ELE4029 1_Scanner Project Report

Afif Danial **2019048586**

This program operates the C-Scanner by modifying the Tiny Compiler and the lex code.

Compilation Environment

Ubuntu 18.0.4.5 LTS

Overview

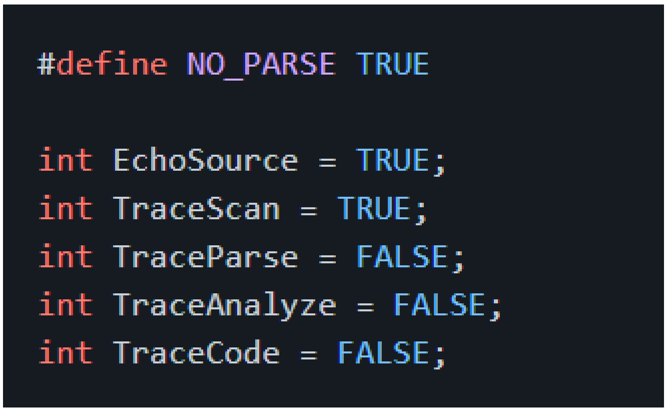
For C-Scanner production, Keyword, Symbol, and Token are defined according to C-grammar, and the character string input using DFA is Lexical Analysis in Token units.

Glossary

- keyword(lower case)
  - if, else, int, return, void, while
- Symbol
  - if, else, int, return, void, while
- Token
  - ID = letter letter *
  - NUM = digit digit*
  - letter = a-z / A-Z
  - digit = 0 - 9

Method 1: C code(scan.c) –“cminus_cimpl”

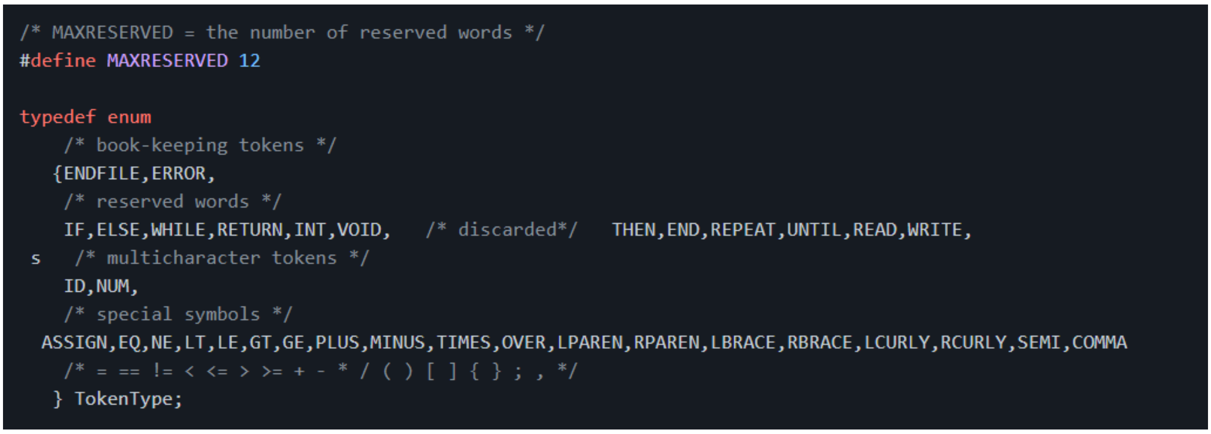
main.c



```
#define NO_PARSE TRUE

int EchoSource = TRUE;
int TraceScan = TRUE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```

Since C-Scanner is the only one that is produced in this program, the flags of **main.c** are modified.

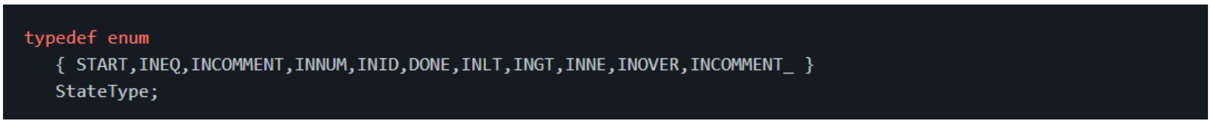
globals.h



