Data Structure & Algorithms
CS210A
Semester II, 2015-16, CSE, IIT Kanpur

Programming Assignment III

Deadline : 11:59 pm of 8th Feb

Note: This programming assignment consists of two parts: The first part has only one question of 25 marks and is compulsory. The second part has two problems and you have to do **only one** of these two problems. First one is an easier problem and won't require much thinking. The second problem will require your analytical and creative skills. Obviously the marks distribution is not same. The first problem carries 50 marks and the second one carries 75 marks.

1 Implementing Stack

You need to provide an implementation of the stack data structure in C. All operations discussed in the class have to be implemented. The items of the stack will be of type string (character array). In the second part of the problem, whenever you need a stack you must use the implementation of the stack developed in this part.

2 Expression Evaluation

2.1 Simple Expressions

We discussed an algorithm for evaluating arithmetic expressions in class using Stacks. You need to implement this algorithm with the following change. /, − have associativity **right to left** instead of left to right. Rest of the operators have the same precedence and associativity as discussed in class.
You may assume that input expression is a valid arithmetic expression involving integers only. Also assume that expression terminates with #.

Sample input and output:

1. `input:`
   `3+8/4/2-7#`
   `output:`
   `0`
2. `input:`
   `5*2+8-7-4#`
   `output:`
   `15`

2.2 Generalized Expression

The aim of this problem is to implement an extension of the algorithm for arithmetic expression evaluation that we discussed in the class. Firstly, let us ask ourselves what is an arithmetic expression. We know that $3+5^6$ is an arithmetic expression whereas neither $3*+6-5$ nor $2*(3+5))$ is an arithmetic expression. Can we give a precise definition of an arithmetic expression ? Think over it. Indeed it is possible through recursion. The following is a recursive definition of an arithmetic expression:

- Each number is an arithmetic expression and its value is the same as the number. Likewise, a variable of type number is also an arithmetic expression and its value is the latest value assigned to it.
- if α is an arithmetic expression then (α) is also an arithmetic expression whose value is the same as the value of α.
- if α and β is an arithmetic expression, then $\alpha \circ \beta$ is also an arithmetic expression where $\circ$ is any arithmetic operator(+, - , * , /, ^).

The above definition can be used to determine if an expression is a valid arithmetic expression. However, as a careful reader might have noticed that the above definition of arithmetic expressions is ambiguous. Indeed, the above definition may lead to multiple possible values of an expression. However, if we specify all the precedence and associativity of the operators, each expression will have a unique value. [In the course on Compilers, you will study a more elegant mechanism of removing the ambiguity by defining the *grammar* of expression suitably]. We shall now extend the domain of arithmetic expression.

In addition to the usual binary operators (+, - , * , /, ^) We introduce an additional operator $=$ called *assignment* operator. Its details are described as follows. If x is a variable and α is an expression, then $x = \alpha$ is also an expression. The value of expression $x = \alpha$ is defined as the value of expression α. The evaluation of $x = \alpha$ involves evaluation of α and assigning this value to variable x. Notice that $=$ is right associative.

Design and implement an algorithm to evaluate a valid generic expression. The challenge lies in incorporating the assignment operator and variables in a seamless manner instead of resorting to case analysis. Your algorithm must make a single scan from left to right. One subtle issue is the following. When you encounter a variable, how to detect whether this variable is to be considered like a number here or as a left hand side of an assignment expression. Think carefully: there is an easy way to detect it. Note that if a variable is to be considered as a number, its value is the latest value assigned to it in the past.

You may assume that only two variables, x and y are allowed in an expression. However, they may appear multiple times in an expression. The initial value of x as well as y is 1. Your output must consist of three lines: value of x in the first line, the value of y in the second line, and the value of the expression in the third line. You may assume that input expression is a valid arithmetic expression and it terminates with #.

Sample input and output:

1. ```
   input:
   3+5^(2*3-(x+y))#
   output:
   1
   1
   628
   ```
2. ```
   input:
   3*6+2^x=2+(x+1)*(3-y)#
   output:
   6
   ```

```
1
82
```

3. ```
   input:
   100+2*x+x=2^4+3*y=3*(2+4^2)-22#
   output:
   112
   32
   214.
   ```

4. ```
   input:
   3+(2*3+x=5+(x+y))+5*x+y#
   output:
   7
   1
   52
   ```

Note: Students from CSE department will see more of expression evaluation or "parsing" during their course on compilers or theory of computation. Compared to that, the above problem is just a toy example. However, its aim is just to address algorithmic aspect of expression evaluation.