1 Introduction

Figure 1 is the overall structure of program execution. After assignment 1 which was supposed to do "Lexical analysis" (see Figure 1); assignment 2 modeled "Syntax analysis" by generating a parse tree ("abstract syntax tree" in the figure). Assignment 3 was the next step *i.e.* "sematntic analysis" and returned an AST. Assignment 6 would deal with "Code generation". Here we peek ahead a look at program execution.

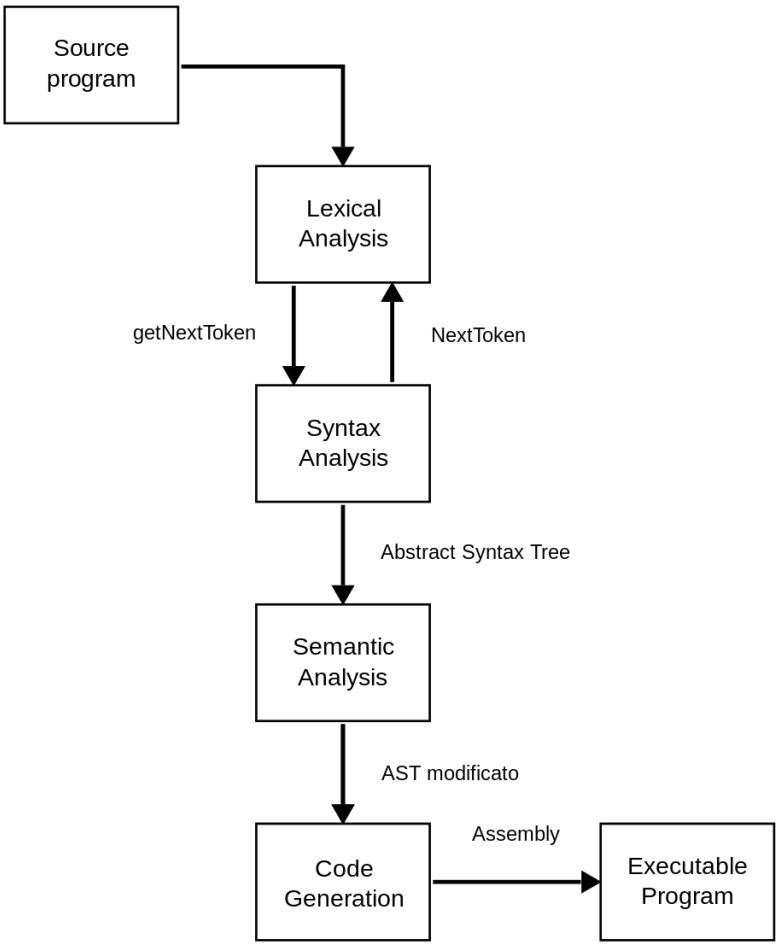

Figure 1: A multipass compiler

2 Machine

Programs usually run on a Macintosh. But that is not necessary. Theoretically, programs require a machine which has an internal state and an interface (mostly in the form of 'instructions') that can change the state. Each program must be translated into such a sequence of 'instructions'. Indeed, these form the 'code' that will be generated in assignment 6. Our machine has many components as described below:

- A random access memory (RAM) for storing 'instructions'

- A index pointing to the current instruction (program-counter)
- A symbol table
- Input and output streams.

The RAM can be seen as an array $\mathbf{R}$ and the program counter as an index pc. The machine repeatedly executes the instruction at $\mathbf{R}[pc]$ and changes its state (which includes pc). It halts upon encountering the END_OF_CODE instruction.

3 Transitions

This section describes the operational semantics of the machine in terms of the state transitions on every instruction. All instructions are given 4-tuples though come fields might have junk (denoted by _). Γ is the symbol table (implemented as a stack) where the activation records are demarcated by the keyword "proc call". The return address is stored alongside this symbol. A variable might appear mulitple times in Γ whence the mapping to be used is the first one from the top. This is denoted as $\Gamma[i \to x]$. The machine starts with the empty stack and $pc = 0$.