```
/* MAXRESERVED = the number of reserved words */
#define MAXRESERVED 12

typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF,ELSE,WHILE,RETURN,INT,VOID,   /* discarded*/   THEN,END,REPEAT,UNTIL,READ,WRITE,
 s   /* 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;
```

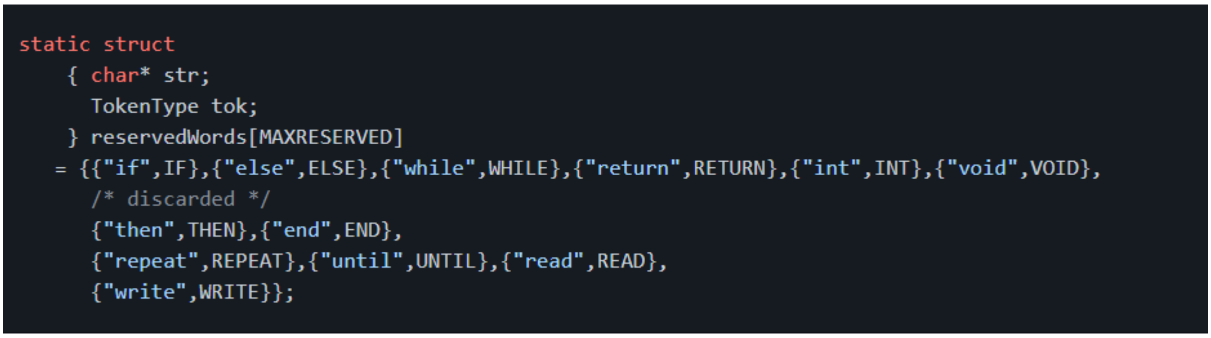
Add keywords and symbols of C- to the **TokenType** enum. At this time, change the value of **MAXRESERVED** to 12, which is the number after adding the keyword. For each enum name or symbol, refer to the code above.

scan.c



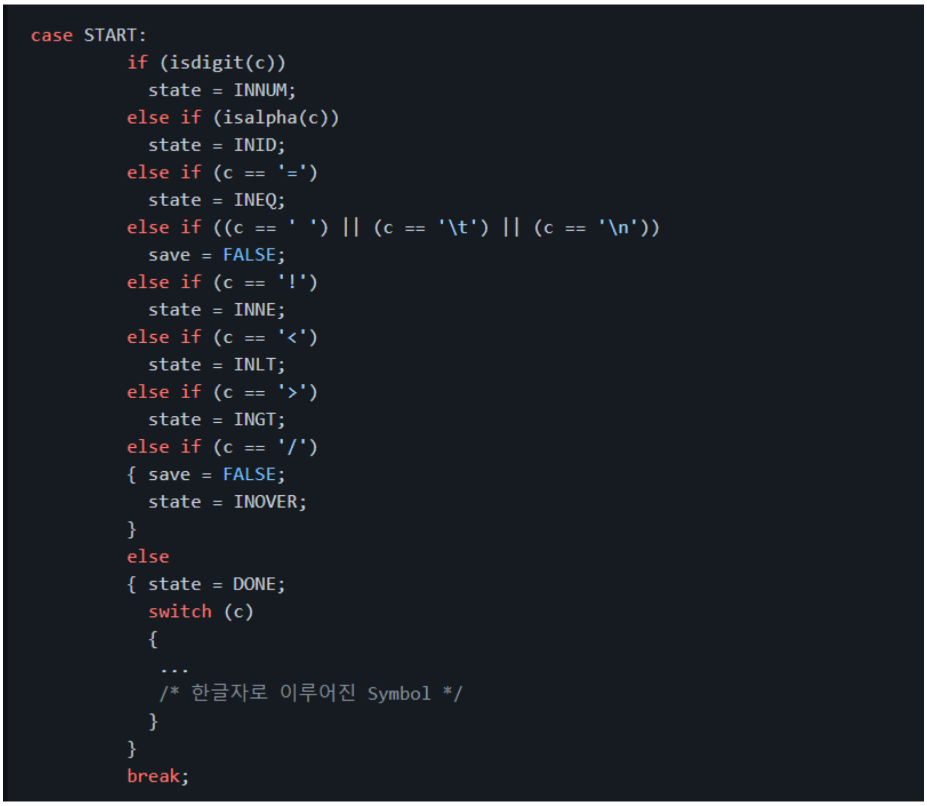
```
typedef enum
   { START,INEQ,INCOMMENT,INNUM,INID,DONE,INLT,INGT,INNE,INOVER,INCOMMENT_ }
   StateType;
```

DFA needs to be configured for C-Scanner, and states for ==, >=, <=, !=, /* */ are added in the state of the existing tiny compiler.



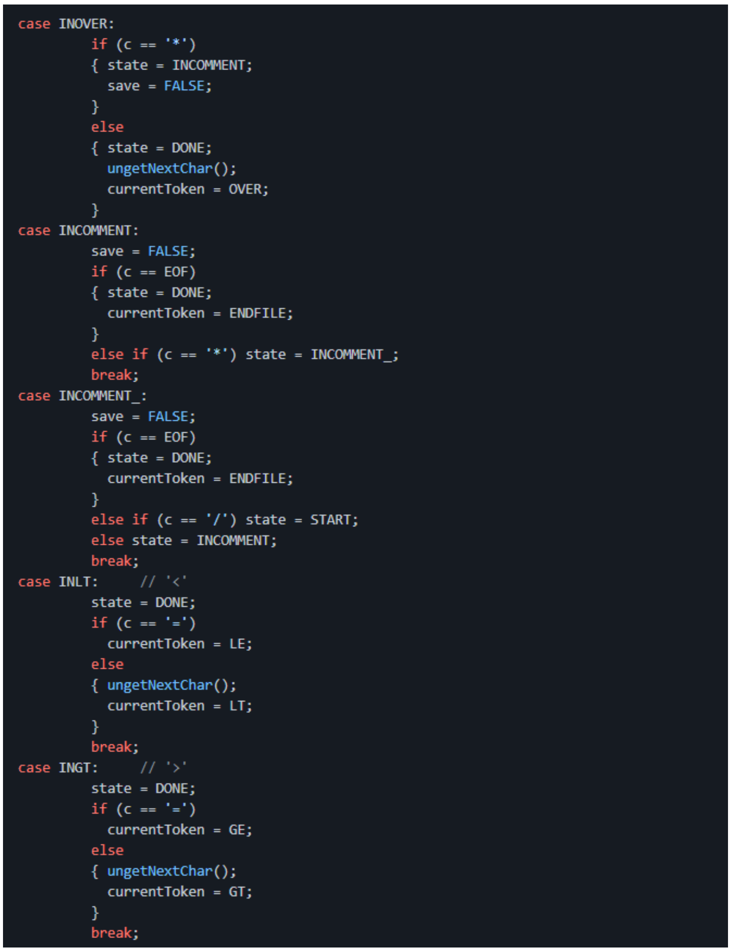
```
static struct
    { char* str;
      TokenType tok;
    } reservedWords[MAXRESERVED]
   = {{"if",IF},{"else",ELSE},{"while",WHILE},{"return",RETURN},{"int",INT},{"void",VOID},
      /* discarded */
      {"then",THEN},{"end",END},
      {"repeat",REPEAT},{"until",UNTIL},{"read",READ},
      {"write",WRITE}};
```

C- keywords are also added to **reservedwords**. Now, to perform DFA of C-, modify the part that processes symbols based on the state added above.

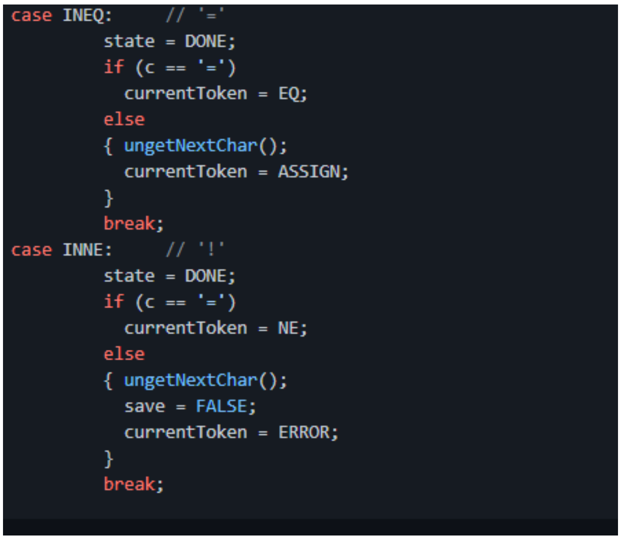


