exp ⇒ exp addop term | term

addop ⇒ + | −

term ⇒ term mulop factor | factor

mulop ⇒ *

factor ⇒ (exp) | number

since 5 non terminals so 5 procedures.

factor ⇒ (exp) | number #(2 productions so if else)

(#procedure for factor non terminal)

```
void factor (void) {
    if (token == '(')
        match ('(')
```

#(terminal so match operation)

```
exp()                          // for an terminal "exp"
match ('(')                    we do a corresponding
else                           match procedure
    match ("number")           }
```

---

addop → + | -

```
addop
void match (void) {
    if (token == '+')
        match ('+')
    else
        match ('-');
}
```

---

mulop → *

```
void mulop (void) {
    match ('*')
}
```

exp & statement are non terminal
if-stmt in grammar of this question

if-stmt ⇒ if (exp) statement [else statement]

```
void if-stmt (void) {
    match ('if')
    match ('(')
    exp()
    match (')')
    statement ()
    if (token == 'else')
        match ('else')
        statement () }
```

exp → exp addop term | term

exp ⇒ term {addop term}

converted to EBNF due to left recursion

procedure:-

```
void exp (void) {
    term ()
    while (token == '+' OR token == '-')
        addop()
        term () }
```

how match operation works?

- void match (expected)

getNextToken()

else

error ("expected token not found");

① var-decl → type decl ; | type decl ; var-decl

type → int | float

decl → id | id, decl | id = n | id = n, decl

② a S a | b S b | a | b