

---

# Project #1. Scanner

2022 Compiler  
Prof. Yongjun Park

---

# Project Goal: Scanner



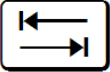
- **C-Minus Scanner Implementations (both 2 methods)**
  - The scanner reads an input source code string, tokenizes it, and returns (prints) recognized tokens.
  - Method 1: **C code**
    - Recognize tokens by **DFA**
    - *scan.c, ... -> cminus\_cimpl*
  - Method 2: **Lex (flex)**
    - Specify lexical patterns by **Regular Expression**
    - *cminus.l, ... -> cminus\_lex*



# Goal: Lexical Convention of C-Minus

- **Reserved Words (Keywords)**
    - *int void if else while return* (lower cases)
  - **Symbols**
    - $+$   $-$   $*$   $/$
    - $<$   $<=$   $>$   $>=$   $==$   $!=$
    - $=$   $;$   $,$
    - $($   $)$   $[$   $]$   $\{$   $\}$
  - **Identifier and Number**
    - *ID* = letter (letter | digit)\*
    - *NUM* = digit digit\*
    - *letter* = a | ... | z | A | ... | Z |
    - *digit* = 0 | ... | 9
- 6 Reserved Words
  - ID and NUM
  - 19 other tokens

# Goal: Lexical Convention of C-Minus

- **Whitespace:** , , 
  - Space, Newline, Tab
  - Ignore other cases (ex: located in beginning and end of line) except whitespace between *ID*, *NUM*, and keywords
- **Comments**
  - Comments (*/\* \*/*) follows normal C notation.
  - No single-line comments
  - Comments cannot be nested
- Please refer “*Kenneth C. Louden* book p. 491-492”

# Requirements

- Output Format

C-MINUS COMPILATION: ./testcase/03\_GeneralCase/array.cm

Line Number

```
1: reserved word: void
1: ID, name= main
1: (
1: reserved word: void
1: )
2: {
3: reserved word: int
3: ID, name= i
3: ;
4: reserved word: int
4: ID, name= size
4: =
4: NUM, val= 3
4: ;
5: reserved word: int
5: ID, name= arr
5: [
```

Token String

- reserved word: %s
- ID, name = %s
- NUM, val = %s
- %s

# Example: C-Minus Code

test.cm

```
/* 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));
}
```

Comments

- **Execute as:**  
\$ ./cminus\_cimpl test.cm  
\$ ./cminus\_lex test.cm
- **Result** should be shown as in the next slide.

C-MINUS COMPILATION: test.cm

```
4: reserved word: int
4: ID, name= gcd
4: (
4: reserved word: int
4: ID, name= u
4: ,
4: reserved word: int
4: ID, name= v
4: )
5: {
6: reserved word: if
6: (
6: ID, name= v
6: ==
6: NUM, val= 0
6: )
6: reserved word: return
6: ID, name= u
6: ;
7: reserved word: else
7: reserved word: return
7: ID, name= gcd
7: (
7: ID, name= v
7: ,
7: ID, name= u
7: -
7: ID, name= u
7: /
7: ID, name= v
7: *
7: ID, name= v
7: )
7: ;
9: }
```

```
11: reserved word: void
11: ID, name= main
11: (
11: reserved word: void
11: )
12: {
13: reserved word: int
13: ID, name= x
13: ;
13: reserved word: int
13: ID, name= y
13: ;
14: ID, name= x
14: =
14: ID, name= input
14: (
14: )
14: ;
14: ID, name= y
14: =
14: ID, name= input
14: (
14: )
14: ;
15: ID, name= output
15: (
15: ID, name= gcd
15: (
15: ID, name= x
15: ,
15: ID, name= y
15: )
15: )
15: ;
16: }
17: EOF
```

# Hint: *main.c*

- *main.c*
  - Modify code to print source & tokens
  - Set **NO\_PARSE**, **TraceScan** to **TRUE**

```
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 TRUE  
12 /* set NO_ANALYZE to TRUE to get a parser-only compiler */  
13 #define NO_ANALYZE FALSE  
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 = TRUE;  
42 int TraceParse = FALSE;  
43 int TraceAnalyze = FALSE;  
44 int TraceCode = FALSE;
```

