Prof. Jingke Li (FAB120-06, lij@pdx.edu), Tue & Thu 12:00-13:15 @ ASRC 230, Lab: Fri 10:00-11:15/11:30-12:45 @ FAB 170

Lab 6: Interpreters

In this lab, you'll have a hands-on practice in implementing an interpreter for the stack-based language SC0. Download from D2L the file `lab6.zip` and unzip it.

The Interpreter Structure

There are three parts to this interpreter implementation:

1. A front-end to read in a SC0 program from a file.
2. An implementation of two storage models used in SC0: the operand stack and the local variable array.
3. The actual interpretation of the SC0 instructions.

Since SC0's instructions are line-based, we can use Java's `BufferedReader` to handle the input of a SC0 program from a file, reading each instruction in as a String, and converting a whole program into an array of Strings:

```
FileInputStream stream = new FileInputStream(args[0]);
BufferedReader reader = new BufferedReader(new InputStreamReader(stream));
ArrayList<String> insts = new ArrayList<String>();
String line;
while ((line = reader.readLine()) != null) {
  if (!(line.isEmpty() || line.trim().equals("")
        || line.startsWith("#")))
    insts.add(line);
}
```

To decode an individual SC0 instruction, we use Java's String `Scanner`:

```
Scanner sc = new Scanner(inst);
String lnum = sc.next();                    // line number (ignored)
String name = sc.next();                    // inst name
int n = sc.hasNextInt() ? sc.nextInt() : 0; // inst operand
```

The two storage models can be implemented with Java's `HashMap` and `Stack`, respectively:

```
static HashMap<Integer,Integer> vars = new HashMap<Integer,Integer>();
static Stack<Integer> stack = new Stack<Integer>();
```

Interpreting Individual Instructions

The standard interpreter's "fetch-and-execute" loop is used to decode and interpret instructions one at a time:

```
int pc = 0;
while (pc < insts.length) {
  ... // decode inst into name and operand n
  int disp = execute(name, n);
  pc += disp;
}
```

Individual intstructions are executed according to their semantics:

```
static int execute(String instName, int n)
{
  int disp = 1;          // default displacement value
  switch (instName) {
  case "ADD":
    int val2 = stack.pop();
    int val1 = stack.pop();
    int res = val1 + val2;
    stack.push(res);
    break;

  case "LOAD":
    val = vars.get(n);  // inst's operand is var's idx
    stack.push(val);
    break;

  case "GOTO":
    disp = n;           // inst's operand is the displacement
    break;

  ...
  }
  return disp;
}
```

Exercise

A starter version of the interpreter is provided in `SC0Interp0.java`. It contains the above-mentioned code portion, as well as the structure of instruction execution routine. Your task is to complete the interpreter by providing execution code for individual instructions.