Project #2. Parser

Parser

- **C-Minus Parser Implementation**

  Implement the parser using Yacc (bison)

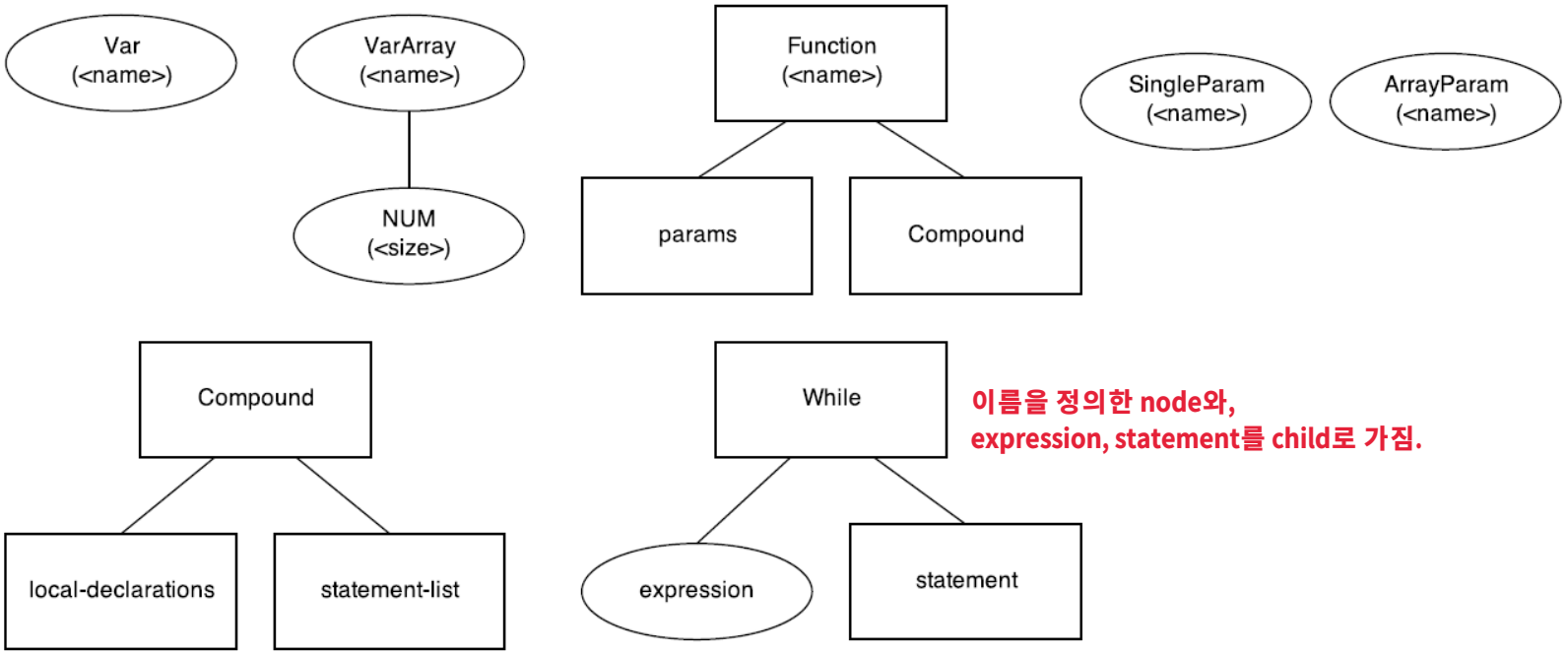
  C-Minus Scanner with Flex should be used.

  Some source code should be obtained using Yacc (bison)

  저번 과제에서 구현한 c minus Flex를 사용함.

