Introduction to *gforth* - DUE 01/15/14 (11:59pm)

Introduction

The output of our compiler will be the input to the "machine." In *C,* the compiler generates an object file, the familiar .o file, that is then **linked** into the *a.out* file. To actually run the program you must still cause the operating system to **load** *a.out* file into memory for the machine to **execute.**

For our project, we will instead use the program *gforth. gforth* is almost completely opposite of *Lisp.* The syntactic format for *gforth* is **postorder.** While *Lisp* has some syntax, *gforth* only has spelling rules. As an example,

our 1+2 expression in *C* would be entered as

1 2 +

in *gforth.*

Objective

Objective 1 is to introduce you to *gforth. Gforth* will be the machine that we code to.

Objective 2 is to emphasize the crucial role of generalized trees and generalized postorder traversal.

Objective 3 is to get you to formulate a generalized expression tree data structure.

Professional Methods and Values

The professional will learn to use many programming languages and paradigms throughout her/his professional career. The hallmark of the professional in this venue is the ability to quickly master a new programming language or paradigm and to relate the new to the old.

Assignment

The below exercises are simple "Hello World" type exercises. These short program segments or small programs are designed to get you to understand how to learn a new language: you pretty much just sit down and try some standard things that you already know must work.

Performance Objectives

In this milestone, you have several clear performance objectives.

1. Learn to run *gforth* on either a Departmental machine or how to install and use *gforth* on your own

machine.
2. Learn the simple programming style of *gforth*. Documentation/Download information is given on the links page.
3. Translate a the infix style expressions in the Initial Forth Exercises 1 - 11 to an expression tree. The output here is a drawn tree.
   You do not need to include the printf statements as part of your tree. The printf is just to make sure you know to print the answer, not just calculate it.
4. Do a postorder traversal of the expression tree to generate the *gforth* input. The output of this step is *gforth* code.
5. Produce running gforth code that evalutes the programs equivalent to initial exercises. The output here is the running of the *gforth* code.

Milestone Report

Your milestone report will include hand written answers to 3 and 4 above.

In addition, your milestone report must include a data structure for an *n*-ary tree and a pseudo-code recursive algorithm to translate an arbitrary instance of these trees into postorder.

Use *handin* on the TEACH website to submit the gforth input file, makefile, and milestone report. We will generate the output file.
Click on these links to see a template for the Milestone Report and Makefile we will use in this class.

Initial Forth Exercises

1. printf("Hello World\n");
2. Bonus 5 pts: print only a certain number of characters from the string…
3. 16 / 32 + 74 * 16 ^ 3 + 5 % 10
4. 16.0 / 32.0 + 74.0 * 16.0 ^ 3.0 + 5 % 10
5. 16.0e0 / 32.0e0 + 74.0e0 * 16.0e0 ^ 3.0e0 + 5 % 10
6. 16 / 32.0 + 74.0 * 16 ^ 3 + 5 % 10
7. y = 16;
   x = 32.0e0;
   y + x - 3.0e0 * 6 / 10.0
8. if 5 < 3 then 7 else 2
9. if 5 > 3 then 7 else 2
10. for (i = 0; i <= 5; i++)
        printf("%d ", i);
11. double convertint(int x)
    { return ((double)x); }
12. int fact(int i)
    {
      if (i <= 0) return 1;
      else return i*fact(i-1);
    }
13. int fib(int i)

```
{
   if(i == 0) return 0;
   else if(i == 1) return 1;
   else return fib(i-1) + fib(i-2);
}
```