```
case START:
         if (isdigit(c))
           state = INNUM;
         else if (isalpha(c))
           state = INID;
         else if (c == '=')
           state = INEQ;
         else if ((c == ' ') || (c == '\t') || (c == '\n'))
           save = FALSE;
         else if (c == '!')
           state = INNE;
         else if (c == '<')
           state = INLT;
         else if (c == '>')
           state = INGT;
         else if (c == '/')
         { save = FALSE;
           state = INOVER;
         }
         else
         { state = DONE;
           switch (c)
           {
            ...
            /* 한글자로 이루어진 Symbol */
           }
         }
         break;
```

States (symbols) consisting of two or more characters, such as == (EQ), != (NE), <= (LE), >= (GE), /* */ (COMMENT) Create and enter that state. If this is not the case, it can be thought as a symbol consisting of Korean characters, thus it is processed according to the symbol.



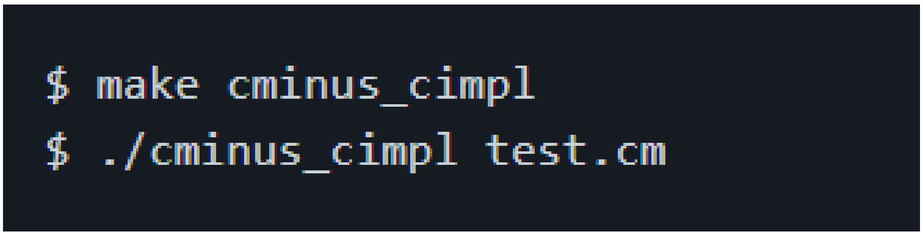
```
case INOVER:
         if (c == '*')
         { state = INCOMMENT;
           save = FALSE;
         }
         else
         { state = DONE;
           ungetNextChar();
           currentToken = OVER;
         }
case INCOMMENT:
         save = FALSE;
         if (c == EOF)
         { state = DONE;
           currentToken = ENDFILE;
         }
         else if (c == '*') state = INCOMMENT_;
         break;
case INCOMMENT_:
         save = FALSE;
         if (c == EOF)
         { state = DONE;
           currentToken = ENDFILE;
         }
         else if (c == '/') state = START;
         else state = INCOMMENT;
         break;
case INLT:     // '<'
         state = DONE;
         if (c == '=')
           currentToken = LE;
         else
         { ungetNextChar();
           currentToken = LT;
         }
         break;
case INGT:     // '>'
         state = DONE;
         if (c == '=')
           currentToken = GE;
         else
         { ungetNextChar();
           currentToken = GT;
         }
         break;
```