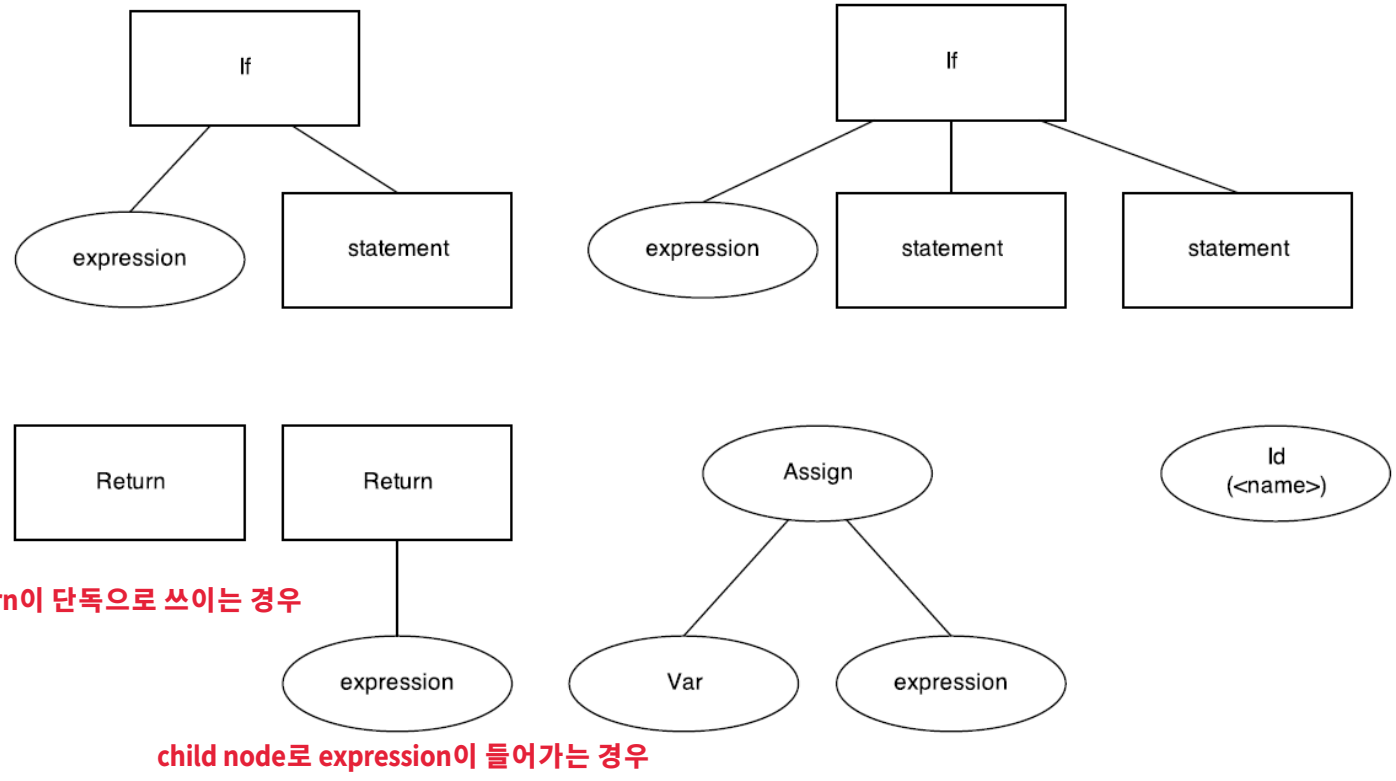
Parser Goal

- **Syntax Tree Definition**



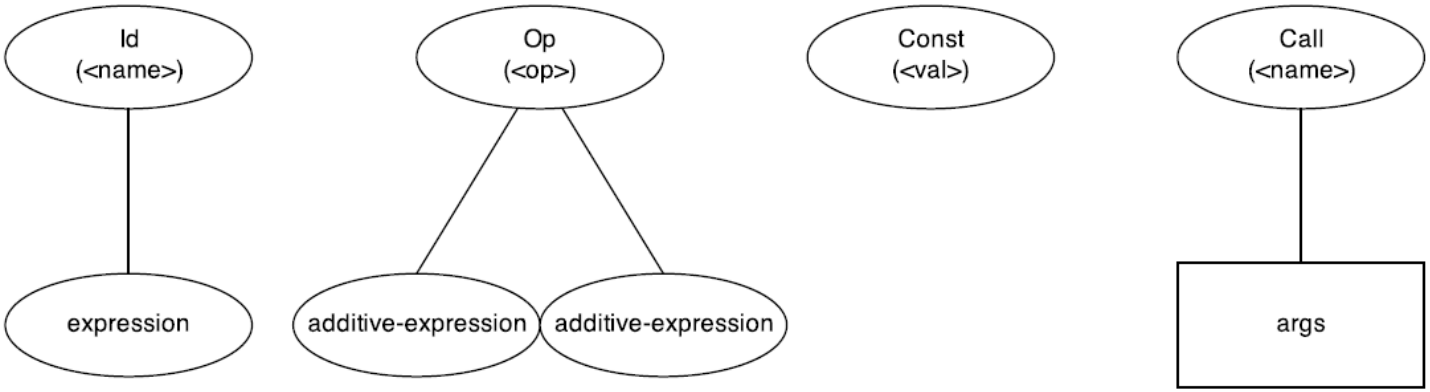
Parser Goal

- **Syntax Tree Definition**



Parser Goal

- **Syntax Tree Definition**



Hanyang University
Division of Computer Science & Engineering

BNF Grammar for C-Minus

- **Appendix A.2**

1. *program* → *declaration-list*
2. *declaration-list* → *declaration-list declaration* | *declaration*
3. *declaration* → *var-declaration* | *fun-declaration*
4. *var-declaration* → *type-specifier* **ID ;** | *type-specifier* **ID [NUM] ;**
5. *type-specifier* → **int** | **void**
6. *fun-declaration* → *type-specifier* **ID (** *params* **)** *compound-stmt*
7. *params* → *param-list* | **void**
8. *param-list* → *param-list* **,** *param* | *param*
9. *param* → *type-specifier* **ID** | *type-specifier* **ID [ ]**
10. *compound-stmt* → **{** *local-declarations statement-list* **}**
11. *local-declarations* → *local-declarations var-declarations* | *empty*
12. *statement-list* → *statement-list statement* | *empty*
13. *statement* → *expression-stmt* | *compound-stmt* | *selection-stmt* | *iteration-stmt* | *return-stmt*
14. *expression-stmt* → *expression* **;** | **;**
15. *selection-stmt* → **if (** *expression* **)** *statement* | **if (** *expression* **)** *statement* **else** *statement*
16. *iteration-stmt* → **while (** *expression* **)** *statement*
17. *return-stmt* → **return ;** | **return** *expression* **;**
18. *expression* → *var* **=** *expression* | *simple-expression*
19. *var* → **ID** | **ID [** *expression* **]**
