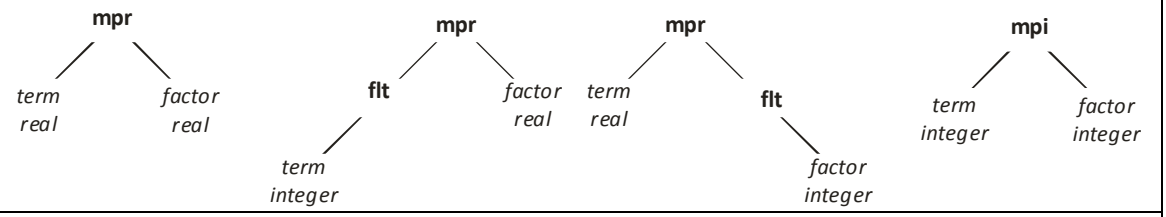
56	*term*	→	*factor*
57	*term*	→	*term mulop factor*
58	*mulop*	→	*****
59	*mulop*	→	**/**
60	*mulop*	→	**div**
61	*mulop*	→	**mod**
62	*mulop*	→	**and**

56	*term*	→	*factor*
			$$=$1

57	*term*	→	*term mulop factor*

Exp term*(*Exp* term*,*string* mulop*,*Exp* factor*)
if **mulop*=**"*"** **return** *term_1*(*term*,*factor*)
if **mulop*=**"/"** **return** *term_2*(*term*,*factor*)
if **mulop*=**"div"** **return** *term_3*(*term*,*factor*)
if **mulop*=**"mod"** **return** *term_4*(*term*,*factor*)
if **mulop*=**"and"** **return** *term_5*(*term*,*factor*)
else *error*(**"invalid mulop"**)

57	*term*	→	*term mulop factor*
58	*mulop*	→	*****



Exp term_1*(*Exp* term*,*Exp* factor*)

1. Semantics for term are dependent on *mulop*. The information given here is for rule 58 mulop → *****.
2. Four expressions are returned depending on the types of input parameters term and factor.

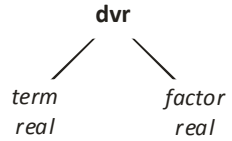
2.1. term has type real. factor has type real.
2.1.1. Create PCode P
mpr
2.1.2. Return Expression E
PCode P created in 2.1.1.
Type real
Left sub expression: parameter term
Right sub expression : parameter factor

2.2. term has type integer. factor has type real.
- 2.2.1. Create PCode P
  - **flt**
- 2.2.2. Create Expression L
  - PCode P created in 2.2.1.
  - Type real
  - Left subexpression: term
- 2.2.3. Create PCode P
  - **mpr**
- 2.2.4. Return Expression E
  - PCode P created in 2.2.3.
  - Type real
  - Left sub expression: expression L created in 2.2.2.
  - Right sub expression : parameter factor

2.3. term has type real. factor has type integer.
- 2.3.1. Create PCode P
  - **flt**
- 2.3.2. Create Expression R
  - PCode P created in 2.3.1.
  - Type real
  - Right subexpression: factor
- 2.3.3. Create PCode P
  - **mpr**
- 2.3.4. Return Expression E
  - PCode P created in 2.3.3.
  - Type real
  - Left sub expression: parameter term
  - Right sub expression : expression R created in 2.3.2.

2.4. term has type integer. factor has type integer.
- 2.4.1. Create PCode P
  - **mpi**
- 2.4.2. Return Expression E
  - PCode P created in 2.4.1.
  - Type integer
  - Left sub expression: parameter term
  - Right sub expression : parameter factor

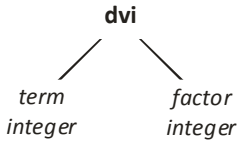
57	*term*	→	*term mulop factor*
59	*mulop*	→	**/**



Exp term_2*(*Exp* term*,*Exp* factor*)

1. Semantics for term are dependent on *mulop*. The information given here is for rule 60 mulop → **/**.
2. Define *mulop* to have the same semantic type has a token so that it is a pointer to a string (string*).
3. Function *term_2* returns an expression
4. Function *term_2* accepts two parameters, *term* and *factor*, both expressions.
5. Both input parameters must have type real.
6. Create PCode P
   dvr
7. Return expression E having
   7.1. PCode P created in step 6
   7.2. Type real
   7.3. Left sub expression: parameter *term*
   7.4. Right sub expression: parameter *factor*.

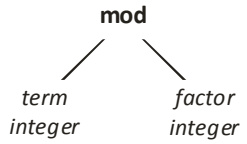
57	*term*	→	*term mulop factor*
60	*mulop*	→	**div**



Exp term_3*(*Exp* term*,*Exp* factor*)

1. Semantics for term are dependent on *mulop*. The information given here is for rule 60 mulop → **div**.
2. Define *mulop* to have the same semantic type as a token so that it is a pointer to a string (string*).
3. Function *term_3* returns an expression
4. Function *term_3* accepts two parameters, *term* and *factor*, both expressions.
5. Both input parameters must have type integer.
6. Create PCode P
   dvi
7. Return expression E having
   7.1. PCode P created in step 6
   7.2. Type integer
   7.3. Left sub expression: parameter *term*
   7.4. Right sub expression: parameter *factor*.

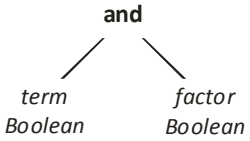
57	*term*	→	*term mulop factor*
61	*mulop*	→	**mod**



Exp term_4*(*Exp* term*,*Exp* factor*)

1. Semantics for term are dependent on *mulop*. The information given here is for rule 61 mulop → **mod**.
2. Define *mulop* to have the same semantic type has a token so that it is a pointer to a string (string*).
3. Function *term_4* returns an expression
4. Function *term_4* accepts two parameters, *term* and *factor*, both expressions.
5. Both input parameters must have type integer.
6. Create PCode P
   mod
7. Return expression E having
   7.1. PCode P created in step 6
   7.2. Type integer
   7.3. Left sub expression: parameter *term*
   7.4. Right sub expression: parameter *factor*.

57	*term*	→	*term mulop factor*
62	*mulop*	→	**and**



Exp term_5*(*Exp* term*,*Exp* factor*)

1. Semantics for term are dependent on *mulop*. The information given here is for rule 62 mulop → **and**.
2. Define *mulop* to have the same semantic type has a token so that it is a pointer to a string (string*).
3. Function *term_5* returns an expression
4. Function *term_5* accepts two parameters, *term* and *factor*, both expressions.
5. Both input parameters must have type Boolean.
6. Create PCode P
   and
7. Return expression E having
   7.1. PCode P created in step 6
   7.2. Type Boolean
   7.3. Left sub expression: parameter *term*
   7.4. Right sub expression: parameter *factor*.