Chapter 21

Interpreters

Interpreter

Pure interpreter:
executes code as it parses it

Compiler-interpreter:
translates to intermediate code and then interprets the intermediate code.

Modified termList method

```
 1 private void termList()
 2 {
 3   int right;
 4
 5   switch(currentToken.kind)
 6   {
 7     case PLUS:
 8       consume(PLUS);
 9       term();
10       right = s.pop();
11       s.push(s.pop() + right);
12       termList();
13       break;
14     case RIGHTPAREN:
15     case SEMICOLON:
16       ;
17       break;
18     default:
19       throw genEx("\"+\", \")\", or \";\"");
20   }
21 }
```

Modified assignmentStatement method

```
 1 private void assignmentStatement()
 2 {
 3  Token t;
 4  int left, expVal;
 5
 6  switch(currentToken.kind)
 7  {
 8    case ID:
 9      t = currentToken;
10      consume(ID);
11      left  = st.enter(t.image);
12      consume(ASSIGN);
13      expr();
14      st.setValue(left, s.pop());
15      consume(SEMICOLON);
16      break;
17    default:
18      throw genEx("<ID>");
19   }
20 }
```

Using the I1 interpreter

```
javac I1.java
java I1 S1
I1 Interpreter written by ...
4107
4107
```

Interpreting statements that transfer control

```
 1   private void whileStatement()
 2   {
 3     Token t;
 4
 5     consume(WHILE);
 6     consume(LEFTPAREN);
 7
 8     // save this position in token chain
 9     t = currentToken;
10     boolean exModeSave = exMode;
11     do
12     {
13       currentToken = t;   // reset position in chain
14       expr();
15       consume(RIGHTPAREN);
16       if (exMode && s.pop() == 0) exMode = false;
17       statement();
18     } while (exMode);
19     exMode = exModeSave;  // reset exMode
20   }
```

Compiler-interpreter

Compile source code to the machine language (s-code) for the s-machine. Then interpret the s-code.

S-code interpreter

```
34        opcode = scode.get(pc++);
35
36        // decode opcode and execute instruction
37        switch(opcode)
38        {
39          case PRINTLN:
40            System.out.println(s.pop());
41            break;
42          case ASSIGN:
43            vtab[scode.get(pc++)] = s.pop();
44            break;
45          case PLUS:
46            right = s.pop();
47            s.push(s.pop() + right);
48            break;
49          case TIMES:
50            right = s.pop();
51            s.push(s.pop() * right);
52            break;
53          case PUSHCONSTANT:
54            s.push(scode.get(pc++));
55            break;
56          case PUSH:
57            s.push(vtab[scode.get(pc++)]);
58            break;
59          case HALT:
60            doAgain = false;
61            break;
62          default:
63            doAgain = false;
64            break;
```

Translating to s-code

```
 1   private void doWhileStatement()
 2   {
 3     int address;
 4
 5     consume(DO);
 6     address = cg.getCurrentAddress();
 7     statement();
 8     consume(WHILE);
 9     consume(LEFTPAREN);
10     expr();
11     cg.emit(JNZ);
12     cg.emit(address);
13     consume(RIGHTPAREN);
14     consume(SEMICOLON);
15   }
```

Translating to s-code

```
 1 private void printArg()
 2 {
 3   Token t;
 4   int sindex;
 5
 6   switch(currentToken.kind)
 7   {
 8     case STRING:
 9       t = currentToken;
10       consume(STRING);
11       sindex = cg.enterString(t.image);
12       cg.emit(PRINTSTRING);
13       cg.emit(sindex);
14       break;
15     case LEFTPAREN:
16     case UNSIGNED:
17     case PLUS:
18     case MINUS:
19     case ID:
20       expr();
21       cg.emit(PRINT);
22       break;
23     default:
24       throw genEx("expression or string");
25   }
26 }
```