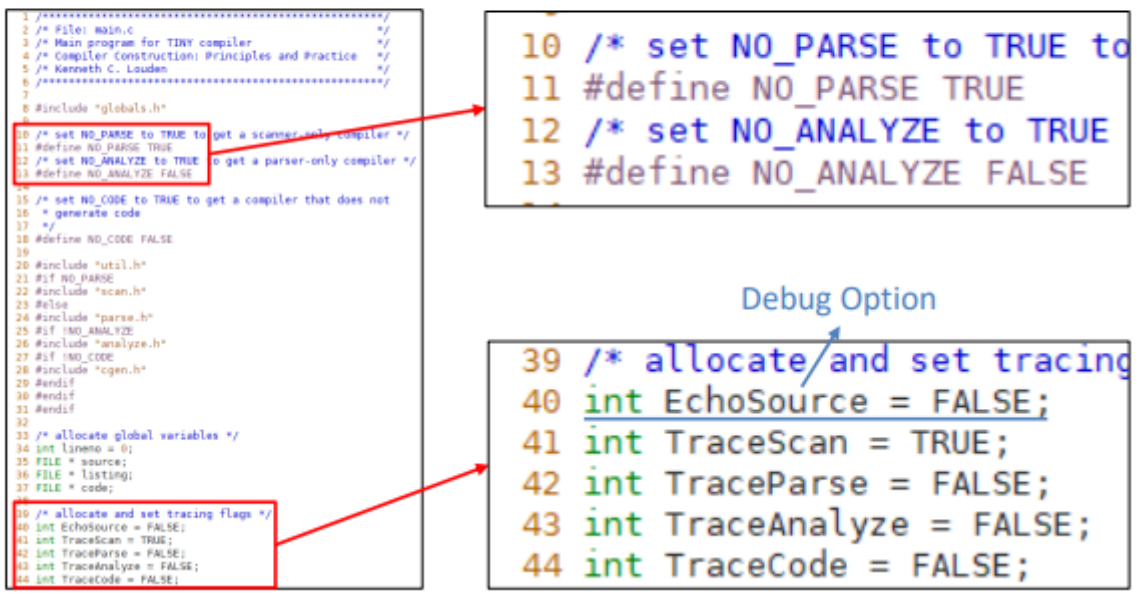
Scanner

0. Environment

- WSL Ubuntu 20.04
- gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
- flex 2.6.4
- GNU Make 4.2.1(Built for x86_64-pc-linux-gnu)

1. Modification

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



- ***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;
```

- Add C-Minus tokens and Remove Tiny tokens

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

- Modify printToken() for C-Minus tokens.

2. Method 1 : C Code 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 };
```

- ***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

Explanation

A getToken() function gets a character from input source using getNextChar() and determines what tokens is derived from the character by transition of state. It starts from START state and goes to DONE state inside of loop. At first, this function checks a character and change state into proper state derived from character. Whenever a new character is encountered, chooses state until the character is unmatched to state. And when it happens, the unmatched character be “unget” and state be DONE.

Modifying getToken()

1. Modifying ID

   ID requirement - {letter}({letter}|{digit})*

   If first character is a letter, then change it’s state from START state to ID state and keep checking after one is a letter or digit until it is not a letter and digit. Finally, when a character is not a letter or digit, call ungetNextChar() function to “unget” the character that I checked recently and change state from ID to DONE. Then ID token is discovered.

2. Modifying 2-char symbols

   < vs <= , > vs >= , = vs == and !=

   If first character is identified, then change state from START to proper state and check second character whether it is ‘=’ or not. Depending on second character, change state into DONE except ‘!=’. Because there is no matching token for just ‘!’, ‘!’ is handled as ERROR.

3. **Comments**

   Comments are more difficult than above 2 cases because there is more cases and it consist of opening comment and optional closing comment.

   When a ‘/’ is encountered on this loop, it can be divided into two states that OVER and COMMENT. So, we have to check next character whether it is ‘*’ or not. If it is ‘*’, then it is the state of COMMENT, otherwise OVER.

   EXCEPTION

1. There is ONLY opening comment.

   All of the characters before EOF are handled as comments, so those are ignored.

2. There is ONLY closing comment.

   Because there is no opening comment, it is not effective token as comment. They are just two tokens as ‘*’ and ‘/’.

4. Add C-Minus Tokens

   To handle addtional C-Minus Tokens, add more states and cases about them.

