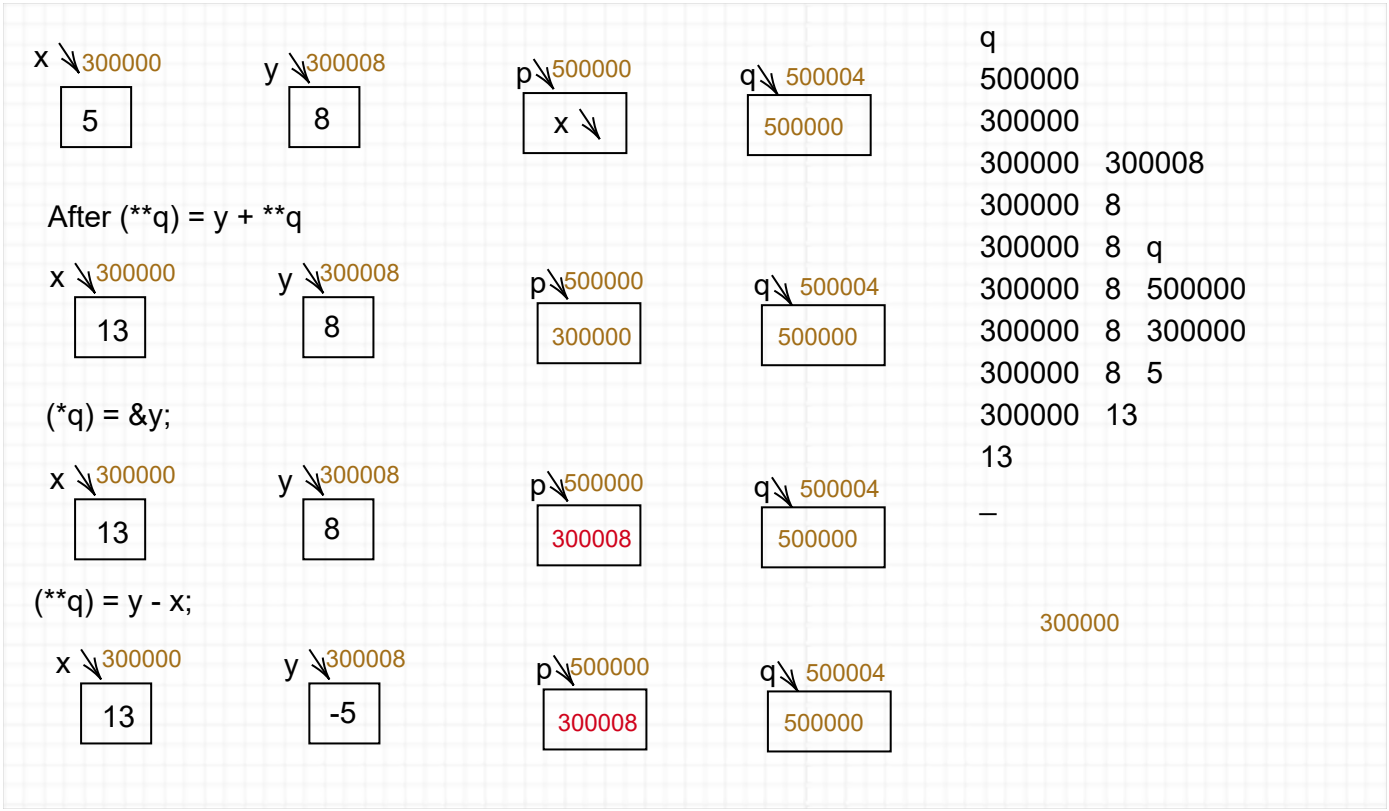
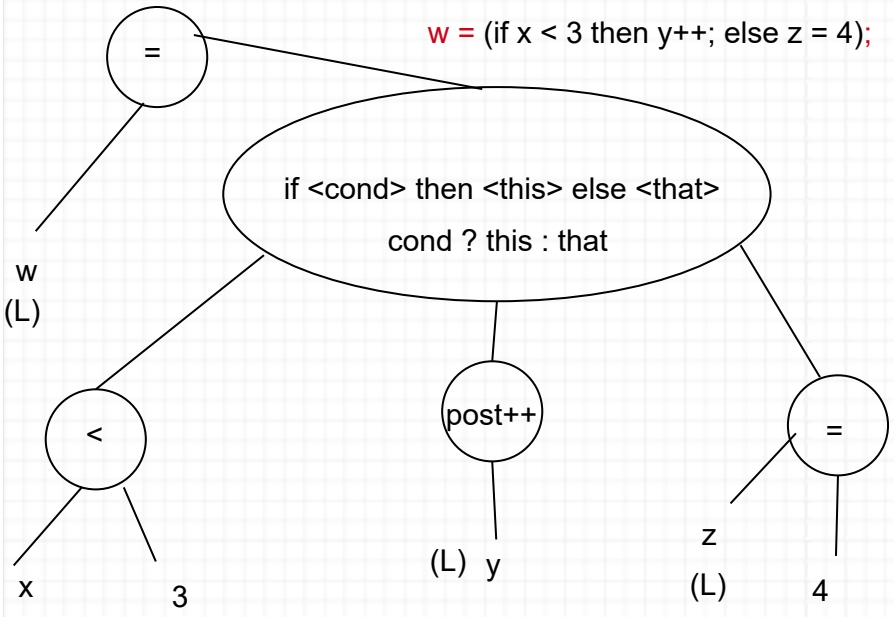
Office Hour and Lec/Rec Examples (may be added to)




```
void main() {
    int x, y, *p, **q;
    x = 5;              x 5 store pop
    y = x+3;            y x fetch 3 + store pop;
    p = &x;             p x store pop;
    q = &p;             q p store pop;
    (**q) = y + **q;  q fetch fetch  y fetch q fetch fetch fetch +    store pop
    (*q) = &y;
    (**q) = y- x;
    printf("Final value of y is %d\n",y);
}
```



