Homework 1 Solution

Question 1

1a

/^(0|1)*[0]{4,}$/

1b

/^1*00*$/

1c

/^(10|1)$/

Question 2

/"([^"\\\n]|\\[^\n])*"/

Question 3

3a

L -> aLb
L -> bb

3b

For n = 0,
The L -> bb is applicable.
Thus, the string is "bb", which satisfies the condition for n = 0.

Assume for some k >= 0, the CFG generates strings of the form (a^k)(b^(k+2)).

For k + 1:

By applying L -> aLb k+1 times we have:
(a^(k+1))L(b^(k+1))

Then, if we apply L -> aLb again, we would have more than k+1 a's.
So, we apply L -> bb, we have:
(a^(k+1))(b^(k+3))

Which satisfies the condition.

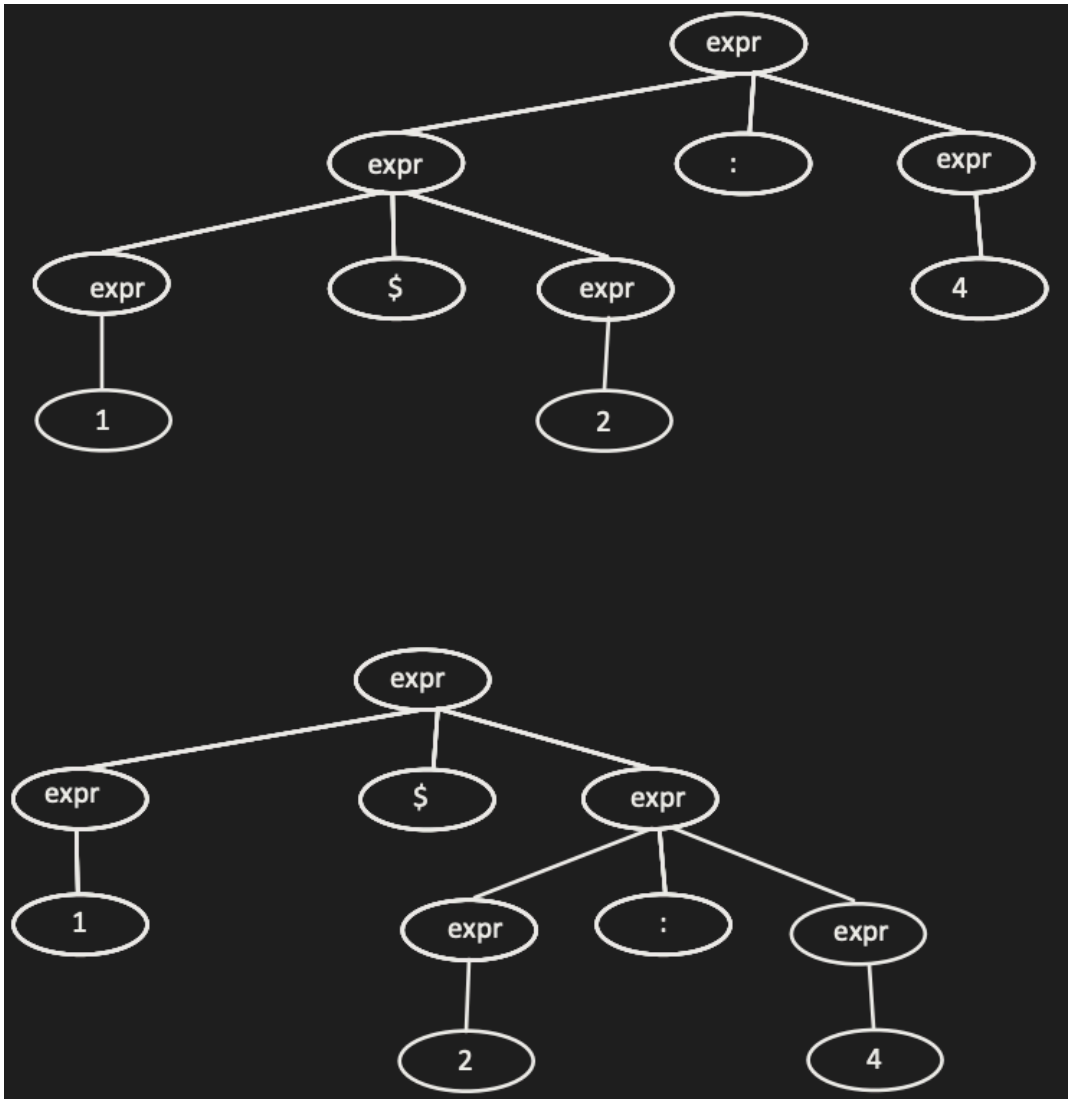
Hence, the CFG describes precisely the language L.

