

Recall: #include "tree.h"

parse.y:

```

%union {
    int y_int;
    ST y_tree;
}
%token <y_int> CONST VAR
%type <y_tree> expr factor term

```

example rules

factor:

```

CONST { $1 = make_const_node($1); }
| VAR { $1 = make_var_node($1); }
| '(' expr ')' { $1 = $2; }
;

```

expr:

```

'+' term
{ $1 = make_binop_node('+', $1, $3); }

```

In tree.c

```

#include "tree.h"
ST make_const_node(int val)
{
    ST ret;
    ret = (ST) malloc(sizeof(ST));
    assert(ret != NULL);
    ret->tag = CONST_NODE;
    ret->u.const_val = val;
    return ret;
}

```

In tree.h

```

typedef struct stn {
    tag_type tag;
    union {
        // ...
    };
} ST;

```

In scan.l

```

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

```

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

Makefile

```

src: parse.o scan.o min.o tree.o
gcc -o min src -lfl

```

parse.c ytab.h: parse.y tree.h

```

bison -t -y -d -v parse.y
mv ytab.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

bison handles syntactical errors

inherited errors can be useful:

Ex: Passing down type info in a declaration

```

D -> T L
T -> int
T -> real
L -> id
L -> L , id

```

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

real x, y, z

$D \rightarrow T L$	$L.type := T.type$
$T \rightarrow \underline{int}$	$T.type := "int"$
$T \rightarrow \underline{real}$	$T.type := "real"$
$L \rightarrow \underline{id}$	$id.type := L.type$
$L \rightarrow L_1, id$	$\begin{cases} L.type := L_1.type \\ id.type := L.type \end{cases}$

Another example: break destination

$P \rightarrow \underline{header} \{ L \}$	$P = "procedure"$
	$L = "list"$
$L \rightarrow \epsilon$	
$L \rightarrow L S$	$S = "statement"$
$S \rightarrow \underline{assign}$	$T = "test"$
$S \rightarrow \underline{if T S \text{ else } S}$	
$S \rightarrow \underline{while T S}$	← generates break dest
$S \rightarrow \{ L \}$	
$S \rightarrow \underline{break}$	← uses break dest

$L \rightarrow L_1 S$	$L_1.bk := L.bk$ $S.bk := L.bk$
$S \rightarrow \underline{if T S_1 \text{ else } S_2}$	$S_1.bk := S.bk$ $S_2.bk := S.bk$
$S \rightarrow \underline{while T S_1}$	$S_1.bk := \text{new label()}$ $(\text{emit } S_1.bk / ":", "$ $\text{after while loop})$
$S \rightarrow \{ L \}$	$L.bk := S.bk$
$S \rightarrow \underline{break}$	emit unconditional jump to S.bk