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

Debug Option

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



# Hint: Token Definitions

- *globals.h*
  - Add C-Minus tokens to *TokenType*
  - You MUST remove Tiny's Tokens (*then, repeat, until, write, read, end*)

```
25 /* MAXRESERVED = the number of reserved words */
26 #define MAXRESERVED 6
27
28 typedef enum
29     /* book-keeping tokens */
30     {ENDFILE,ERROR,
31     /* reserved words */
32     IF,ELSE,WHILE,RETURN,INT,VOID,
33     /* multicharacter tokens */
34     ID,NUM,
35     /* special symbols */
36     ASSIGN,EQ,NE,LT,LE,GT,GE,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,LBRACE,RBRACE,LCURLY,RCURLY,SEMI,COMMA
37 } TokenType;
```

---

# Hint: Print Tokens

- *utils.c*
  - Need to modify *printToken()* for C-Minus tokens
  - Check slide **[Requirements: Output Format]**



---

# Method 1: C Implementation

- **C-Minus Scanner Implementations** (both 2 methods)
  - The Scanner reads an input source code string, tokenizes it, and returns (prints) recognized tokens.
  - Method 1: **C code**
    - Recognize tokens by **DFA**
    - *scan.c, ... -> cminus\_cimpl*
  - Method 2: **Lex (flex)**
    - Specify Lexical Patterns by *Regular Expression*
    - *cminus.l, ... -> cminus\_lex*



# Makefile

```
1 # Makefile for C-Minus Scanner
2
3 CC = gcc
4
5 CFLAGS =
6
7 OBJS = main.o util.o scan.o
8
9 .PHONY: all clean
10 all: cminus_cimpl
11
12 clean:
13     -rm -vf cminus_cimpl *.o
14
15 cminus_cimpl: $(OBJS)
16     $(CC) $(CFLAGS) -o $@ $(OBJS)
17
18 main.o: main.c globals.h util.h scan.h
19     $(CC) $(CFLAGS) -c -o $@ $<
20
21 scan.o: scan.c globals.h util.h scan.h
22     $(CC) $(CFLAGS) -c -o $@ $<
23
24 util.o: util.c globals.h util.h
25     $(CC) $(CFLAGS) -c -o $@ $<
```

# Makefile for C-Minus Scanner

CC = gcc

CFLAGS =

OBJS = main.o util.o scan.o

.PHONY: all clean

all: cminus\_cimpl

clean:

-rm -vf cminus\_cimpl \*.o

cminus\_cimpl: \$(OBJS)

\$(CC) \$(CFLAGS) -o \$@ \$(OBJS)

main.o: main.c globals.h util.h scan.h

\$(CC) \$(CFLAGS) -c -o \$@ \$<

scan.o: scan.c globals.h util.h scan.h

\$(CC) \$(CFLAGS) -c -o \$@ \$<

util.o: util.c globals.h util.h

\$(CC) \$(CFLAGS) -c -o \$@ \$<

# Hint: DFA Implementation

- *scan.c*
  - Reserved word should be added for C-Minus

```
60 /* lookup table of reserved words */
61 static struct
62 {
63     char* str;
64     TokenType tok;
65 } reservedWords[MAXRESERVED] = {
66     {"if", IF},
67     {"else", ELSE},
68     {"while", WHILE},
69     {"return", RETURN},
70     {"int", INT},
71     {"void", VOID},
72 };
```

---

# Hint: DFA Implementation

- ***scan.c***
  - *getToken()* should be modified for C-Minus tokens
    - It represents DFA for scanner.
  - ***StateType state*** variable represents current state in DFA
    - You should add your custom states to scan C-Minus tokens into StateType
    - Note: “==”, “<=”, “>=”
    - Hint: add INEQ, INLT, INGT, INNE, INOVER, INCOMMENT, INCOMMENT\_
  - ***TokenType currentToken*** variable represents a recognized token.
  - *getNextChar()* reads a character
  - *ungetNextChar()* undoes a read character