```
case INEQ:     // '='
         state = DONE;
         if (c == '=')
           currentToken = EQ;
         else
         { ungetNextChar();
           currentToken = ASSIGN;
         }
         break;
case INNE:     // '!'
         state = DONE;
         if (c == '=')
           currentToken = NE;
         else
         { ungetNextChar();
           save = FALSE;
           currentToken = ERROR;
         }
         break;
```

Since symbols composed of two or more characters belong to IN~ state, if the next character is a character suitable for the corresponding symbol, the current state is set as the state of the corresponding symbol. At this time, comments are processed by dividing the state (INCOMMENT->INCOMMENT_) when the first slash (over) appears (INOVER->INCOMMENT) and when the second *(times) appears.

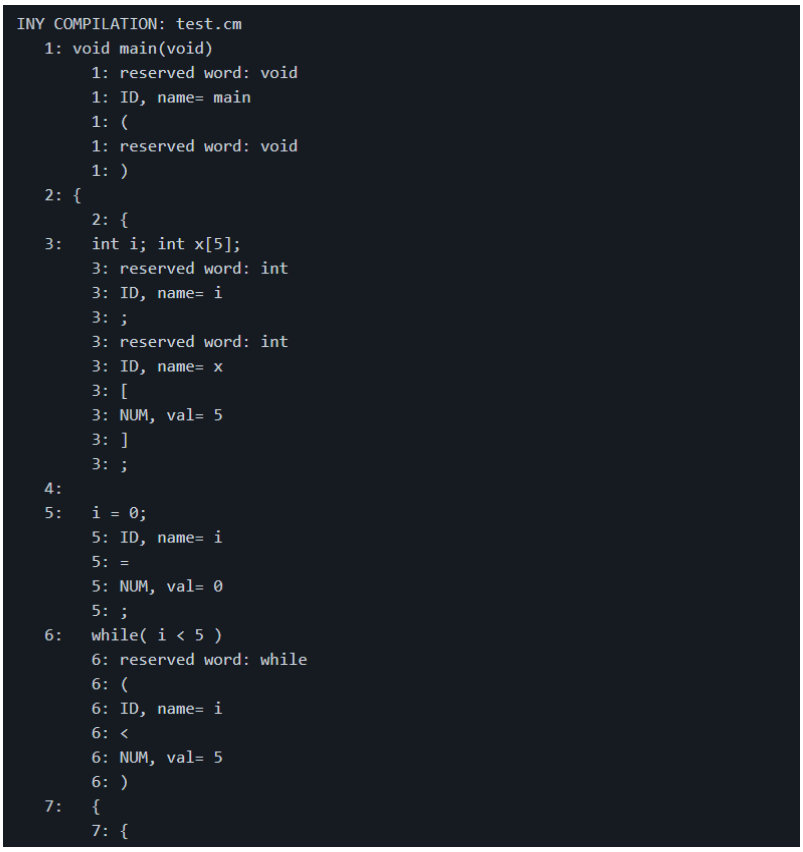
Implementation



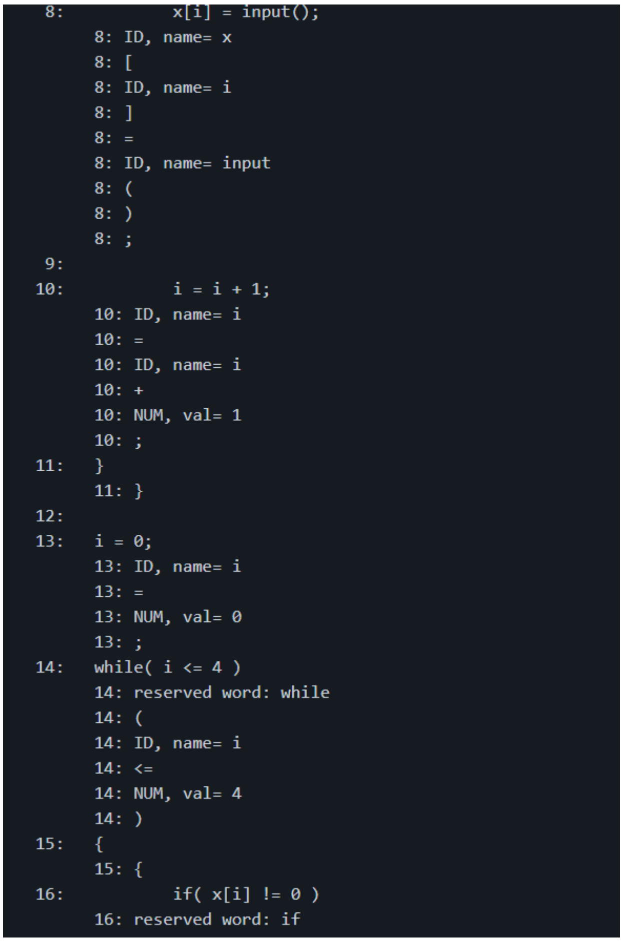
```
$ make cminus_cimpl
$ ./cminus_cimpl test.cm
```

Result

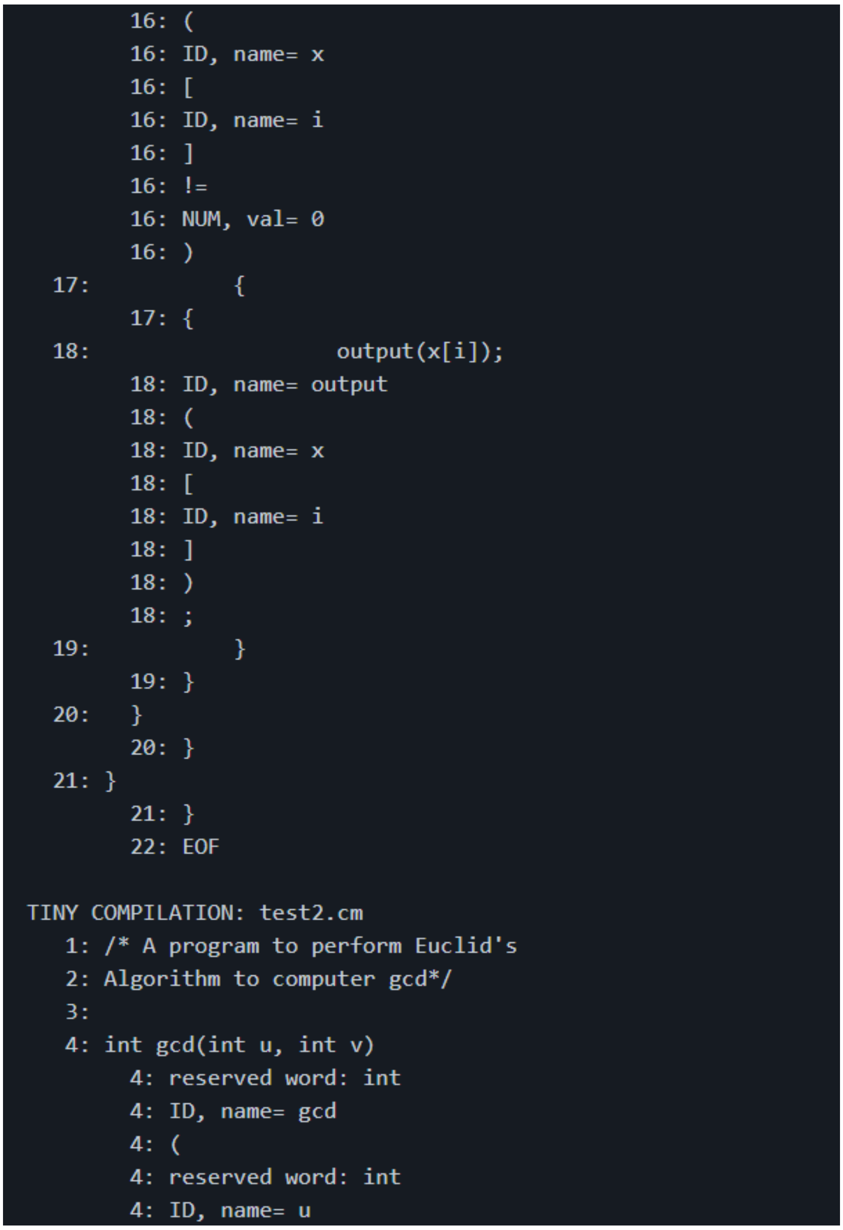
You can see the result at the screenshot below



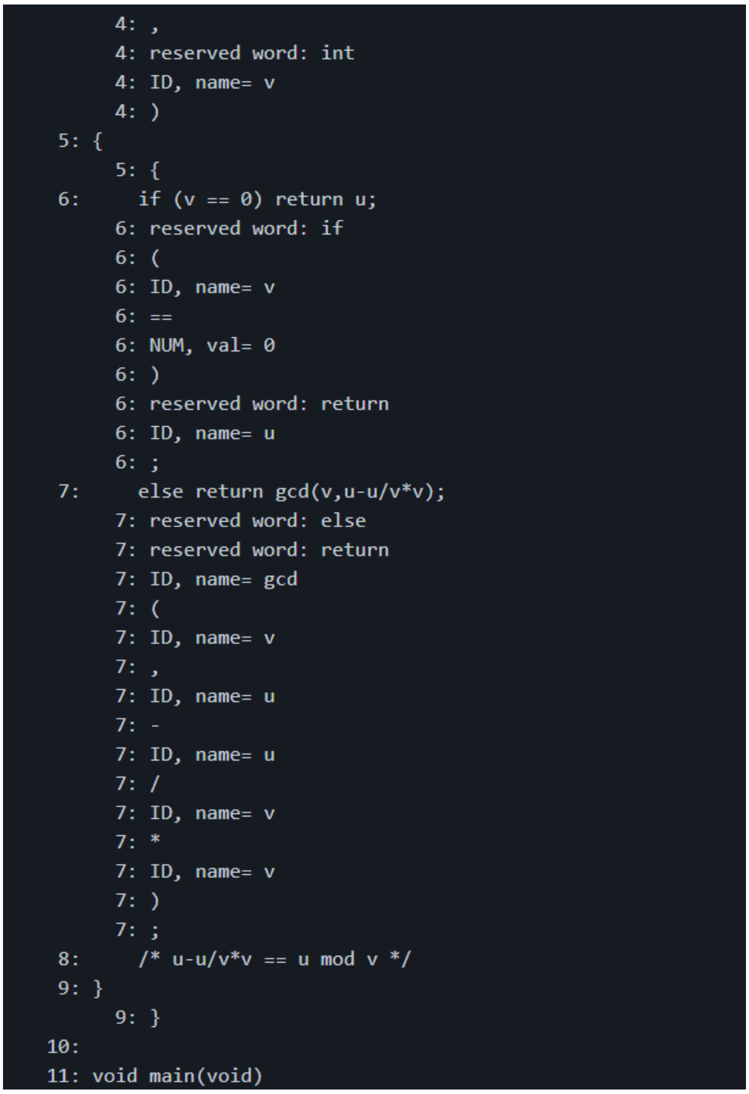
```
INY COMPILATION: test.cm
   1: void main(void)
        1: reserved word: void
        1: ID, name= main
        1: (
        1: reserved word: void
        1: )
   2: {
        2: {
   3:   int i; int x[5];
        3: reserved word: int
        3: ID, name= i
        3: ;
        3: reserved word: int
        3: ID, name= x
        3: [
        3: NUM, val= 5
        3: ]
        3: ;
