Time spent:15 minutes

Answer 1:

Given: a non-terminal <t> it will generate variables.

The exponentiation operation denoted by "^" has higher priority and is right associative.

The multiplication operation denoted by juxtaposition has lower priority in comparison and is left associative.

To build:

A grammar for algebraic expression involving multiplication and exponentiation.

Let <e>, <e1>, <e2> represent non-terminals and <t> represent terminal for the expression.

Expression for multiplication we have keeping in mind that it is left associative we allow it to expand to the left and hence apply nonterminal to the left.

We have:

<e> -> <t>|<e> <t>---- (1)

Expression for exponentiation, we have keeping in mind that it Is right associative, we leave it to expand to the right and hence apply nonterminal to the right.

<e> -> <t>|<t>^<e>----- (2)

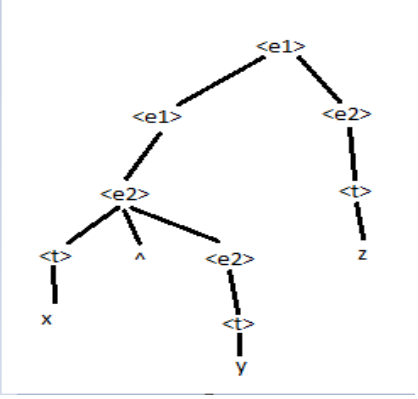
Now an expression for the combination, using (1) and (2) we have can be as follows:

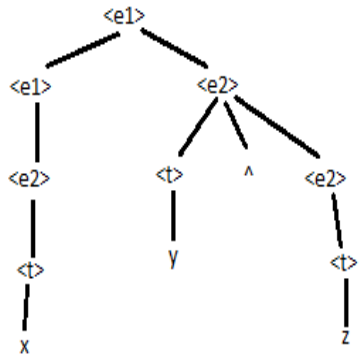
<e1> -> <e2>|<e1> <e2>

<e2> -> <t>|<t> ^ <e2>

This ensures that both the left and right associativity of respective operators are held and exponentiation has higher priority over the multiplication. Examples for the above said:

Consider graph to build x^y z and xy^z from left to right respectively


<e1>
<e1>
<e2>
<e2>
<t>
^
<e2>
<t>
x
<t>
y
z


<e1>
<e1>
<e2>
<e2>
<t>
^
<e2>
<t>
x
y
<t>
z

Time spent: 1 hour (30 minutes to collect ideas and 30 minute to write it down)

2.

First class functions are those functions which support passing functions as parameters i.e. as arguments to a function and as return values. Garbage collection is the automatic memory management wherein any memory that is left unused will get cleared instead of it being locked up. This helps in saving a lot of memory leaks. First class functions require some runtime and virtual machine support. First class functions rewrite several times to the compiler.

In languages with immutable data garbage collection would seem more efficient as there are no cycle sin variable references. Although in some languages the variables are on the heap the references to these variables are in the stack and hence it is needed. Garbage collectors help in making the accessibility to free variables faster .and guarantees locality of reference. Now when we consider the first class functions stack based implementation the clear understanding of the need of a garbage collector arises.

When function is returned by the function call then it is called upwards function arguments and when the functions are passed as the argument to the function then it is called downward functional arguments. The state of the caller is preserved when the function calls are made. The state is preserved in order so that once the call returns the control flows back in the sequential manner. In majority of the programs this sequence is stored in the activation records and these are pushed and popped corresponding to the calls being made and when calls return.

A problem with the upward functional argument arises when the calling function has referred to function state of an exited function after it has returned. Hence we see that the activation record must not be deallocated when the functions return.

As a solution to this problem we allocated all activation records from heap and hence ensure that the garbage collection would ensure that it is deallocated whenever it is no longer needed. And hence it is seen that the first order functions are found in languages with garbage collection as it enhances its implementation.

References apart from text book:


- http://en.wikipedia.org/wiki/First-class_function

Time spent:10 minutes for code+30 minutes for documentation.

3. Name of the language: C

The implementation that I have tested for is from right to left.

Source Code: Attached file aman.c is the source code for the evaluation.

Output: Output file as well as the screen shots is attached.

Compilers used are: GCC and Turbo C.

In the code that I have implemented I made use of the increment operator as they help us judge the order.

As per the output generated the order seems to be from right to left as the last variable seems to have been assigned the first value that is after increment i.e. it is been assigned 1 and so on.

The documentation of C that is attached with this email shows that there is no particular order in which the evaluation is defined and also there were several links which stated that in C and C++ order of evaluation is undefined.

I have implemented the same code in two compilers i.e. turbo c and the GCC compiler and have found that for both of them the output seems to keep it from right to left.

Also the order of evaluation depends on the order in which the code is written and the functional flow f the code and hence we can say that the order of evaluation in C is not fixed.

References used:

- Attached documentation of c
- http://stackoverflow.com/questions/621542/compilers-and-argument-order-of-evaluation-in-c

  Hence to conclude the order of evaluation depends on the style in which the program runs and is also compiler dependent in C.

Time spent in doing: 1 hour to understand the needs+ 30 minutes to code.

4. The overall understanding of the code is as follows.

Here we had to implement an iterator helper class that would perform certain functions as to

- Create a thread which acts the control for the execution.
- Perform the yield function wherein the yield function would calculate the index for the data on which the loop has to perform some action.
- This yield must wait rather sleep till it is been triggered by a get function form the range class.
- The get function will help to obtain the value of the next element if available.
- Here the thread is put to sleep till a notify is obtained from the range class.

The range class is another class in which the following operations are to be performed.

- We create an object of the iterator helper class here.
- The range class must have a means by which a check on the existence of the next element can be made.
- Range must implement in a means by which value of the next existing element can be obtained.

The code that I have written would perform these similar functions but it is not in separate procedural pattern.
Here I have made a check for the next element in the array and the yield function is performed only by this such that only if there is an element in this array the thread will be created. Each of these threads is made to run 10 times for random time intervals of sleep. The code is as attached in the file.

The links and references used for the coding are as follows:

- http://www.roseindia.net/java/thread/thread-creation.shtml
- http://www.java-samples.com/showtutorial.php?tutorialid=235
- http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Iterator.html
- http://mindprod.com/jgloss/iterator.html