
Project #2. Parser

Parser

- **C-Minus Parser Implementation**

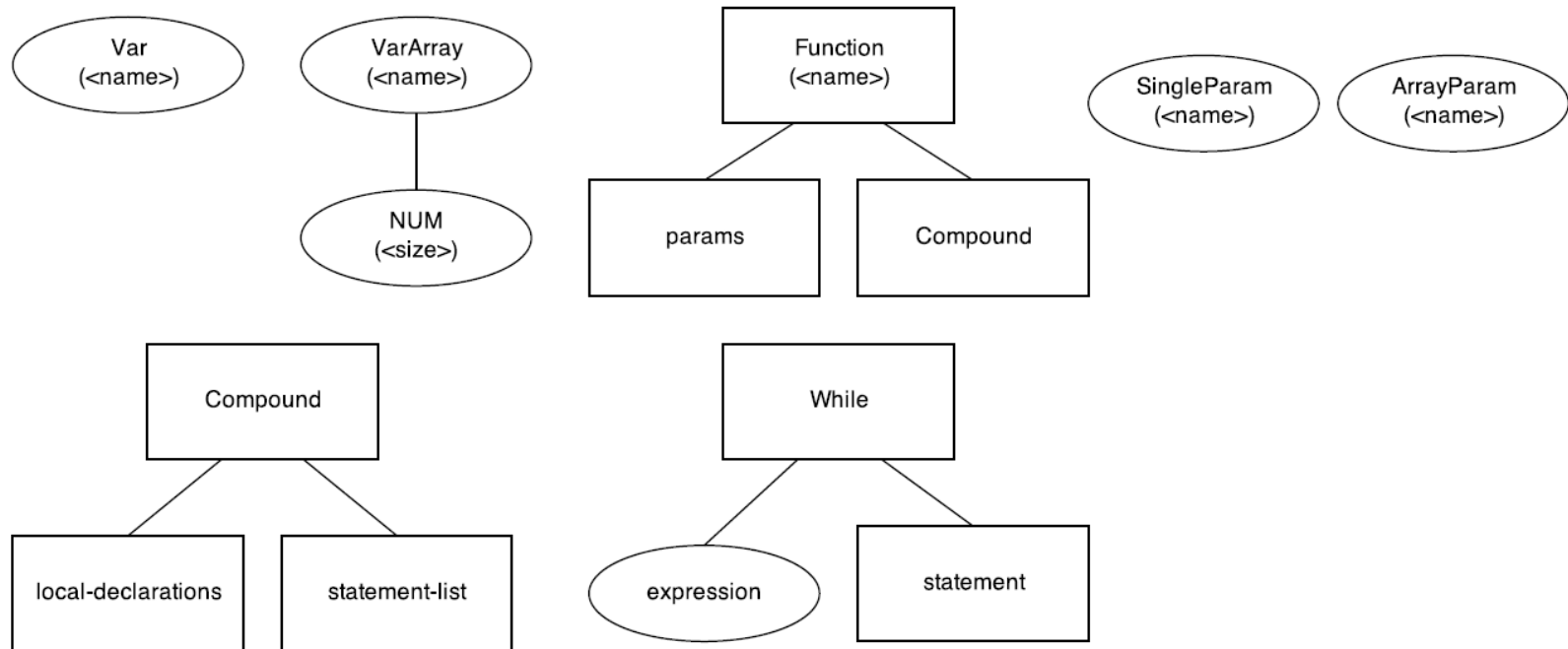
Implement the parser using Yacc (bison)

C-Minus Scanner with Flex should be used.

