Project #2. Parser

2022 Compiler

Prof. Yongjun Park

Hanyang University
Division of Computer Science & Engineering

Project Goal: Parser

- **C-Minus Parser Implementation using *Yacc (Bison)***
  - The Parser reads an input source code string, tokenizes and parses it with C-Minus grammar, and returns (prints) **abstract syntax tree (AST)**.
  - C-Minus scanner with LEX should be used.
    - Using Source codes that uploaded.
  - Some source code should be obtained using Yacc.
    - Yacc takes a grammar in BNF form as input and generate a LALR(1) parser.
    - Ambiguous grammar will cause conflicts.
  - *cminus.y, … -> cminus_parser*

BNF Grammar for C-Minus

- **Implement in cminus.y**

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

- Ambiguity in the grammar *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;** }

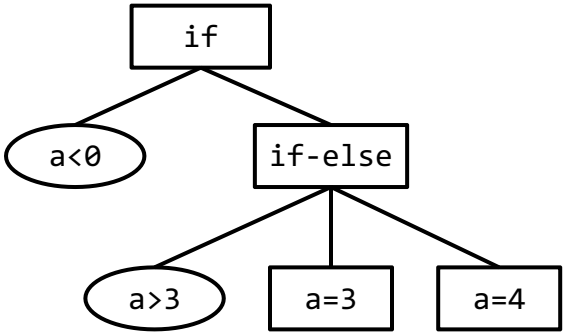
(2) void main(void) { if(a < 0) **if (a > 3) a = 3; else a = 4;** }

(2)

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

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      If Statement:
        Op: <
          Variable: name = a
          Const: 0
      If-Else Statement:
        Op: >
          Variable: name = a
          Const: 3
        Assign:
          Variable: name = a
          Const: 3
        Assign:
          Variable: name = a
          Const: 4
```

- Rule: Associate the else with the nearest if



Hanyang University
Division of Computer Science & Engineering

Project Goal: AST and Output Format

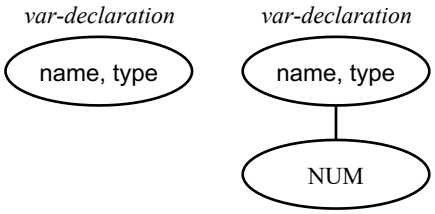
*** Type (*type-specifier, ...*)**

```
<Format: Type>
(type = )int
(type = )void
(type = )int[]
(type = )void[]
```

*** Operator (*relop, addop, mulop*)**

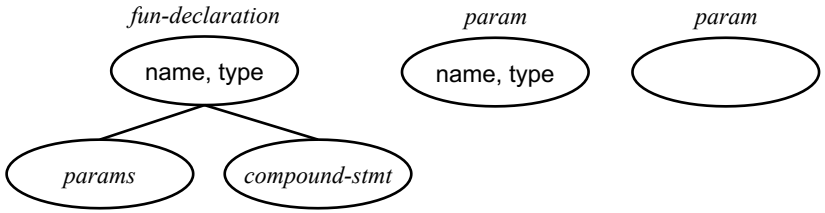
```
<Format: Operator (used in Binary Operator Expression>
+
-
*
/
<=
!=
…
```

*** Variable Declaration (*var-declaration*)**




```
<Format: Variable Declaration>
Variable Declaration: name = %s, type = %s
  /* Child Node: Array Size */
```

*** Function Declaration (*fun-declaration*)**

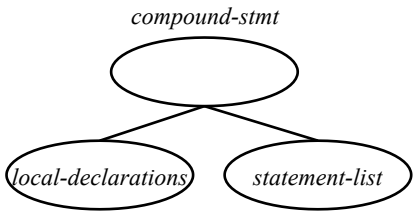



```
<Format: Function Declaration>
Function Declaration: name = %s, return type = %s
  /* Child Node: Parameters */
  /* Child Node: Compound Statement */

