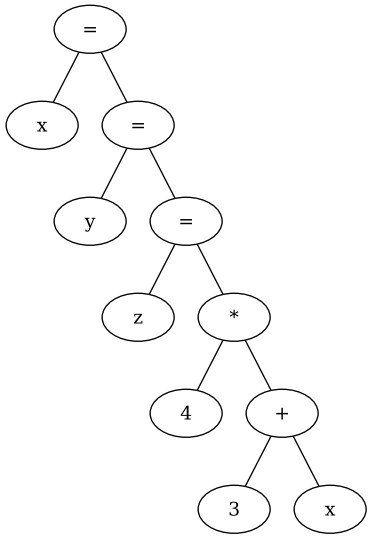
All students should be sure they've completed the reading for `lga-register-allocation.pdf` **for this assignment** and the next lecture.

Distribute the following questions across the members of your group. You will share your solutions (and most importantly the *method* of your solutions) during the next lecture period. Divide up the questions so that **each** question has at least two solutions from different group members.

1. Change REGISTERNEEDS and TREECG (either as presented in lecture or the text's version) so that it treats assignment (=) as a standard binary operator. Be sure your logic supports multiple assignment such as

```
x = y = z = 4*(3+x)
```



2. If the CPU has only a JUMP instruction and no dedicated Return Address register, how can **caller/callee** protocol presented `in lecture` be modified to support function calls. Be explicit in your changes, describing the specific machine instructions that should be executed, when, and by which entity.
3. Using the `pretest loop` AST structure in figure 7.15 of the text, suppose the predicate takes seven instructions for the truth value to be calculated, and the loop body requires 31 instructions. How are these instruction blocks, a test instruction such as `IFZERO R3,@` or `IFNOTZERO R3,@`, and a `JUMP @` instruction sequenced in memory for the control structure to run correctly? Specify the addresses (`@`s) in `(PC,#)` (program counter register-offset form).

4. Recall that the `treeCG` algorithm has the convenient property of always storing the calculated expression result in $head(regList)$; this is called **register targeting**. `lga-register-allocation.pdf` asked (q1) how this algorithm property can be used to the compiler writer's advantage when an expression contains a function call. Review this solution from your group, and then consider if there are any pitfalls when the compiled source looks like

```
int bar( int q, int r, int s );
int foo( int a, int b, int c, int d )
{
    ⋮
    int l = bar( b+c*d, a-d*c, c*d-a );
    ⋮
    return x;
}
```

   and the architecture specifies the first two integer arguments of functions are passed in registers `R1` and `R2`.

5. Page 540, question 10.

   For some clarity, at the right is an example sequence of TREECG instructions for the sum

   $$z = a + b + c + d + e$$

   and $regList = \{R1, R3, R5\}$.

   Notice how there are now instructions issued (executed) at times 2, 5, 8 or 11. This is because the LOADs have stalled waiting for data to arrive from the cache.

Instruction	Issued	Ready
LOAD R1,@a	0	2
LOAD R3,@b	1	3
ADD R1,R1,R3	3	
LOAD R3,@c	4	6
ADD R1,R1,R3	6	
LOAD R3,@d	7	9
ADD R1,R1,R3	9	
LOAD R3,@e	10	12
ADD R1,R1,R3	12	
STORE R1,@z	13	···

   Show how $R5$ can be incorporated into this same pattern of LOADS and ADDs to arrive at the same result in z sooner.