Some source code should be obtained using Yacc (bison)

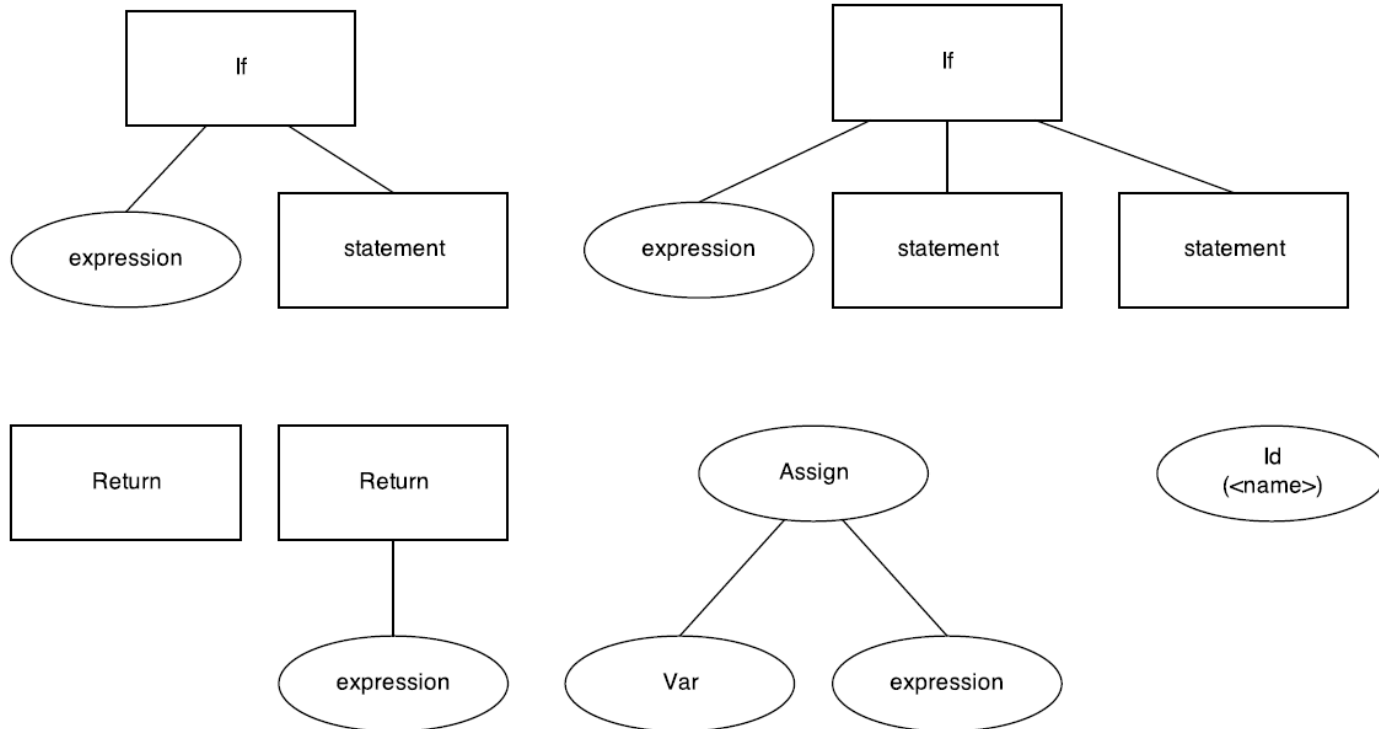
Parser Goal

- Syntax Tree Definition



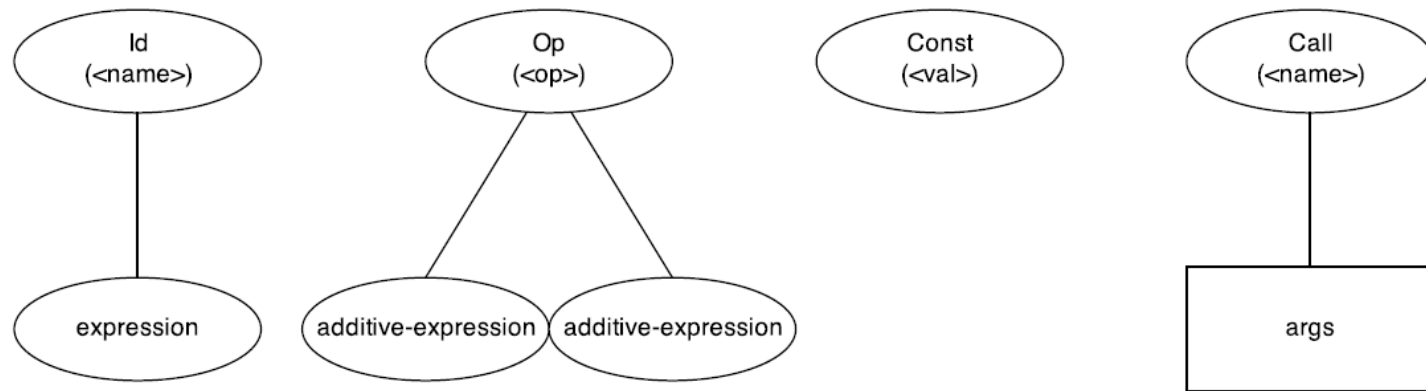
Parser Goal

- Syntax Tree Definition



Parser Goal

- Syntax Tree Definition



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*

Dangling Else Problem

