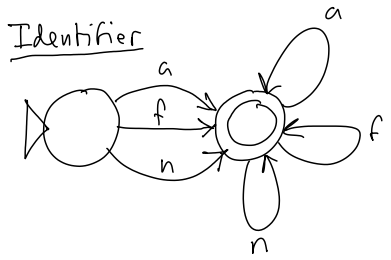
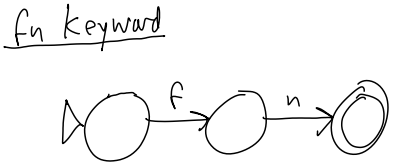
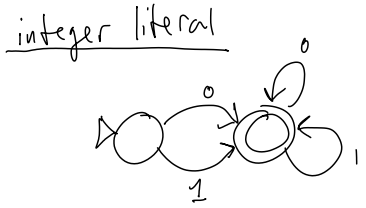
Question 1.

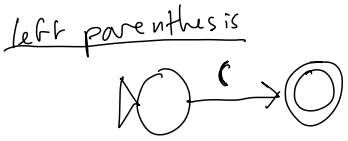
Kind of token	regular expression
identifier	(a|f|n)+
fn keyword	fn
integer literal	(0|1)+
left parenthesis	\(
right parenthesis	\)
whitespace	␣+

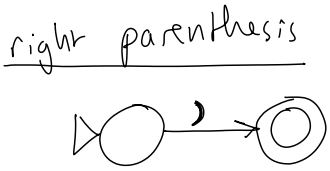
Question 2.

