Final Exam Solutions – Keerthana Ravichandran

Problem 1:

A grammar for algebraic expression involving multiplication and exponentials:

We know that a grammar consists of non-terminals which is a subset of a language and terminals which are the final symbols that occur in a string. eg : x,y,z, *, ^.

A grammar can be represented using expressions(E), terms(T), factors (F) and we know that:

E -> T
T -> F
F -> (E)
F -> x
F -> y
F -> z
So an algebraic expression like a = x * (y ^ z) can be represented as:

E => E * E
  => T * E
  => F * T
  => x * F
  => x * (E)
  => x * (E ^ T)
  => x * (T ^ F)
  => x * (F ^ z)
  => x * (y ^ z)

This grammar shows exponentiation has higher precedence than multiplication and multiplication is represented as left- associative and exponentiation is right associative.

Problem 2:

First class values and need for garbage collection:

First class objects means the language supports passing functions as arguments to other functions returning them as values from other functions and assigning them to variables and strings in data structures.
Eg: map function in Haskell takes a function and list of values as parameters and applies the function to each and every element in list and gives us a list of values.

First class subroutines with nested scope has an extra complexity that a reference to a subroutine may outlive the execution of scope in which that routine was declared.
If local objects were destroyed at the end of each scope's execution, then the referencing environment captured in a long lived closure might become full of dangling references. To avoid this, local objects should have unlimited extent. So they have to be allocated on a heap.

These first class values require the ability to compute new values at run time. First class subroutines' behavior is determined dynamically and automatic garbage collection is an important service for runtime system supporting applications and first class functions require runtime interpretation of code.

Generally heap stores all methods of a class, constructors and attributes whereas stack stores all local variables, types and reference to variables in heap. If there is no garbage collection, then all local variables will be allocated in stack and removed once the method is executed. And thus does not let us use the function again once it exits.

Haskell for example has been more declarative as it has a garbage collector.
Eg: consider a function, let x = a: x in x

here x is a cyclic data structure and x is referenced by itself and never deallocated if it has no garbage collection.

In Haskell, the map function might be applied to millions of records eg the list [1..1000] must be kept in memory until the user changes the command; so the lifetime of this must be determined dynamically, and this is why dynamic memory management is necessary and hence Haskell requires a garbage collector, since garbage collection is the highest-performance automatic dynamic memory manager.

Another example in Haskell is since it has a lazy evaluation pattern for example:

```
f :: Integer -> Integer
f n = product [1..n]
```
We can allocate the list [1..n] on the stack and deallocate it after f returns, but this would change f from using O(1) memory (under garbage collection) to O(n) memory and the time required to access such a big list is also more.

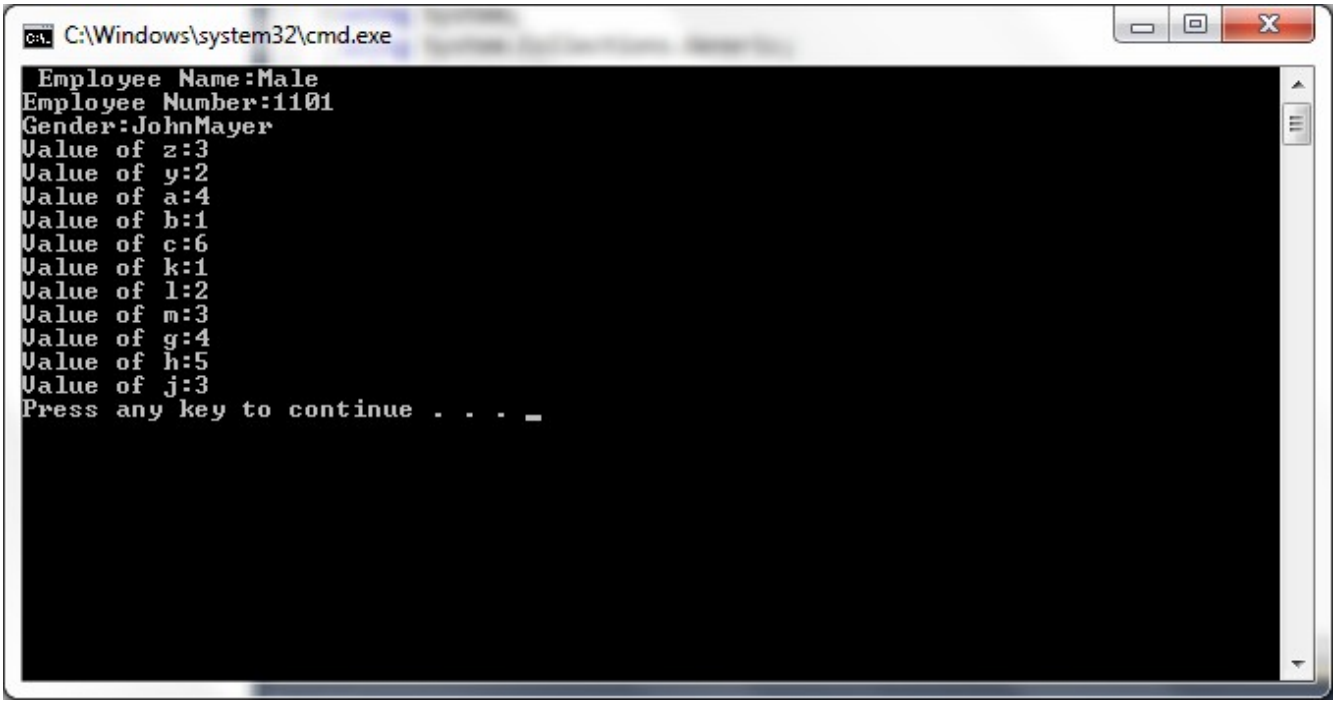
A garbage collector is never necessary, provided you have sufficient memory. However, in reality, we don't have infinite memory, and so we need some method to reclaim memory that is no longer needed.