20. *simple-expression* → *additive-expression relop additive-expression* | *additive-expression*
21. *relop* → **<=** | **<** | **>** | **>=** | **==** | **!=**
22. *additive-expression* → *additive-expression addop term* | *term*
23. *addop* → **+** | **-**
24. *term* → *term mulop factor* | *factor*
25. *mulop* → ***** | **/**
26. *factor* → **(** *expression* **)** | *var* | *call* | **NUM**
27. *call* → **ID (** *args* **)**
28. *args* → *arg-list* | *empty*
29. *arg-list* → *arg-list* **,** *expression* | *expression*

Hanyang University
Division of Computer Science & Engineering

Dangling Else Problem

- Ambiguous(Conflict) in 13, 15

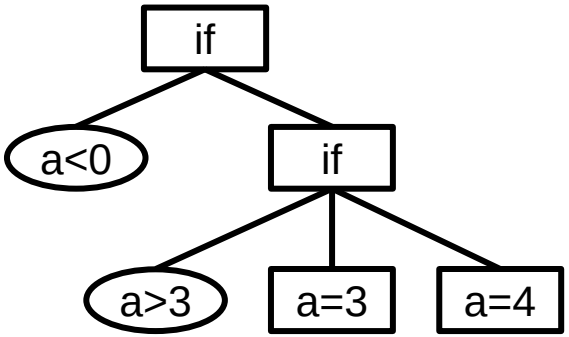
```
/* dangling else example */
void main(void) { if( a < 0 )   if ( a > 3 ) a = 3;   else a = 4; }
```

(1) void main(void) { if(a < 0) if (a > 3) a = 3; else a = 4; }

(2) void main(void) { if(a < 0) if (a > 3) a = 3; else a = 4; } 아래가 맞음

(2)

- Rule: Associate the else with the nearest if



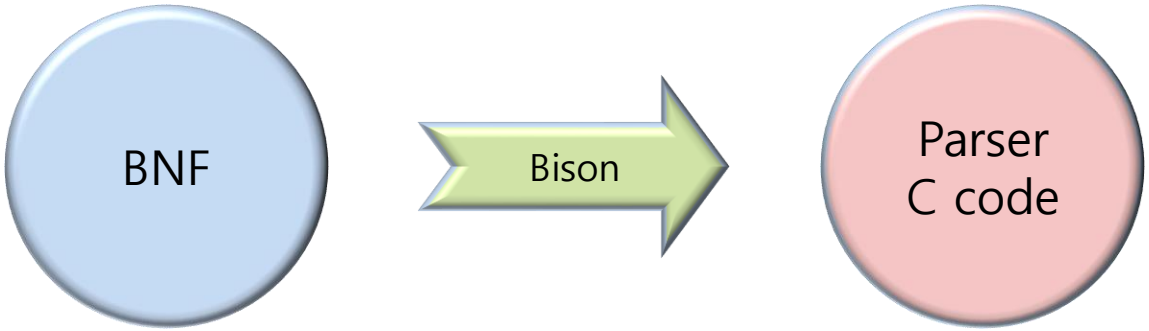

```
C-MINUS COMPILATION: test.cm

Syntax tree:
  Function declaration, name : main, return type : void
    Single parameter, name : (null), type : void
    Compound statement :
      If (condition) (body)
        Op : <
          Id : a
          Const : 0
        If (condition) (body) (else)
          Op : >
            Id : a
            Const : 3
          Assign : (destination) (source)
            Id : a
            Const : 3
          Assign : (destination) (source)
            Id : a
            Const : 4
```

