Q1

We eliminate the tail recursion function call by converting it into a simple jump back to the beginning of the function.

We can only do such an optimisation when the recursive call is in tail-call position and should be the last instruction within the function.

This optimisation could save space in the stack and avoid the stack overflow for a large number of calls in depth.

Q2

1 + ((2 * 3) + (4 - 3))

Q3

ldc 3 → load constant 3

iload 3 → load the 4th variable assigned

istore 1 → save the top of the stack into the 2nd variable

ifeq label → single stack operand! if equal to 0 then make jump to else branch. (Rmr opposite)

if_icmpge label → if the 2 values on the stack are ≥ then jump to "label". (used for opposite <)

Q4

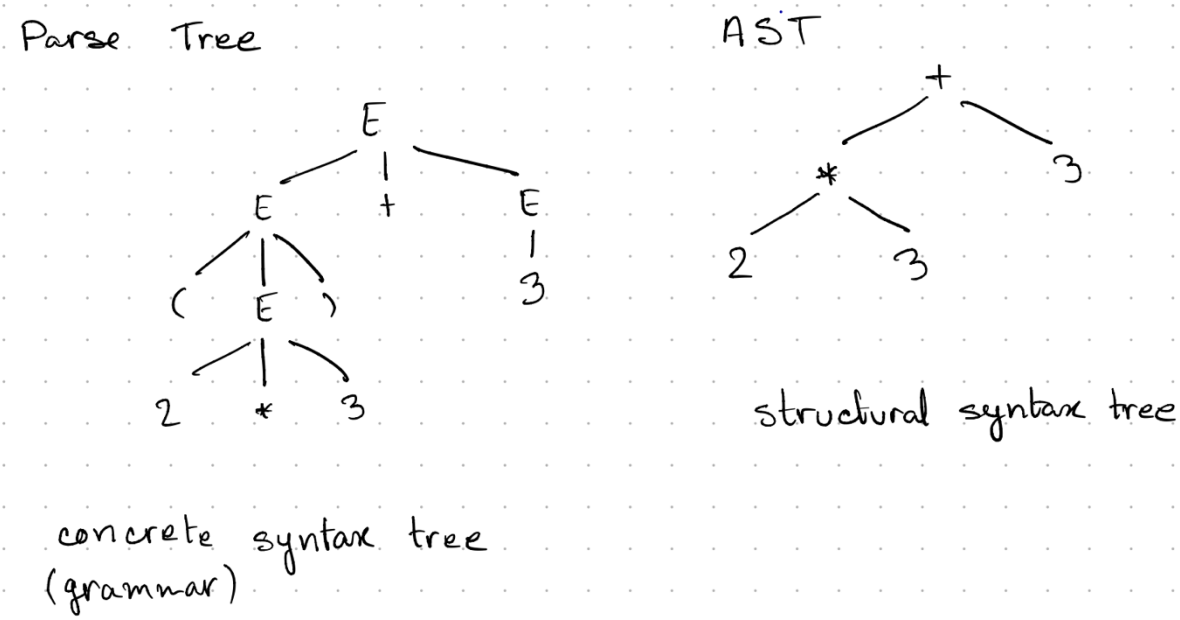
```
bar (int: x) {
    if (x == 0) then { return 0}
    else {
        if (x == 1) then { return 1}
        else { return bar(x-1) + bar (x-2) }
    }
}
```

Q5

((a * a) + ((a * 2) * b)) + ((a * 2) * b)

Q6

The AST contains only the essential information about the program constructs (the tree for generating code), while the parse-tree records how the input string is parsed.



Q7

matcher r s def = nullable(ders s r). The Brzozowski matcher will do the derivative to each string according to the regular expression described and then distinguish whether the result is an empty string, if yes, then match, otherwise false.

Q8

Question 8:

You need to explain how & what the compile functions in the compiler work.

Talk about: compile_aexp
compile_stmt
compile_bexp

Each taking a Aexp/Stmt/Bexp as argument. More interesting is the second argument which is the environment. The environment records the index (memory location) for each variable (where the value needs to be stored and read from).

- Compile_aexp: traverses the tree and generates the instructions in post_order fashion. It doesn't change the environment, since no new variable is created by Aexp.
- Compile_bexp: works similar, but introduces jumps when the condition is not true. It then jumps to the given label.
- Compile_stmt: changes the environment so it needs to be carefully threaded through the recursion. In the assignment you need to check whether the variable already exists or a new index needs to be created.