- Ambiguous(Conflict) in 13, 15

```
/* dangling else example */
```

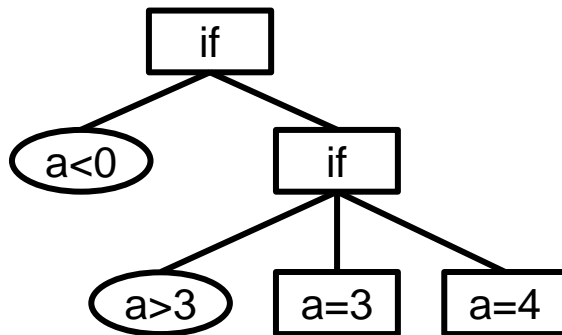
```
void main(void) { if( a < 0 ) if ( a > 3 ) a = 3; else a = 4; }
```

(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; }

(2)

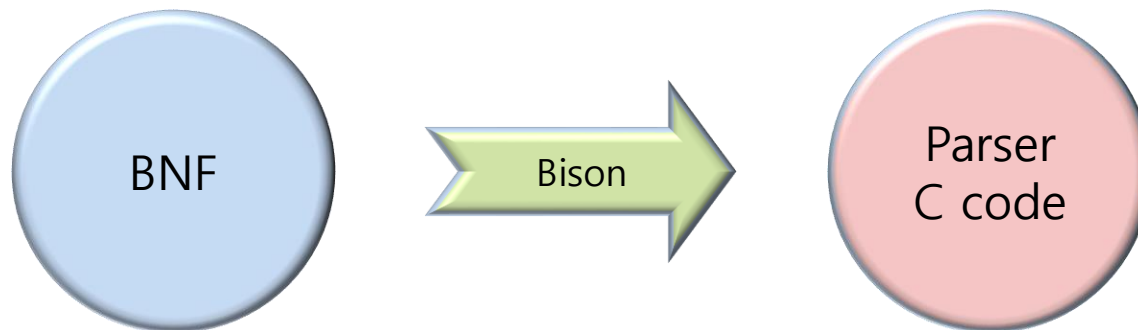
- Rule: Associate the else with the nearest if