3. Method 2: LEX(FLEX)

```
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

Explanation

A LEX is program that automatically describe lexer. In LEX, there are three important sections.

1. Definition Section

   This Section is for header, declaration of variables, setting name using regular expression.

2. Rule Section

   Here is a space for setting rule for tokens what state will be returned and what actions will be taken.

3. Subroutine Section

   This is a section to define functions that is not in lex library. A newly defined function is included in lex.yy.c when this is created.

Modifying cminus.l

1. Definition Section

   To meet C-Minus Lexical Convention, I adjust identifier as {letter}({letter}|digit})*.

2. Rule Section

   Adding and Removing keyword, symbols.

   Modifying comments is that I checked two characters, ‘c’ and ‘temp’, to identify ‘*’ and ‘/’. The loop is keep repeating until “*/” is founded or EOF is encountered.

3. Subroutine Section

   There is no small modification about ERROR to terminate program with error message.

4. Test & Result

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

[Input file test.1.txt]

```
root@DESKTOP-UATIUE2:~/gitlab/2022_ele4029_2018008240/loucomp# ./cminus_cimpl test.1.txt

C-MINUS COMPILATION: 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
        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
```

[Result of ./cminus_cimpl test.1.txt]

```
root@DESKTOP-UATIUE2:~/gitlab/2022_ele4029_2018008240/loucomp# ./cminus_lex test.1.txt

C-MINUS COMPILATION: 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
        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
```

[Result of ./cminus_lex test.1.txt]

```
void main(void)
{
  int i; int x[5];

  i = 0;
  while( i < 5 )
  {
    x[i] = input();

    i = i + 1;
```

```
  }

  i = 0;
  while( i <= 4 )
  {
    if( x[i] != 0 )
    {
      output(x[i]);
    }
  }
}
```

[Input file test.2.txt]

```
root@DESKTOP-UAT1UE2:~/gitlab/2022_ele4029_2018008240/1_Scanner# ./cminus_cimpl test.2.txt

C-MINUS COMPILATION: test.2.txt
       1: reserved word: void
       1: ID, name= main
       1: (
       1: reserved word: void
       1: )
       2: {
       3: reserved word: int
       3: ID, name= i
       3: ;
       3: reserved word: int
       3: ID, name= x
       3: [
       3: NUM, val= 5
       3: ]
       3: ;
       5: ID, name= i
       5: =
       5: NUM, val= 0
       5: ;
       6: reserved word: while
       6: (
       6: ID, name= i
       6: <
       6: NUM, val= 5
       6: )
       7: {
       8: ID, name= x
       8: [
       8: ID, name= i
       8: ]
       8: =
       8: ID, name= input
       8: (
       8: )
       8: ;
       10: ID, name= i
       10: =
       10: ID, name= i
       10: +
       10: NUM, val= 1
       10: ;
       11: }
       13: ID, name= i
       13: =
       13: NUM, val= 0
       13: ;
       14: reserved word: while
       14: (
       14: ID, name= i
       14: <=
       14: NUM, val= 4
       14: )
       15: {
       16: reserved word: if
       16: (
       16: ID, name= x
       16: [
       16: ID, name= i
       16: ]
       16: !=
       16: NUM, val= 0
       16: )
       17: {
       18: ID, name= output
       18: (
       18: ID, name= x
       18: [
       18: ID, name= i
       18: ]
       18: )
       18: ;
       19: }
       20: }
       21: }
       22: EOF
```

[Result of ./cminus_cimpl test.2.txt]

```
root@DESKTOP-UAT1UE2:~/gitlab/2022_ele4029_2018008240/1_Scanner# ./cminus_lex test.2.txt

C-MINUS COMPILATION: test.2.txt
       1: reserved word: void
       1: ID, name= main
       1: (
       1: reserved word: void
       1: )
       2: {
       3: reserved word: int
       3: ID, name= i
       3: ;
       3: reserved word: int
       3: ID, name= x
       3: [
       3: NUM, val= 5
       3: ]
       3: ;
       5: ID, name= i
       5: =
       5: NUM, val= 0
       5: ;
       6: reserved word: while
       6: (
       6: ID, name= i
       6: <
       6: NUM, val= 5
       6: )
       7: {
       8: ID, name= x
       8: [
       8: ID, name= i
       8: ]
       8: =
       8: ID, name= input
       8: (
       8: )
       8: ;
       10: ID, name= i
       10: =
       10: ID, name= i
       10: +
       10: NUM, val= 1
       10: ;
       11: }
       13: ID, name= i
       13: =
       13: NUM, val= 0
       13: ;
       14: reserved word: while
       14: (
       14: ID, name= i
       14: <=
       14: NUM, val= 4
       14: )
       15: {
       16: reserved word: if
       16: (
       16: ID, name= x
       16: [
       16: ID, name= i
       16: ]
       16: !=
       16: NUM, val= 0
       16: )
       17: {
       18: ID, name= output
       18: (
       18: ID, name= x
       18: [
       18: ID, name= i
       18: ]
       18: )
       18: ;
       19: }
       20: }
       21: }
       22: EOF
```

[Result of ./cminus_lex test.2.txt]