   4:
   5:   i = 0;
        5: ID, name= i
        5: =
        5: NUM, val= 0
        5: ;
   6:   while( i < 5 )
        6: reserved word: while
        6: (
        6: ID, name= i
        6: <
        6: NUM, val= 5
        6: )
   7:   {
        7: {
```



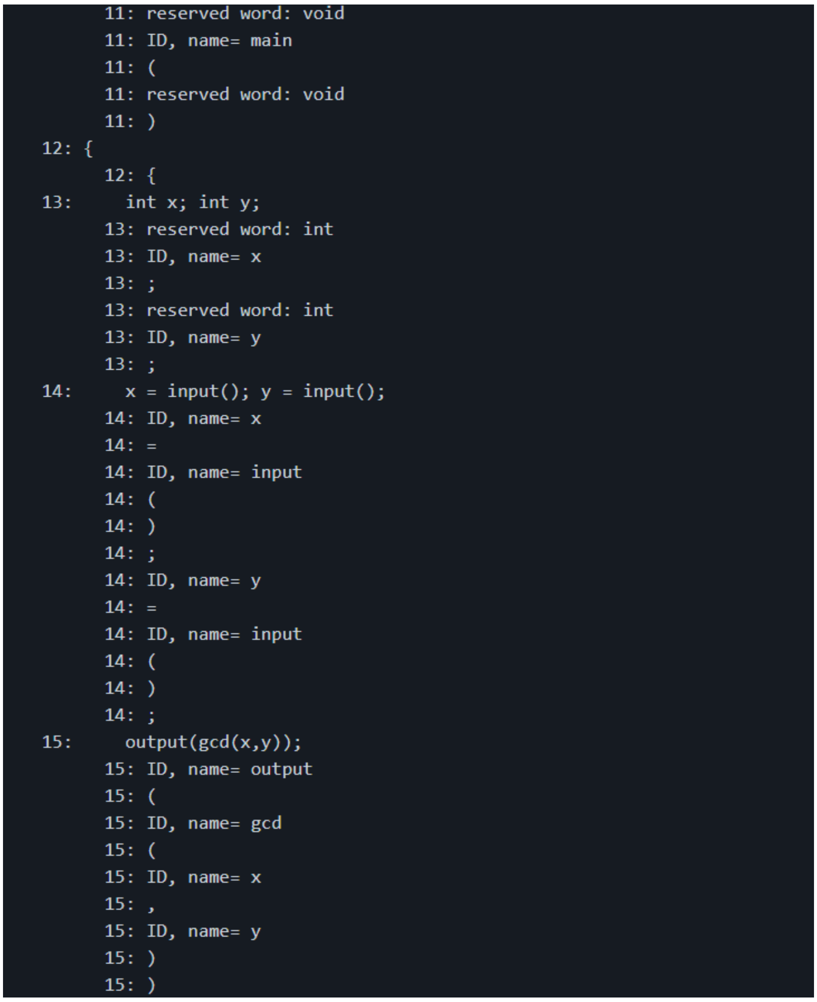
```
   8:           x[i] = input();
        8: ID, name= x
        8: [
        8: ID, name= i
        8: ]
        8: =
        8: ID, name= input
        8: (
        8: )
        8: ;
   9:
  10:           i = i + 1;
        10: ID, name= i
        10: =
        10: ID, name= i
        10: +
        10: NUM, val= 1
        10: ;
  11:   }
        11: }
  12:
  13:   i = 0;
        13: ID, name= i
        13: =
        13: NUM, val= 0
        13: ;
  14:   while( i <= 4 )
        14: reserved word: while
        14: (
        14: ID, name= i
        14: <=
        14: NUM, val= 4
        14: )
  15:   {
        15: {
  16:           if( x[i] != 0 )
        16: reserved word: if
```



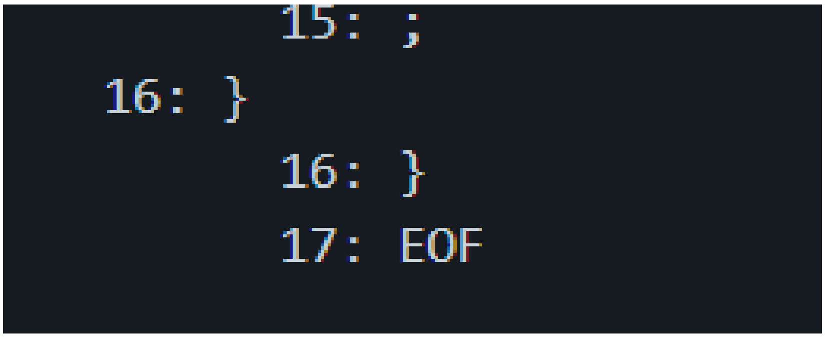
```
        16: (
        16: ID, name= x
        16: [
        16: ID, name= i
        16: ]
        16: !=
        16: NUM, val= 0
        16: )
  17:           {
        17: {
  18:                   output(x[i]);
        18: ID, name= output
        18: (
        18: ID, name= x
        18: [
        18: ID, name= i
        18: ]
        18: )
        18: ;
  19:           }
        19: }
  20:   }
        20: }
  21: }
        21: }
        22: EOF