<Format: Parameters>
Parameter: name = %s, type = %s
Void Parameter
```

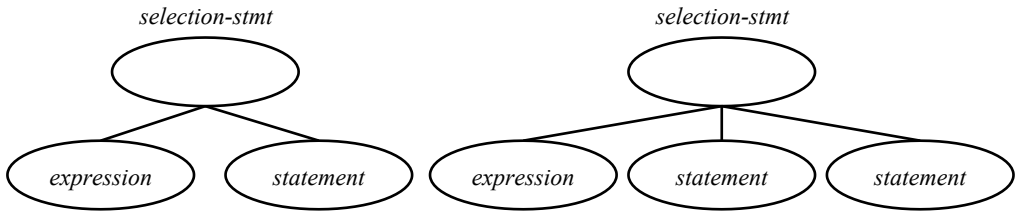
Project Goal: AST and Output Format

*** Compound Statement (*compound-stmt*)**



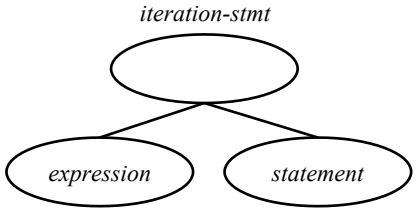

```
<Format: Compound Statement>
Compound Statement:
  /* Child Node: Local Declarations */
  /* Child Node: Statement Lists */
```

*** If/If-Else Statement (*selection-stmt*)**



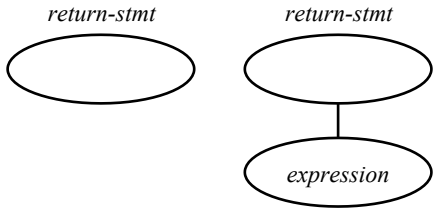

```
<Format: If/If-Else Statement>
If Statement:
  /* Child Node: Condition Expression */
  /* Child Node: Then-Statement */
If-Else Statement:
  /* Child Node: Condition Expression */
  /* Child Node: Then-Statement */
  /* Child Node: Else-Statement */
```

*** While Statement (*iteration-stmt*)**




```
<Format: While Statement>
While Statement:
  /* Child Node: Condition Expression */
  /* Child Node: Loop Body Statement */
```

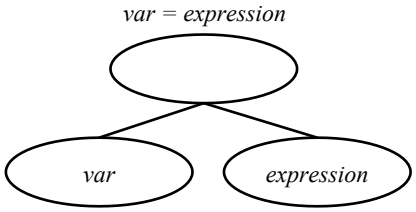
*** Return Statement (*return-stmt*)**




```
<Format: Return Statement>
Non-value Return Statement
Return Statement:
  /* Child Node: Return Expression */
```

Project Goal: AST and Output Format

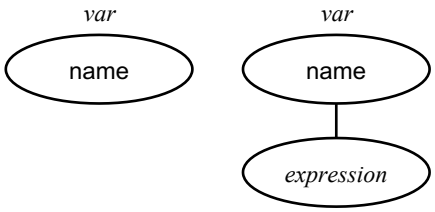
*** Assignment Expression (*var = expression*)**



<Format: Assignment Expression>

```
Assign:
  /* Child Node: Variable */
  /* Child Node: Expression */
```

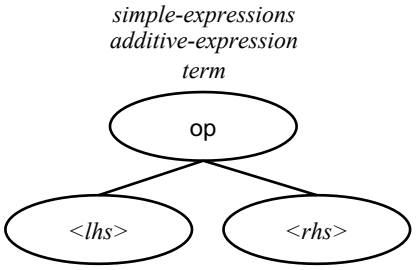
*** Variable Accessing & Array Indexing Expression (*var*)**



<Format: Variable Accessing & Array Indexing>

```
Variable: name = %s
  /* Child Node: Array Index Expression */
```

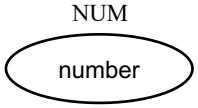
*** Binary Operator Expression**
(*simple-expression, additive-expression, term*)



<Format: Binary Operator Expression>

```
Op: %s
  /* Child Node: Left Hand Side */
  /* Child Node: Right Hand Side */
```

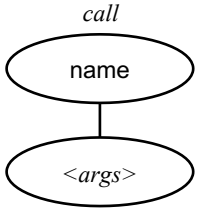
*** Constant Expression (NUM)**



<Format: Constant Expression>

