

1. [15] Assuming that \blacktriangle is a right-associative operator having precedence 1 and that ∇ is left associative and has precedence 2 (i.e., higher than 1), translate the following infix expression

$a \blacktriangle b \nabla c \nabla d \blacktriangle e \blacktriangle f$

into

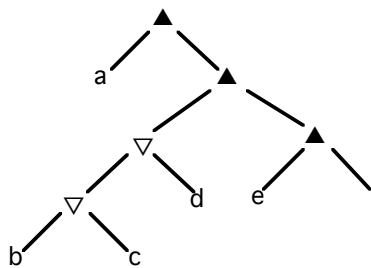
- a. prefix notation

$(a \blacktriangle (((b \nabla c) \nabla d) \blacktriangle (e \blacktriangle f)))$
 $(\blacktriangle a (\blacktriangle (\nabla (\nabla b c) d) (\blacktriangle e f)))$
 $\blacktriangle a \blacktriangle \nabla \nabla b c d \blacktriangle e f$

- b. postfix notation

$(a \blacktriangle (((b \nabla c) \nabla d) \blacktriangle (e \blacktriangle f)))$
 $(a ((b c \nabla) d \nabla) (e f \blacktriangle) \blacktriangle) \blacktriangle$
 $a b c \nabla d \nabla e f \blacktriangle \blacktriangle \blacktriangle$

- c. an abstract syntax tree



2. [15] Given the following BNF grammar:

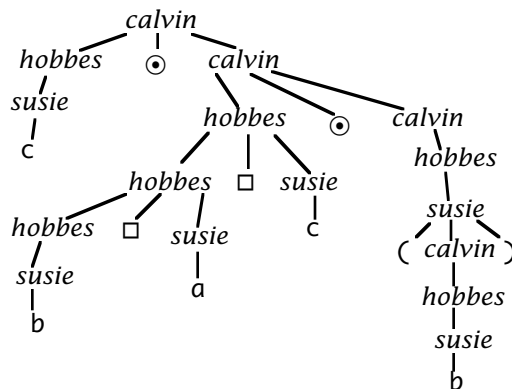
$\langle \text{calvin} \rangle ::= \langle \text{hobbes} \rangle \odot \langle \text{calvin} \rangle \mid \langle \text{hobbes} \rangle$

$\langle \text{hobbes} \rangle ::= \langle \text{susie} \rangle \mid \langle \text{hobbes} \rangle \square \langle \text{susie} \rangle$

$\langle \text{susie} \rangle ::= a \mid b \mid c \mid (\langle \text{calvin} \rangle)$

- a. give a parse tree for the following string:

$c \odot b \square a \square c \odot (b)$



- b. circle the correct entries in the following table:

<u>operator</u>	<u>associativity</u>	<u>precedence</u>
\odot	left <input checked="" type="checkbox"/> right neither	<input checked="" type="checkbox"/> lower higher same
\square	<input checked="" type="checkbox"/> left right neither	lower <input checked="" type="checkbox"/> higher same

3. [15] Show the steps in the evaluation of the following expression

$(\backslash x y z \rightarrow \text{if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1) 2 (2 + 8 - 5) (6 + 7)$

using each of the following evaluation strategies (assume all operators associate to the left):

a. leftmost innermost

$(\backslash x y z \rightarrow \text{if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1) 2 (2 + 8 - 5) (6 + 7)$
 $\rightsquigarrow (\backslash x y z \rightarrow \text{if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1) 2 (10 - 5) (6 + 7)$
 $\rightsquigarrow (\backslash x y z \rightarrow \text{if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1) 2 5 (6 + 7)$
 $\rightsquigarrow (\backslash x y z \rightarrow \text{if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1) 2 5 13$
 $\rightsquigarrow \text{if } 2 > 0 \text{ then } 5 * 5 + 2 * 5 + 1 \text{ else } 13 - 1$
 $\rightsquigarrow \text{if True then } 5 * 5 + 2 * 5 + 1 \text{ else } 13 - 1$
 $\rightsquigarrow 5 * 5 + 2 * 5 + 1$
 $\rightsquigarrow 25 + 2 * 5 + 1$
 $\rightsquigarrow 25 + 10 + 1$
 $\rightsquigarrow 35 + 1$
 $\rightsquigarrow 36$