TINY COMPILATION: test2.cm
   1: /* A program to perform Euclid's
   2: Algorithm to computer gcd*/
   3:
   4: int gcd(int u, int v)
        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: {
      5: {
 6:     if (v == 0) return u;
      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:     else return gcd(v,u-u/v*v);
      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: ;
 8:     /* u-u/v*v == u mod v */
 9: }
      9: }
10:
11: void main(void)
```



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

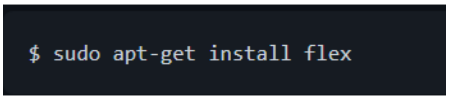


```
      15: ;
16: }
      16: }
      17: EOF
```

Method 2: Lex(flex)(cminus.l) –"cminus_lex"

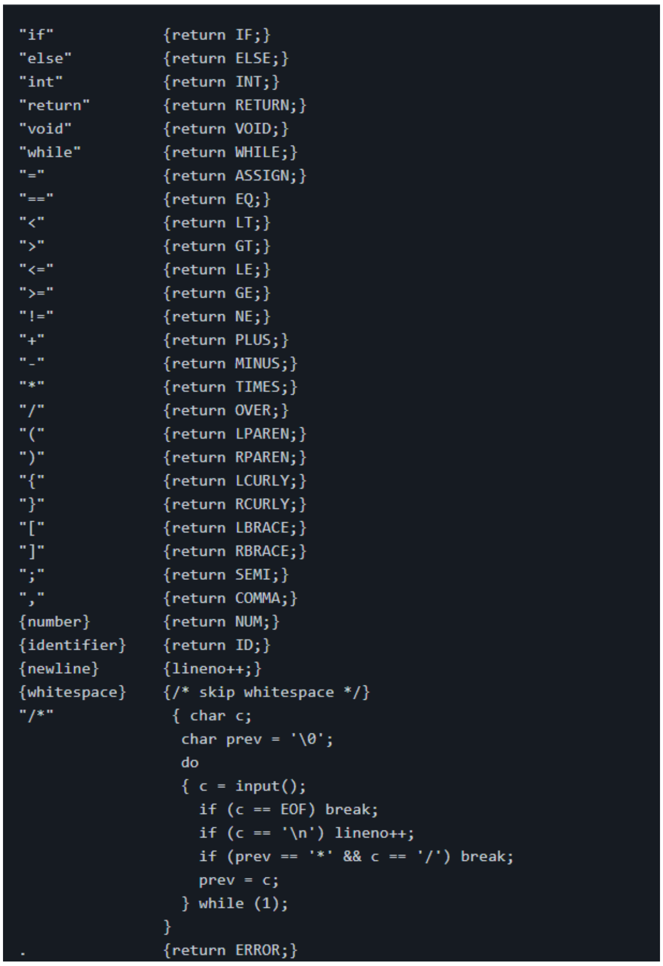
Setup

We have to install flex on our ubuntu by typing this command on the terminal.