Haskell requires automatic, dynamic memory management: automatic memory management is necessary because manual memory management is unsafe; dynamic memory management is necessary because for some programs, the lifetime of an object can only be determined at runtime and the scope need not be saved when the scope completes execution; the space can be reclaimed in dynamic memory management.

Problem 3:

Argument evaluation order:

(1) The language is chose is C#.
(2) Implementation to test: The argument evaluation order
(3) Source code is attached as a separate file.
(4) Output of the program : Screen shot attached:



(a) In the first method, setval(), the first argument is an integer and rest two are strings. When I call the method, I pass the gender value as second argument and name as the third argument. But in method declaration, I pass name as second and gender as third. So in the output employee name gets gender value and vice versa. This shows that the order of evaluation is left to right and nothing else.

(b) In the second method sample(), I declare x, y,z as arguments in same order but while calling the function, I pass 1 which goes to x by default, z and y. Since I have explicitly mentioned the name of the arguments before passing the value, the order does not matter here.

© In the third method sample2(), I declare j as 1 and pass j+3, j, j=5 as arguments. The same order is maintained and the value of first argument does not affect the value of the second argument though the same variable is used. Here too we can say that order of evaluation is left to right.

(d) In the fourth method sample3(), I declare I as 0 and pass ++i, ++i, ++i as arguments. So in the method evaluation the incremented 'i' values are passed to k,l,m as 1,2,3 i.e from left to right only. It does not take any other order.

(e) In the fifth method sample4(), like previous one I pass ++k, ++k and ++k but I explicitly mentioned which variable should take which value. So j gets the first incremented value then g and then h.

From this we can conclude that if the argument name is not mentioned explicitly, then the default order of evaluation is from left to right. But if the variable arguments are mentioned in the declaration of the method, then it takes those values and does not evaluate from left to right. The arguments are still evaluated in the order they're written .

The index expressions of argument list are evaluated from left to right. But if the evaluation of index expression causes an exception, then no further evaluations are executed.

I checked the documentation of C# and it says that the default order is left to right.

Link to documentation :
<<http://blogs.msdn.com/b/oldnewthing/archive/2007/08/14/4374222.aspx >>

Problem 4:

Source code file attached.

In this program, I first created a final1 class which creates an object for Range class and the thread execution for that starts.
The range class creates an object for iterator helper class and thus both class work in a synchronized manner.
Since range class implements Runnable interface, first the run() method in range class is executed. It calls the yield method in IteratorHelper class. It just gets the value of the for loop in run() method and prints it. I have tried here to use wait() and notify(). If the system waits for along time, it throws an exception which is captured and appropriate error message is printed out. So it works like Range class generates some set of numbers and IteratorHelper class meanwhile gets the value and prints it.

Inside yield( ),wait( ) is called. This causes its execution to suspend until the Range notifies you that some data is ready. When this happens, execution inside yield( ) resumes. After the data has been obtained, yield( ) calls notify( ). This tells that it is okay to put more data in the queue. Inside get( ), wait( ) suspends execution until the IteratorHelper has removed the item from the queue. When execution resumes, the next item of data is put in the queue, and notify( ) is called. This tells the IteratorHelper that it should now remove it.

I am not sure about done() method. So I am setting it like if the for loop in range class does not return any integer value, then it returns false and there are no more values to read and print.