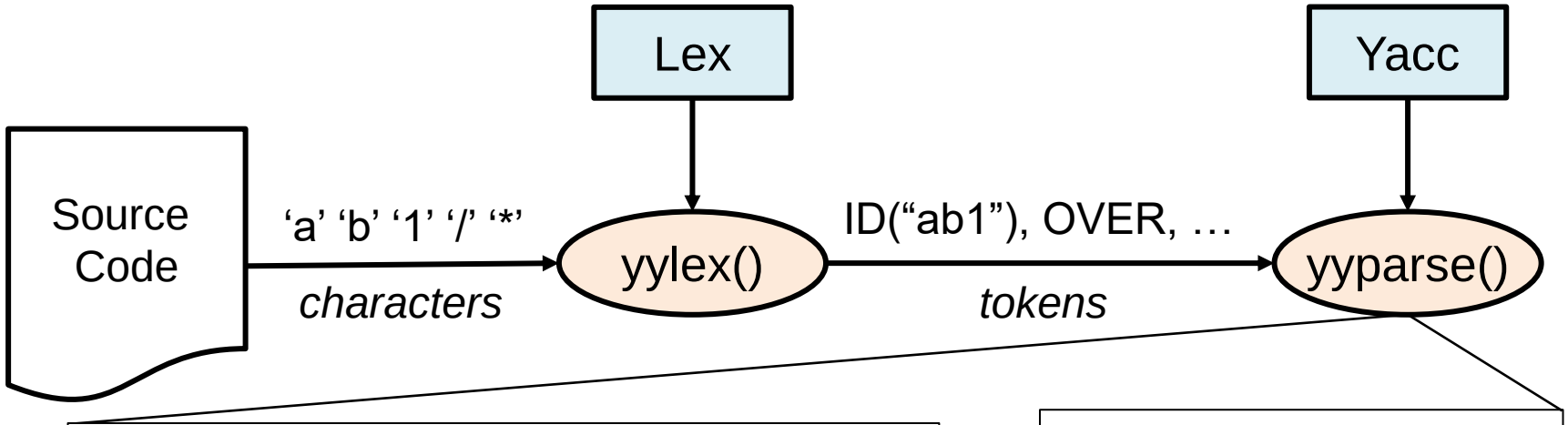
Yacc (bison)

- **Yacc: Parser generator for UNIX**
  - Yet Another Compiler Compiler
  - Bison: GNU Project parser generator (yacc replacement)
- **Input BNF** 을 입력하면
- **Output: C-code of parser for the input BNF**

Parser형태의 c code가 생성된다.

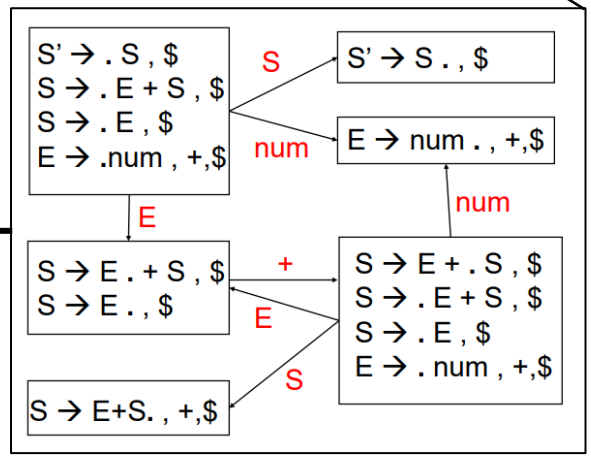


Yacc (bison), LALR(1) Parser



derivation	stack	input stream	action
(1+2+(3+4))+5		(1+2+(3+4))+5	shift
(1+2+(3+4))+5	(	1+2+(3+4))+5	shift
(1+2+(3+4))+5	(1	+2+(3+4))+5	reduce E→ num
(E+2+(3+4))+5	(E	+2+(3+4))+5	reduce S→ E
(S+2+(3+4))+5	(S	+2+(3+4))+5	shift
(S+2+(3+4))+5	(S+	2+(3+4))+5	shift
(S+2+(3+4))+5	(S+2	+(3+4))+5	reduce E→ num
(S+E+(3+4))+5	(S+E	+(3+4))+5	reduce S → S+E
(S+(3+4))+5	(S	+(3+4))+5	shift
(S+(3+4))+5	(S+	(3+4))+5	shift
(S+(3+4))+5	(S+(	3+4))+5	shift
(S+(3+4))+5	(S+(3	+4))+5	reduce E→ num

Shift-Reduce Parsing



Parsing Table

Yacc (bison) source description

Definitions

Tokens (Priority, Associativity)

%%

Rules (BNF syntax)

Parsing Rules with C/C++ Codes
($$, $1, … are the pointers to YYSTYPE objects)

%%

~~Subroutines~~

(You don't need to modify this part)

Yacc (bison) source example – tiny.y

- **rules**

```
  $$       $1 $2 $3   $4   $5
if_stmt     : IF exp THEN stmt_seq END
                { $$ = newStmtNode(IfK);
                  $$->child[0] = $2;
                  $$->child[1] = $4;
                }
            | IF exp THEN stmt_seq ELSE stm
                { $$ = newStmtNode(IfK);
                  $$->child[0] = $2;
                  $$->child[1] = $4;
                  $$->child[2] = $6;
                }
            ;
```

```
State 30

   10 if_stmt: IF exp THEN . stmt_seq END
   11        | IF exp THEN . stmt_seq ELSE stmt_seq END

    error   shift, and go to state 1
    IF      shift, and go to state 2
    REPEAT  shift, and go to state 3
    READ    shift, and go to state 4
    WRITE   shift, and go to state 5
    ID      shift, and go to state 6

    stmt_seq     go to state 41
    stmt         go to state 9
    if_stmt      go to state 10
    repeat_stmt  go to state 11
    assign_stmt  go to state 12
    read_stmt    go to state 13
    write_stmt   go to state 14
```

```
State 51

   10 if_stmt: IF exp THEN stmt_seq END .

    $default  reduce using rule 10 (if_stmt)
```

Yacc (bison) source example – tiny.y

- **rules**

YYSTYPE* (treeNode*)

