Scanning & Parsing Tools

- Scanning => lex
- Parsing => yacc

yacc

yacc – Unix tool (Bison – Window version)

- **Yet Another Compiler Compiler**

- LALR
- C code

A yacc grammar file has four main sections

```
%{
C declarations
%}

yacc declarations

%%
Grammar rules
%%

Additional C code
```

contains declarations that define terminal and nonterminal symbols, specify precedence, and so on.

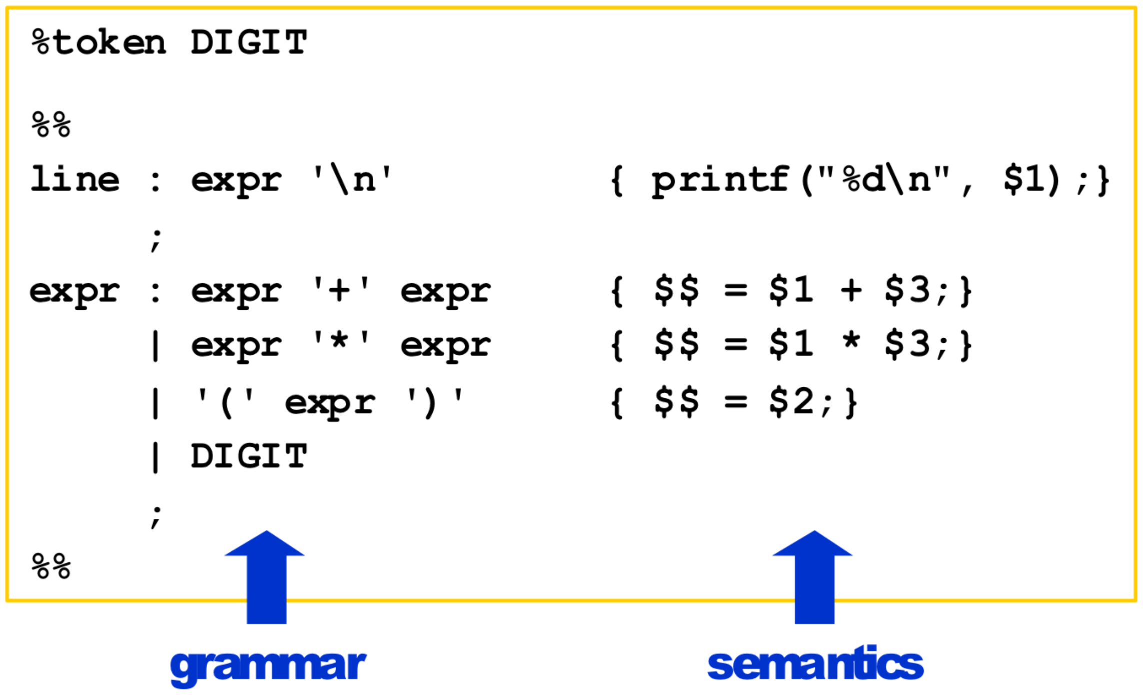
The grammar rules section

- contains one or more yacc grammar rules of the following general form:

```
result: components...     {C statements}

        ;

exp:      exp '+' exp
  ;

result:    rule1-components...
      | rule2-components...
      ...
      ;
result:                /*empty */
      | rule2-components...
      ;
```

Example: expression interpreter

- input



```
%token DIGIT

%%
line : expr '\n'          { printf("%d\n", $1);}
     ;
expr : expr '+' expr      { $$ = $1 + $3;}
     | expr '*' expr      { $$ = $1 * $3;}
     | '(' expr ')'       { $$ = $2;}
     | DIGIT
     ;
%%
```

- Yacc has a stack of values - referenced ‘$i’ in semantic actions

- Input file (desk0)

```
%%
line : expr '\n'            { printf("%d\n", $1);}
     ;
expr : expr '+' expr        { $$ = $1 + $3;}
     | expr '*' expr        { $$ = $1 * $3;}
     | '(' expr ')'         { $$ = $2;}
     | DIGIT
     ;
```

```
> make desk0
bison -v desk0.y
desk0.y contains 4 shift/reduce conflicts.
gcc -o desk0 desk0.tab.c
>
```

Conflict resolution in yacc

- Conflict **shift-reduce** – prefer **shift**

- Conflict **reduce-reduce** – chose first production

```
%%
line : expr '\n'            { printf("%d\n", $1);}
     ;
expr : expr '+' expr        { $$ = $1 + $3;}
     | expr '*' expr        { $$ = $1 * $3;}
     | '(' expr ')'         { $$ = $2;}
     | DIGIT
     ;
%%
```

- Run yacc
- Run desk0

```
> desk0
2*3+4
14
```

Operator priority in yacc

- From low to great

```
%token DIGIT
%left '+'
%left '*'

%%
line : expr '\n'          { printf("%d\n", $1);}
     ;
expr : expr '+' expr      { $$ = $1 + $3;}
     | expr '*' expr      { $$ = $1 * $3;}
     | '(' expr ')'       { $$ = $2;}
     | DIGIT
     ;
%%
```

- Use

```
>lex spec.lxi
>yacc –d spec.y
>gcc lex.yy.c y.tab.c -o result –lfl
>result<InputProgram
```

- More on

  https://pubs.opengroup.org/onlinepubs/009695399/utilities/yacc.html

Example