编译原理 - 作业(4)：语义分析

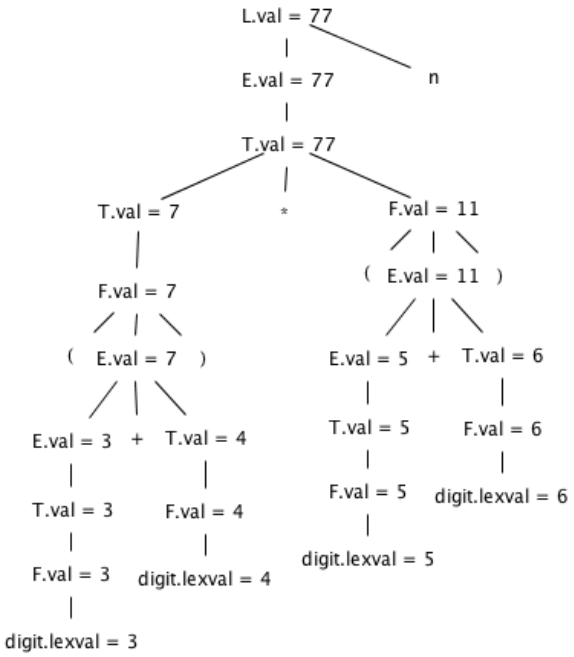
截至时间：2022.5.19/周四 上课前（14:20）
提交方式：超算习堂（https://easyhpc.net/course/144）

Q1: (P309, Exercise 5.1.1) For the SDD below, give annotated parse trees for the following expressions:

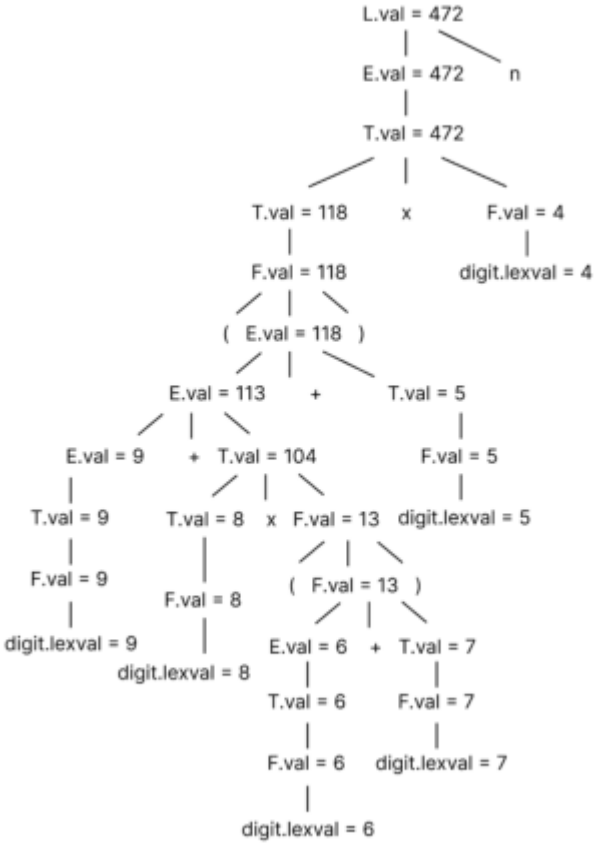
PRODUCTIONS	SEMANTIC RULES
1) $L \rightarrow E$ **n**	L.val = E.*val*
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val$ x $F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow (E)$	$F.val = E.val$
7) $F \rightarrow$ digit	F.*val* = digit.*lexval*

(1) (3 + 4) * (5 + 6) **n**
(2) (9 + 8 * (7 + 6) + 5) * 4 **n**

（1）



（2）



Q2: (p323, Exercises 5.3.1) Below is a grammar for expressions involving operator + and integer of floating-point operands. Floating-point numbers are distinguished by having a decimal point:

$E \rightarrow E + T \mid T$

$L \rightarrow$ num . num | num

Give an SDD to determine the type of each term T and expression E.

