Jason Cross

Milestone 1 Report

The biggest and most obvious purpose of this milestone was to get us working with gforth and to start becoming familiar with it so that we can eventually use it as the output language of our compilers. Also, we've primarily only worked with infix languages during in all of our previous CS classes, so this is exposing us to a postfix language for the first time. Through making the trees, I think we were meant to analyze various ways that languages can be parsed and how this parsing can affect the meaning of the code/language being read in as well as give us practice generating these expression trees and ways they can be traversed.

Since these problems weren't the most complex, a lot of the problem solving was learning how common syntax and semantics in C were expressed in Gforth. Aside from that I tried to parse the input and structure my trees in a manner that made sense to me and could easily align with the Gforth syntax. Along with that, I had to be aware of the data types of the values being input and how/when they need to be expressed or converted to be properly handled by Gforth.

The primary testing that I did was printing out all of the output that my code got and comparing it to the answers I got doing the problems out manually. The last 3 problems (10, 11, and 12) had the possibility of outputting different answers depending on the input. For these I had the stutest file plug in different values to help ensure the code was correct.

I learned a good bit of Gforth syntax and how that language structures its functions and loops and how important the integer and float stacks are when working in Gforth. This assignment was also a useful refresher on constructing and traversing different kinds of syntax trees. Although it didn't enforce too much as to why the timing of converting integers to floats was important, as part of the assignment guidelines I had to be aware of when the proper time to move integers to the float stack.

A Data Structure for n-ary tree:

```
struct Node {
	char[] data;
	Node *child[N];
}
```

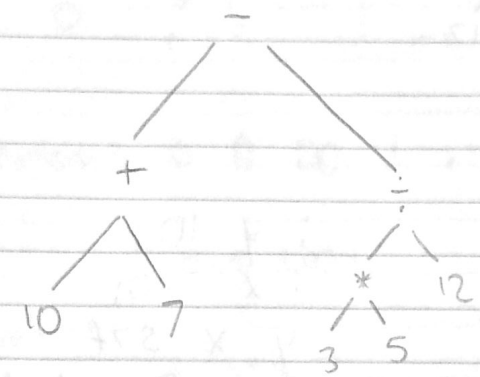
Pseudo Code to traverse in to postorder:

```
void traverse(Node *root) {
	Print data
	for(i = 0; i < N; i++) {
		traverse(root->child[i]);
	}
}
```

1) only a print so no tree needed

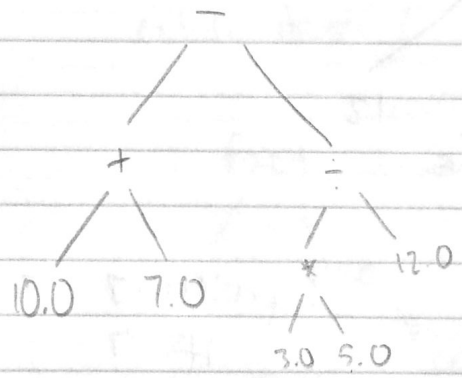
cr." Hello World"

2)



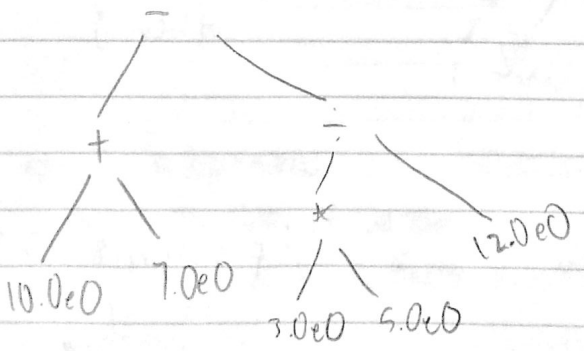
10 7 + 35 * 12 ÷ - .

3)



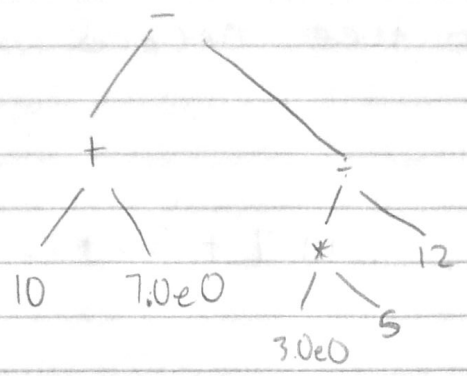
10.0e 7.0e f+ 3.0e 5.0e f* 12.0e f÷ f- f.