```
Const: %d
```

*** Call Expression (*call*)**



<Format: Call Expression>

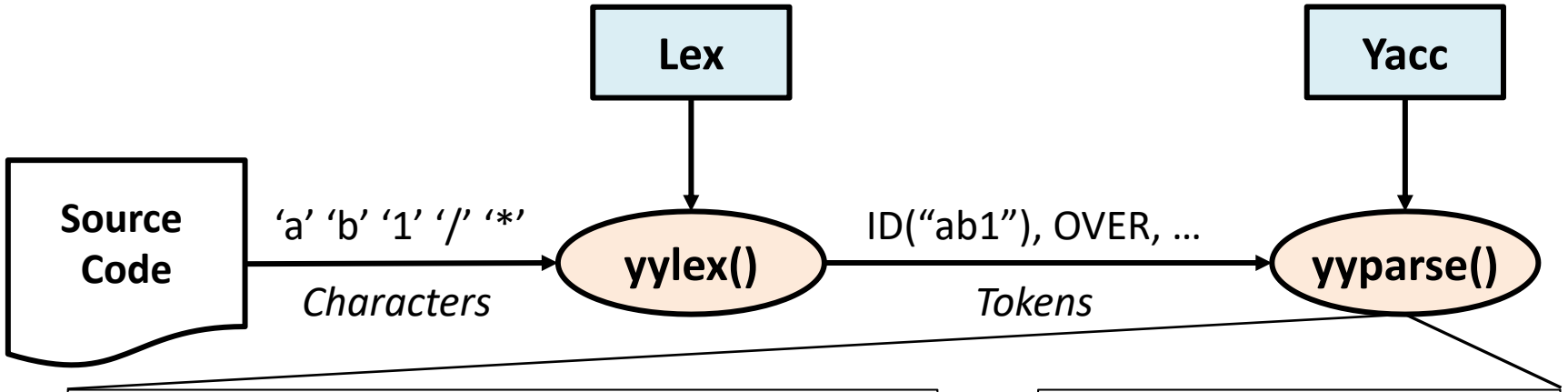
```
Call: function name = %s
  /* Child Node: Arguments */
```

Yacc (Bison)

- **Parser Generator for UNIX**
  - **Yacc**: Yet Another Compiler Compiler
  - **Bison**: GNU project parser generator (upward compatible with Yacc)

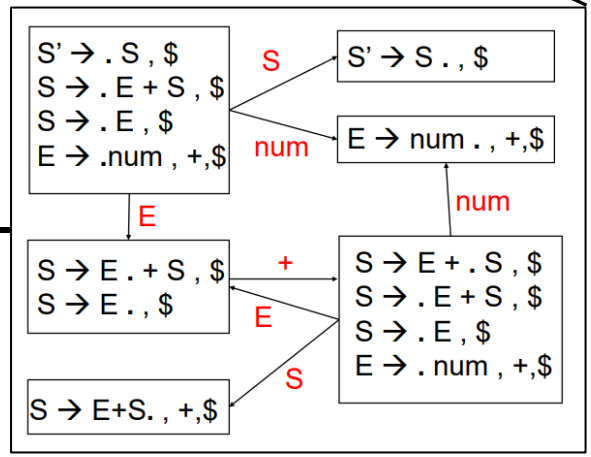
  - Input : A context-free grammar in BNF form
  - Output: C-code of parser for the input grammar

Yacc: 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 Source Structure

Definitions ← Tokens (Priority, Associativity)

%%

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

Fill the Codes

%%

Subroutines ← (You don't need to modify this part)

Yacc Example: *tiny.y*

- **Rules**

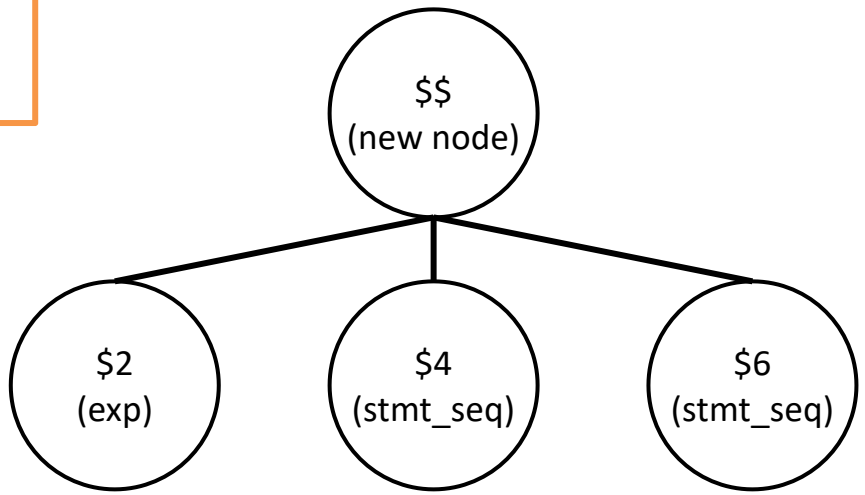
Pointer to *if_stmt* (non-terminal) → $$

YYSTYPE (*TreeNode**)

