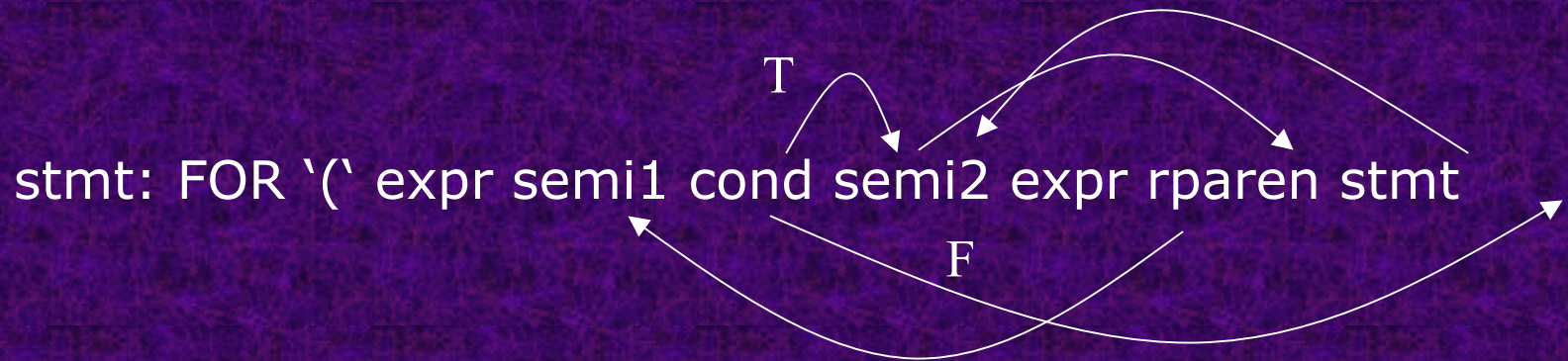


For Statement



{ branch semi2; fixup cond with end; fixup rparen with semi1; fixup semi2 with rparen; }

semi1: ';' { pick off address; }

semi2: ';' { emit branch, STOP (fixup later); pick off address; }

rparen: ')' { emit branch, STOP (fixup later); pick off address; }

Comma Expression

expr : '-' expr %prec UNARYMINUS

'-' expr . '-'

UNARYMINUS represents a reduction instead of an operator and is given very high precedence.

Comma, Detecting the Problem

After adding:

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

Notice that you get a number of shift reduce errors.

1. Eliminate some of them by giving comma precedence at the top of `C.y`.
2. Remaining shift reduce errors manifest themselves by having `fprint` output only one item in a list. For example: `printf(a,b,c)` prints out only `a`.
3. Go to `y.output` and do a search on `red'n`.
4. Notice there is a shift/reduce error on the production `prlist: expr` with the token comma (`,`).
5. The compiler doesn't know whether to reduce `expr` to `prlist` (which will cause the expression to be printed out) or shift in the comma (which will cause the remainder of the expression to be treated as a comma expression).

Comma, Solution

prlist: expr %prec PL

Give PL higher precedence than comma at top of C.y.

Ternary Operator, ? :

$\text{expr}_1 ? \text{expr}_2 : \text{expr}_3$

If expr_1 is true do expr_2 ; otherwise do expr_3

? :, Problem

We currently have:

stmt: cond '?' stmt colonpart

colonpart: ':' stmt

The above productions should be changed to

stmt: cond '?' expr colonpart

colonpart: ':' expr

and

expr: cond '?' expr colonpart

However, the change generates shift/reduce errors.

? :, Shift/Reduce Errors

Should:

$a > b ? c : d + e;$

Be interpreted as:

$(a > b ? c : d) + e;$

or

$a > b ? c : (d + e);$

Declarations, code.c

- Copy current code.c to old_code.c
- Copy new_code.c to code.c
- The new code.c will handle declarations.
The only difference is in the `assign()` and `eval()` functions.

Declarations, Productions

```
decs:      decs type id_list `;`  
          |  $\epsilon$   
type:     D_INT  
          | D_FLOAT  
id_list:  id_list `,' ID  
          | ID
```

Place decs NT in RHS of production(s)
where you want declarations to appear

Declarations, lex.1

```
inum          {digit}+
fnum          ({digit}+\.{digit}*)|({digit}*\.{digit}+)

{inum} {
    sscanf(yytext, "%lf", &yylval.val);
    yylval.sym = install("",S_INT,yylval.val);
    printf("%s",yytext);
    return NUMBER; }

{fnum} {
    sscanf(yytext, "%lf", &yylval.val);
    yylval.sym = install("",S_FLOAT,yylval.val);
    printf("%s",yytext);
    return NUMBER; }

{id}         ???
```

Pointers

```
void main() {  
    int a, *b, *c, d;  
    a=73; b = &a; c = b;  
    printf(a, *b, *c);  
    *c = 142;  
    Printf(a, *b, *c);  
    d=6; c = &d; *b = *c;  
    printf(a, *b, *c, d);  
    *b = 59;  
    printf(a, *b, *c, d); }  
}
```

Pointer Declarations

```
id_list: id_list `,' id_type  
        | id_type
```

```
id: ID { $1->type = T;  
        $1->ptr=0; }  
    | *ID { $1->type = T;  
          $1->ptr=1; }
```

Pointer Instructions

```
void main() {  
    int a, *b, *c, d;  
    a=73; b = &a; c = b; *b = 24; a = *b;
```

constpush

73

varpush

a

assign

varpush

a

varpush

b

assign

varpush

b

eval

varpush

c

assign

constpush

24

varpush

b

eval

assign

varpush

b

eval

eval

varpush

a

assign

Boolean Operations

cond: dexpr

dexpr: cexpr

| dexpr cexpr

cexpr: expr EQ expr

| expr NE expr

| expr GT expr

| ...

| cexpr AND cexpr

| cexpr OR cexpr

Boolean Operations (Second Alternative)

cond: cexpr

cexpr: expr EQ expr
| expr NE expr
| expr GT expr
| ...
| cexpr AND cexpr
| cexpr OR cexpr