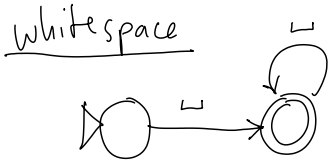




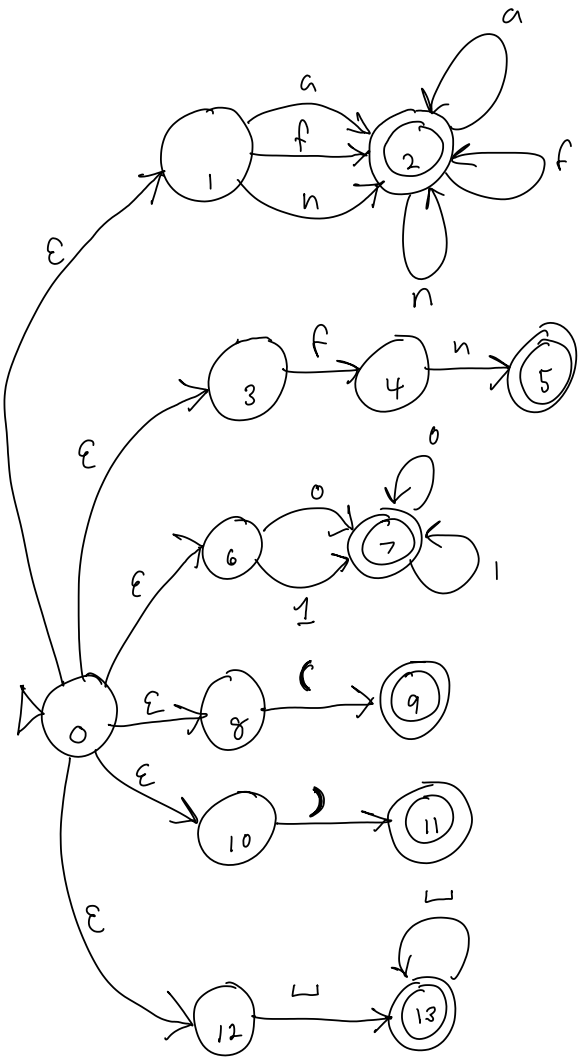




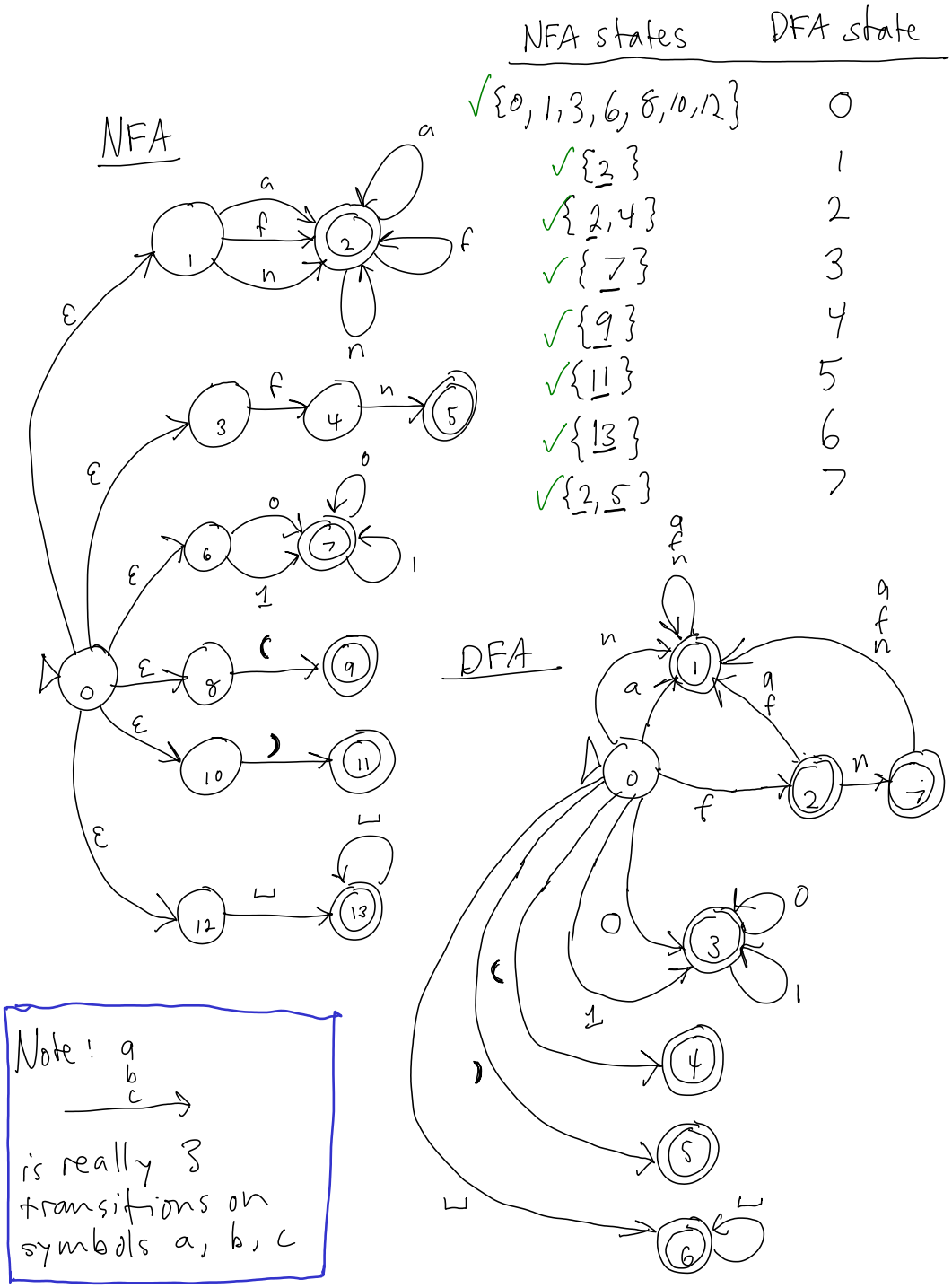




Question 3.



Question 4.


NFA states
DFA state
{0, 1, 3, 6, 8, 10, 12}
0
{2}
1
{2, 4}
2
{7}
3
{9}
4
{11}
5
{13}
6
{2, 5}
7
NFA
DFA
Note: a, b, c
is really 3 transitions on symbols a, b, c

Question 5 (628).

Note: "accepting" state means the same thing as "final" state.

DFA state 7 represents the reachable NFA state set { 2, 5 }. NFA state 2 is the accepting state for identifiers, and NFA state set 5 is the accepting state for the **fn** keyword. If we want to prioritize recognition of the **fn** keyword, we simply make this DFA state correspond to matching an occurrence of **fn** rather than identifier. In flex, we'd do this because in the lexer specification, the pattern for **fn** was earlier than the pattern for identifier. This is how flex resolves ambiguities: when an ambiguity exists, the earliest pattern "wins".

All other reachable NFA state sets have an accepting state corresponding to at most 1 token type, so there are no other ambiguities.

Question 5 (428), 6 (628).