```
$$      $1 $2 $3   $4   $5
if_stmt : IF exp THEN stmt_seq END
            { $$ = 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;
            }
        ;
```

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 Example: *tiny.y*

- **Variables**

Type for AST Nodes (defined in *globals.h*)

```
15  #define YYSTYPE TreeNode *
16  static char * savedName; /* for use in assignments */
17  static int savedLineNo;  /* ditto */
18  static TreeNode * savedTree; /* stores syntax tree for later return */
19  static int yylex(void); // added 11/2/11 to ensure no conflict with lex
```

AST Root (returned by *parse()*)

```
30  program     : stmt_seq
31                { savedTree = $1;}
32                ;
```


Hanyang University
Division of Computer Science & Engineering

Yacc Example: *tiny.y*

- **Definitions**

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

- **Priority**
  - Top Line < Bottom Line

- **Associativity**
  - ***%left, %right, %noassoc*** instead of ***%token***
  - Example: *%left PLUS MINUS TIMES OVER*

Yacc Usages

- **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**
  - https://www.gnu.org/software/bison/manual

Hint: Build with Makefile

```
# Makefile for C-Minus
#
# ./lex/tiny.l        --> ./cminus.l (from Project 1)
# ./yacc/tiny.y       --> ./cminus.y
# ./yacc/globals.h    --> ./globals.h

CC = gcc (clang)

CFLAGS = -W -Wall

OBJS = main.o util.o lex.yy.o y.tab.o

.PHONY: all clean
all: cminus_parser

clean:
    rm -vf cminus_parser *.o lex.yy.c y.tab.c y.tab.h y.output

cminus_parser: $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -o $@ -lfl(-ll)

main.o: main.c globals.h util.h scan.h parse.h y.tab.h
    $(CC) $(CFLAGS) -c main.c

util.o: util.c util.h globals.h y.tab.h
    $(CC) $(CFLAGS) -c util.c

scan.o: scan.c scan.h util.h globals.h y.tab.h
    $(CC) $(CFLAGS) -c scan.c

lex.yy.o: lex.yy.c scan.h util.h globals.h y.tab.h
    $(CC) $(CFLAGS) -c lex.yy.c

lex.yy.c: cminus.l
    flex cminus.l

y.tab.h: y.tab.c

y.tab.o: y.tab.c parse.h
    $(CC) $(CFLAGS) -c y.tab.c

y.tab.c: cminus.y
    yacc -d -v cminus.y
```

In case of Mac,
Use Clang and – ll instead

Hint: where to see?

- **main.c**
  - Modify code to print *only* syntax tree
  - *NO_ANALYZE*, *TraceParse*

```
/****************************************************/
/* 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;
```

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

```
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?

- **globals.h**
  - Overwrite your *globals.h* with *yacc/globals.h.*
  - "Syntax tree for parsing" should be updated to meet C-Minus Spec.
  - You can define your own AST.
    - You **can** modify/add/remove *NodeKind*, *StmtKind*, *ExpKind*, *ExpType*, and ***TreeNode*.** (You only should follow the output AST format specified in project goal slide. The Internal implementation is FREE.)
    - FAQ: What is the difference between *StatK* and *ExpK*?
      - It depends on your implementation. (= They are not important in C-Minus implementation) You can even remove *NodeKind* (the statement/expression classification) and integrate *StmtKind* and *ExpKind*.
  - *TreeNode** is used to define YYSTYPE in *cminus.y*

Hint: where to see?

