CS 260
Spring 2015
Project 5

- The purpose of this project is to use stacks to implement a small programming language, in the same way that stacks are used to implement much larger real-world programming languages. Your task is to write a Python3 program project5.py that reads and interprets programs written in the Nano programming language, as described below.
- Every value in Nano has type integer. Each Nano identifier consists of a single lowercase letter. Each Nano integer literal consists of a single digit.
- The Nano programming language provides nested scopes, assignment statements, arithmetic expressions, and output statements.
- Each Nano program begins in the global (outermost) scope. Each pair of matching braces { … } delimits a new nested scope. The first statement in each scope declares all the identifiers that are local to this scope. Its syntax consists of the @ symbol followed by a comma-separated list of identifiers, and terminates with a semicolon. All local identifiers are initialized to value 0.
- An assignment statement in Nano consists of an identifier, the = symbol, an arithmetic expression, and terminates with a semicolon.
- An arithmetic expression in Nano contains one or more identifiers and/or integer literals, which are combined using arithmetic operators. Operators include +, -, *, /, and ^ (exponentiation). The three precedence levels are +, - (lowest) and *, / (middle) and ^ (highest). All operators are left-associative, except ^ which is right-associative. Pairs of matching parentheses (…) can be placed around subexpressions to override these precedence and associativity rules.
- An output statement in Nano indicates one or more identifiers whose current value is to be displayed. Its syntax consists of the ? symbol followed by a comma-separated list of identifiers, and terminates with a semicolon. Each identifier is displayed on a separate output line in this format: identifier = value
- In a Nano program, all whitespace characters (blanks, tabs, newlines) are ignored.
- You may assume that each input file is a syntactically and semantically correct Nano program. Three example Nano programs are shown on the next page.
- Your interpreter program will need to use two stacks to correctly interpret Nano programs: an environment stack and an expression stack.
- The environment stack holds all the currently existing scopes. Each scope is a list of pairs of each local identifier and its current value. Each left brace causes a new empty scope to be pushed onto the stack, and each right brace causes the top scope to be popped from the stack. Each identifier listed within an @ statement is added to the current local scope (at top of the environment stack), along with its initial value 0. Each identifier within an assignment statement or print statement is first searched in the current local scope (at top of the environment stack) before searching within nonlocal scopes (proceeding in order down the stack).
- The expression stack is used to evaluate each arithmetic infix expression. When each identifier or integer literal is encountered, push its value onto the expression stack. Also push each left parenthesis onto this stack. When a right parentheses is seen, perform the operations on the stack down to its matching left parentheses. When each new operator is seen, perform the

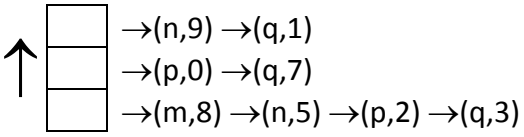
operations on the stack that have higher precedence, or that have the same precedence if left associative, and then push the new operator onto the stack. When a semicolon is seen, perform all the remaining operations on the stack.

- Here are three example Nano programs and their corresponding outputs, generated as follows:
  python3 project5.py < input1.nano > output1.txt
  python3 project5.py < input2.nano > output2.txt
  python3 project5.py < input3.nano > output3.txt