```
$ sudo apt-get install flex
```

cminus.l

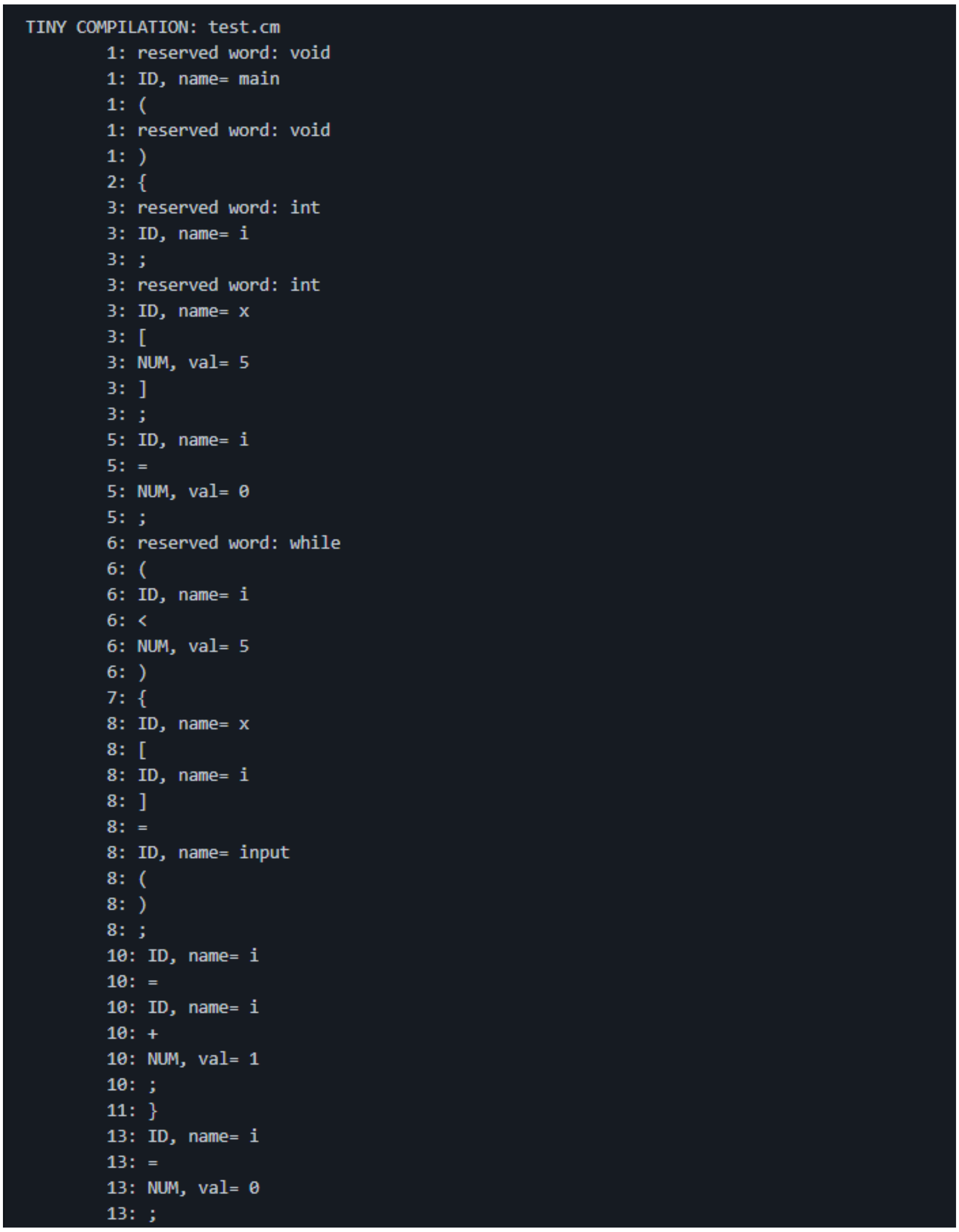