Question 4

4a

expr => expr '$' expr
    => '(' expr ')' '$' expr
    => '(' expr 'then' expr 'else' expr ')' '$' expr
    => '(' 1 'then' expr 'else' expr ')' '$' expr
    => '(' 1 'then' '#' expr 'else' expr ')' '$' expr
    => '(' 1 'then' '#' 2 'else' expr ')' '$' expr
    => '(' 1 'then' '#' 2 'else' 3 ')' '$' expr
    => '(' 1 'then' '#' 2 'else' 3 ')' '$' 4

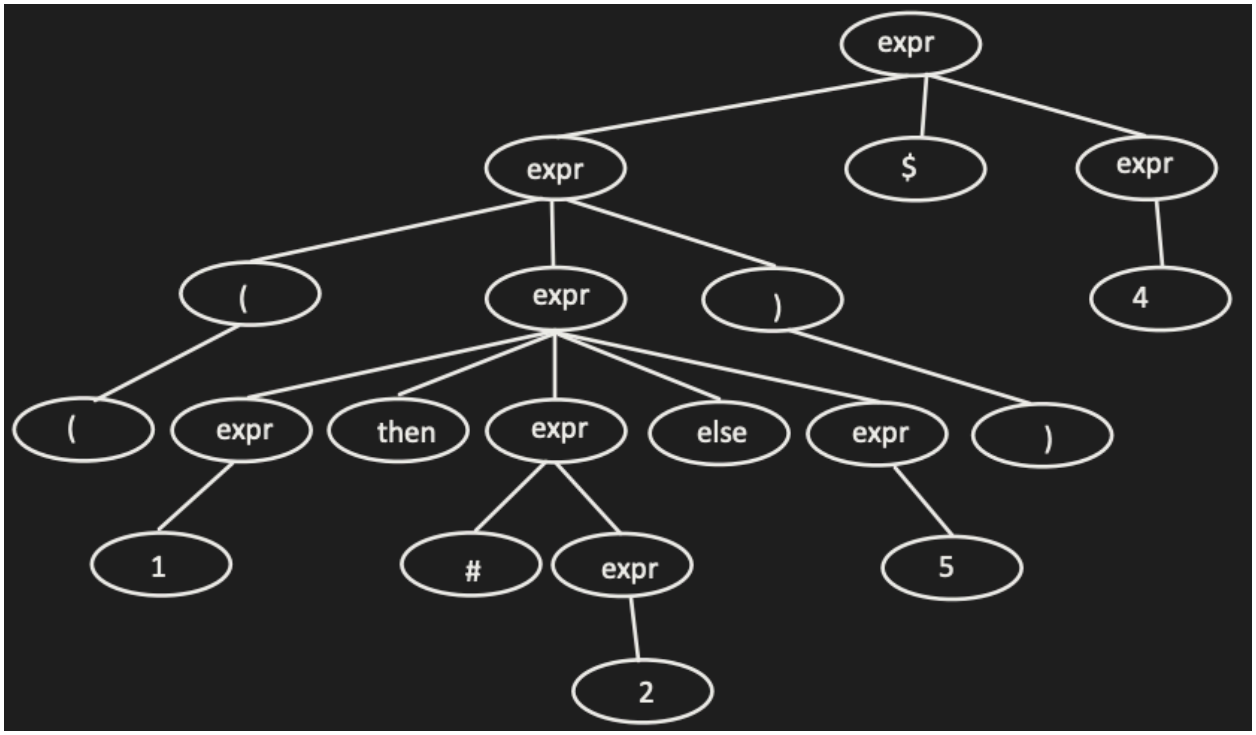
4b



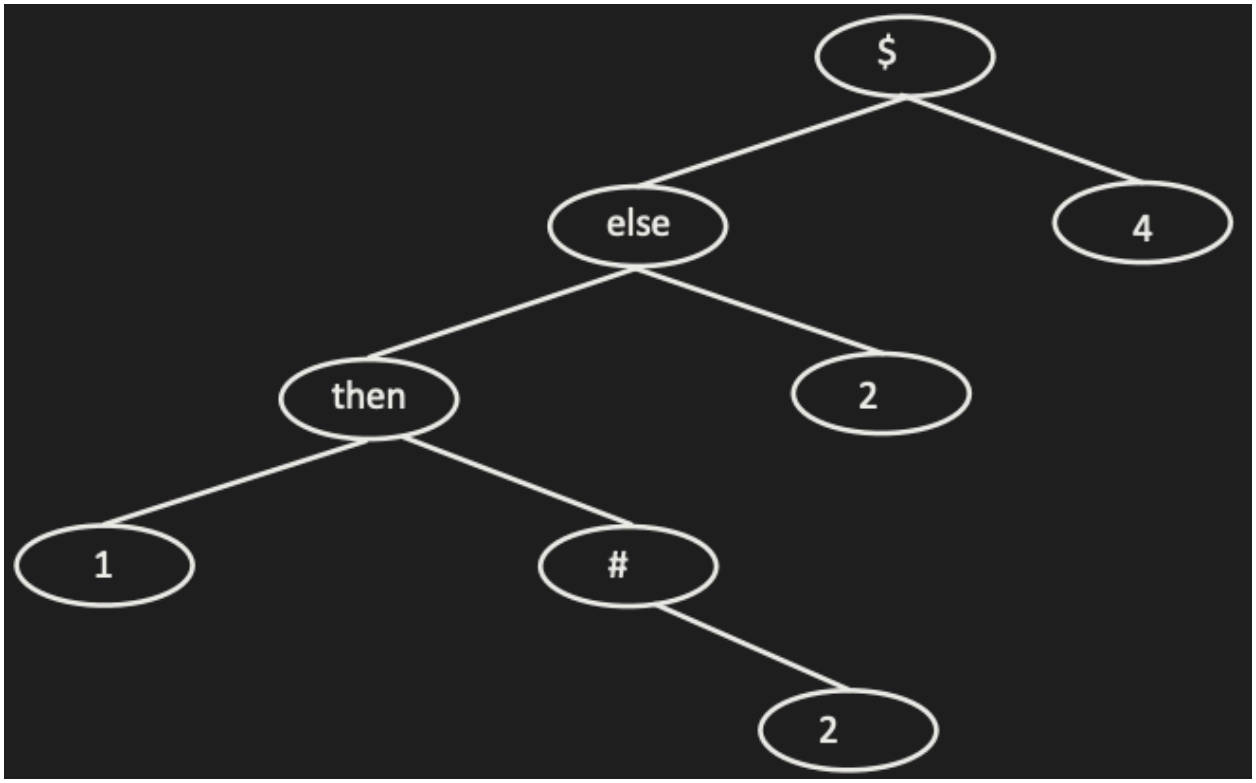
4c

```
expr
  : expr 'then' expr 'else' expr2
  | expr2
  ;
expr2
  : expr3 '$' expr2
  | expr3
  ;
expr3
  : expr4 ':' expr3
  | expr4
  ;
expr4
  : '#' expr
  | INT
  | '(' expr ')'
  ;
```

4d



4e



4f

```
expr
  : expr2 ( 'then' expr2 'else' expr2 )*
  ;
expr2
  : expr3 ( '$' expr3 )*
  ;
expr3
  : expr4 ( ':' expr4 )*
  ;
expr4
  : ( '#' )* expr
  | INT
  | '(' expr ')'
  ;
```

4g

```
expr() {
  expr2()
  if (peek('then')) {
    consume('then')
    expr2()
    consume('else')
    expr2()
  }
}

expr2() {
  expr3()
  while (peek('$')) {
    consume('$')
    expr3()
  }
}

expr3() {
  expr3()
  while (peek(':')) {
    consume(':')
    expr3()
  }
}

expr4() {
  if (peek('#')) {
    consume ('#')
    expr()