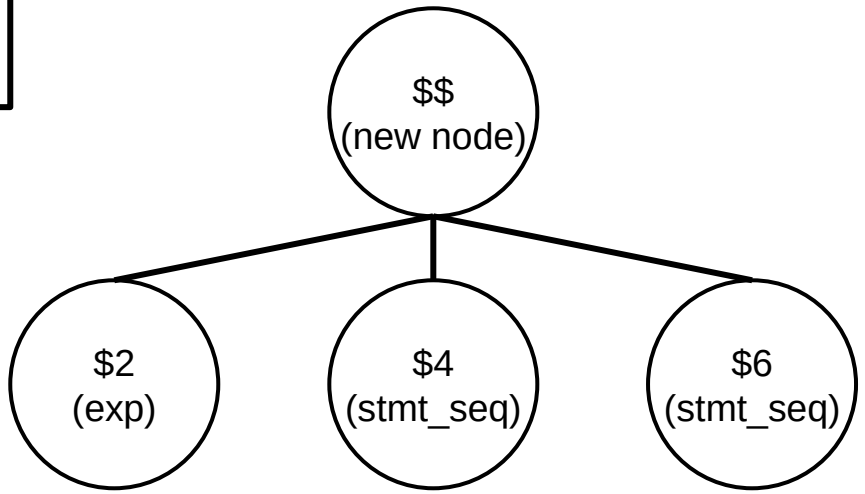
Pointer to if_stmt (non-terminal)

```
$$      $1 $2 $3   $4   $5
if_stmt : IF exp THEN stmt_seq END
            { $$ = newStmtNode(IfK);
              $$->child[0] = $2;
              $$->child[1] = $4;
            }
        | IF exp THEN stmt_seq ELSE stmt_seq END
            { $$ = newStmtNode(IfK);
              $$->child[0] = $2;
              $$->child[1] = $4;
              $$->child[2] = $6;
            }
        ;
```

$2 $4 $6

Executed at REDUCE

```
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp;} kind;
     union { TokenType op;
             int val;
             char * name; } attr;
     ExpType type; /* for type checking of exps */
   } TreeNode;
```



Yacc (bison) source example – tiny.y

- **YYSTYPE, savedTree, yylex()**

```
#define YYSTYPE TreeNode *
static char * savedName;
static int savedLineNo;
static TreeNode * savedTree;
static int yylex(void);
```

AST Node Type
(defined in globals.h)

AST root
(returned by parse())

```
program     : stmt_seq
                { savedTree = $1;}
```

Yacc (bison) source example – tiny.y

- **definitions**

```
%token IF THEN ELSE END REPEAT UNTIL READ WRITE
%token ID NUM
%token ASSIGN EQ LT PLUS MINUS TIMES OVER LPAREN RPAREN SEMI
%token ERROR
```

- Priority
  - Top Line < Bottom Line
- Associativity
  - **%left, %right, %noassoc** instead of **%token**
  - Example: *%left PLUS MINUS TIMES OVER*

Yacc (bison) Usage & Manual

Usage: yacc [options] filename

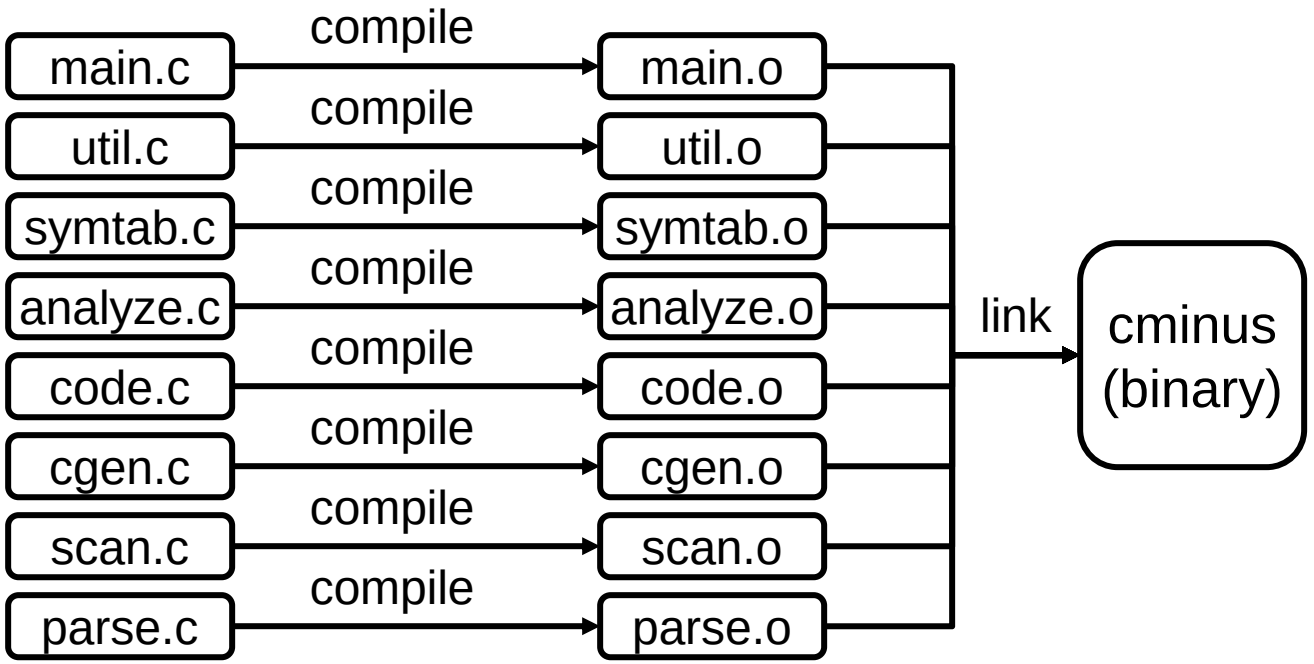
Options:

