Exercises 4 (Interpretation) – Solutions

4A. *(SVM interpreter)*

(a)

```
case DIV: {
  int w2 = data[--sp];
  int w1 = data[--sp];
  if (w2 == 0)
    status = FAILED;
  else
    data[sp++] = w1 / w2;
  break; }
```

(b)

```
case LOADG: {
  int d = code[pc++]<<8 | code[pc++];
  if (d < 0 || d >= data.length)
    status = FAILED;
  else
    data[sp++] = data[d];
  break; }
```

And similarly for `LOADL`, `STOREL`, `STOREG`.

(c)

```
case JUMP: {
  int c = code[pc++]<<8 | code[pc++];
  if (c < 0 || c >= cl)
    status = FAILED;
  else
    pc = c;
  break; }
```

And similarly for `JUMPF`, `JUMPT`.

4B. *(Mini-Basic VM)*

(a) A mini-Basic virtual machine would include a code store that contains one command per cell, together with a program counter that indicates the next command to be executed. It would also include a data store with 26 cells, one cell per variable. Finally, it would include a status register. On each iteration of the interpreter, the next command would be fetched, analysed, and executed.

(b) Option 1: represent each command by a *string*. There would be no delay before program execution starts. Each command must be analysed by a lexer and a parser just before execution. This would be slow – a significant performance issue since a command might be analyzed and executed frequently.

Option 2: represent each command by a *list of tokens*. There would be a short delay (lexing all commands) before program execution starts. Each command must be analysed by a parser only. Analysis of commands when executed would be somewhat faster than in option 1.

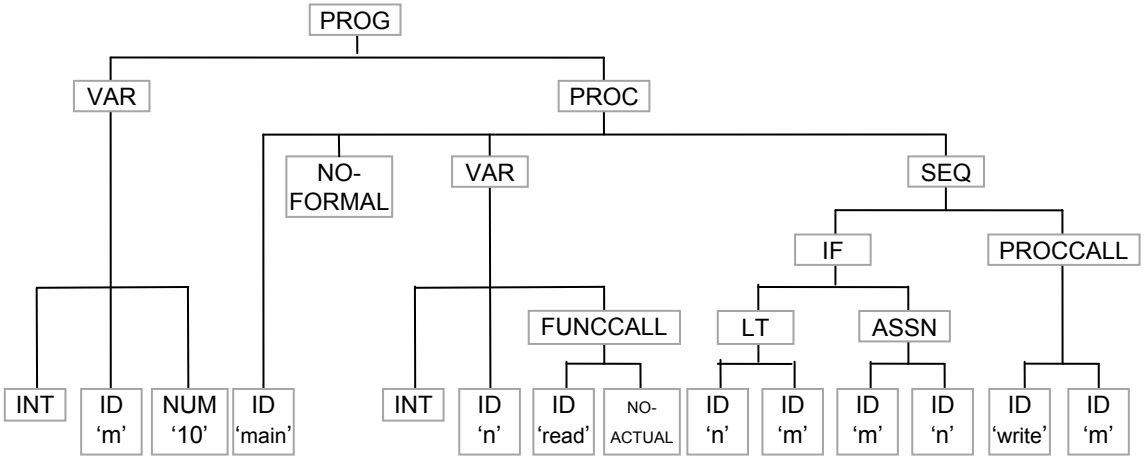
Option 3: represent each command by an *AST*. There would be a longer delay (lexing and parsing all commands) before program execution starts. Analysis of commands when executed would be faster than in option 1 or 2.

In option 2, commands could be lexed as they are edited. Similarly, in option 3, commands could be lexed and parsed as they are edited. Such integration of the editor and interpreter would eliminate any delay immediately before execution starts.

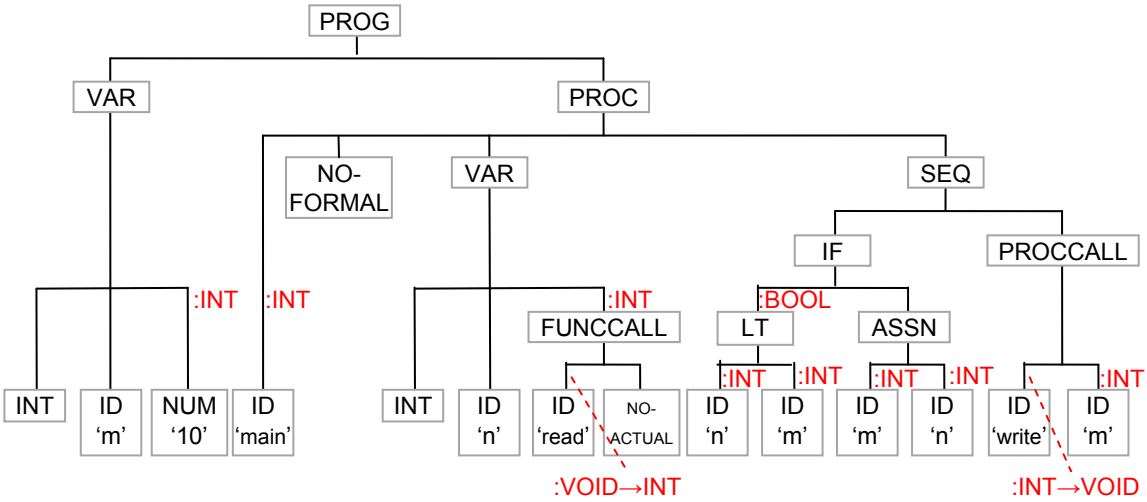
Exercises 5 (Compilation) – Solutions

5A. *(Fun compiler)*

(a) AST after parsing:

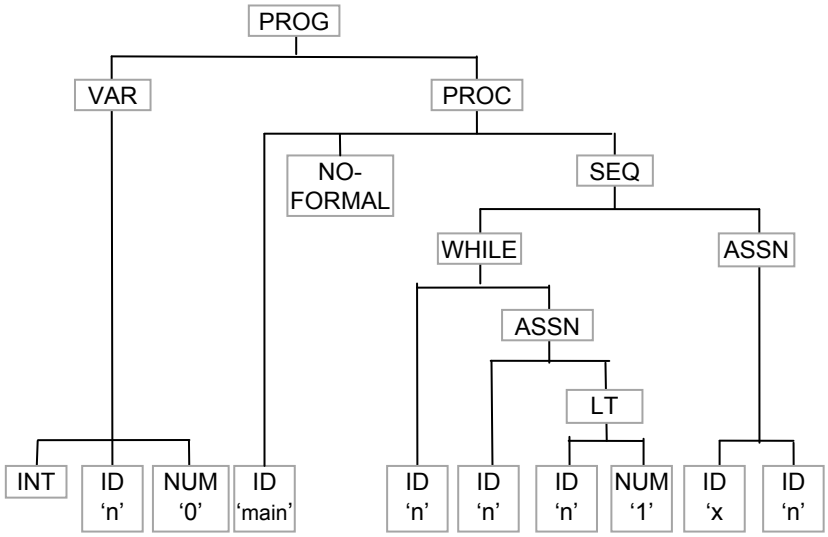


(b) AST after contextual analysis:



5B. *(Fun compiler – error detection)*

(a) AST after syntactic analysis:



(b) AST after contextual analysis: