Flex & Yacc

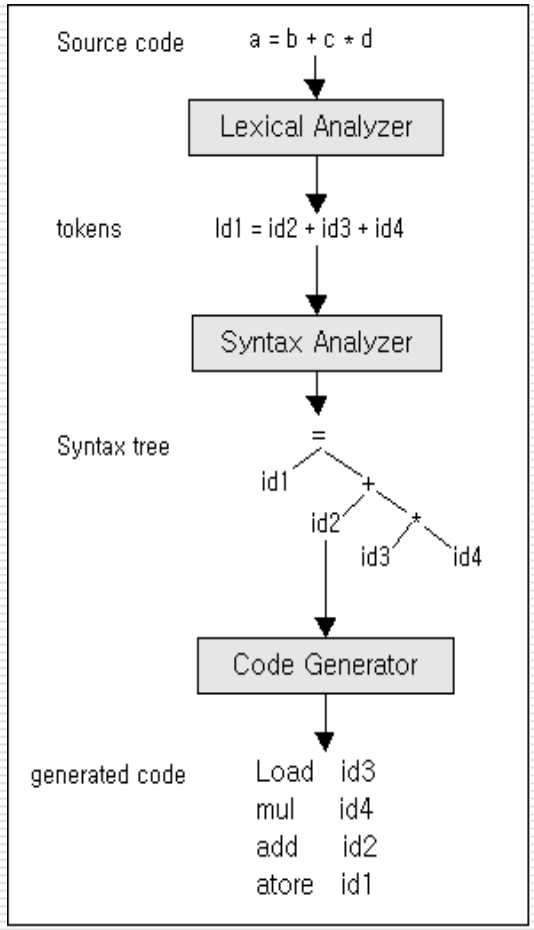
CS420
Keonchoel Shin

Flex

- Extended Lex by Vern Paxon
- Fast lexical analyzer generator
- Find matching string pattern and make token
- Input – rules
  - (regular expressions & C code)
- Output – lex.yy.c (yylex())
  - Compiled and linked with '-lfl' library

    #> flex ex1.l

    #> gcc lex.yy.c –o ex1 -lfl



Simple Examples

```
%{
#include <stdio.h>
%}
%%
stop printf("Stop command receive₩n");
start printf("Start command receive₩n");
%%
```

```
start program
Start command receive
 program
stopped
Stop command receive
ped
```

```
   int num_lines=0;
   int num_chars=0;
%%
₩n ++num_lines; ++num_chars;
.    ++num_chars;
%%
main(int argc, char** argv)
{
   if(argc>1)
     yyin = fopen(argv[1], “r”);
   yylex();
   printf(“# of lines=%d, #of
chars=%d₩n”, num_lines,
num_chars);
}
```

Format of Input File

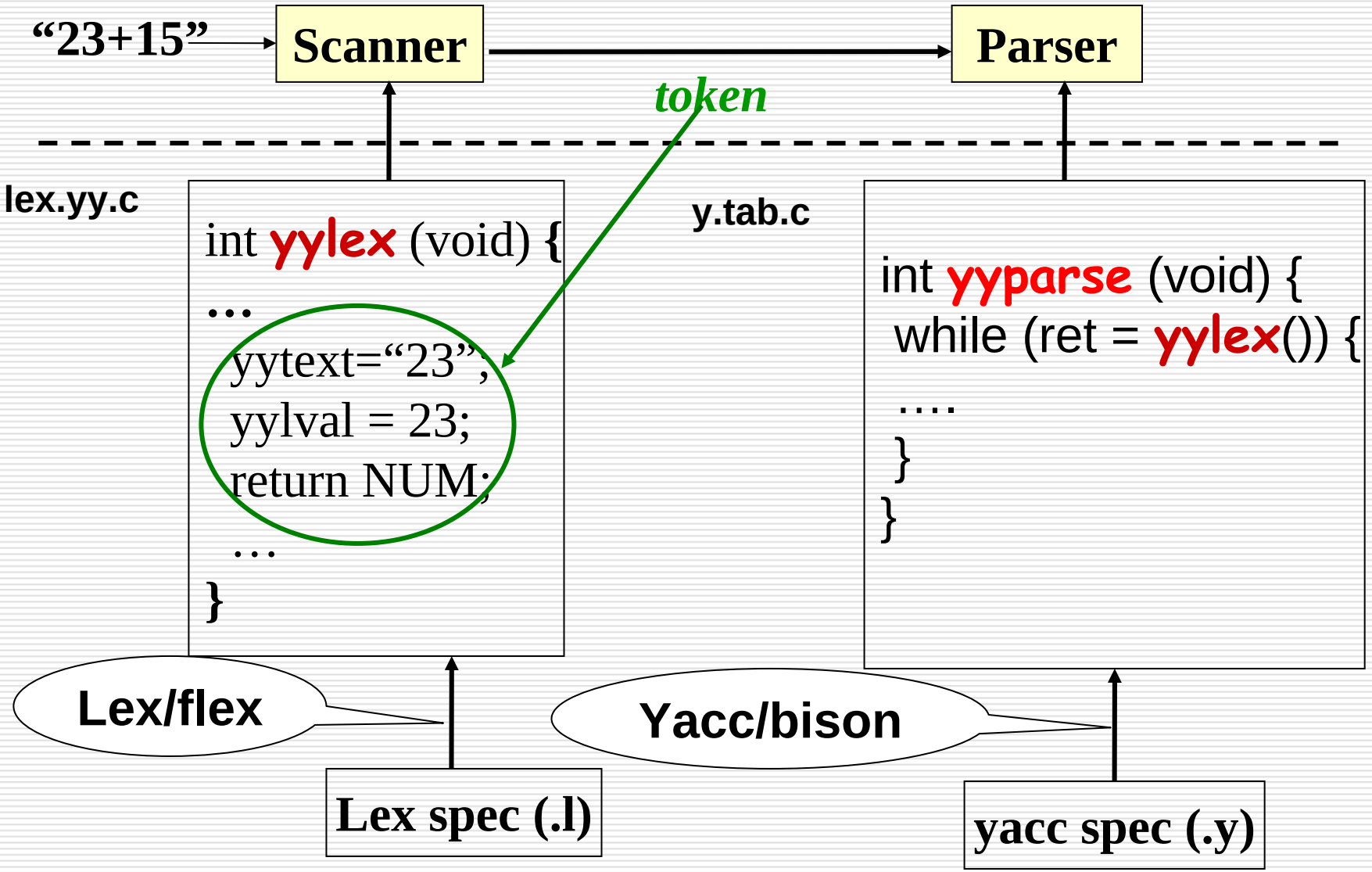
```
Definitions
%%
Rules
%%
User code
```

- Definitions (name definition)
  - Ex) DIGIT [0-9], ID [a-z][a-z0-9]*
  - Definition can be referred to using {name}
    - Ex) {DIGIT}+"."{DIGIT}*
  - any indented text or text enclosed in %{, %} is copied to the output
- Rules (pattern action)
  - Action must begin on the same line
  - Longest match
    - More applicable rules, favor 1st
- User code (optional)
  - Simply copied to the output

Patterns

x	{name}
.	(r)
[xyz]	rs
[abj-oZ]	r|s
[^A-Z\n]	r/s
r*	^r
r+	r$
r?	[\t\n]+

Scanner and Parser



Bison

- GNU ver. of Yet Another Compiler-Compiler(YACC)
- LALR(1) Parser Generator
- Check if the input is syntactically well-formed
- Input: Extended Backus-Naur Form (BNF)
  - Layout

    %{
    Prologue
    %}
    Bison declarations
    %%
    Grammar rules
    %%
    Epilogue
- Output – filename.tab.c, filename.tab.h (yyparse())
  - Need --defines (-d) option & --verbose (-v)

    #> bison -d -v test.y
    #> gcc test.tab.c lex.yy.c -o test –lfl
    or #> g++ test.tab.c lex.yy.c -o test –lfl

Simple Examples

calc.l

```
/* Definitions */
%{
#include <stdlib.h>
#include “calc.tab.h”
%}
NUMBER [0-9]+
%%
/* Rules */
{NUMBER} { yylval = atoi(yytext);
return NUMBER; }
“+”      { return PLUS; }
“*”      { return MULT; }
%%
/* User Code */
```

calc.y

```
/* Definitions */
%{
#include <stdio.h>
%}
%token NUMBER PLUS MULT
%%
/* Rules */
expr: NUMBER          { $$ = $1; }
    | expr PLUS expr  { $$ = $1 + $3; }
    | expr MULT expr  { $$ = $1 * $3; }
    ;
%%
/* User Code */
void yyerror(char *s) { printf(“%s₩n”, s); }
```

Ambiguity

- Specify associativity

```
%token NUMBER PLUS MULT
=>
%token NUMBER
%left PLUS
%left MULT
```

- Change the grammar rules

```
expr0: NUMBER           { $$ = $1; }
    ;
expr1: expr0            { $$ = $1; }
    | expr1 MULT expr0 { $$ = $1 * $3; }
    ;
expr2: expr1            { $$ = $1; }
    | expr2 PLUS expr1 { $$ = $1 + $3; }
    ;
```

- If else conflict can be OK in bison

Tips for Bison & Flex

- For Bison
  - Token and Nonterminal symbol have type
    - %union {
      Exp* exp;
      char* str;
      }
      %token <str> ID
      %type <exp> Expression
  - %prec change the precedence
    - %left '-'
      %left NEG
      expr: '-' expr %prec NEG

- For Flex
  - Single character can be used as token
  - yylval.str = strndup(yytext, yyleng);

SymbolTable

- use STL vector and map
- Class
  - id
  - Parent class id
  - Method table
  - Field table
- Method
  - id
  - Return type
  - isPublic
  - Local variable list ?
  - Parameter list
- Variable
  - id
  - isPublic
  - Type
- Need insert, query, print method

Requirements of Project

- Requirements
  - Non nested comments
  - Line number of error code
  - Field hiding
  - Method overriding
- Optional but extra points
  - Nested comments
  - Error recovery
  - Method overloading

Work Environment of Project

- cc1~cc4 server
  - ssh(22), sftp
  - should backup personally
- cd01~cd40 user account
  - Passwd: cs420
- Project webpage
  - Class webpage->class BBS
  - id: student number, passwd: 0000
  - All material about project will be posted

Output of project

- minijava.l
- minijava.y
- minijava program for test
- Symbol table files
  - result must be written in a file
- Readme file
  - optional requirements implemented
  - anything that TA should know