-d	**write definitions (y.tab.h)**
-o output_file	**(default "y.tab.c")**
-t	**add debugging support**
-v	**write description (y.output)**

- **Manual**
  http://www.gnu.org/software/bison/manual/ **(English)**

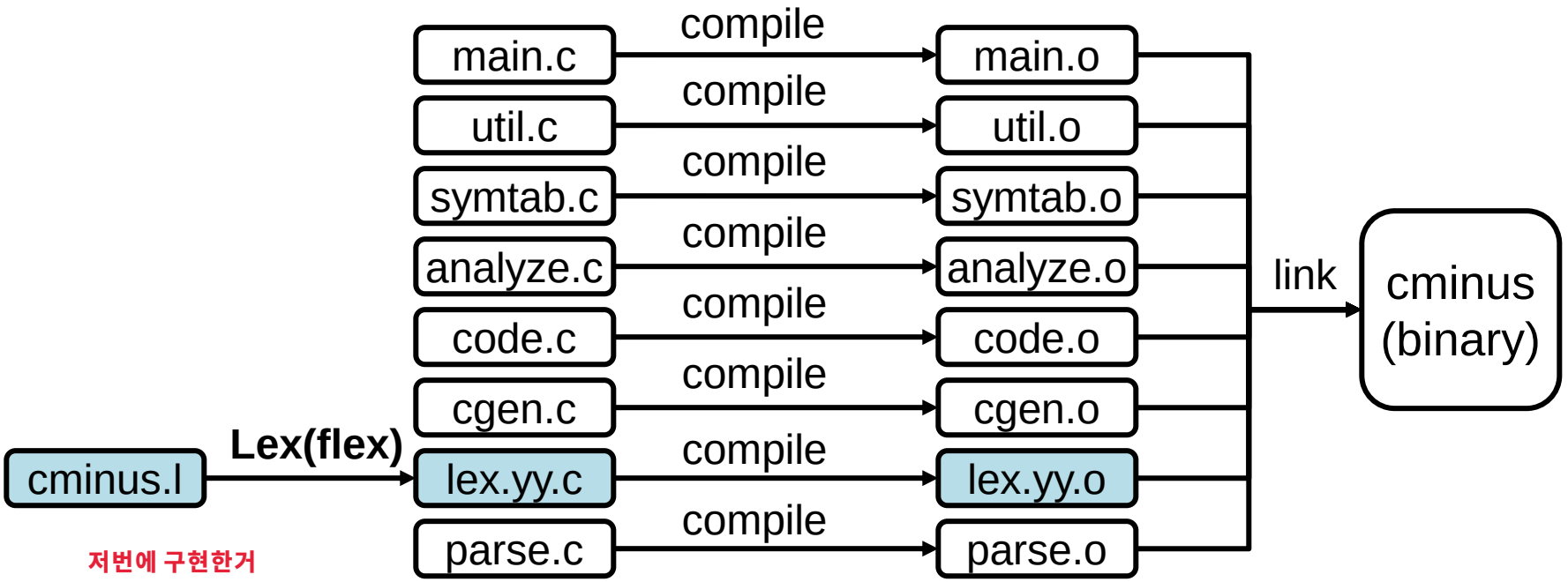
Hint: How to build?

- **Using c-implementation**
  (= original tiny compiler build structure)



