Lambda Calculus and Operational Semantics

1. As in lecture, let us define the following:

```
true = λt. λ f. t
false = λt. λ f. f
or = λa. λb. a a b
not = λb. b false true
```

   Using the above encodings, evaluate the following expression using normal order evaluation:

```
or (false) (not false)
```

2. Using the simple imperative language shown in lecture, write out the derivation trees for the following expressions and statements using operational semantics

P → *S*
 	
S → skip | *S*; *S* | *V* = *A*
| **if** *B* **then** *S* **else** *S* **end**
| **while** *B* **do** *S* **end**
 	
A → *N* | *V* | (*A* + *A*) | (*A* − *A*) | (*A* * *A*)
 	
B → true | false | (*A* <= *A*) | (*B* **and** *B*) | **not** *B*
 	
V → *Identifier*
 	
N → *IntegerLiteral*
	`x = -1`
	
	`1. ((x * 3) - 2)`
	
	`2. ((x <= 3) and (0 <= x))`

3. Suppose we wanted to add the ternary conditional (B ? A : A) to our language, where (b ? a_1 : a_2) evaluates to a_1 if b evaluates to true and a_2 otherwise. Write rules for the evaluating a ternary conditional using big-step operational semantics.