w = (if x < 3 then y++; else z = 4);

```
w
w x
w 7
w 7 3
w false
w false y
w false 13
w false 13 y
w false 13 y y
w false 13 y 13
w false 13 y 13 1
w false 13 y 14
w false 13 14
w false 13
w false 13 z
w false 13 z 4
w false 13 4
w 4
4
```

Say x=7, y=13, z=4.

y 300008: 13

y 300008: 14

z 300088: 4

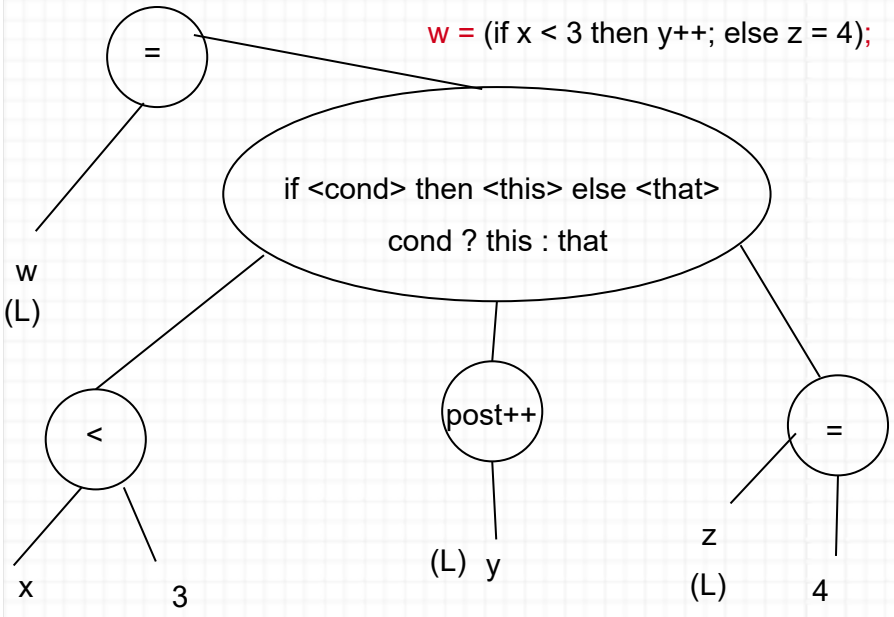
w 299992: 4

Unannotated Postfix traversal: w x 3 < y post++ z 4 = (? :) =

Annotated: (LVar w) (RVar x) 3 < (LVar y) post++ (LVar z) 4 = (? :) =

w x fetch 3 < y fetch y y fetch 1 + store pop z 4 store if-then-else store pop

Same thing with x initialized to 2 instead of 7:



w = (if x < 3 then y++; else z = 4);

```
w
w x
w 2
w 2 3
w true
w true y
w true 13
w true 13 y
w true 13 y y
w true 13 y 13
w true 13 y 13 1
w true 13 y 14
w true 13 14
w true 13
w true 13 z
w true 13 z 4
w true 13 4
w 13
13
```

Say x=2, y=13, z=4.

y 300008: 13

y 300008: 14

z 300088: 4

w 299992: 13

Unannotated Postfix traversal: w x 3 < y post++ z 4 = (? :) =

Annotated: (LVar w) (RVar x) 3 < (LVar y) post++ (LVar z) 4 = (? :) =

w x fetch 3 < y fetch y y fetch 1 + store pop z 4 store if-then-else store pop

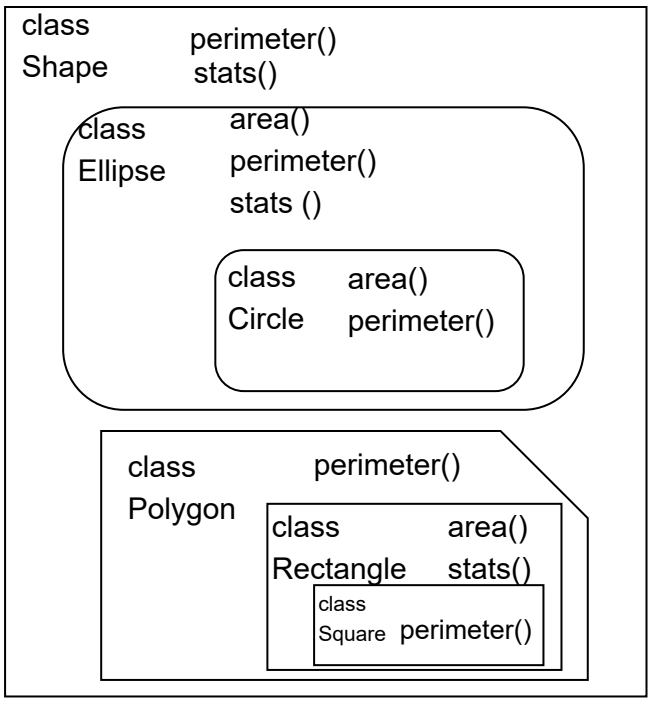
Relating to a lecture example:

w false 8
w false 8 z
w false 8 z 4
w false 8 4
w 4
4
–

w
w x
w 5 3
w false
w false y
w false 8
w false 8 y y
w false 8 y 8 1
w flase 8 y 9
w false 8 9
w false 8

Inheritance/Overriding in a Class Hierarchy vis-a-vis Nested Scopes:

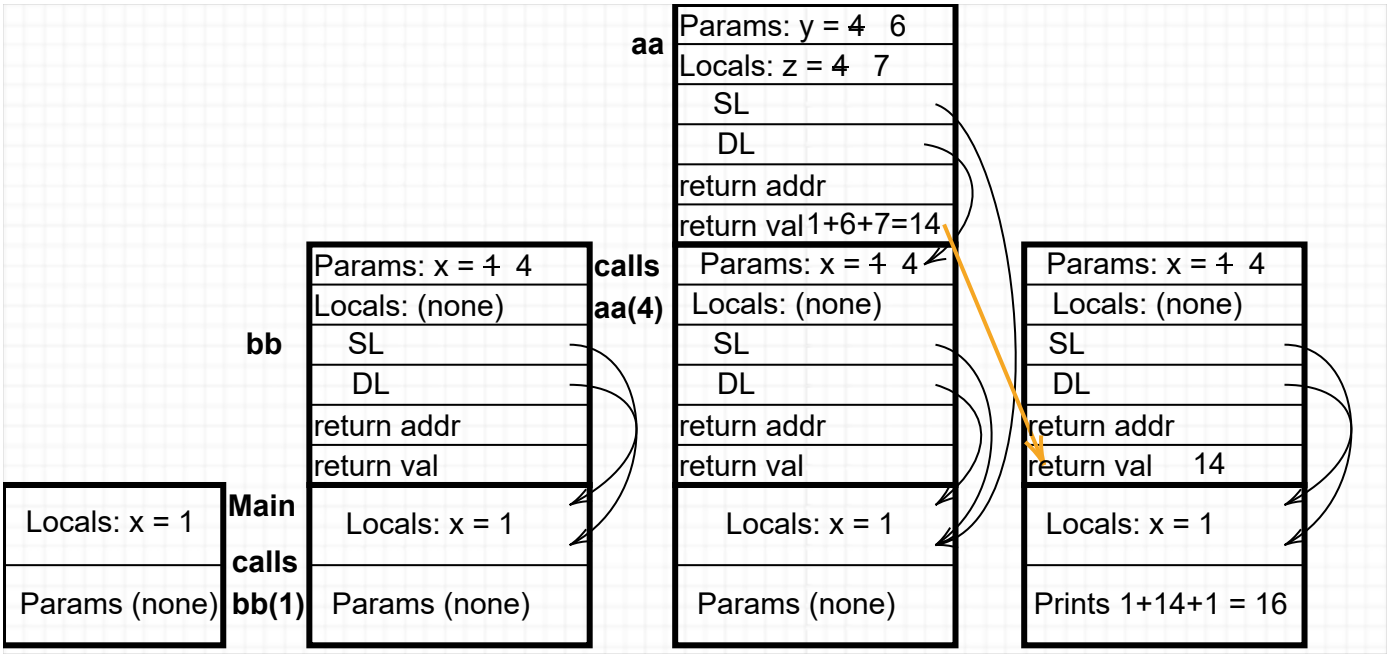
```
class Shape {
   internal Shape(...) {...}    //"internal" ≡ default scope in Java
   internal /*virtual*/ float Perimeter() {...}
   internal /*virtual*/ void Stats() {
      Console.WriteLine("Perimeter is: " + Perimeter());
   }
}
class Ellipse : Shape {
   internal Ellipse(...) {...}
   internal /*virtual*/ float Area() {...}
   internal /*override*/ float Perimeter() {...}
   internal /*override*/ void Stats() {
      Console.WriteLine("Perimeter is: " + Perimeter());
      Console.WriteLine("Area is: " + Area());
   }
}

class Circle : Ellipse {
   internal Circle(...) {...}
   internal /*override*/ float Area() {...}
   internal /*override*/ float Perimeter() {...}
}
class Polygon : Shape {
   internal Polygon(...) {...}
   internal /*override*/ float Perimeter() {...}
}
class Rectangle : Polygon {
   internal Rectangle(...) {...}
   internal /*override*/ float Area() {...}
   internal /*override*/ void Stats() {
      Console.WriteLine("Perimeter is: " + Perimeter());
      Console.WriteLine("Area is: " + Area());
   }
}
class Square : Rectangle {
   internal Square(...) {...}
   internal /*override*/ float Perimeter() {...}
}
```