4)



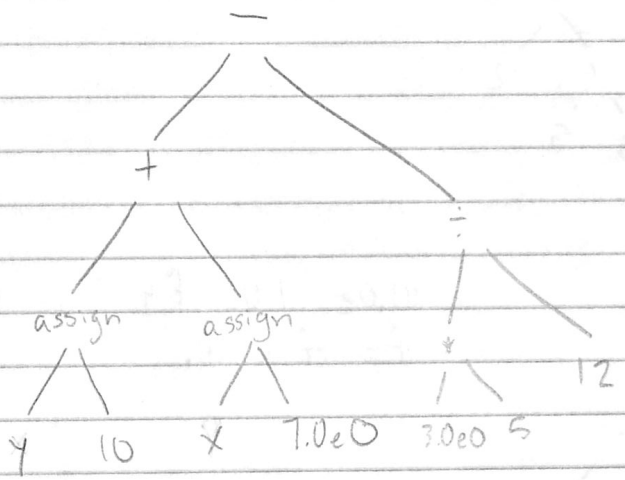
1e1 7e0 f+ 3e0 5e0 f* 1.2e1 f÷ f- f.

5)



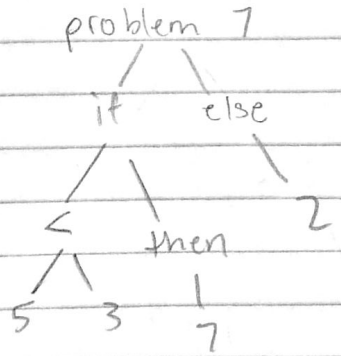

```
10 7e0 s>f fswap f+
3e0 5 s>f f* 12 s>f
f÷ f- f.
```

6)



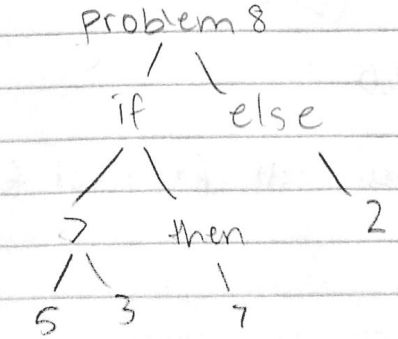

```
: y 10;
: x 7e0;
y x s>f fswap f+
3e0 5 s>f f* 12 s>f
f÷ f- f.
```

7)



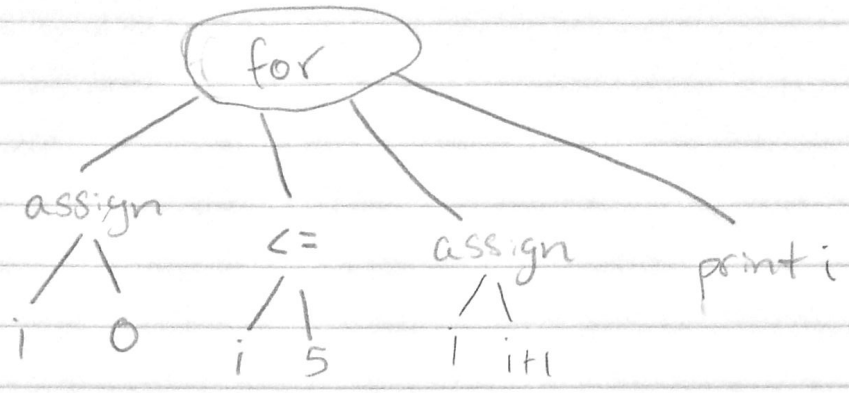

```
: problem7  5 3 <
    if 7
    else 2
    then ;
```

8)




```
: problem8  5 3 >
    if 7
    else 2
    then ;
```

9)



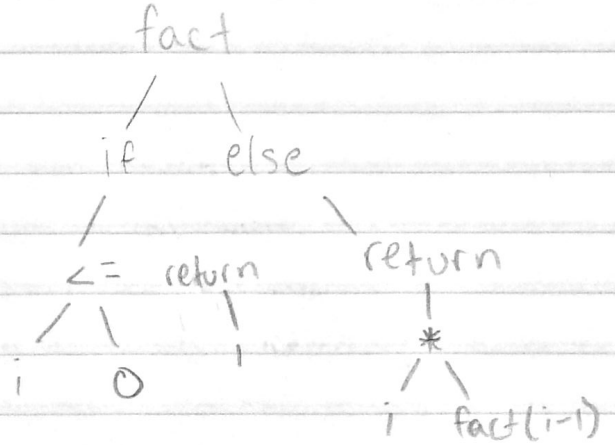

```
: problem9 6 0 DO I . LOOP ;
```

10)

```
convertint
    |
  int x
    |
return double x
```

```
: convertint
  s>f ;
```

11)




```
: fact recursive
  dup 0 <= if drop 1
  else dup 1 - fact *
  then;
```