p2. Lexical Analysis

2025 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**

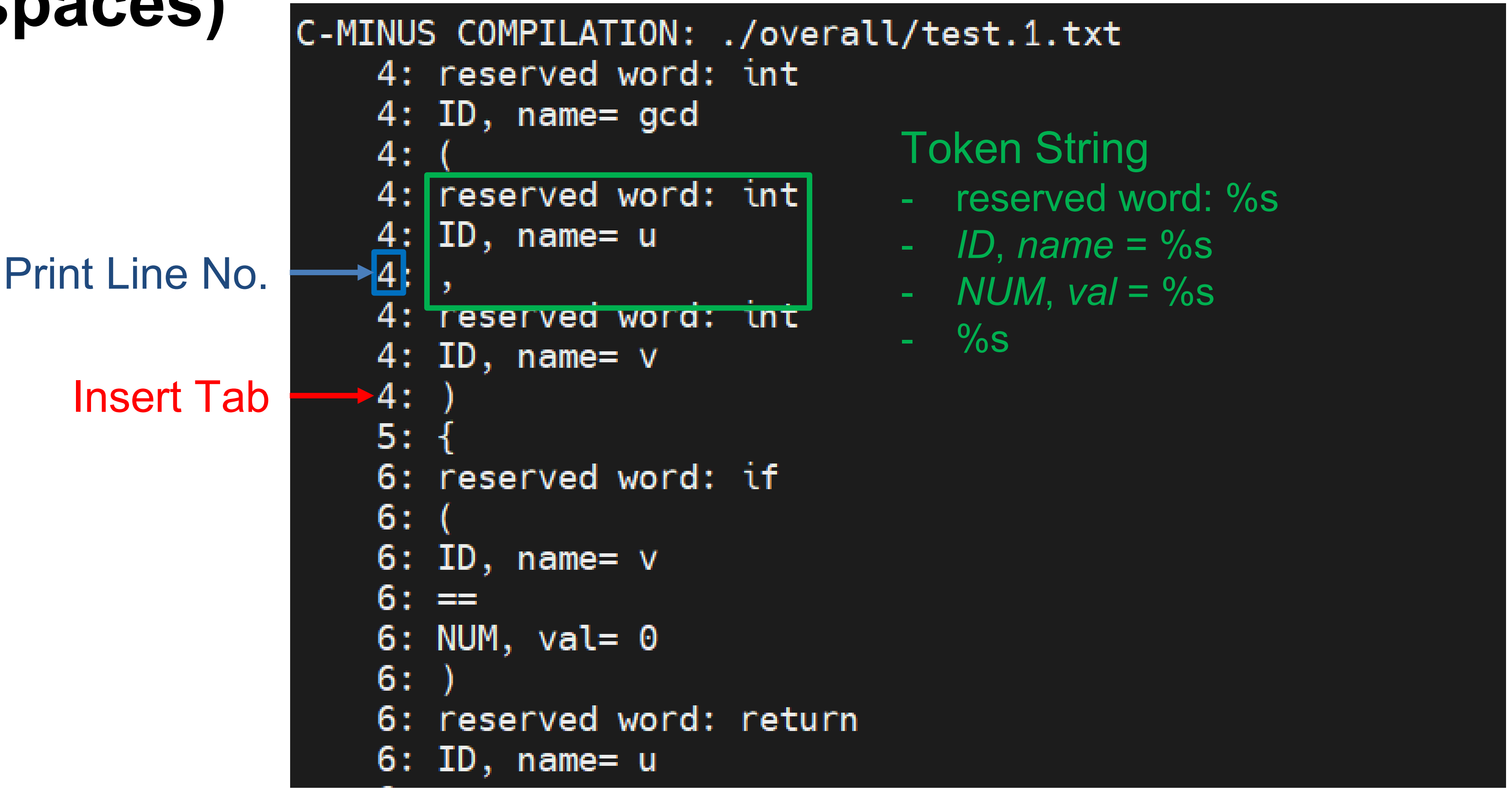
C-Minus Lexical Convention - 2

- **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
    - /* /* */ ➔ The /* in the middle is ignored

Requirement: Output Format

- **You must Obey the format: You will get penalty (do not mix up tabs and spaces)**



```
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
```

