Recall:

parse.y:

```
%{ #include "tree.h"
%union {
    int y_int;
    ST y_tree;
}
%token <y_int> CONST VAR
%type <y_tree> expr factor term

example rules                    prototypes in tree.h
factor:
    CONST  { $$ = make_constnode($1); }
  | VAR    { $$ = make_varnode($1); }
  | '(' expr ')'  { $$ = $2; }
  ;

expr:
    expr '+' term
        { $$ = make_binopnode('+', $1, $3); }
```

type of a semantic stack entry

In tree.c

```
#include "tree.h"

ST make_constnode(int val)
{
    ST ret;
    ret = (ST) malloc(sizeof(struct stn));
    assert(ret != NULL);
    ret->tag = CONST_NODE;
    ret->u.constval = val;
    return ret;
}
```

tree.h

```
typedef struct stn {
    tagtype tag;
    union {
        ...
    } u;
} ST_NODE, *ST;
```

In scan.l:

```
%{
#include "tree.h"
#include "y.tab.h"
...

[0-9]+   { yylval.y_int = atoi(yytext);
           return CONST;
         }
```

Makefile

```
see : parse.o scan.o main.o
    gcc parse.o scan.o tree.o -o see -ll

parse.c ytab.h : parse.y tree.h
    bison -t -y -d -v parse.y
    mv y.tab.c parse.c

scan.c : scan.l tree.h ytab.h
    flex scan.l
    mv lex.yy.c scan.c

parse.o : parse.c
    gcc -c parse.c

scan.o : scan.c
    gcc -c scan.c
    :
    :

clean:
    -rm *.o parse.c scan.c ytab.h see
```

can omit implicit rules

bison handles synthesized attrs

inherited attrs can be useful:

Ex: Passing down type info in a declaration

D = "declaration"
T = "type"
L = "list of names"

D → T L

T → int

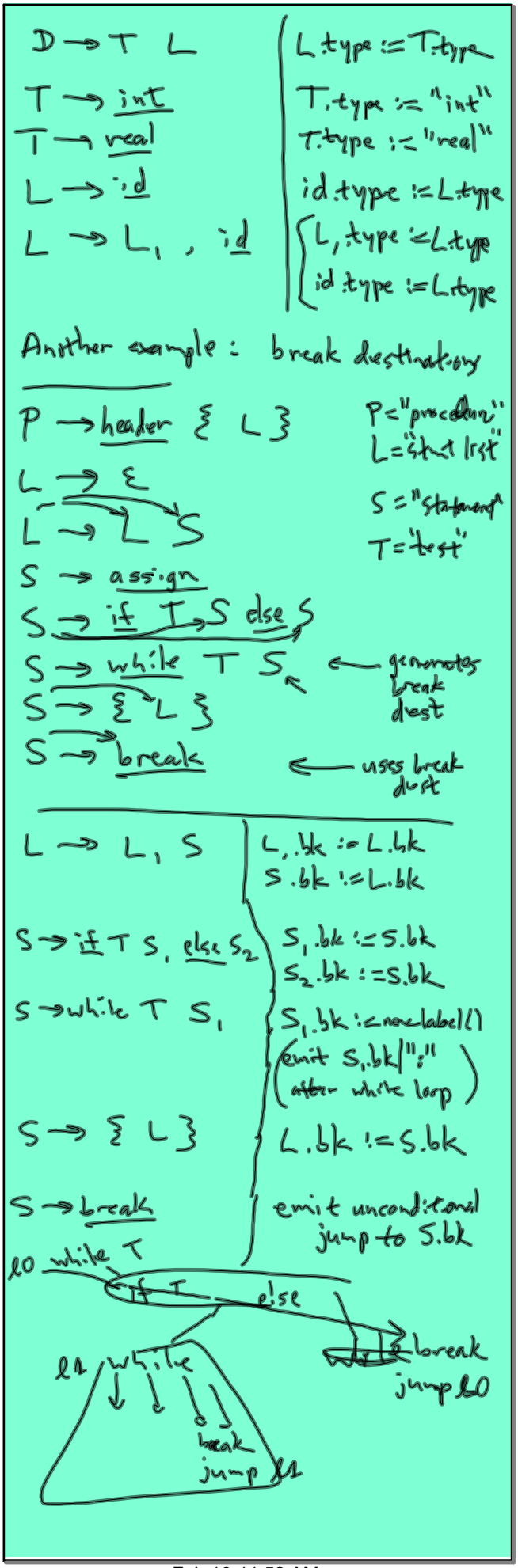
T → real

L → id

L → L , id

real x, y, z

Feb 16-11:01 AM



Feb 16-11:52 AM