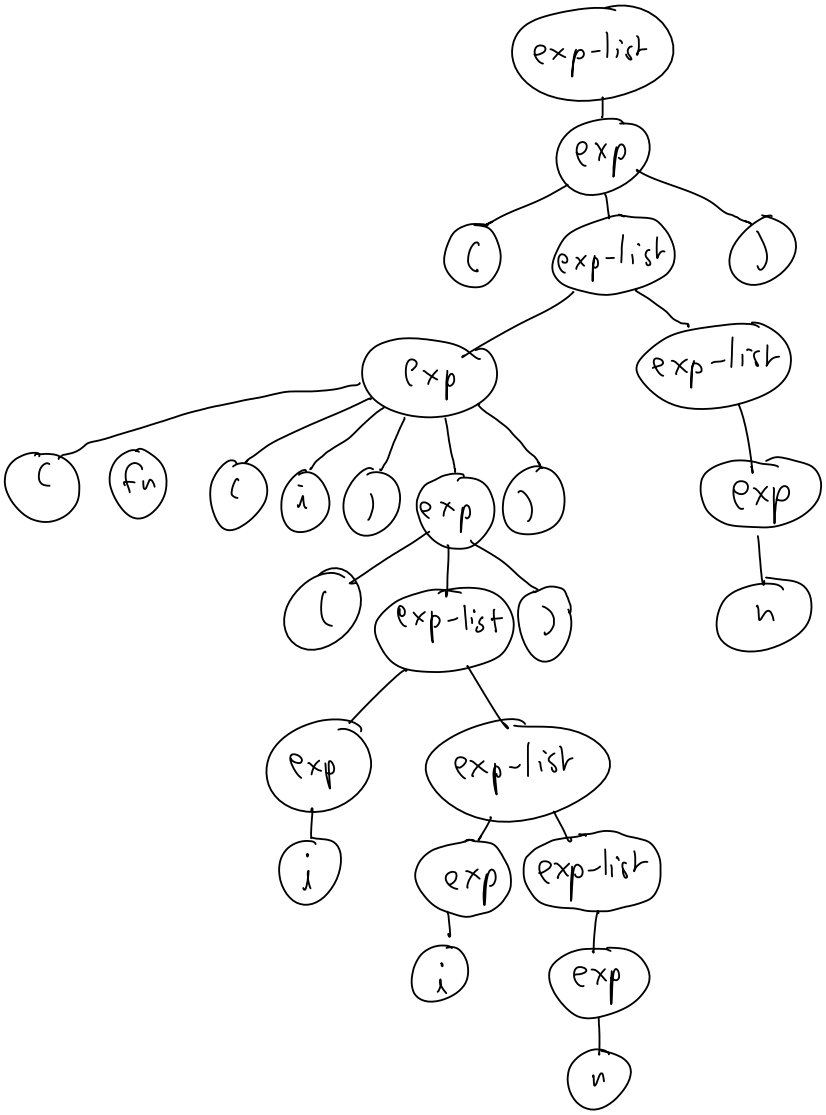
For brevity: i is identifier, n is integer-literal.

So, we want to derive $\boxed{(\ (\ \textbf{fn}\ (\ i\ )\ (\ i\ i\ n\ )\ )\ n\ )}$. (Because "a" and "fff" are both identifiers, and "101" and "110" are both integer literals.)

Working string	Production
exp-list	exp-list → exp
exp	exp → (exp-list)
(exp-list)	exp-list → exp exp-list
(exp exp-list)	exp → (**fn** (i) exp)
((**fn** (i) exp) exp-list)	exp → (exp-list)
((**fn** (i) (exp-list)) exp-list)	exp-list → exp exp-list
((**fn** (i) (exp exp-list)) exp-list)	exp → i
((**fn** (i) (i exp-list)) exp-list)	exp-list → exp exp-list
((**fn** (i) (i exp exp-list)) exp-list)	exp → i
((**fn** (i) (i i exp-list)) exp-list)	exp-list → exp
((**fn** (i) (i i exp)) exp-list)	exp → n
((**fn** (i) (i i n)) exp-list)	exp-list → exp
((**fn** (i) (i i n)) exp)	exp → n
((**fn** (i) (i i n)) n)	

Question 6 (428), 7 (628).

Note: i is identifier, and n is integer literal



Question 7 (428), 8 (628).

For brevity: i is identifier, n is integer-literal.

FIRST(exp) = { i, n, (}

FOLLOW(exp) = { i, n, (,), *eof* }

Question 8 (428), 9 (628).

Note that building a parse tree isn't explicitly shown.

```
parseExp()
    t = peek()
    if (t is null)
        error("unexpected end of input")
    if (t is identifier)
        -- Apply exp → identifier
        expect(identifier)
    else if (t is integer-literal)
        -- Apply exp → integer-literal
        expect(integer-literal)
    else if (t is "(")
        expect("(")
        t = peek()
        if (t is null)
            error("unexpected end of input")
        else if (t is fn)
            -- Apply exp → ( fn ( identifier ) exp )
            expect("fn")
            expect("(")
            expect(identifier)
            expect(")")
            parseExp()
            expect(")")
        else
            -- Apply exp → ( exp-list )
            parseExpList()
            expect(")")
```

Question 10 (628).

No, because the FIRST+ sets of the $\boxed{exp \rightarrow \textbf{(}\ exp\text{-}list\ \textbf{)}}$ and $\boxed{exp \rightarrow \textbf{(}\ \textbf{fn}\ \textbf{(}\ \textbf{identifier}\ \textbf{)}\ exp\ \textbf{)}}$ productions both contain "(". Another way to look at this is to recognize that distinguishing these productions requires two tokens of lookahead.