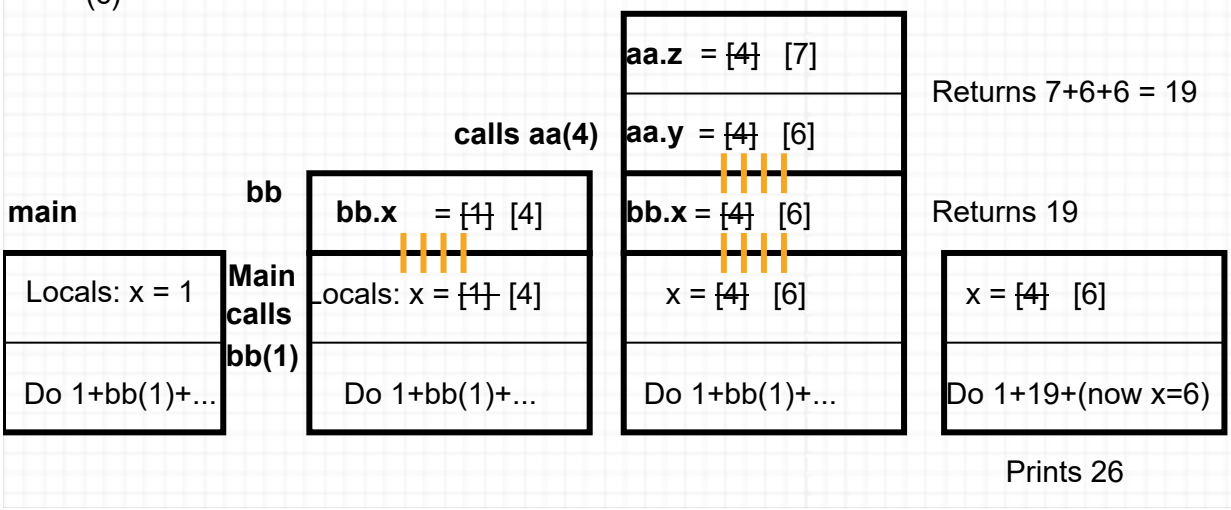
Main difference from static scoping is that e.g. if an `Ellipse` variable `e` makes a call `e.area()` when it is holding a `Circle` object, the `Circle` version of `area()` will be called---even if the code of the call is in class `Ellipse`. (This presumes all the methods are virtual, as in Java.)

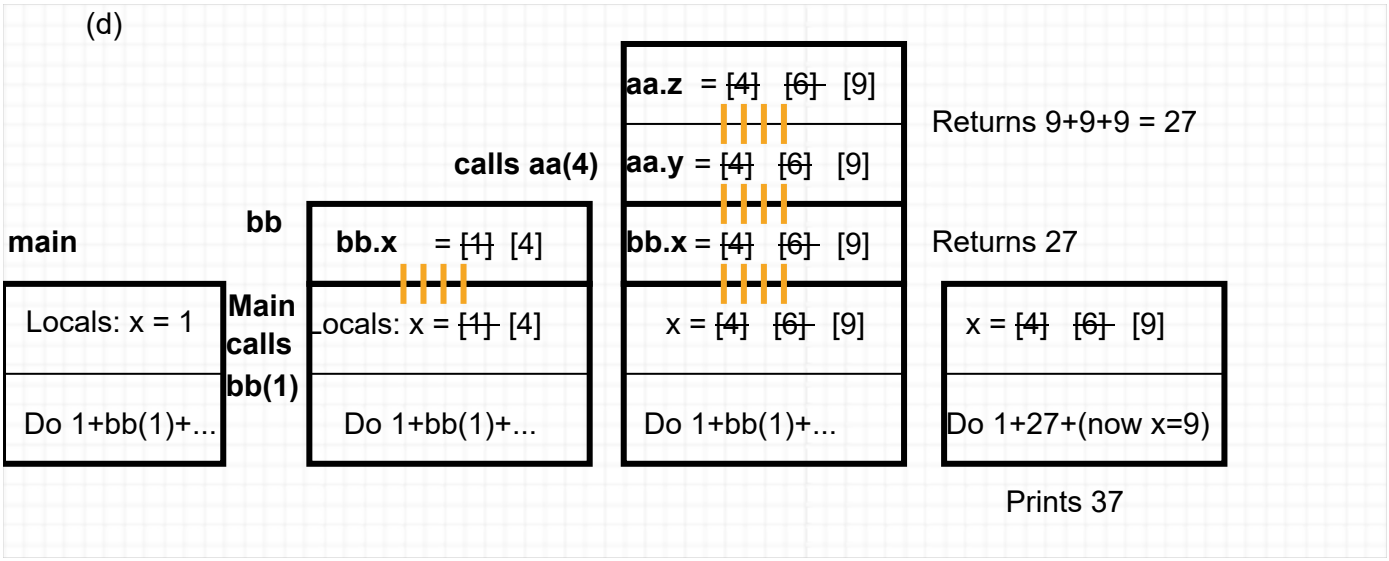
Diagrams in the Prelim II key, plus an extra on the original typo in part 1(d).

