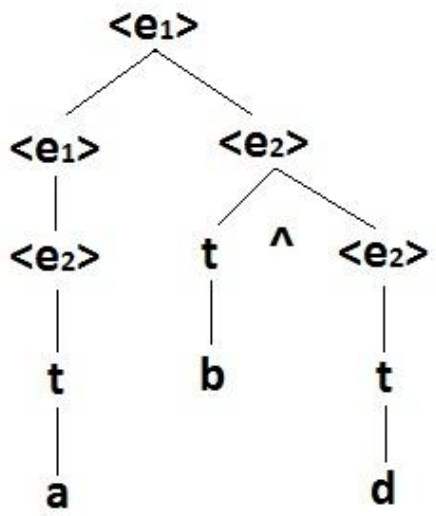
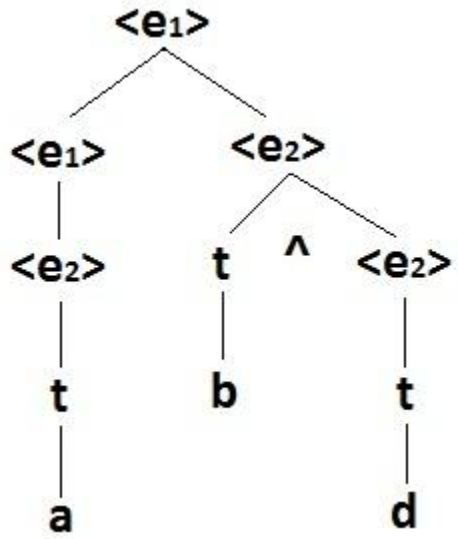
CptS 580 Final by Neeharika Vempally

Amount of time spent:
4.5days (mostly for the 4th question), a few hrs for the others.

1. The grammar is as follows:
   $<e_1> \rightarrow <e_2> \mid <e_1> <e_2>$
   $<e_2> \rightarrow <t> \mid <t> \text{ ^ } <e_2>$
   Tree: ab^d



a^bd:



CptS 580 Final by Neeharika Vempally

Amount of time spent: 4.5days (mostly for the 4th question), a few hrs for the others.

2. **A small introduction to Garbage Collection (Automatic and Manual) and first class functions:**

First class functions: First class functions can be treated as values, they can be assigned into a variable, Store functions in collections, Use functions as return values of other functions etc. First class functions differ from closure support. For example in a language like C, first class functions are supported through pointers. But C does not support any notion of nested function, and hence does not support closures as well.

For example in F#: Like how we can name integers, strings and other built-in types, we can name first class functions. This is called "binding" an identifier to a value. F# uses let expressions to do this which has the syntax of let <identifier> = <value>. (ex: let num = 4)

We can name a function by using: let squareIt2 n = n * n (squareIt2 is the function)

Garbage Collector: A garbage collector reclaims memory that is no longer in use by the program. The basic principles of garbage collection are to find data objects in a program that cannot be accessed in the future and reclaim the resources used by those objects. Many languages such as Java and C# implement garbage collection as part of the language specification or for practical implementation (formal languages). Others are designed for use with manual memory management. Manual Memory management refers to using of manual instructions to identify and deallocate memory. Manual memory management could cause bugs such as memory leak if used incorrectly. But it does offer superior performance.

Values returned from functions may contain references to other values that were created within the same function or might've been passed as a parameter, or referenced by something. In C, if programmers don't get things right it could cause dangling pointers.

Functional programming style is built around closure, lambdas and other functions, therefore, without a Garbage collector you can't use closures as freely as you could with a Garbage collector.

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.

For ex:

CptS 580 Final by Neeharika Vempally

Amount of time spent:
4.5days (mostly for the 4th question), a few hrs for the others.

```
main = loop (Just [1..1000]) where
 loop :: Maybe [Int] -> IO ()
 loop obj = do
   print obj
   resp <- getLine
   if resp == "clear"
    then loop Nothing
    else loop obj
```

The list [1…1000] must be kept in memory until the user types “clear” so the lifetime of this must be determined dynamically, and this is where dynamic memory management comes into picture. Garbage collection is necessary since it is the highest performance automatic dynamic memory manager. The standard implementation techniques applied to Haskell actually require a GC more than most other languages, since they never mutate previous values, instead creating new, modified values based on the previous ones. Since this means the program is constantly allocating and using more memory, a large number of the values will be discarded as time goes on.

CptS 580 Final by Neeharika Vempally

Amount of time spent:
4.5days (mostly for the 4th question), a few hrs for the others.

3. The language that I used was C. Given the function f(a, b, c). From the main function call f(a++, a++, a++). Now the smallest number of a, b, c will tell us which was evaluated 1st and the next largest was evaluated 2nd and the largest of the 3 was evaluated last.

This program prints out the evaluation order as Right to Left.
But according to C documentation: The function parameter evaluation is undefined. It varies from compiler to compiler.

```
int i = 0;
printf( "%d   %d   %d\n", i++, i++, i++ );
```

produces the following output on GCC 4.01 (i386) and Visual Studio 2003:

2 1 0 (right to left)

However, GCC 4.01 (PPC) produces:

0 1 2 (left to right)

Source:
http://www.eetimes.com/discussion/programming-pointers/4023961/Evaluating-Function-Arguments
http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V40F_HTML/AQTLTBTE/DOCU_056.HTM
http://orangejuiceliberationfront.com/funny-thing-about-c-parameter-evaluation-order/

CptS 580 Final by Neeharika Vempally

Amount of time spent:
4.5days (mostly for the 4th question), a few hrs for the others.

4. Need to create an iterator for a list of items.

Any class that implements a runnable has to override a run() method.

the get() method will be the one that has the preceding 'synchronized' keyword

- Java Thread yield is a static method
- It works on same priority threads
- It makes same priority thread from running to runnable state.
- Then other same priority thread comes to running state from runnable state

Multithreading is used to replace event loop programming by dividing the tasks into discrete and logical units. Treads also do away with polling. Polling is usually implemented by a loop that is used to check some condition repeatedly. Once the condition is true, appropriate action is taken. This wastes CPU time.
To avoid polling, Java includes an interprocess communication mechanism via the **wait( )**, **notify( )**, and **notifyAll( )** methods. These methods are implemented as **final** methods in **Object**, so all classes have them. All three methods can be called only from within a **synchronized** method.

- **wait( )** tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls **notify( )**.
- **notify( )** wakes up the first thread that called **wait( )** on the same object.
- **notifyAll( )** wakes up all the threads that called **wait( )** on the same object. The highest priority thread will run first.

Referenced: http://www.oracle.com/technetwork/java/javase/documentation/index.html