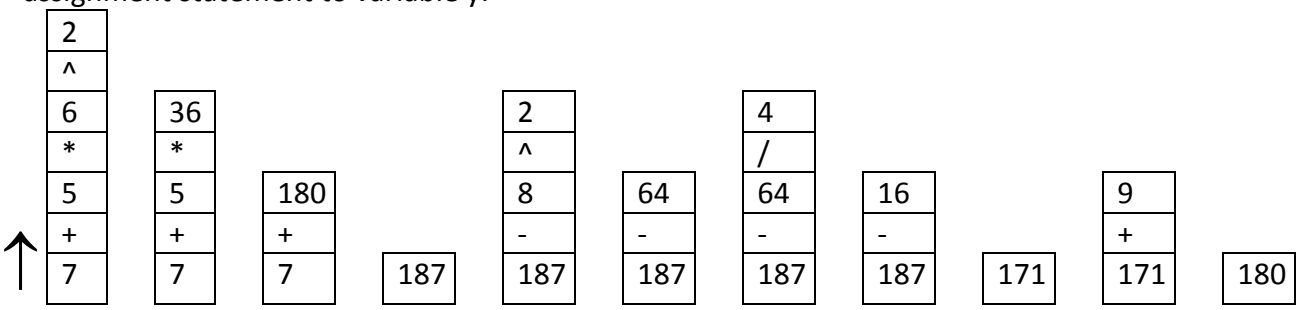
input1.nano and output1.txt		input2.nano and output2.txt		input3.nano and output3.txt	

@ u, v, w, x, y, z;
? u, v, w, x, y, z;
u = 9-2-4+7;
v = 8*6/2/4*5;
w = 2^2^3;
x = 2^4*5-7*3^2;
y = 7+5*6^2-8^2/4+9;
z = ((1+2)*4)^(6/(8-5));
? u, v, w, x, y, z;
	u = 0
	v = 0
	w = 0
	x = 0
	y = 0
	z = 0
	u = 10
	v = 30
	w = 256
	x = 17
	y = 180
	z = 144
		@a,b,c,d;
		a=2;
		b=3;
		?a,b,c,d;
		{ @b,d;
		b=4;
		d=5;
		c=a*(b+d);
		}
		?a,b,c,d;
		{ @a,c;
		a=6;
		c=7;
		d=b+a*c;
		}
		?a,b,c,d;
			a = 2
			b = 3
			c = 0
			d = 0
			a = 2
			b = 3
			c = 18
			d = 0
			a = 2
			b = 3
			c = 18
			d = 45
				@ m,n,p,q;
				m = 0;
				n = 1;
				p = 2;
				q = 3;
				{ @ p,q;
				m = 4;
				n = 5;
				p = 6;
				q = 7;
				{ @ n,q;
				m = 8;
				n = 9;
				p = 0;
				q = 1;
				? m,n,p,q;
				}
				? m,n,p,q;
				}
				? m,n,p,q;
					m = 8
					n = 9
					p = 0
					q = 1
					m = 8
					n = 5
					p = 0
					q = 7
					m = 8
					n = 5
					p = 2
					q = 3

- You should read these Nano programs and make sure you understand the Nano programming language syntax and semantics before you write the Nano interpreter. In particular, make sure you can figure out the contents of the environment stack and the expression stack at each step.
- Here are the contents of the environment stack for input3.nano, immediately before its first output statement. Top of stack is local scope, and bottom of stack is global scope.



- Here are the contents of the expression stack for input1.nano, at several places during the assignment statement to variable y:



- You are permitted/encouraged to use scanner.py when reading the Nano input file. To help you get started, here is a pseudo-code outline that shows the main structure of the interpreter.

```
while c = fetch the next visible character:
        if c is left brace:
                push new scope on environment stack;
        else if c is right brace:
                pop current scope from environment stack;
        else if c is @ symbol:
                read list of identifiers and add them to current scope;
        else if c is question mark:
                read list of identifiers, and search for and display their current values;
        else if c is lowercase letter:
                c must be the left side of an assignment statement;
                read the right side of the assignment statement;
                evaluate the right side using the expression stack;
                update the value of the left side identifier in the environment stack;
```

The following rules apply to all projects in this course:

- We now provide a virtual machine so you can test your program in the same environment where it will be graded. This server is "`cs260.ua.edu`". Regardless where you primarily develop your program, you must use this server to test your projects before submitting. Login using your Bama userid and password. You might first need to install the SSH secure shell and file transfer clients, for example at http://helpdesk.ua.edu/~recovery/ssh.htm. Also, if you wish to access the server from off-campus, you'll need to install a VPN client, which is available at https://mybama.ua.edu on the Tech tab.
- During grading, your program will be tested on the `cs260.ua.edu` server. Example scripts and input/output files will be provided on this server in the /projects directory.
- **You are responsible for ensuring that your program works correctly for all possible inputs. The provided input/output files are NOT guaranteed to catch every error in your program. We will test your program using other data that was not provided to you in advance. So you should read the assignment carefully, do not make any extra assumptions about the I/O formats, ask for clarification if you are in doubt, and thoroughly test your program by creating additional data beyond the files we provide.**

- We provide a common platform that all students must use to test their programs before submitting, and we will use this same platform to evaluate each submitted program. Your program will NOT be evaluated based on how it performs in any other environment, because we will not attempt to replicate different environments used by each individual student.
- In previous semesters, some students did not follow the instructions and/or did not adequately test their programs before submitting, so they lost points for errors that they could have easily detected and corrected. This semester, it is required to test the final version of your program on the server before you submit it. All the scripts and input/output files are provided on the server in the /projects directory. When you run the script, it will produce a file "checksums.txt" that you must submit along with your program. When grading your project, we will use the contents of this file to verify that you actually did test the final version of your program on the server.
- You are permitted to verbally discuss ideas with classmates or other people, but sharing any code is strictly forbidden. **Do not look at your classmates' or anybody else's code, and do not show anybody your code!** However, you are encouraged to verbally discuss the projects with your classmates, provided nobody involved is looking at any code during the discussion.
- You must write all your own code for each project. Do not take code from any other source. If you happen to find a similar program online or in a book, you are NOT permitted to copy portions of that program into your program. The purpose of the projects is to learn how to write programs for data structures and algorithms, not to practice googling for solutions.
- I**f your code is too similar to another student's code or to anybody else's code, you will receive an invitation to discuss the situation in the Dean's office.** We use the MOSS system which detects plagiarism based on programs that are too similar. The penalty for plagiarism is typically a score of 0 on the project or a grade of F in the course.
- You should strive to make each program as efficient as possible. During testing, we will allow each program to run for a reasonable amount of time, but if your program takes much longer to run than we think it should, we will eventually decide your program is too inefficient (or possibly in an infinite loop), and in this case we may terminate the execution of your program before it ends.
- Place all your program's source files and the file "checksums.txt" (generated by the script) into a single zip file, give it a name of the form LastName_FirstName_ProjectNumber.zip (example: Doe_John_1.zip), and upload this zip file using Blackboard. Make sure your zip file includes the latest version of all your source files. Your zip file should contain the file "checksums.txt" and all the source files needed to run your program, and these files should all be located at the top level of your zip file. The zip file should not contain any other subfolders. Do not submit projects via email.
- Each assignment will be due by 11:59pm on the due date specified on Blackboard. Please aim to submit your project earlier than the due date in case you encounter unforeseen circumstances at the last minute. There will be a substantial deduction for late submissions. Points will also be deducted if you do not follow all the instructions exactly, because this can make it difficult or impossible to grade your project.

- After the projects are graded, your score will be uploaded onto Blackboard, along with a grading report that summarizes how your score was calculated. Also, any additional input/output files used when grading the projects will be placed in the /projects directory on the `cs260.ua.edu` server.
- To completely understand your project's score, you will likely need to run your program on the server using the new input/output files. If you think your project grade was computed incorrectly, you must inform us **within at most one week** from when your score is posted on Blackboard. You must stop by one of the instructors' offices during office hours and show us that your program runs correctly using the script and input/output files located on the server. If a mistake was made when grading your program, we will be glad to correct it.
- However, note that the following statements (which we hear quite frequently) will NOT cause us to increase your project score:
  - "I spent many hours working on the project, so I want a higher score."
  - "My program contains over 1000 lines of code, so I want a higher score."
  - "I didn't follow all the instructions because I thought they were optional."
  - "My program shouldn't be tested on any valid inputs that I forgot to consider."
  - "I didn't understand the project description, and I didn't ask for clarification."