# Hint: DFA Implementation

- *scan.c*

```
79 TokenType getToken(void)
80 { /* index for storing into tokenString */
81     int tokenStringIndex = 0;
82     /* holds current token to be returned */
83     TokenType currentToken;
84     /* current state - always begins at START */
85     StateType state = START;
86     /* flag to indicate save to tokenString */
87     int save;
```

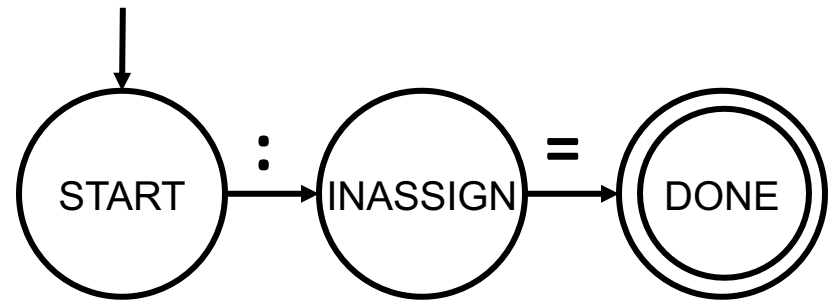
# Hint: DFA Implementation

- *scan.c*
  - Example: existing “:=” (ASSIGN) token in Tiny
    - It is NOT a C-Minus ASSIGN Token, refer as just example.

```
while (state != DONE)
{
    int c = getNextChar();
    save = TRUE;

    switch (state)
    {
        case START:
            if (c == ':')
                state = INASSIGN;
            /* ... */

        case INASSIGN:
            state = DONE;
            if (c == '=') currentToken = ASSIGN;
            else
            {
                ungetNextChar();
                save = FALSE;
                currentToken = ERROR;
            }
            break;
        /* ... */
    }
}
```





---

# Method 2: Lex Implementation

- **C-Minus Scanner Implementations** (both 2 methods)
  - The Scanner reads an input source code string, tokenizes it, and returns (prints) recognized tokens.
  - Method 1: C code
    - Recognize tokens by *DFA*
    - *scan.c, ... -> cminus\_cimpl*
  - Method 2: Lex (flex)
    - Specify Lexical Patterns by *Regular Expression*
    - *cminus.l, ... -> cminus\_lex*



---

# Lex / Flex

- **(Fast) Lexeme Analysis**

- Automatically generates a target scanner based on input Regex
- Usually work with *yacc* (*bison*)

- **Install**

- Ubuntu: `$ sudo apt-get install flex`
- MacOS : `$ brew install flex`

- **Usage**

- `$ flex [Lex filename]`
- *lex.yy.c* will be created

- **Manual**

- `$ man flex`
- `$ info flex`



```

16 digit      [0-9]
17 number     {digit}+
18 letter     [a-zA-Z]
19 identifier {letter}+
20 newline    \n
21 whitespace [ \t]+

```

```

22
23 %%
24

```

```

25 "if"        {return IF;}
26 "then"      {return THEN;}
27 "else"      {return ELSE;}
28 "end"       {return END;}
29 "repeat"    {return REPEAT;}
30 "until"     {return UNTIL;}
31 "read"      {return READ;}
32 "write"     {return WRITE;}
33 {whitespace} {/* skip whitespace */}
34 "{"         { char c;
35             do
36             { c = input();
37               if (c == EOF) break;
38               if (c == '\n') lineno++
39             } while (c != '}');
40             }
41 .           {return ERROR;}

```

```

42
43 %%
44

```

```

45 TokenType getToken(void)
46 { static int firstTime = TRUE;
47   TokenType currentToken;
48   if (firstTime)
49   { firstTime = FALSE;

```

- **Definition Section**

- C header / declaration, Regex naming, ...