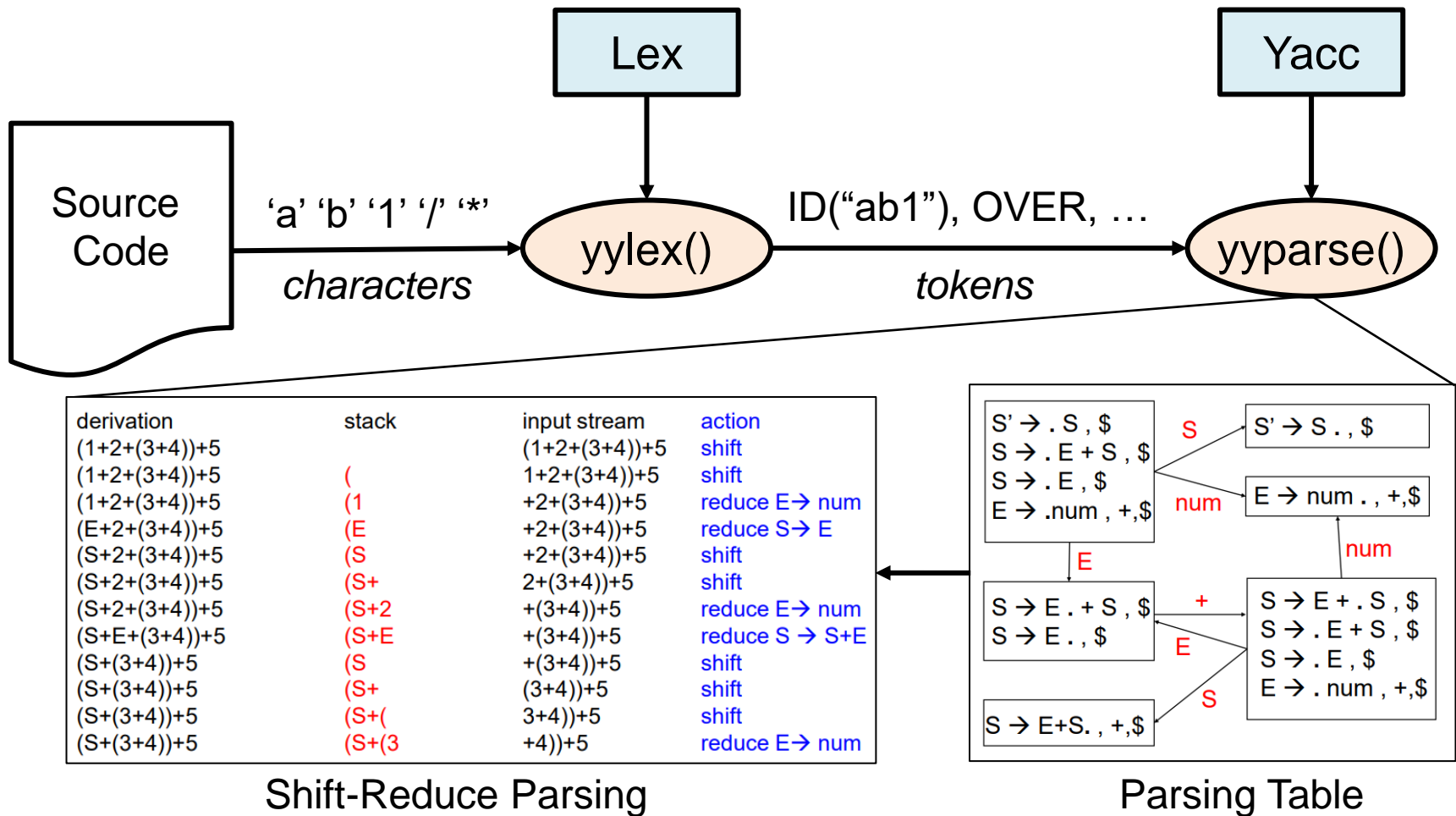
```
C-MINUS COMPILATION: test.cm
Syntax tree:
Function declaration, name : main, return type : void
Single parameter, name : (null), type : void
Compound statement :
If (condition) (body)
  Op : <
  Id : a
  Const : 0
If (condition) (body) (else)
  Op : >
  Id : a
  Const : 3
  Assign : (destination) (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**
- **Output: C-code of parser for the input BNF**



Yacc (bison), LALR(1) Parser