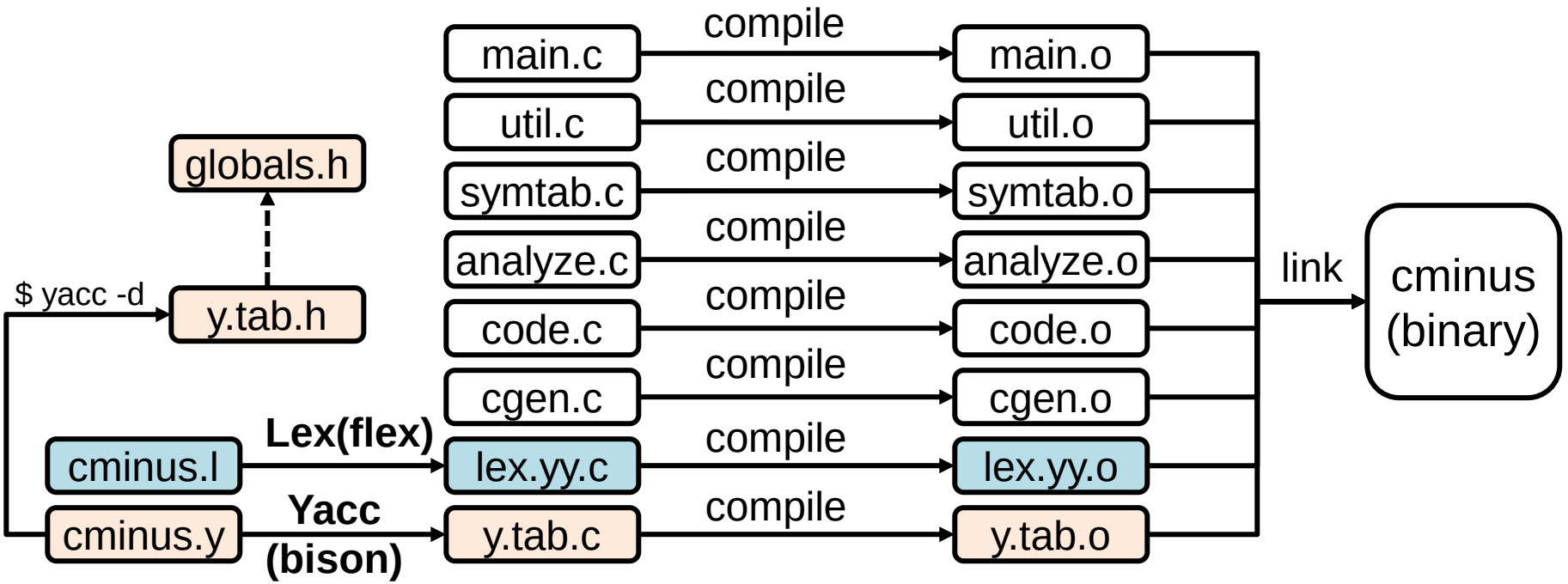
Hint: How to build?

- **Using Lex for scanner**



Hint: How to build?

- **Using Yacc for parser(this project)**



기존 flex 그대로 쓰고 Yacc도 사용.

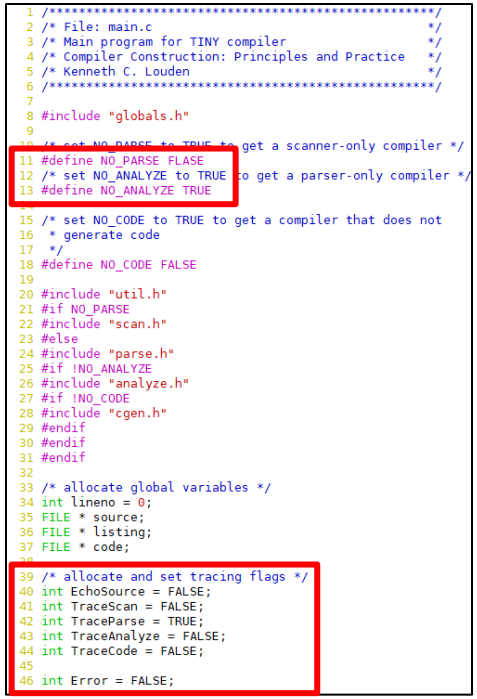
Hint: Build with Makefile

Makefile

```
1   # ./lex/tiny.l --> ./cminus.l
  1 # ./yacc/tiny.y --> ./cminus.y
  2 # ./yacc/globals.h --> ./globals.h
  3
  4 CC = gcc
  5 CFLAGS =
  6
  7 OBJS = main.o util.o lex.yy.o y.tab.o symtab.o analyze.o code.o cgen.o
  8
  9 all: cminus
 10
 11 cminus: $(OBJS)
 12     $(CC) $(CFLAGS) $(OBJS) -o $@ -lfl
 13
 14 main.o: main.c globals.h y.tab.h util.h scan.h parse.h analyze.h cgen.h
 15     $(CC) $(CFLAGS) -c main.c
 16
 17 util.o: util.c util.h globals.h y.tab.h
 18     $(CC) $(CFLAGS) -c util.c
 19
 20 lex.yy.c: cminus.l
 21     flex cminus.l
 22
 23 lex.yy.o: lex.yy.c globals.h y.tab.h util.h scan.h
 24     $(CC) $(CFLAGS) -c lex.yy.c
 25
 26 y.tab.c: cminus.y
 27     yacc -d -v cminus.y
 28
 29 y.tab.h: y.tab.c
 30
 31 y.tab.o: y.tab.c globals.h y.tab.h util.h scan.h parse.h
 32     $(CC) $(CFLAGS) -c y.tab.c
 33
 34 symtab.o: symtab.c symtab.h
 35     $(CC) $(CFLAGS) -c symtab.c
 36
 37 analyze.o: analyze.c globals.h y.tab.h symtab.h analyze.h
 38     $(CC) $(CFLAGS) -c analyze.c
 39
 40 code.o: code.c code.h globals.h y.tab.h
 41     $(CC) $(CFLAGS) -c code.c
 42
 43 cgen.o: cgen.c globals.h y.tab.h symtab.h code.h cgen.h
 44     $(CC) $(CFLAGS) -c cgen.c
 45
 46 clean:
 47     rm -vf $(OBJS) lex.yy.c y.tab.h y.tab.c cminus
 48
NORMAL  Makefile
"Makefile" 49L, 1038C
```

Hint: where to see?