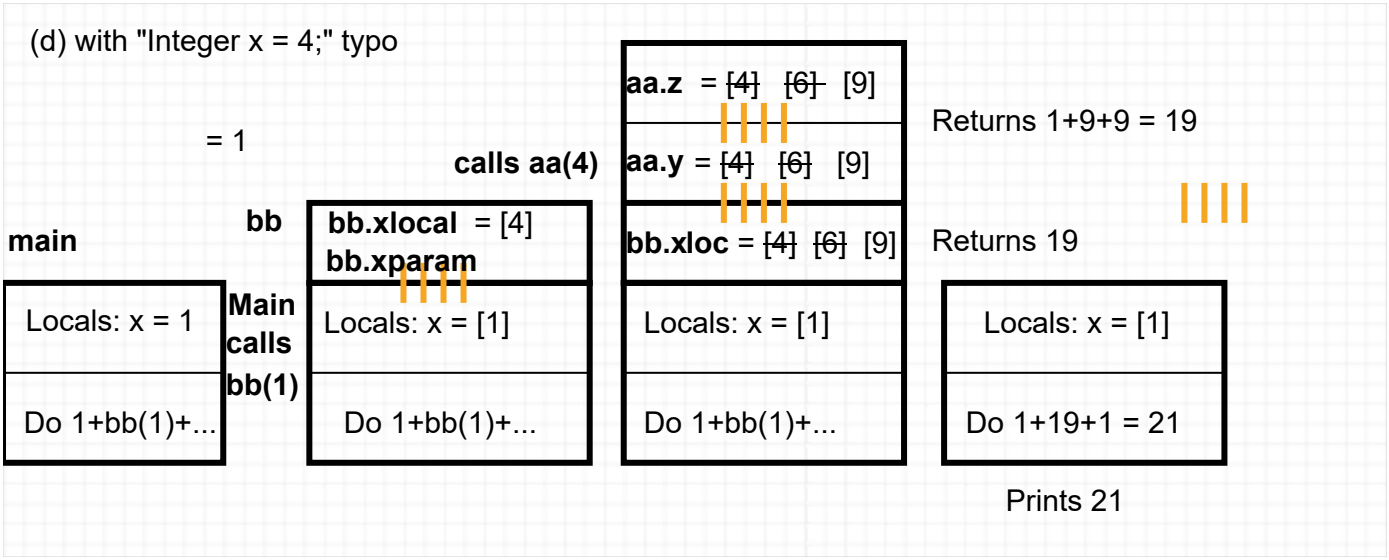
```
void main() {
   int x = 1;
   int aa(int y) {
      int z = y;
      y += 2;
      z += 3;
      return x+y+z;
   }
   int bb(int x) {
      x = 4;
      return aa(x);
   }
   /* body of main is here */
   print(x + bb(x) + x);
}
```