	产生式	语义规则
1)	E -> E_1 + T	E.type = E_1.type === float || T.type === float ? float : int
2)	E -> T	E.type = T.type
3)	T -> num.num	T.type = float
4)	T -> num	T.type = int

Q3: (p317, Exercises 5.2.4) This grammar generates binary numbers with a “decimal” point:

$S \rightarrow L . L \mid L$

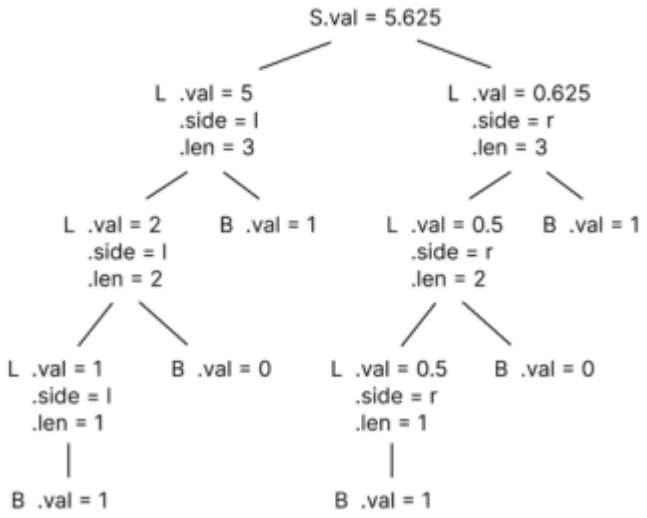
$L \rightarrow L B \mid B$

$B \rightarrow 0 \mid 1$

(1) Design an L-attributed SDD to compute *S.val*, the decimal number value of an input string. For example, the translation of string 101.101 should be the decimal number 5.625. Hint: use an inherited attribute *L.side* that tells which side of the decimal point a bit is on.

(2) Draw the annotated parse tree of 101.101.

	产生式	语义规则
1)	S -> L_1.L_2	L_1.isLeft = true L_2.isLeft = false S.val = L_1.val + L_2.val
2)	S -> L	L.isLeft = true S.val = L.val
3)	L -> L_1B	L_1.isLeft = L.isLeft L.len = L_1.len + 1 L.val = L.isLeft ? L_1.val * 2 + B.val : L_1.val + B.val * 2^(-L.len)
4)	L -> B	L.len = 1 L.val = L.isLeft ? B.val : B.val/2
5)	B -> 0	B.val = 0
6)	B -> 1	B.val = 1



Q4: For the code snippet below：

```
1:  int x = 0;
2:  float y = 0.0;
3:  while (x < 10) {
```

```
4:   int y, z;
5:   y = x;
6:   z = 0;
7:   while (y < 10) {
8:     z = z + y;
9:     y = y + 1;
10:  }
11: }
12: y = x * 1.0;
```

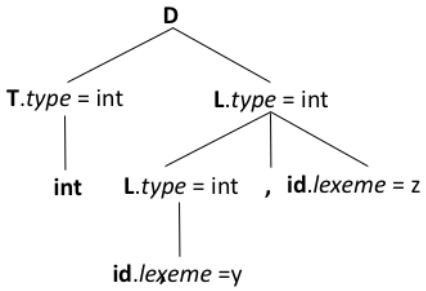
Regarding the semantic analysis of variable type, we consider the following simplified grammar and syntax-directed translation (SDT):

1) $D \rightarrow T$ { $L.type = T.type$ } L
2) $T \rightarrow$ int { $T.type$ = int }
3) $T \rightarrow$ float { $T.type$ = float }
4) $L \rightarrow$ { $L_1.type = L.type$ }
   L_1, id { *addtype*(id.*entry*, $L.type$)
5) $L \rightarrow$ id { *addtype*(id.*entry*, $L.type$)

(1) In the above SDT, both T and L have attribute 'type'. The type attribute is synthesized or inherited? Please explain.

T. *type*是综合属性；L. *type*是继承属性。

(2) For Line 4 of the code snippet：int y, z; Construct the annotated parse tree based on the above SDT.



(3) For Lines 3, 7 and 11 of the code snippet, list the valid variables (name and type) in symbol table.

Locations	Symbol table entries
Line (3)	{x: int, y: float}
Line (7)	{x: int, y: int, z: int}
Line (11)	{x: int, y: float}