- **main.c**
  - To modify code to print *only* Syntax Tree
  - NO_ANALYZE, TraceParse



```c
/****************************************************/
/* File: main.c                                     */
/* Main program for TINY compiler                   */
/* Compiler Construction: Principles and Practice   */
/* Kenneth C. Louden                                */
/****************************************************/

#include "globals.h"

/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE FLASE
/* set NO_ANALYZE to TRUE to get a parser-only compiler */
#define NO_ANALYZE TRUE

/* set NO_CODE to TRUE to get a compiler that does not
 * generate code
 */
#define NO_CODE FALSE

#include "util.h"
#if NO_PARSE
#include "scan.h"
#else
#include "parse.h"
#if !NO_ANALYZE
#include "analyze.h"
#if !NO_CODE
#include "cgen.h"
#endif
#endif
#endif

/* allocate global variables */
int lineno = 0;
FILE * source;
FILE * listing;
FILE * code;

/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = FALSE;
int TraceParse = TRUE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;

int Error = FALSE;
```

```c
/* set NO_PARSE to TRUE to ge
#define NO_PARSE FLASE
/* set NO_ANALYZE to TRUE to
#define NO_ANALYZE TRUE
```

```c
/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = FALSE;
int TraceParse = TRUE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;

int Error = FALSE;
```

Hint: where to see?

- **util.c**
  - printTree function should be updated to print C-Minus Syntax Tree
- **globals.h**
  - Overwrite your globals.h with yacc/globals.h
  - "Syntax tree for parsing" should be updated to meet C-Minus Spec
- **yacc/tiny.y**
  - Baseline of cminus.y
- **Other files(analyze.c, cgen.c, …)**
  - If need

Example (Syntax tree)

```
/* A program to perform Euclid's
   Algorithm to computer gcd */

int gcd (int u, int v)
{
    if (v == 0) return u;
    else return gcd(v,u-u/v*v);
    /* u-u/v*v == u mod v */
}

void main(void)
{
    int x; int y;
    x = input(); y = input();
    output(gcd(x,y));
}
```

```
C-MINUS COMPILATION: ./test/test.1.cm

Syntax tree:
  Function declaration, name : gcd, return type : int
    Single parameter, name : u, type : int
    Single parameter, name : v, type : int
    Compound statement :
      If (condition) (body) (else)
        Op : ==
          Id : v
          Const : 0
        Return :
          Id : u
        Return :
          Call, name : gcd, with arguments below
            Id : v
            Op : -
              Id : u
              Op : *
                Op : /
                  Id : u
                  Id : v
                Id : v
  Function declaration, name : main, return type : void
    Single parameter, name : (null), type : void
    Compound statement :
      Var declaration, name : x, type : int
      Var declaration, name : y, type : int
      Assign : (destination) (source)
        Id : x
        Call, name : input, with arguments below
      Assign : (destination) (source)
        Id : y
        Call, name : input, with arguments below
      Call, name : output, with arguments below
        Call, name : gcd, with arguments below
          Id : x
          Id : y
```

Some Comments

- **You don't need to generate exactly same output. If you generate the right result, it will be okay.**

- **You don't need to care about Semantics, just Syntax analyzer will be okay.**

Some Comments

```
/* Semantic Error Example */
/* (1) uninitialized variables a and b (2) undefined variable c */
int main ( void )
{
    int a;
    int b;
    c = a + b;
}
```

- **For this example, this code will be parsed correctly even though the code has some semantic error.**

Report

- **Guideline**
  - Compilation method and environment
  - Explanation about how to implement and how to operate
  - Some explanation about the modified code
  - Example and Result Screenshot

- **File format**
  - MS Word, HWP, PDF, …
  - GitLab Wiki Not Allowed
    (If you want, write report in markdown and **take screenshot** and submit in other formats(PDF, JPEG, …))

Submission

- **Submission directory in repository: 2_Parser**
  (Please submit all your codes and reports into the submission directory)

- **Questions**
  compiler.teachingassistant@gmail.com

- **Parser submission deadline**
  - **11/25 (wed) 23:59:59**

Contact (Prof. Yongjun Park)

- **Submission**
  - Where: Using GitLab
    - https://hconnect.hanyang.ac.kr
    - Git Project:
      https://hconnect.hanyang.ac.kr/2020_ELE4029_11784/2020_ELE4029_Student#.git
    - Example URL: https://hconnect.hanyang.ac.kr/2020_ELE4029_11784/2020_ELE4029_2018000000.git
    - The Submission Directory is in Repo: 1_Scanner, **2_Parser**, 3_Semantic, …
  - Teaching Assistant
    - **compiler.teachingassistant@gmail.com**
    - If you don't have the GITLAB account, please let him know the account information after creation.
  - **What to submit**
    - **All the source codes and the report**

Q&A