- **Rule Section**

- Token rule (Regex) and action (C codes)
- You can use "rule" or {name} for token rule
- The return in action will become return of **yylex()**

- **Subroutine Section**

- User defined functions

# Makefile

```
1 # Makefile for C-Minus Scanner
2 # ./lex/tiny.l --> ./cminus.l
3
4 CC = gcc
5
6 CFLAGS = -W -Wall
7
8 OBJS = main.o util.o scan.o
9 OBJS_LEX = main.o util.o lex.yy.o
10
11 .PHONY: all clean
12 all: cminus_cimpl cminus_lex
13
14 clean:
15     -rm -vf cminus_cimpl cminus_lex *.o lex.yy.c
16
17 cminus_cimpl: $(OBJS)
18     $(CC) $(CFLAGS) -o $@ $(OBJS)
19
20 cminus_lex: $(OBJS_LEX)
21     $(CC) $(CFLAGS) -o $@ $(OBJS_LEX) -lfl
22
23 main.o: main.c globals.h util.h scan.h
24     $(CC) $(CFLAGS) -c -o $@ $<
25
26 scan.o: scan.c globals.h util.h scan.h
27     $(CC) $(CFLAGS) -c -o $@ $<
28
29 util.o: util.c globals.h util.h
30     $(CC) $(CFLAGS) -c -o $@ $<
31
32 lex.yy.o: lex.yy.c globals.h util.h scan.h
33     $(CC) $(CFLAGS) -c -o $@ $<
34
35 lex.yy.c: cminus.l
36     flex -o $@ $<
```

# Makefile for C-Minus Scanner

# ./lex/tiny.l --> ./cminus.l

CC = gcc

CFLAGS = -W -Wall

OBJS = main.o util.o scan.o

OBJS\_LEX = main.o util.o lex.yy.o

.PHONY: all clean

all: cminus\_cimpl cminus\_lex

clean:

-rm -vf cminus\_cimpl cminus\_lex \*.o lex.yy.c

cminus\_cimpl: \$(OBJS)

\$(CC) \$(CFLAGS) -o \$@ \$(OBJS)

cminus\_lex: \$(OBJS\_LEX)

\$(CC) \$(CFLAGS) -o \$@ \$(OBJS\_LEX) -lfl

main.o: main.c globals.h util.h scan.h

\$(CC) \$(CFLAGS) -c -o \$@ \$<

scan.o: scan.c globals.h util.h scan.h

\$(CC) \$(CFLAGS) -c -o \$@ \$<

util.o: util.c globals.h util.h

\$(CC) \$(CFLAGS) -c -o \$@ \$<

lex.yy.o: lex.yy.c globals.h util.h scan.h

\$(CC) \$(CFLAGS) -c -o \$@ \$<

lex.yy.c: cminus.l

flex -o \$@ \$<

---

# Hint: Difference in Lex Version

- ***globals.h, main.c, util.c***
  - Same as in DFA implementation
- ***scan.c***
  - This file is not used because the body of *getToken()* will be automatically generated using Flex
- ***cminus.l***
  - Start from copying *tiny.l* and properly modify it



---

# 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: **Comments** are also one of key check point.
  - Note: Make sure there are no **segmentation fault** or **infinite loop** on any inputs.
- **Report** : **10%**



---

# Report

- **Guideline ( $\leq 5$  pages)**

- Compilation environment and method
- Brief explanations about how to implement and how it operates
- Examples and corresponding result screenshots

- **Format**

- Any visible formats such as PDF, MS Word, HWP, ... are allowed
  - PDF format is recommended
- GitLab wiki is not allowed
  - Instead, write in markdown format and submit as PDF



---

# Submission

- **Deadline: 11/4 (Fri.) 23:59:59**
- **Submission**
  - Submit all the source codes and the report in the **1\_Scanner** directory
  - [https://hconnect.hanyang.ac.kr/2022\\_ele4029\\_12271/2022\\_ele4029\\_\[Student ID\].git](https://hconnect.hanyang.ac.kr/2022_ele4029_12271/2022_ele4029_[Student ID].git)
- **Questions**
  - E-mail: [compiler.teachingassistant@gmail.com](mailto:compiler.teachingassistant@gmail.com)
    - Please provide all questions related with projects to TAs.



---

# FAQ

- **Handling EOF**

- Whether the input source is valid or not, the scanner program must be terminated without any runtime errors such as segmentation fault or infinite loop.
- Even if the input source ends without closing brace when the scanner reads EOF, for example, all tokens except the missing brace must be printed normally.
- Please consider that Lex can detect '\0' (0) instead of EOF (-1) in the end of the input source in some environment.

---

# Q&A