$$
\begin{array}{rlll}
\langle \mathsf{DECLARE\_INT}, i, \_, \_\rangle \vdash & \langle \Gamma, pc\rangle & \to & \langle \Gamma[i \to \_], pc+1\rangle \\
\langle \mathsf{DECLARE\_BOOL}, i, \_, \_\rangle \vdash & \langle \Gamma, pc\rangle & \to & \langle \Gamma[i \to \_], pc+1\rangle \\
\langle \mathsf{DECLARE\_PROC}, p, \mathrm{start}, \_\rangle \vdash & \langle \Gamma, pc\rangle & \to & \langle \Gamma[p \to \mathrm{start}], pc+1\rangle \\
\langle \mathsf{PRINT}, i, \_, \_\rangle \vdash & \langle \Gamma[i \to x], pc\rangle & \to & o.\mathsf{push}(\#x) \colon \langle \Gamma[i \to x], pc+1\rangle \\
\langle \mathsf{READ}, \_, \_, i\rangle \vdash & \langle \Gamma, pc\rangle & : i.\mathsf{pop}() = v \to & \langle \Gamma[i \to v], pc+1\rangle \\
\langle \mathsf{CALL}, p, \_, \_\rangle \vdash & \langle \Gamma[p \to pc_s], pc\rangle & \to & \langle \Gamma[p \to pc_s][\text{proc call}, pc], pc_s\rangle \\
\langle \mathsf{IF}, i, \mathrm{loc}, \_\rangle \vdash & \langle \Gamma[i \to x], pc\rangle & \to & \#x = \mathbf{tt} \colon \langle \Gamma[i \to x], pc+1\rangle \\
\langle \mathsf{IF}, i, \mathrm{loc}, \_\rangle \vdash & \langle \Gamma[i \to x], pc\rangle & \to & \#x = \mathbf{ff} \colon \langle \Gamma[i \to x], \mathrm{loc}\rangle \\
\langle \mathsf{GOTO}, \_, \mathrm{loc}, \_\rangle \vdash & \langle \Gamma, pc\rangle & \to & \langle \Gamma, \mathrm{loc}\rangle \\
\langle \mathsf{RETURN}, \_, \_, \_\rangle \vdash & \langle \Gamma[\text{proc call}, pc_s]\Gamma', pc\rangle & \to & \langle \Gamma, pc_s+1\rangle \\
\langle \mathsf{ASSIGN}, v, \_, i\rangle \vdash & \langle \Gamma, pc\rangle & \to & \langle \Gamma[i \to v], pc+1\rangle \\
\langle \mathsf{ASSIGN}, i_1, \_, i_{\mathrm{dest}}\rangle \vdash & \langle \Gamma, pc\rangle & \to & v = \#i_1 \colon \langle \Gamma[i_{\mathrm{dest}} \to v], pc+1\rangle \\
\langle \mathsf{OPER\_BINARY}, i_1, i_2, i_{\mathrm{dest}}\rangle \vdash & \langle \Gamma, pc\rangle & \to & v = \#i_1 \circ \#i_2 \colon \langle \Gamma[i_{\mathrm{dest}} \to v], pc+1\rangle \\
\langle \mathsf{OPER\_UNARY}, i_1, \_, i_{\mathrm{dest}}\rangle \vdash & \langle \Gamma, pc\rangle & \to & v = \circ(\#i_1) \colon \langle \Gamma[i_{\mathrm{dest}} \to v], pc+1\rangle \\
\langle \mathsf{END\_OF\_CODE}, \_, \_, \_\rangle \vdash & \langle \Gamma, pc\rangle & \to & \textbf{Execution complete}
\end{array}
$$

Here OPER_BINARY $\in \{\mathsf{PLUS}, \mathsf{MINUS}, \mathsf{MULT}, \mathsf{DIV}, \mathsf{MOD}, \mathsf{GEQ}, \mathsf{GT}, \mathsf{LEQ}, \mathsf{LT}, \mathsf{NEQ}, \mathsf{EQ}, \mathsf{AND}, \mathsf{OR}\}$ is a binary operator; OPER_UNARY $\in \{\mathsf{UPLUS}, \mathsf{UMINUS}, \mathsf{NOT}\}$ is a unary operator; i and o represent input and output streams. **Note** that all assignment duplicate contents (the other alternative could be duplicating references).

4 I/O Specifications

- **Input:** The input would be a file having one instruction on each line. The four fields in an instruction would be separated by a single whitespace. The first line in the file is the instruction at index 0 and so on. A simple input file implementing factorial is attached as *input.txt*.
- **Output:** The output should be a file having the contents of the pushed into the output stream in a linear order. The output corresponding to *input.txt* ($n = 5$) is in *output.txt*.

5 Instructions for Submission

- **All submissions must be through moodle. No other form of submission will be entertained.**
- **No submissions will be entertained after the submission portal closes.**
- **Sometimes there are two deadlines possible – the early submission deadline (which we may call the "lifeline") and the final "deadline". All submissions between the "lifeline" and the "deadline" will suffer a penalty to be determined appropriately.**

6 What to Submit?

- You will create one folder which will have 2 files, program file and the writeup file.
- The program file should be named with your Kerberos ID. For example, if kerberos id is 'cs1140999' then the file name should be cs1140999.sml or cs1140999.ocaml. The writeup should be named as "writeup.txt".
- Both the files should be present in one folder. Your folder also should be named as your Kerberos ID. For example, if kerberos id is 'cs1140999' then the folder should be called cs1140999.
- The first line of writeup should contain a numeral indicating language preferred with 0-ocaml, 1-sml and 2-haskell.
- For submission, the folder containing the files should be zipped(".zip" format). Note that, you have to zip folder and NOT the files.
- This zip file also should have name as your Kerberos ID. For example, if kerberos id is 'cs1140999' then the zip file should be called cs1140999.zip.
- Since the folder has to be zipped the file cs1140999.zip should actually produce a new folder cs1140999 with files (cs1140999.sml or cs1140999.ocaml) and writeup.txt.

  Hence the command *"unzip -l cs1140999.zip"* should show

  cs1140999/cs1140999.sml
  cs1140999/writeup.txt
- After creating zip, you have to convert ".zip" to base64(.b64) format as follows (for example in ubuntu):
  base64 cs1140999.zip > cs1140999.zip.b64 will convert .zip to .zip.b64
  This cs1140999.zip.b64 needs to be uploaded on moodle.
- After uploading, please check your submission is up-to the mark or not, by clicking on evaluate. It will show result of evaluation. If folder is as required, there will be no error, else REJECTED with reason will be shown. So, make sure that submission is not rejected.