Parser

Routines

- Constructor / initializer: Creates a `Parser` and opens the input (source VM code) file
- Getting the current instruction:

  `hasMoreLines():` Checks if there is more work to do (boolean)

  `advance():` Gets the next instruction and makes it the *current instruction* (string)
- Parsing the *current instruction*:

  `commandType():` Returns the type of the current command (string constant):

  `C_ARITHMETIC` if the current command is an arithmetic-logical command;

  `C_PUSH`, `C-POP`, `C_LABEL`, `C_GOTO`, `C_IF`, `C_FUNCTION`, `C_RETURN`, `C_CALL`

  if the current command is any of these command types

  `arg1():` Returns the first argument of the current command;
  In the case of `C_ARITHMETIC`, the command itself is returned (string)

  `arg2():` Returns the second argument of the current command (int);
  Called only if the current command is `C_PUSH`, `C-POP`, `C_FUNCTION`, or `C_CALL`

Examples:

current command	
`add, neg, eq, …`	`commandType()` returns `C_ARITHMETIC`; `arg1()` returns `"add"`, `"neg"`, `"eq"`,…
`push local 3`	`commandType()` returns `C_PUSH`; `arg1()` returns `"local"`; `arg2()` returns `3`
`call foo 17`	`commandType()` returns `C_CALL`; `arg1()` returns `"foo"`; `arg2()` returns `17`

Parser API (detailed)

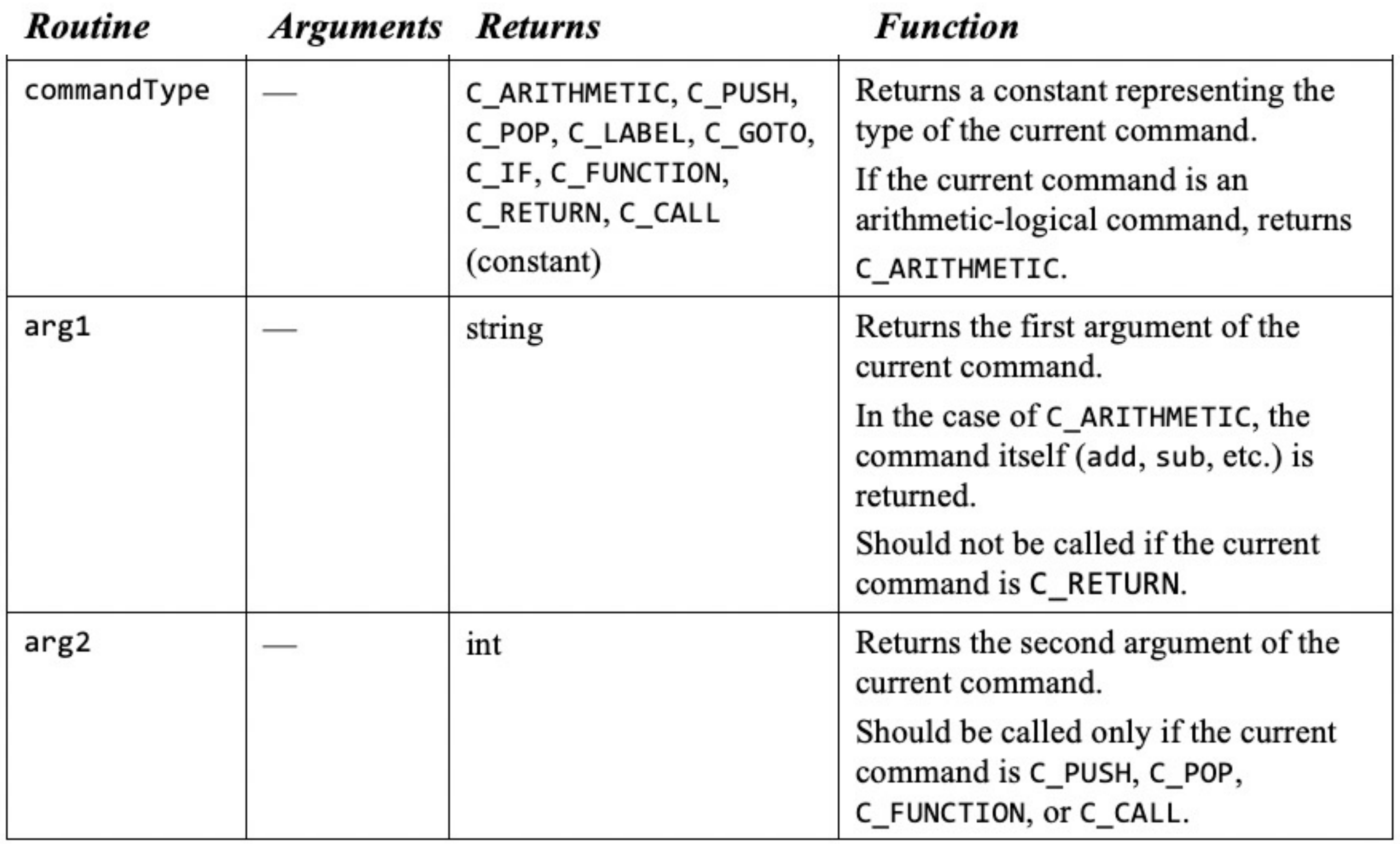
- Handles the parsing of a single `.vm` file
- Reads a VM command, parses the command into its lexical components, and provides convenient access to these components
- Ignores white space and comments

Routine	***Arguments***	***Returns***	***Function***
constructor	input file / stream	—	Opens the input file/stream, and gets ready to parse it.
`hasMoreLines`	—	boolean	Are there more lines in the input?
`advance`	—	—	Reads the next command from the input and makes it the *current command*. This method should be called only if `hasMoreLines` is true. Initially there is no current command.

(continues in the next slide)

Parser API (detailed)

- 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 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` (constant)	Returns a constant representing the type of the current command. If the current command is an arithmetic-logical command, returns `C_ARITHMETIC`.
`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`.

CodeWriter API

Generates assembly code from the parsed VM command

Routine	***Arguments***	***Returns***	***Function***
constructor	output file / stream	—	Opens an 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-logical command.
`WritePushPop`	command (`C_PUSH` or `C_POP`), segment (string), `index` (int)	—	Writes to the output file the assembly code that implements the given push or pop command.
`close`	—	—	Closes the output file.

Notes

- The components/fields of each VM command are supplied by the Parser routines;
- Before committing it to code, write and debug *on paper* the assembly code that each VM command should generate;
- More routines will be added to this module in Project 8, for handling all the commands of the VM language.