# Lab #3

parser.y token recognizer

Replace with actual Mini GISL grammar

Grammar rules from spec

Add trace parse func

Handle operator precedence properly

what does Bison do?

Reads grammar rules, generates shift reduce parser

Builds parse table that decides when to shift / reduce / apply a grammar rule

%% sections separate declarations, grammar, C code

```
non-terminal :  production1 { action1 }
              | production2 { action2 }
              ;
```

yTRACE() trace macro defined in file

parser.y

{ actions } executes when Bison reads that rule

print out what rule was applied

Grammar has ambiguity with if/else statements

Bison uses shift preference by default

Match else with nearest if

## Operator precedence

```
lowest               %left     OR
                     %left     AND
                     %nonassoc EQ NEQ '<' LEQ '>' GEQ
                     %left     '+' '-'
                     %left     '*' '/'
right associative    %right    '^'
highest              %right    '!' UMINUS
```

1+ 2*3

1+( 2*3)

Need higher precedence of unary minus over binary minus

| '-' %prec UMINUS

-2 +3

(-2) +3

Explicit binary op

collapse rules into expression non-terminal

## Conflicts

Shift reduce conflict

Reduce-reduce conflict

Token type mismatches?

% non assoc used for comp operators instead of %left / %right
→ Can't chain operators → a<b<c syntax error

Bison defaults to shift for shift reduce conflict

Bison is parser generator, not parser
Only write grammar rules, Bison generates C code for parser
↳ uses parse table for shift/reduce decisions

make → bison → parser.c, parser.tab.h

## CFG to bison syntax

CFG from specification to Bison rules & syntax

eg. declaration → type ID ';'

declaration
 : type ID ';'
  { yTRACE("..."); }

Left recursive rules for declarations & statements (bottom up parsing)
Epsilon prod as empty alternatives
Maintain grammar structure as specified

## Operator precedence

initial implementation had used non terminals (binary_op, unary_op) for grouping operators
prevented bison from applying precedence declarations

Bison precedence mechanism only works when it sees actual tokens
in prod rules, NOT non terminals that wrap tokens

Inline all operators directly into expression rules

% non assoc prevents chaining as explained
L conflict after this fix

## Syntax vs Semantics

Parser only validates syntax, not meaning / semantics
int x = true ✓
Semantically invalid

## Trace parser functionality for debugging parsing

Added trace prints

## Verification

Test cases

Declaration
Type variants
Operator precedence
Nested scopes
Control flow
Constructor / function
Array subscript