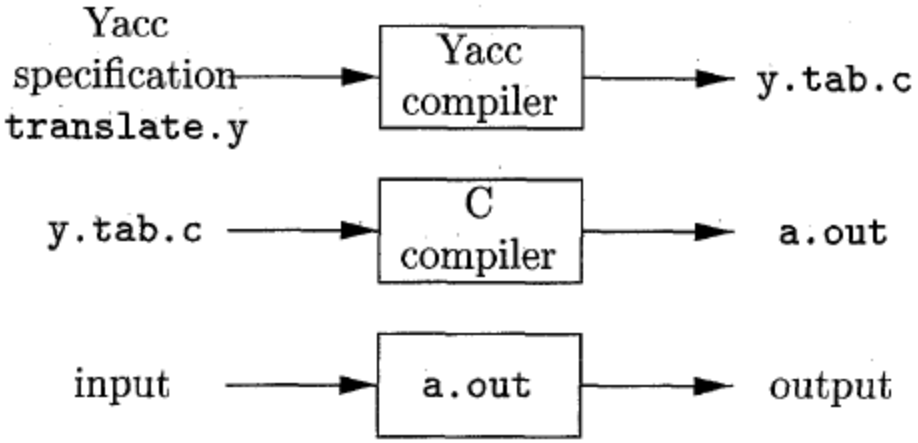
Project Part 2: Parser

Project Part 2: Parser



Command line: ***yacc translate.y***

Command line: ***yacc y.tab.c -ly***

yacc is one example, in this project we will use ***bison.***

Input to Bison

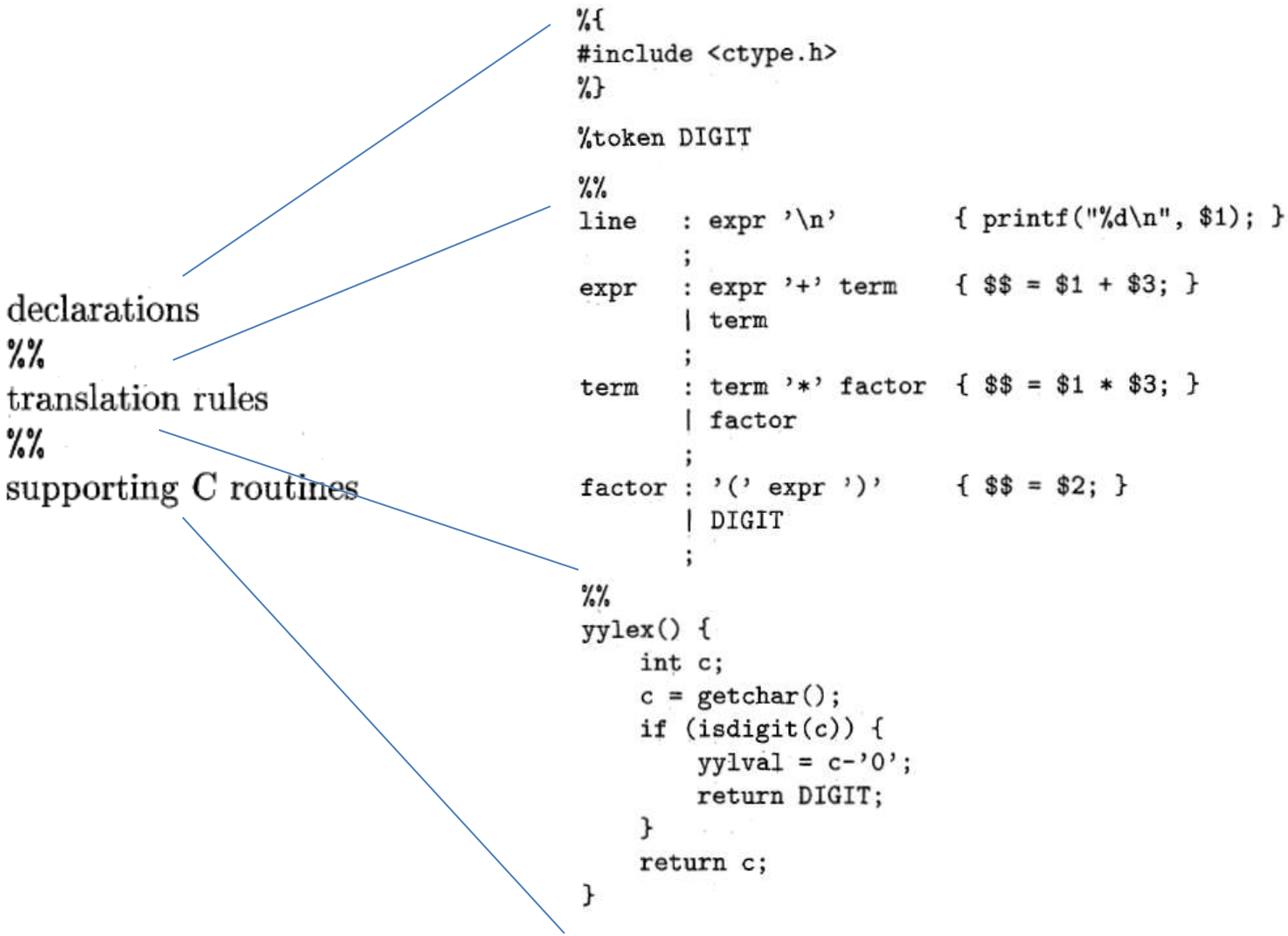
declarations
%%
translation rules
%%
supporting C routines

```
%{
#include <ctype.h>
%}

%token DIGIT

%%
line   : expr '\n'        { printf("%d\n", $1); }
       ;
expr   : expr '+' term    { $$ = $1 + $3; }
       | term
       ;
term   : term '*' factor  { $$ = $1 * $3; }
       | factor
       ;
factor : '(' expr ')'     { $$ = $2; }
       | DIGIT
       ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

Input to Bison




```
%{
#include <ctype.h>
%}

%token DIGIT

%%
line   : expr '\n'        { printf("%d\n", $1); }
       ;
expr   : expr '+' term     { $$ = $1 + $3; }
       | term
       ;
term   : term '*' factor   { $$ = $1 * $3; }
       | factor
       ;
factor : '(' expr ')'      { $$ = $2; }
       | DIGIT
       ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

Input to Bison

declarations
%%
translation rules
%%
supporting C routines

```
%{
#include <ctype.h>
%}

%token DIGIT

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

%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

semantic actions (sequence of C statements)

Default semantic action is $$=$1

unquoted strings not declared as tokens are taken as nonterminals

$$ attribute value of production head

$i attribute value of *ith* symbol in production body

Input to Bison

declarations
%%
translation rules
%%
supporting C routines

```
%{
#include <ctype.h>
%}

%token DIGIT

%%
line   : expr '\n'        { printf("%d\n", $1); }
       ;
expr   : expr '+' term    { $$ = $1 + $3; }
       | term
       ;
term   : term '*' factor  { $$ = $1 * $3; }
       | factor
       ;
factor : '(' expr ')'     { $$ = $2; }
       | DIGIT
       ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

bison + flex

- First use flex to generate yy.lex.c without *main* function
- In input to bison, put #include “yy.lex.c” in the declaration part of the file
- *bison translate.y*
- The command line: *gcc y.tab.c –ly -ll*