COS 126 **General Computer Science** **Fall 2013**

Programming Exam 2

This programming exam has 2 parts, weighted as indicated. The exam is semi-open: you may use the course website, the booksite, any direct links from those two, the textbook, any printed or written notes, and your past coursework on your computer. You may not use other websites. No communication is permitted, except with course staff members.

Upload your code to the dropbox. As with the COS 126 assignments, you can submit your code multiple times for testing, but only the last version will be graded.

Your code will be graded primarily on correctness. You will lose a substantial number of points if your program does not compile or has the wrong API. However, efficiency, clarity, and code style are also factors in your grade. Remember to include headers and comments in your code. Headers MUST include your name, netID and precept number.

Print your name, netID, and precept number on this page (now), and write out and sign the Honor Code pledge before turning in this paper. It is a violation of the Honor Code to discuss this exam until everyone in the class has taken the exam. You have 90 minutes to complete the test. **Do not remove this exam from the exam room.**

Write out and sign the Honor Code pledge before turning in the test:
"I pledge my honor that I have not violated the Honor Code during this examination."

Pledge: _____

_____

Signature: _____

Name: _____

NetID: _____

Precept: _____

P01	12:30 TTh	Donna Gabai
P01A	12:30 TTh	David Pritchard
P01B	12:30 TTh	Aleksey Boyko
P02	1:30 TTh	Doug Clark
P02A	1:30 TTh	Sachin Ravi
P02B	1:30 TTh	Aleksey Boyko
P02C	1:30 TTh	Borislav Hristov
P03	2:30 TTh	Doug Clark
P04	7:30 TTh	Kevin Lee
P05	1:30 WF	Donna Gabai
P05A	1:30 WF	Judi Israel
P05B	1:30 WF	Dave Pritchard
P06	2:30 WF	Kevin Lai
P06A	2:30 WF	Judi Israel

Problem	**Point Value**
`MiniPro`	22
`MiniPro2`	8
Total	30

In this exam you will create an interpreter that is capable of simulating programs in a very simple programming language called MiniPro. MiniPro only handles integer arithmetic with 32-bit two's complement integers, the same as Java's `int` type.
In Part 1, you will handle storing and printing variables. In Part 2, you will handle calculations and a control flow statement. You can submit Part 1 without doing Part 2. But because the two parts are related, you may want to briefly skim Part 2 before starting to implement Part 1.

Part 1

For this part, you will deal with programs that only have two kinds of statements: assignment statements, and println statements. Here is a sample MiniPro program with these two kinds of statements.

```
x = 13                  (this is line 0)
y = x                   (this is line 1)
x = 39                  (this is line 2)
x println               (this is line 3)
y println               (this is line 4)
```

This particular program defines a variable `x` and sets its value equal to 13. Then it defines `y` equal to `x` (which will define a variable `y` and set the value of `y` to 13). It then redefines the value of `x` to be 39. Finally, it prints the value of `x` and the value of `y` — so executing the program prints 39, then 13, on separate lines.

You will create a class that allows the user to execute the lines of the program one at a time.

In order to make your work more straightforward, we will assume that the client has already broken the program into lines, with each line broken into space-separated tokens (variables, commands, or integers). For example, the above program will be represented as

```
String[][] sampleProgram = {{"x", "=", "13"},
                            {"y", "=", "x"},
                            {"x", "=", "39"},
                            {"x", "println"},
                            {"y", "println"}};
```

So each token is a `String`, each line is a `String[]`, and the whole program is a `String[][]`. (Note: the number of lines in the whole program can be determined using `sampleProgram.length` which is 5 in this example. To determine the number of tokens on line `i`, use `sampleProgram[i].length`.)

You must create a class `MiniPro` with the API below.

```
MiniPro(String[][] program) // Create interpreter for this program (don't execute it yet!)
int valueOf(String v) // Return the current value of the variable named v. If no
                      // such variable is currently defined, throw a RuntimeException.
int programCounter()  // Return the number of the line that will execute next.
void step()           // Execute the line whose number equals the value of the
                      // program counter. Then, increment the program counter.
boolean isDone()      // Is the program done?
```

We give an electronic version of the API, that you may copy, at:

`http://www.cs.princeton.edu/~cos126/MiniPro`

You may add a test `main()` method to help with debugging if you like (but it is completely optional). Here is one sample test `main()` you may use; it executes the `sampleProgram` mentioned previously.

```
public static void main(String[] args) {

   String[][] sampleProgram =  // the program in a format ready for the interpreter:
        {{"x", "=", "13"},     // line 0
         {"y", "=", "x"},      // line 1
         {"x", "=", "39"},     // line 2
         {"x", "println"},     // line 3
         {"y", "println"}};    // line 4

   MiniPro mp = new MiniPro(sampleProgram);
   StdOut.println(mp.isDone());         // should print false
   StdOut.println(mp.programCounter()); // should print 0 (0 always first line)

   // run thru program
   mp.step();                           // line 0 sets x = 13
   StdOut.println("currently x is " + mp.valueOf("x")); // prints "currently x is 13"
   mp.step();                           // line 1 sets y = 13
   mp.step();                           // line 2 sets x = 39
   mp.step();                           // line 3 prints x value which is 39
   mp.step();                           // line 4 prints y value which is 13

   StdOut.println(mp.isDone());         // should print true (program is finished)
}
```