b. leftmost outermost

$(\backslash x y z \rightarrow \text{if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1) 2 (2 + 8 - 5) (6 + 7)$
 $\rightsquigarrow \text{if } 2 > 0 \text{ then } (2 + 8 - 5) * (2 + 8 - 5) + 2 * (2 + 8 - 5) + 1 \text{ else } (6 + 7) - 1$
 $\rightsquigarrow \text{if True then } (2 + 8 - 5) * (2 + 8 - 5) + 2 * (2 + 8 - 5) + 1 \text{ else } (6 + 7) - 1$
 $\rightsquigarrow (2 + 8 - 5) * (2 + 8 - 5) + 2 * (2 + 8 - 5) + 1$
 $\rightsquigarrow (10 - 5) * (2 + 8 - 5) + 2 * (2 + 8 - 5) + 1$
 $\rightsquigarrow 5 * (2 + 8 - 5) + 2 * (2 + 8 - 5) + 1$
 $\rightsquigarrow 5 * (10 - 5) + 2 * (2 + 8 - 5) + 1$
 $\rightsquigarrow 5 * 5 + 2 * (2 + 8 - 5) + 1$
 $\rightsquigarrow 25 + 2 * (2 + 8 - 5) + 1$
 $\rightsquigarrow 25 + 2 * (10 - 5) + 1$
 $\rightsquigarrow 25 + 2 * 5 + 1$
 $\rightsquigarrow 25 + 10 + 1$
 $\rightsquigarrow 35 + 1$
 $\rightsquigarrow 36$

c. lazy

$(\backslash x y z \rightarrow \text{if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1) 2 (2 + 8 - 5) (6 + 7)$
 $\rightsquigarrow \text{let } x = 2; y = 2 + 8 - 5; z = 6 + 7 \text{ in if } x > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1$
 $\rightsquigarrow \text{let } x = 2; y = 2 + 8 - 5; z = 6 + 7 \text{ in if } 2 > 0 \text{ then } y * y + 2 * y + 1 \text{ else } z - 1$
 $\rightsquigarrow \text{let } y = 2 + 8 - 5; z = 6 + 7 \text{ in if True then } y * y + 2 * y + 1 \text{ else } z - 1$
 $\rightsquigarrow \text{let } y = 2 + 8 - 5 \text{ in } y * y + 2 * y + 1$
 $\rightsquigarrow \text{let } y = 10 - 5 \text{ in } y * y + 2 * y + 1$
 $\rightsquigarrow \text{let } y = 5 \text{ in } y * y + 2 * y + 1$
 $\rightsquigarrow 5 * 5 + 2 * 5 + 1$
 $\rightsquigarrow 25 + 2 * 5 + 1$
 $\rightsquigarrow 25 + 10 + 1$
 $\rightsquigarrow 35 + 1$
 $\rightsquigarrow 36$

4. [15] Rewrite the following Haskell function definition

```
f :: Int -> [Int] -> Int
f n xs = if not (null xs) && n==0 then head xs
         else if not (null xs) && n > 0 then f (n-1) (tail xs)
         else if not (null xs) then 1
         else 2
```

in two different ways, preserving the order of the tests:

- a. [5] using guard syntax

```
f n xs
| not (null xs) && n==0 = head xs
| not (null xs) && n > 0 = f (n-1) (tail xs)
| not (null xs)         = 1
| otherwise             = 2
```

- b. [10] using patterns wherever you can, and guard syntax otherwise

```
f 0 (x:_) = x
f n (_:xs) | n>0 = f (n-1) xs
f _ (_:_) = 1
f _ _ = 2
```

5. [15] Define a Haskell function `match :: String -> String -> Bool` so that `match xs ys` returns True if and only if

`xs!!n == ys!!n` (Note: This does **not** imply that you should use `(!!)` in your solution.)

for all `n` such that

$0 \leq n < \text{length } xs$.

Then use `match` to define a function `search` so that `search as bs` returns a list of the positions in `as` at which `bs` occurs. For example,

```
search "Now is the winter of our discontent" "nt" ~> [13,30,33]
```

```
search "Now is the winter of our discontent" "g" ~> []
```

Include `search`'s type in your answer.

A solution:

```
match :: String -> String -> Bool
match [] _ = True
match (_:_) [] = False
match (a:as) (b:bs) = a==b && match as bs

search :: String -> String -> [Int]
search as bs = scan as 0
  where
    scan :: String -> Int -> [Int]
    scan [] n = []
    scan xs n
    | match bs xs = n : rest
    | otherwise = rest
    where
      rest = scan (tail xs) (n+1)
```