

Comma Expression

To add the comma expression to your compiler, add the production:

```
expr: expr ',' expr
```

This enables you to recognize statements like:

`a = b, c, d;` which in effect just sets `a` to `d` and ignores `b,c`

Stack Problem

- In its current form, the comma expression leaves a value on the runtime stack after it is executed. Therefore, if you execute the comma expression repeatedly, such as in a loop, you could run yourself out of stack memory and have the program terminate abnormally.
- It is up to you to solve this problem.

Shift/Reduce Errors

- Inserting the comma expression will cause shift/reduce errors
- Giving ‘,’ a precedence at the top of c.y will eliminate some of the errors, but not all of them. It is up to you to determine which precedence to give ‘,’.

Remaining Shift/Reduce Errors

You get a hint of what causes the remaining shift/reduce errors when you print out multiple variables.

For example, `fprint(a,b,c,d)` prints out only the variable `a`.

Detecting the Problem

- Go to the file `y.output` and do a search on the string `red'n`
- Notice there is a shift/reduce error on the production `pexpr: expr` where comma `(,)` is the lookahead token
- The compiler doesn't know whether to reduce the production `pexpr: expr` (which causes the first expression to be printed out) or shift the comma in (which is what we should do to print everything out)

Shift/Reduce - Solution

Put % prec PL at the end of the production
pexpr: expr

pexpr: expr %prec PL

This gives the production expr: pexpr the name
PL

Then, make PL higher in precedence than
comma (,) at the top of c.y