The URL given earlier has a link to this test `main()` code. Here is the output of a sample run:

```
% java-introcs MiniPro
false
0
currently x is 13
39
13
true
```

In defining the class and constructor, note that each `MiniPro` instance needs to remember:

- the program counter (the number of the line that will be executed next).
- the table of names and values of all variables defined so far.
- the program. We do *not* require you to make a defensive copy of the program.

In implementing `step()`, you will have to examine the line to be executed and take an appropriate action based on its contents. For part 1, each line is a `String[]`, with one of these two forms:

- `{<variable name>, "=", <integer or variable name>}`
- `{<variable name>, "println"}`

One way of distinguishing assignment statements (= statements) from println statements is to examine the second token. Remember that *you should not use* `==` *with* `String` *objects.* Use the `.equals()` method of `String` objects instead.

Variable names will consist of one or more lower-case characters. For the assignment statement, `step()` will need to determine whether the third token is a variable name or an integer. Using the regular expression `[a-z]+` with the `matches()` method of the `String` class can help you distinguish integers from variable names — here is an example just to illustrate.

```
String aVarName = "sum";
StdOut.println(aVarName.matches("[a-z]+")); // true
String aNumber = "126";
StdOut.println(aNumber.matches("[a-z]+")); // false
```

You do not have to check for invalid syntax. You do not have to check for programs that would crash due to undefined variables; e.g., we *do not* care what happens when your interpreter runs

```
String[][] badProgram = {{"undefinedvar", "println"}};
```

We won't try to call `programCounter()` or `step()` on any `MiniPro` instance that is done.

The above test `main()`, and more test clients including one that reads from `StdIn`, are provided at

`http://www.cs.princeton.edu/~cos126/MiniPro`

These are the same tests that we run when you upload `MiniPro.java` to the dropbox. *Submit your code using the link on the Assignments page.*

Part 2

You will now write a more advanced version of the class you wrote for Part 1. The API will be the same, using the class name `MiniPro2` instead. The new commands that `MiniPro2` can handle are: three arithmetic commands `+=`, `-=`, and `*=`; and one control flow command `pos?jump`.

Lines using the arithmetic commands `+=`, `-=` and `*=` will have the syntax

```
{<variable name>, "+=", <integer or variable name>}
{<variable name>, "-=", <integer or variable name>}
{<variable name>, "*=", <integer or variable name>}
```

They should behave the same as in Java: `x += y` means "add `y` to `x`," `x -= y` means "subtract `y` from `x`," and `x *= y` means "multiply `x` by `y`." Here is a sample execution demonstrating this.

```
String[][] mathProgram = {{"x", "=", "7"},
                          {"y", "=", "10"},
                          {"x", "+=", "y"},
                          {"x", "*=", "3"},
                          {"x", "println"}};
MiniPro2 mp2 = new MiniPro2(mathProgram);
while (!mp2.isDone()) mp2.step();
```

This program should print out 51, because (7+10)*3 is 51.

The other new command is called `pos?jump`. Its syntax is

```
{<variable name>, "pos?jump", <integer or variable name>}
```

The meaning of `x pos?jump v` is the following: if `x` is positive, "jump" `v` lines; if the variable `x` is less than or equal to 0, take no special action. "Jumping" means to move forward `v` lines (or backward if `v` is negative — an example is given later on).

Here is an example `MiniPro2` program to compute the absolute value; there is a version at the left and one at the right, with two different initial values.

```
x = -10
x pos?jump 2
x *= -1
x println
```

```
x = 126
x pos?jump 2
x *= -1
x println
```

In the left version, `x` is not positive so the `pos?jump` command does nothing special, then `x` is multiplied by -1 (giving 10), and 10 is printed. In the right version, `x` is positive so we jump forward 2 lines (instead of the normal 1). The `x *= -1` line is never executed. The interpreter therefore prints 126.

Here is an example program to compute the sum $10 + 9 + 8 + \cdots + 1$:

```
i = 10
sum = 0
sum += i
i -= 1
i pos?jump -2
sum println
```

When `i` is positive, we jump back to the `sum += i` line. Executing this program will cause it to print out 55 (which is $10 + 9 + 8 + \cdots + 1$).

If you happen to jump to beyond the end of the program *or* before the beginning, subsequent calls to the `isDone()` method should return `true`.

Tests including the above examples are provided at

`http://www.cs.princeton.edu/~cos126/MiniPro`

These are the same tests that we run when you upload `MiniPro2.java` to the dropbox. *Submit your code using the link on the Assignments page.*