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(lfK);
            $$->child[0] = $2;
            $$->child[1] = $4;
          }
        | IF exp THEN stmt_seq ELSE stmt_seq END
          { $$ = newStmtNode(lfK);
            $$->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

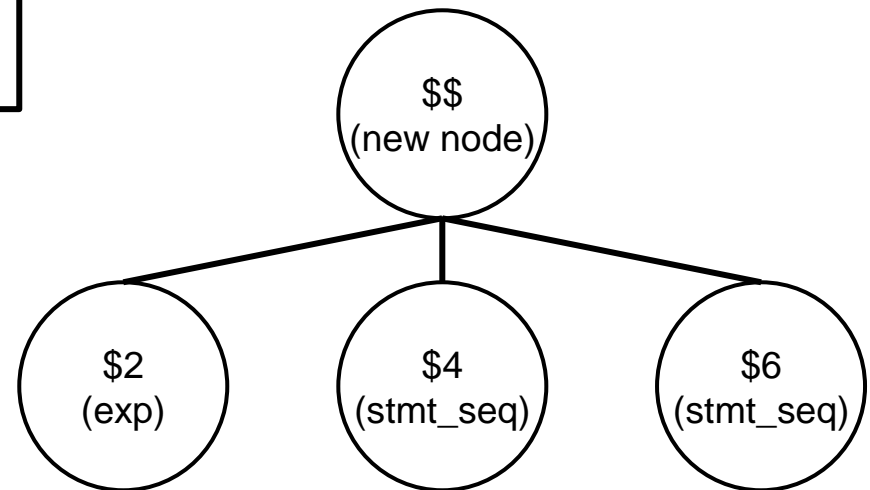
Pointer to if_stmt (non-terminal) → **\$\$**

YYSTYPE* (treeNode*) → **\$1 \$2 \$3 \$4 \$5**

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

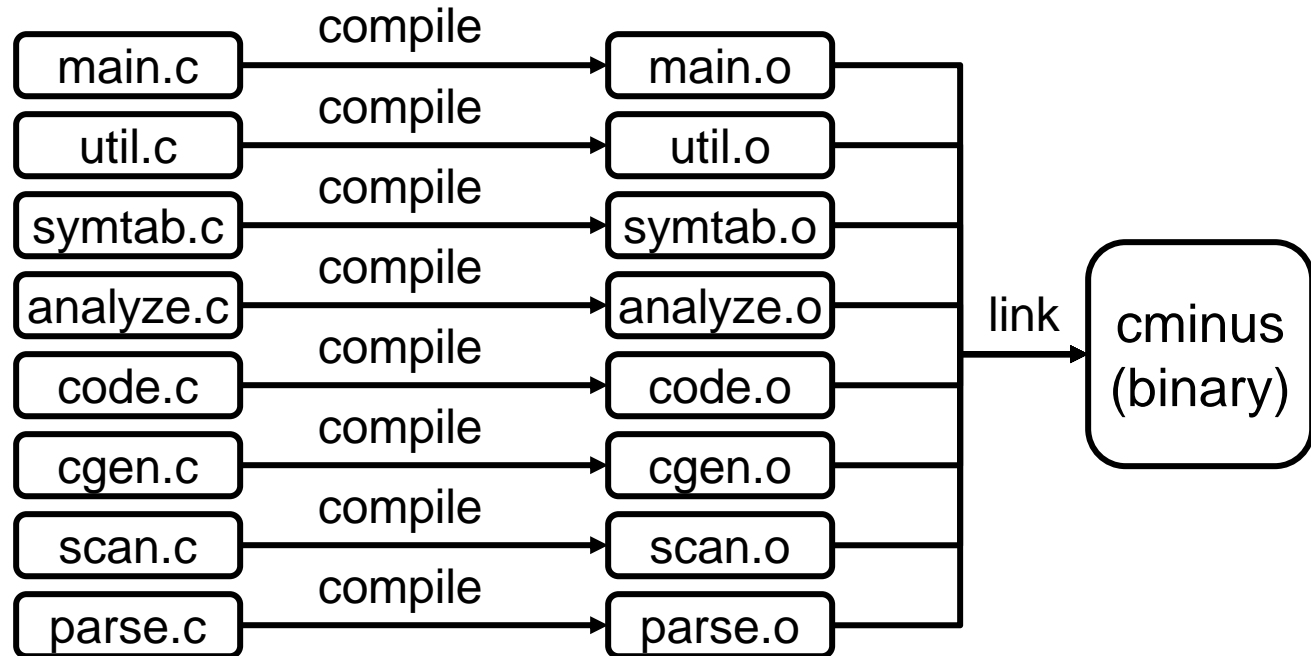
- Manual

