

Project #1: Scanner

2024 Fall

Hunjun Lee

Hanyang University

Project Goal

- **You will implement a C-Minus scanner in two methods**
 - The scanner (1) reads an input source code string, (2) tokenizes the string, and (3) returns or prints the identified tokens
 - **Method #1: Using a custom C code**
 - Recognizes tokens by DFA
 - You will modify scan.c code to implement your lexical rules
 - **Method #2: Lex (Flex)**
 - Specify lexical patterns by Regular Expression
 - Modify cminus.l code to define your lexical rules

C-Minus Lexical Convention - 1

- There are six reserved words (keywords)

– *int void if else while return* (lower cases)

- There are 19 symbols

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

- Identifier and number rules

- *ID* = letter (letter | digit)*
- *NUM* = digit digit*

C-Minus Lexical Convention - 2

- **Whitespaces:**

- Spaces, newlines, tabs
- Ignore whitespaces located at the beginning and end of line
- Use whitespaces in between the letters/digits/strings to distinguish tokens

- **Comments**

- Comments (`/* */`) follows a normal C notation
- There is no single-line comments such as (`//`)
- Comments cannot be nested

Requirement: Output Format

- Output format

C-MINUS COMPILATION: ./overall/test.1.txt

```
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
```

Print Line No. →

Insert Tab →

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

```

Modify main.c file

- main.c
 - Modify code to print source & tokens
 - Set **NO_PARSE** and **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;
```


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;
```

Print Tokens

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

Compiling

- **We provide a separate Makefile to compile the program**

- make cminus_cimpl to compile c-based implementation
- make cminus_lex to compile lex-derived program

Method #1: C Implementation

- You will implement a C-Minus scanner in two methods
 - The scanner (1) reads an input source code string, (2) tokenizes the string, and (3) returns or prints the identified tokens
 - **Method #1: Using a custom C code**
 - Recognizes tokens by DFA
 - You will modify scan.c code to implement your lexical rules
 - **Method #2: Lex (Flex)**
 - Specify lexical patterns by Regular Expression
 - Modify cminus.l code to define your lexical rules

DFA Implementation - 1

- ***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 };
```

DFA Implementation - 2

- ***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

DFA Implementation - 3

```
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;
```

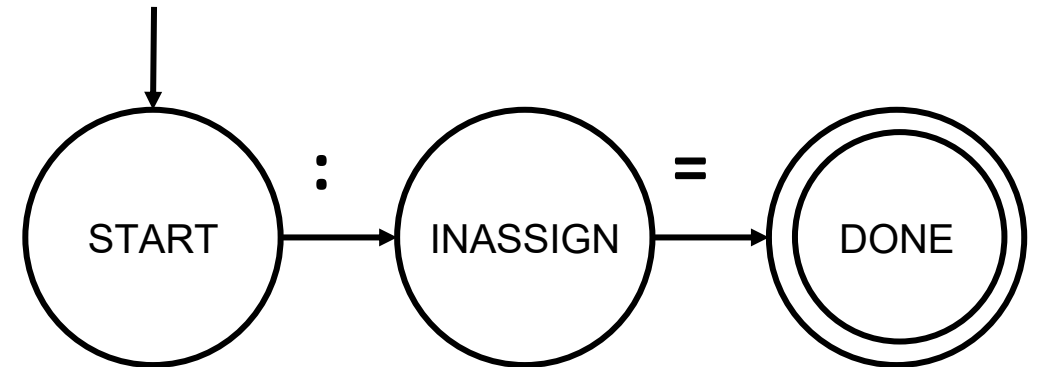
DFA Implementation - 4

- **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

- You will implement a C-Minus scanner in two methods
 - The scanner (1) reads an input source code string, (2) tokenizes the string, and (3) returns or prints the identified tokens
 - Method #1: Using a custom C code
 - Recognizes tokens by DFA
 - You will modify scan.c code to implement your lexical rules
 - Method #2: Lex (Flex)
 - Specify lexical patterns by Regular Expression
 - Modify cminus.l code to define your lexical rules

Lex / Flex - 1

- **(Fast) Lexeme Analysis**

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

- **Install**

- `apt-get install flex`

- **Usage**

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

```

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

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 to 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 (≤ 5 pages)**

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

- **Format**

- Use PDF with the filename as follows

Submission

- **Deadline: 9/30 (Mon.) 23:59:00**
- **Submission**
 - Submit all the source codes in a single zip file and report as a pdf file
 - Format + Name:
 - Report: [Student No]_Project1.pdf
 - Code: do not modify any name and compress all the codes into a single zip file and the name should be
 - [Student No]_Project1.zip
- **Questions**
 - Upload any questions to the LMS