aa
Params: y = 4 6
Locals: z = 4 7
SL
DL
return addr
return val 1+6+7=14
bb
Params: x = 1 4
Locals: (none)
SL
DL
return addr
return val
calls aa(4)
Params: x = 1 4
Locals: (none)
SL
DL
return addr
return val
Params: x = 1 4
Locals: (none)
SL
DL
return addr
return val 14
Main
Locals: x = 1
calls bb(1)
Params (none)
Locals: x = 1
Params (none)
Locals: x = 1
Params (none)
Locals: x = 1
Prints 1+14+1 = 16

(c)

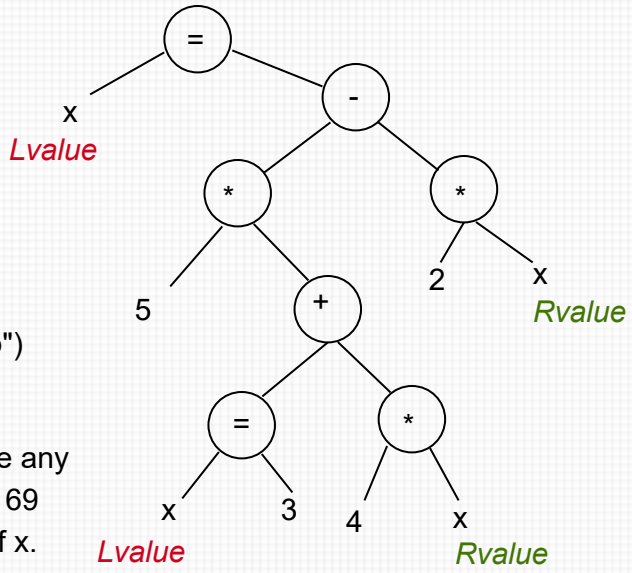

aa.z = [4] [7]
Returns 7+6+6 = 19
calls aa(4)
aa.y = [4] [6]
bb
main
bb.x = [1] [4]
bb.x = [4] [6]
Returns 19
Main
Locals: x = 1
Locals: x = [1] [4]
x = [4] [6]
x = [4] [6]
calls bb(1)
Do 1+bb(1)+...
Do 1+bb(1)+...
Do 1+bb(1)+...
Do 1+19+(now x=6)
Prints 26


(d)
main
Locals: x = 1
Do 1+bb(1)+...
Main calls bb(1)
bb
calls aa(4)
bb.x = [1] [4]
Locals: x = [1] [4]
Do 1+bb(1)+...
aa.z = [4] [6] [9]
aa.y = [4] [6] [9]
bb.x = [4] [6] [9]
x = [4] [6] [9]
Do 1+bb(1)+...
Returns 9+9+9 = 27
Returns 27
x = [4] [6] [9]
Do 1+27+(now x=9)
Prints 37


(d) with "Integer x = 4;" typo
= 1
main
Locals: x = 1
Do 1+bb(1)+...
Main calls bb(1)
bb
calls aa(4)
bb.xlocal = [4]
bb.xparam
Locals: x = [1]
Do 1+bb(1)+...
aa.z = [4] [6] [9]
aa.y = [4] [6] [9]
bb.xloc = [4] [6] [9]
Locals: x = [1]
Do 1+bb(1)+...
Returns 1+9+9 = 19
Returns 19
Locals: x = [1]
Do 1+19+1 = 21
Prints 21

(a) Expression Tree for

x = 5*((x = 3) + 4*x) - 2*x;



(b)

L-to-R Postorder Traversal;
only *Rvalue* adds a "fetch":

x 5 x 3 store 4 x fetch * + *
2 x fetch * - store (; adds "pop")

(c)

In this order, 3 is stored to x before any fetch, so value 5*(3+4*3) - (2*3) = 69 does not depend on initial value of x.