<http://www.gnu.org/software/bison/manual/> (English)



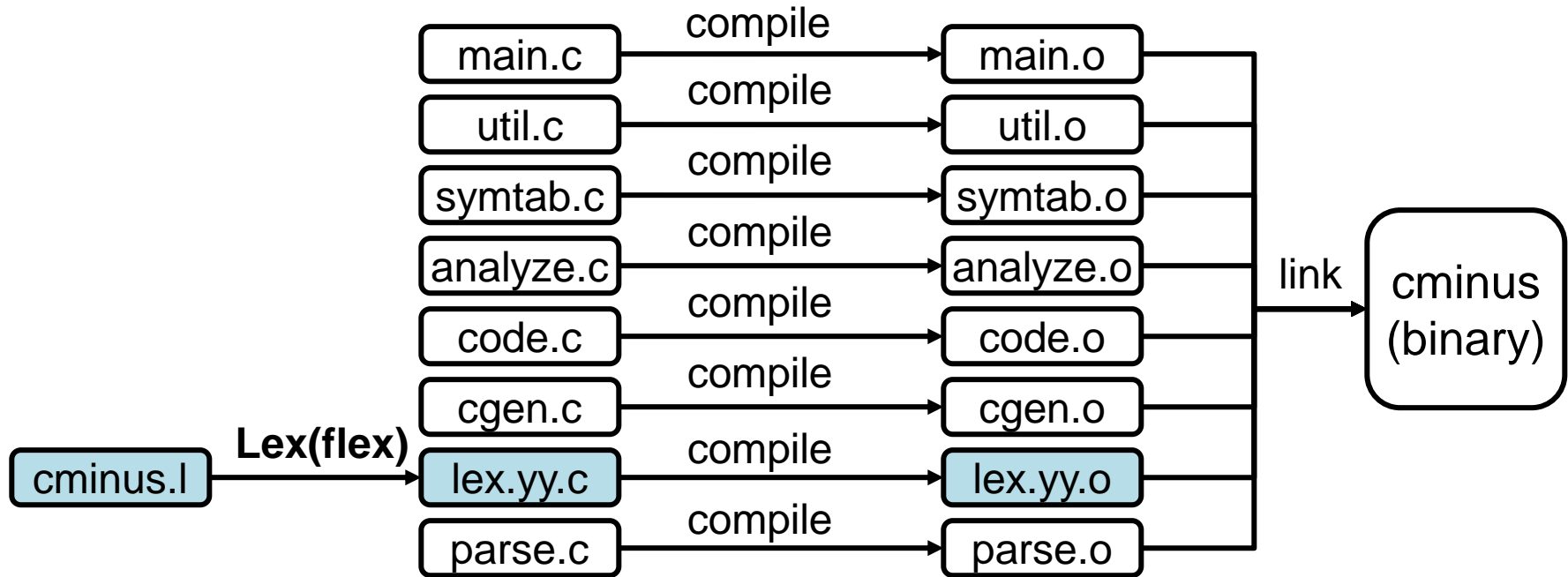
Hint: How to build?

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



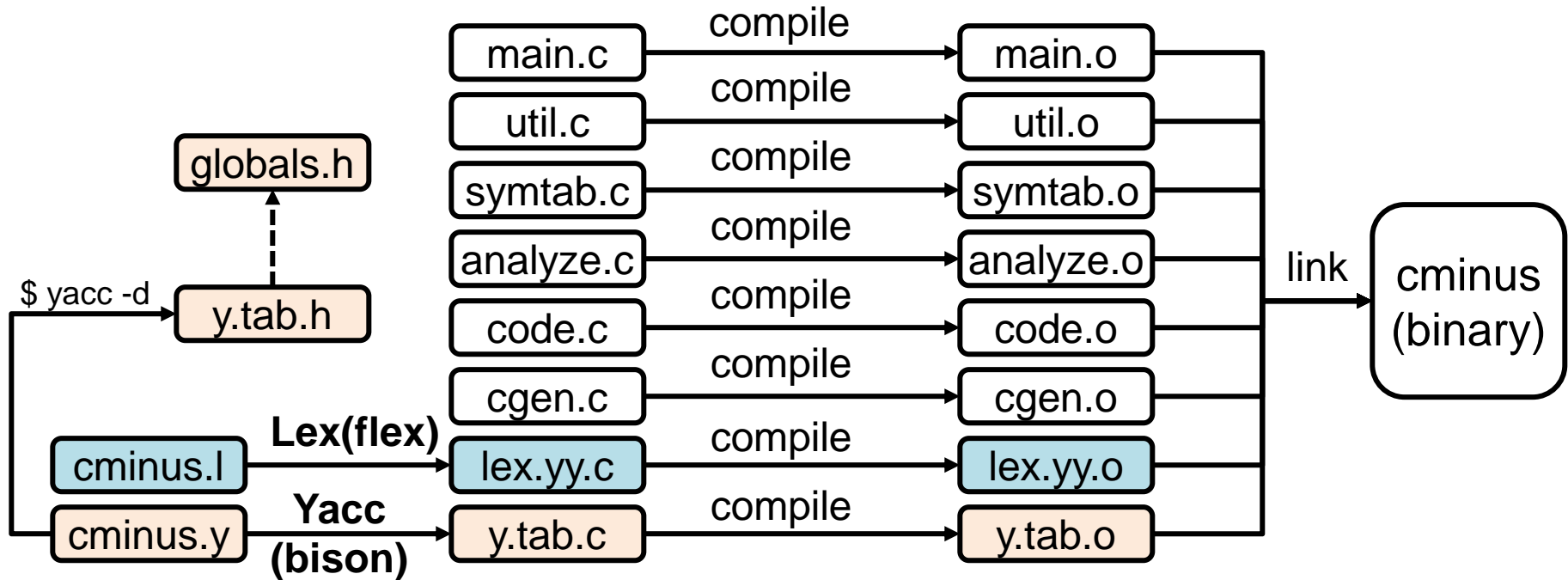
Hint: How to build?

- Using Lex for scanner



Hint: How to build?

- Using Yacc for parser(this project)



Hint: Build with Makefile

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



Hint: where to see?

- **main.c**

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