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));
}
```

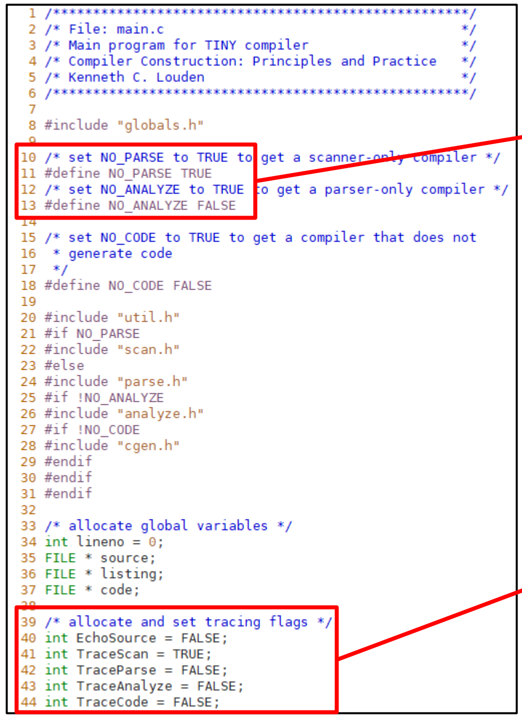
Comments

- **Execute as:**
```
$ ./cminus_cimpl test.cm
$ ./cminus_lex test.cm
```
- **Result** should be shown as in the next slide.

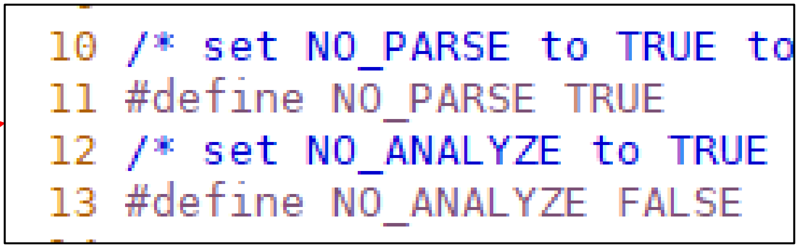
```
C-MINUS COMPILATION: test.cm
          4: reserved word: int
          4: ID, name= gcd
          4: (
          4: reserved word: int
          4: ID, name= u
          4: ,
          4: reserved word: int
          4: ID, name= v
          4: )
          5: {
          6: reserved word: if
          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

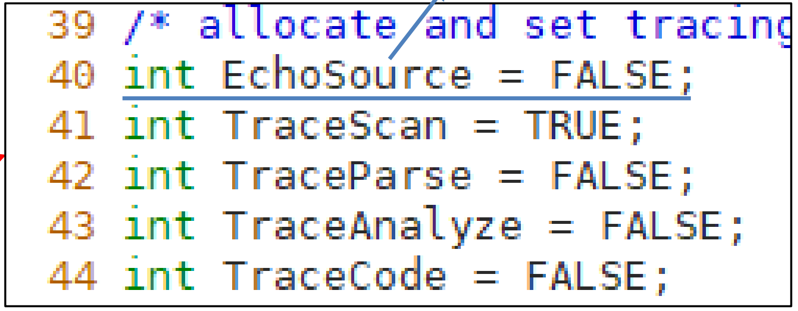


```c
/*****************************************************/
/* File: main.c                                      */
/* Main program for TINY compiler                    */
/* Compiler Construction: Principles and Practice    */
/* Kenneth C. Louden                                 */
/*****************************************************/

#include "globals.h"

/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE TRUE
/* set NO_ANALYZE to TRUE to get a parser-only compiler */
#define NO_ANALYZE FALSE

/* 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 = TRUE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```



