Project #1. Scanner

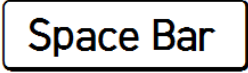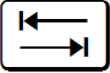## 2023 Compiler

Prof. Eul Gyu Lm

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*   둘 다 구현

  – 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**   심볼들 token화
  - **+   -   *   /**
  - **<   <=   >   >=   ==   !=**
  - **=   ;   ,**
  - **( ) [ ] { }**

- **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*

Hanyang University
Division of Computer Science & Engineering

Goal: Lexical Convention of C-Minus

- **Whitespace:** [Space Bar] , [Enter] , [Tab]

  - 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

  - When a comment is opened with '/*' but not closed with ' */', it results in EOF.
    EOF출력

  Please refer "*Kenneth C. Louden* book p. 491-492"

Requirements

- **Output Format**

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

Line Number

\t

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

\t

```
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: }
```

\t

```
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
```

Hanyang University
Division of Computer Science & Engineering

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 $@ $<
```

```
$make clean
$make all
$./cminus_cimpl ./example/test.cm
$./cminus _lex ./example/test.cm
```

Hanyang University
Division of Computer Science & Engineering

Hint: *main.c*

- *main.c*
  - Modify code to print source & tokens
  - Set **NO_PARSE**, **TraceScan** to **TRUE**

Debug Option

Hanyang University
Division of Computer Science & Engineering

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,      형식에 맞게 수정
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*

```
∨ 1_Scanner
  > example
  ≡ cminus.l
  C globals.h
  C main.c
  M Makefile
  C scan.c
  C scan.h
  C util.c
  C util.h
```

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: additional state **INEQ, INLT, INGT, INNE, INOVER, INCOMMENT, INCOMMENT_**
      useless state **INASSIGN**
      incomment = /
      incomment_ = /*
  - ***TokenType currentToken*** variable represents a recognized token.

  - *getNextChar*() reads a character
  - *ungetNextChar*() undoes a read character

Hanyang University
Division of Computer Science & Engineering

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;
```
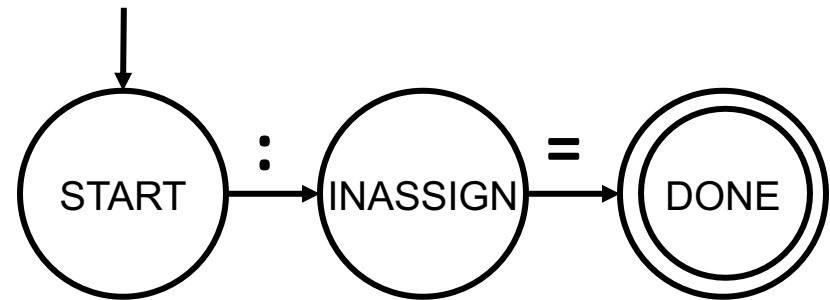
Hanyang University
Division of Computer Science & Engineering

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();    1개의 char 읽어오기
    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;
        /* ... */
}
```

Hanyang University
Division of Computer Science & Engineering

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: $ 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 %%
```

string에 따라 token으로 변환 - role section 잘 정의

```
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 %%
```

얘는 바꿀 필요 없음

```
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

Hanyang University
Division of Computer Science & Engineering

Hint: Difference in Lex Version

- ### *globals.h, main.c, util.c*

  - Same as in DFA implemenation

- ### *scan.c*

  - This file is not used because the body of *getToken*() will be automatically generated using Flex

- ### *cminus.l*

  - 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** (≤ **3 pages**)

  – Compilation environment and method 개발환경, method 필수

  – Brief explanations about how to implement and how it operates

 – Examples and corresponding result screenshots

- **Format**

  – PDF format

Submission

- **Deadline: 10/16 (Mon.) 23:59:59**

- **Submission**

  – Place all the <u>source codes</u> in the **StudentID/1_Scanner** directory

  – Place <u>report</u> 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.
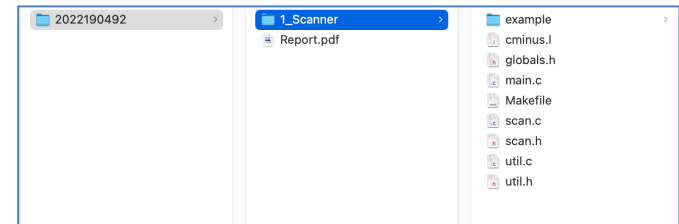
FAQ

- **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.

    \0또는 -1이면 EOF이게 코드에 넣을 것
  - Please consider that Lex cand detect '\0'(0) instead of EOF(-1) in the end of the input source in some environment.

Q&A