  } else if (peek(INT)) {
    consume(INT)
  } else {
    consume('(')
    expr()
    consume(')')
  }
}
```

5

5a

```
return a + b*x;     //refs to a, b, x.
a: 6
b: 5
x: 2
```

return b + h(a)*x;   //refs to a, b, x.
a: 3
b: 3
x: 2

return a + g(x);   //refs to a, x.
a: 1
x: 2

5b

//1
a: <1,0>

//2
x: <1,-3>

//3
a: <2,0>
b: <2,1>

//4
y: <2,-3>

//5
b: <3,0>

//6
a: <3,-3>

5c

Since h() has depth 3, and x has depth 1, the compiler would follow (3 - 1 = 2) static chains to reach x's depth, then offset by -3 to retrieve it.

6

6a

It is possible to support subroutine calls without using a stack. However, there are limitations: difficult to handle nested or recursive calls.

6b

Not totally valid. It is possible to write many arithmetic expressions involving the four binary operators +, -, * and / unambiguously without needing to use parentheses. But without parentheses, we will not be able to write some expressions with precedences overwritten.

6c

It is valid. Some notable languages: C, Java, and Perl5

6d

It is valid. "." symbol are used in dotted pairs in Lisp. However, we can also create pairs with the cons functions.

6e

Not valid. They are likely be allocated on a heap.