```c
/* set NO_PARSE to TRUE to
#define NO_PARSE TRUE
/* set NO_ANALYZE to TRUE
#define NO_ANALYZE FALSE
```

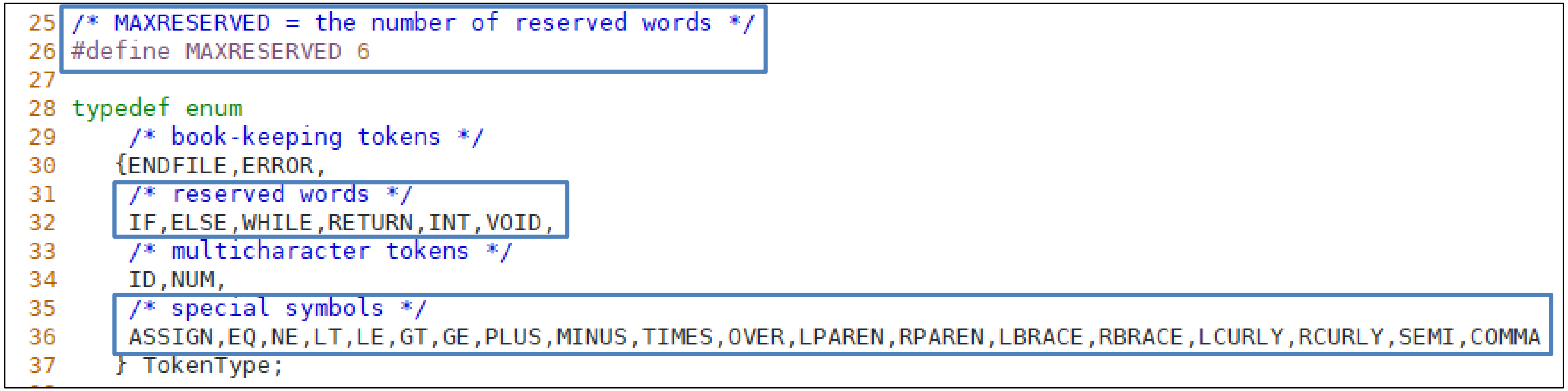
Debug Option



