant
    e. **For each .vm file**:
        i. **Define** and **initialize** a counter that will be used to count the number of **logical** commands in the current .vm file.
        ii. **Store** in some global variable the name of the .vm file that is currently being processed (without the .vm suffix)
        iii. **Read** the file, line-by-line; for each line decide which helper function to call based on the first word in the line (remember that the first word in each line of VM code is an operation name).

iv. You can use http://rosettacode.org/wiki/File_input/output to see how to read from files in different languages.

f. Write **helper** functions to deal with each type of command. See table below for the specifications:

i. Note, there is only **a single output** file that all helper functions will write to.

What the function should do	Parameters	Helper function	Command	
It should print the following to the end of the output file: **command: add** or **command: sub** or **command: neg**	None	handle_add handle_sub handle_neg	**add** **sub** **neg**	**Arithmatic**
It should print the following to the end of the output file: **command: eq** or **command: gt** or **command: lt** After that, it should print the current **counter** value and increment the counter. For example **counter: 3**	None	handle_eq handle_gt handle_lt	**eq** **gt** **lt**	**Logic**
It should print the following to the end of the output file: **command: push segment: <s> index: <i>** or **command: pop segment: <s> index: <i>** For example, for the input push static 2 We should see in the file **command: push segment: static index: 2**	First parameter named **segment** of type **string** Second parameter, named **index** of type **int**	handlePush handlePop	**push <s> <i>** **pop <s> <i>**	**Memory access** A command with 3 words should call a helper function to process the remaining words

g. At the end **of each input file**, close the file and print **to the screen** the message "End of input file: XXX", substituting the name of the current file for XXX (eg. "End of input file: InputB.vm")
h. At the **end of the program**, print to the screen the message "Output file is ready: YYY.asm", substituting the name of your .asm file for YYY (eg. "Output file is ready: Tar0.asm")

Sample input .vm files and the expected output.

In the current example we first processed InputA.vm and then InputB.vm, but that is coincidental.

Output file: Tar0.asm	**Input file 2: InputB.vm**	**Input file 1: InputA.vm**
command: push segment: static index: 0 command: pop segment: local index: 4 command: gt counter: 1 command: push segment: constant index: 7 command: gt counter: 2 command: pop segment: this index: 2 command: push segment: constant index: 3 command: add command: push segment: argument index: 1 command: eq counter: 3 command: pop segment: that index: 2 command: push segment: constant index: 3 command: lt counter: 1 command: pop segment: pointer index: 1 command: push segment: constant index: 43 command: push segment: local 4 command: neg command: sub command: eq counter: 2	pop that 2 push constant 3 lt pop pointer 1 push constant 43 push local 4 neg sub	push static 0 pop local 4 gt push constant 7 gt pop this 2 push constant 3 add push argument 1 eq
Output to screen:		
End of input file: InputA.vm End of input file: InputB.vm Output file is ready: Tar0.asm		

Good luck