```
1 /******  
2 /* File: main.c  
3 /* Main program for TINY compiler  
4 /* Compiler Construction: Principles and Practice  
5 /* Kenneth C. Louden  
6 /******  
7  
8 #include "globals.h"  
9  
10 /* set NO_PARSE to TRUE to get a scanner-only compiler */  
11 #define NO_PARSE FALSE  
12 /* set NO_ANALYZE to TRUE to get a parser-only compiler */  
13 #define NO_ANALYZE TRUE  
14  
15 /* set NO_CODE to TRUE to get a compiler that does not  
16 * generate code  
17 */  
18 #define NO_CODE FALSE  
19  
20 #include "util.h"  
21 #if NO_PARSE  
22 #include "scan.h"  
23 #else  
24 #include "parse.h"  
25 #if !NO_ANALYZE  
26 #include "analyze.h"  
27 #if !NO_CODE  
28 #include "cgen.h"  
29 #endif  
30 #endif  
31 #endif  
32  
33 /* allocate global variables */  
34 int lineno = 0;  
35 FILE * source;  
36 FILE * listing;  
37 FILE * code;  
38  
39 /* allocate and set tracing flags */  
40 int EchoSource = FALSE;  
41 int TraceScan = FALSE;  
42 int TraceParse = TRUE;  
43 int TraceAnalyze = FALSE;  
44 int TraceCode = FALSE;  
45  
46 int Error = FALSE;
```

```
10 /* set NO_PARSE to TRUE to ge  
11 #define NO_PARSE FALSE  
12 /* set NO_ANALYZE to TRUE to  
13 #define NO_ANALYZE TRUE  
14
```

```
39 /* allocate and set tracing flags */  
40 int EchoSource = FALSE;  
41 int TraceScan = FALSE;  
42 int TraceParse = TRUE;  
43 int TraceAnalyze = FALSE;  
44 int TraceCode = FALSE;  
45  
46 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