```c
/* allocate and set tracing
int EchoSource = FALSE;
int TraceScan = TRUE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
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 …)



```c
/* MAXRESERVED = the number of reserved words */
#define MAXRESERVED 6

typedef enum
     /* book-keeping tokens */
    {ENDFILE,ERROR,
     /* reserved words */
     IF,ELSE,WHILE,RETURN,INT,VOID,
     /* multicharacter tokens */
     ID,NUM,
     /* special symbols */
     ASSIGN,EQ,NE,LT,LE,GT,GE,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,LBRACE,RBRACE,LCURLY,RCURLY,SEMI,COMMA
    } 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

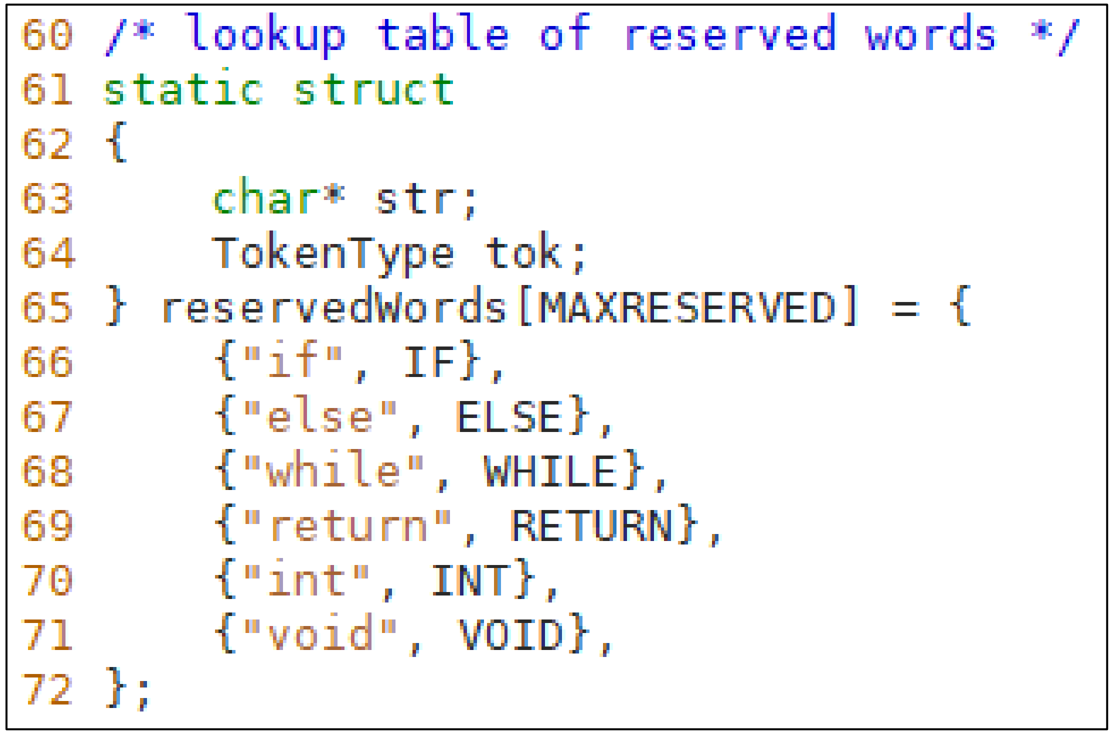
- **You will implement a C-Minus scanner in two methods**
  - The scanner (1) reads an input source code string, (2) tokenizes the string, and (3) returns or prints the identified tokens

  - **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

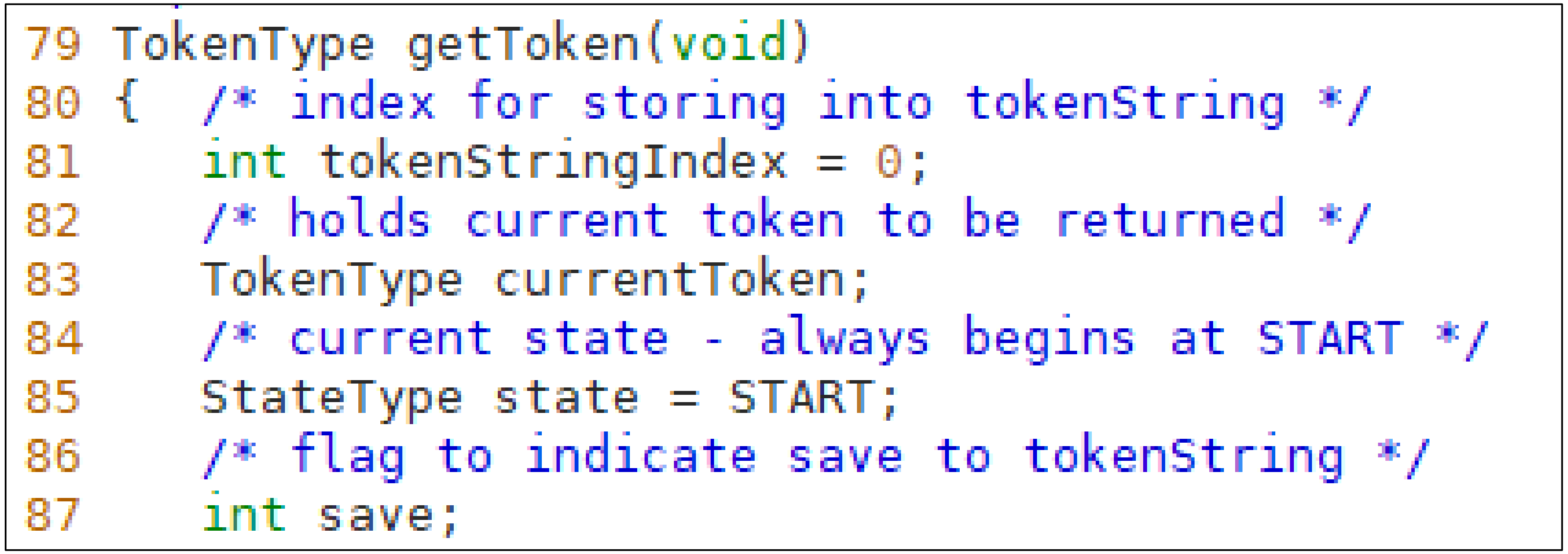