- **cminus.y**
  - ~~Write C-Minus tokens in the definition section.~~
    - ~~Consider priority and associativity.~~
  - Define a C-Minus grammar and reduce actions for each rules.
    - *YYSTYPE* (the type of *$$*, *$1*, …) is defined as *TreeNode**.

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.1.txt

Syntax tree:
  Function Declaration: name = gcd, return type = int
    Parameter: name = u, type = int
    Parameter: name = v, type = int
    Compound Statement:
      If-Else Statement:
        Op: ==
          Variable: name = v
          Const: 0
        Return Statement:
          Variable: name = u
        Return Statement:
          Call: function name = gcd
            Variable: name = v
            Op: -
              Variable: name = u
              Op: *
                Op: /
                  Variable: name = u
                  Variable: name = v
                Variable: name = v
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = x, type = int
      Variable Declaration: name = y, type = int
      Assign:
        Variable: name = x
        Call: function name = input
      Assign:
        Variable: name = y
        Call: function name = input
      Call: function name = output
        Call: function name = gcd
          Variable: name = x
          Variable: name = y
```

Some Comments

- **You should generate exactly same output.**

- **REMOVE ALL YACC CONFLICTS EVEN IF IT IS JUST WARNING**
  - PENALTIES FOR EACH CONFLICT: Shift/Shift, Shift/Reduce, Reduce/Reduce
  - But you can still ignore warnings related with gcc/clang compilation.

- **Check output formats (should be distinguishable):**
  - *If* without *Else* statement and *If-Else* Statement
  - No Parameter (*void*) and Parameters
  - *Return* statement without value and *return* statement with value

Some Comments

- **How to implement Lists? (*declaration-list, statement-list, param-list, …*)**
  - Hint: see declaration-list in cminus.y
- **How to store attributes of *TreeNode* such as *ID* (=*name*), *type* and *op*?**
  - Consideration: *TokenString* may not contain "string of the ID token" when reduce.
  - Intra-Rule action (performed at shift) such as [*assign_stmt*] in tiny is not recommended.
  - Passing values using explicit casting with *void** is not recommended. (but it is possible)
  - Do not update variables handled by scanner such as *TokenString*. Use *copyString*().
- **Keep and set the line number attribute of *TreeNode* for Project 3.**

Some Comments

- **You don't need to care about Semantics, just Syntax analyzer will be okay. (Analyzing semantics is for Project 3.)**

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

```
/* Semantic Error Example */
/* (1) void-type variable a, b
 * (2) uninitialized variable c (and b)
 * (3) undefined variable d */

int main ( void a[] )
{
    void b;
    int c;
    d[1] = b + c;
}
```

```
C-MINUS COMPILATION: ./error_test.cm

Syntax tree:
  Function Declaration: name = main, return type = int
    Parameter: name = a, type = void[]
    Compound Statement:
      Variable Declaration: name = b, type = void
      Variable Declaration: name = c, type = int
      Assign:
        Variable: name = d
          Const: 1
        Op: +
          Variable: name = b
          Variable: name = c
```

Evaluation

- **Evaluation Items**
  - **Compilation** (Success / Fail): **20%**
    - Please describe in the report how TA can build your project.
  - **Correctness** check for several testcases: **70%**
    - Note: Make sure there are no segmentation fault or infinite loop on any inputs.
  - **Report** : **10%**

Report

- **Guideline (≤ 5 pages)**
  - Compilation environment and method
  - Brief explanations about how to implement and how it operates
  - Examples and corresponding result screenshots

- **Format**
  - PDF format

Submission

- **Deadline: 11/27 (Mon.) 23:59:59**
- **Submission**
  - Place all the **source codes** in the **StudentID/2_Parser** directory
  - Place **report** in the **StudentID** directory
  - Zip the **StudentID** directory
  - Upload the zip file to the **LMS system**
- **Questions**
  - E-mail: **ted6345@hanyang.ac.kr**
    - Please provide all questions related with projects to TAs.

Q&A