```c
/* lookup table of reserved words */
static struct
{
    char* str;
    TokenType tok;
} reservedWords[MAXRESERVED] = {
    {"if", IF},
    {"else", ELSE},
    {"while", WHILE},
    {"return", RETURN},
    {"int", INT},
    {"void", VOID},
};
```

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
    - Ex) “<“ vs. “<=“

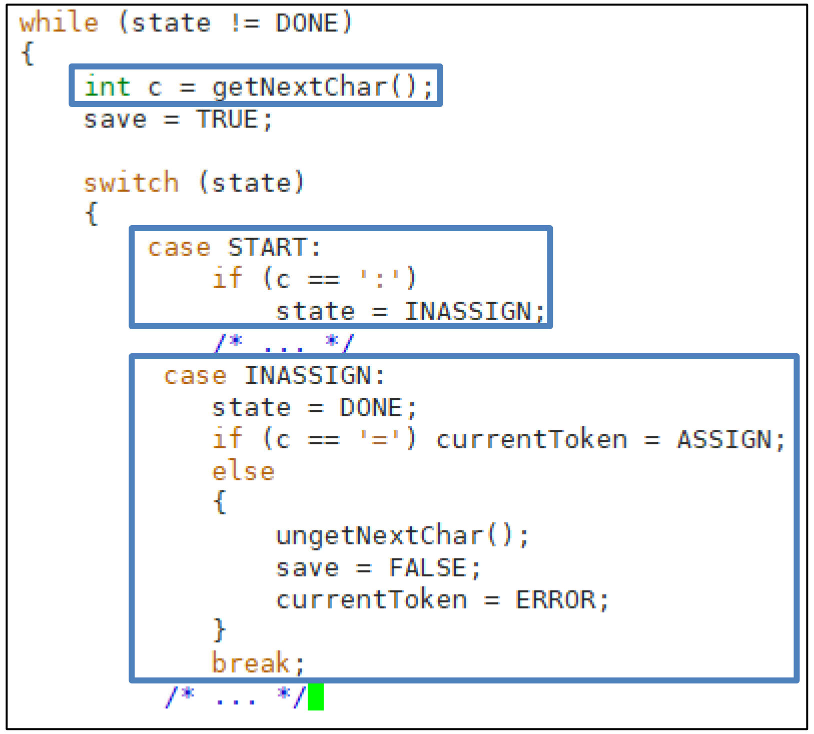
DFA Implementation - 3



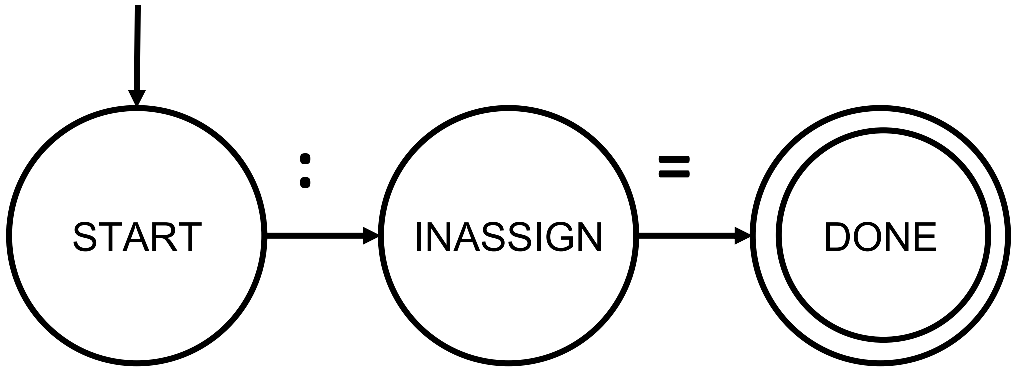
```
TokenType getToken(void)
{  /* index for storing into tokenString */
   int tokenStringIndex = 0;
   /* holds current token to be returned */
   TokenType currentToken;
   /* current state - always begins at START */
   StateType state = START;
   /* flag to indicate save to tokenString */
   int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

DFA Implementation - 4

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



```c
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;
        /* ... */
```



$$\begin{tikzcd} \arrow[r] & \text{START} \arrow[r, ":"] & \text{INASSIGN} \arrow[r, "="] & \text{DONE} \end{tikzcd}$$

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

```
digit        [0-9]
number       {digit}+
letter       [a-zA-Z]
identifier   {letter}+
newline      \n
whitespace   [ \t]+

%%

"if"            {return IF;}
"then"          {return THEN;}
"else"          {return ELSE;}
"end"           {return END;}
"repeat"        {return REPEAT;}
"until"         {return UNTIL;}
"read"          {return READ;}
"write"         {return WRITE;}
{whitespace}    {/* skip whitespace */}
"{"             { char c;
                  do
                  { c = input();
                    if (c == EOF) break;
                    if (c == '\n') lineno++;
                  } while (c != '}');
                }
.               {return ERROR;}

%%

TokenType getToken(void)
{ static int firstTime = TRUE;
  TokenType currentToken;
  if (firstTime)
  { 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 *cminus.l to the working directory* 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: Should be careful about tabs and spaces
    - 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: 10/03 (Fri.) 23:59:00**
- **Submission**
  - Submit all the **l, c and h files** in a single zip file and **report** as a pdf file
  - Do not submit binary files, temporary files, text files, etc
    - May negatively affect your grade in an automatic grading system
  - Format + Name:
    - Report: [Student No].pdf
    - Code: do not modify any name and compress all the codes into a single zip file and the name should be
      - [Student No].zip