```
"if"            {return IF;}
"else"          {return ELSE;}
"int"           {return INT;}
"return"        {return RETURN;}
"void"          {return VOID;}
"while"         {return WHILE;}
"="             {return ASSIGN;}
"=="            {return EQ;}
"<"             {return LT;}
">"             {return GT;}
"<="            {return LE;}
">="            {return GE;}
"!="            {return NE;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
";"             {return SEMI;}
","             {return COMMA;}
{number}        {return NUM;}
{identifier}    {return ID;}
{newline}       {lineno++;}
{whitespace}    {/* skip whitespace */}
"/*"             { char c;
                  char prev = '\0';
                  do
                  { c = input();
                    if (c == EOF) break;
                    if (c == '\n') lineno++;
                    if (prev == '*' && c == '/') break;
                    prev = c;
                  } while (1);
                }
.               {return ERROR;}
```

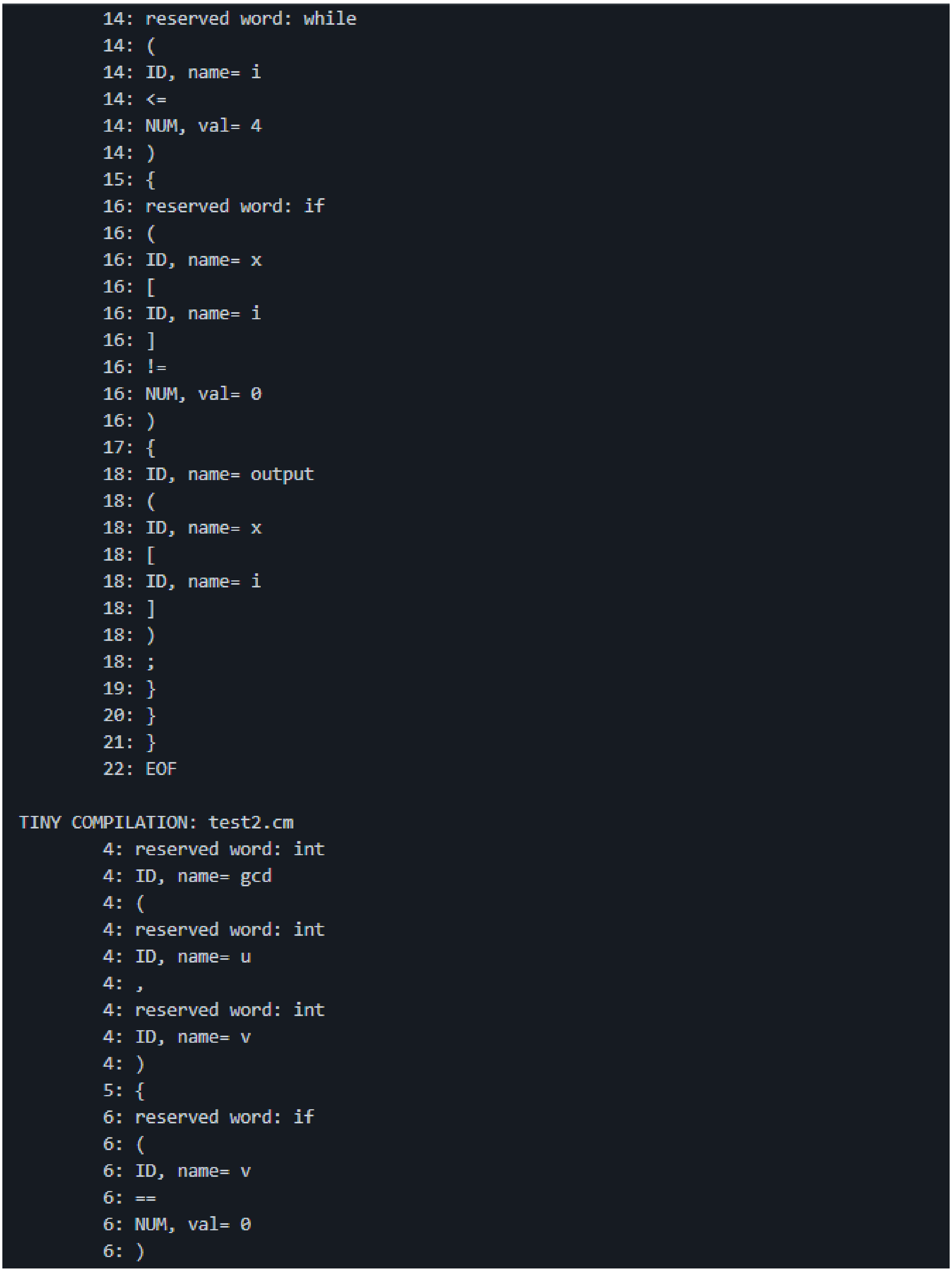
Because it automatically creates a C-lexer using flex, I only need to add C- keywords and symbols. At this time, since the comment had compared two characters, thus it is processed by declaring the prev variable that stores at the previous character.

Result

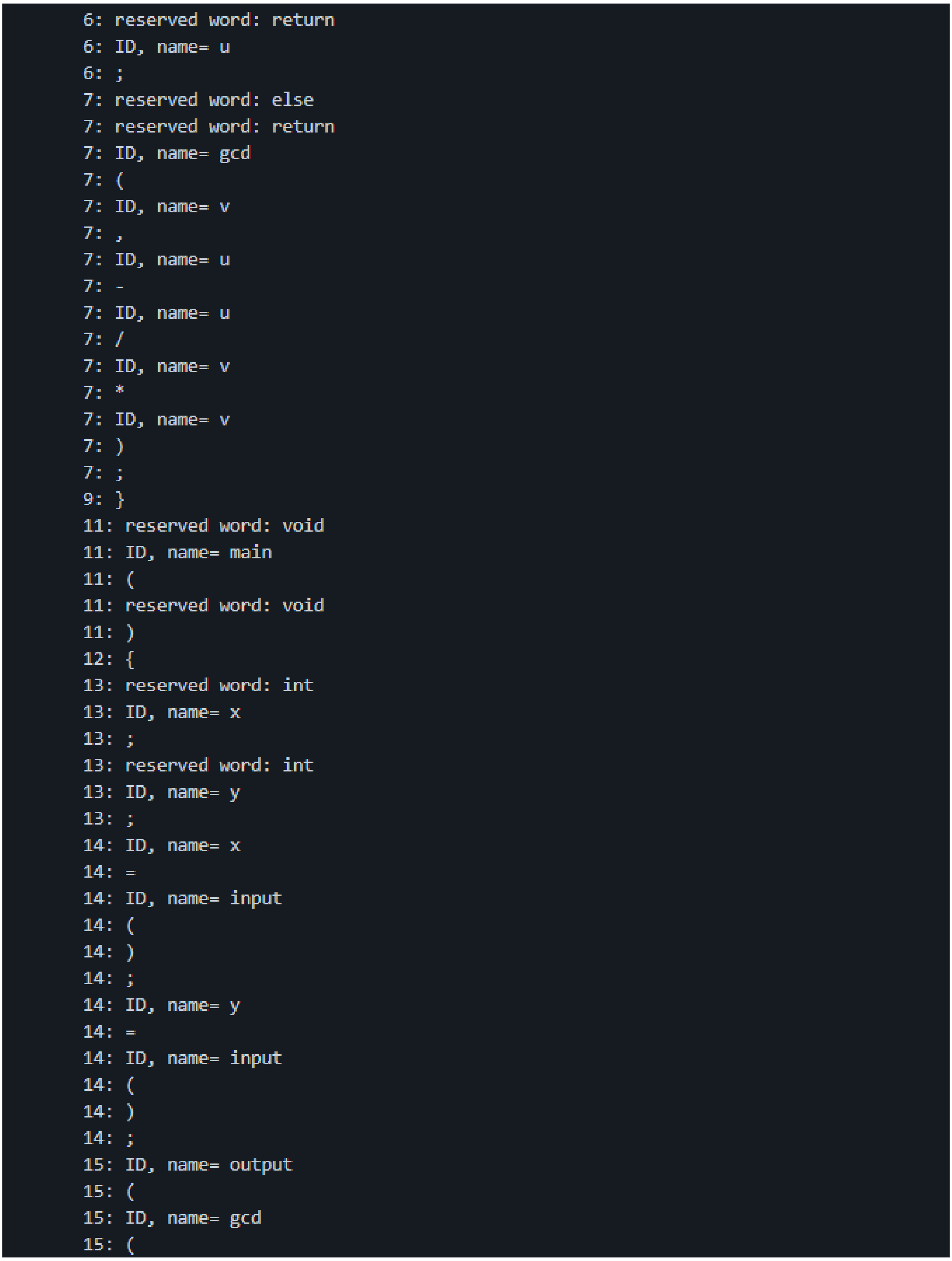
You can see the result at the screenshot below



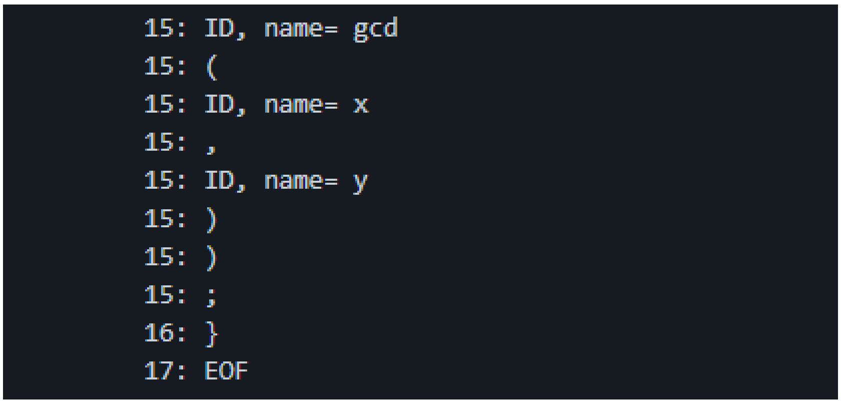
```
TINY COMPILATION: test.cm
        1: reserved word: void
        1: ID, name= main
        1: (
        1: reserved word: void
        1: )
        2: {
        3: reserved word: int
        3: ID, name= i
        3: ;
        3: reserved word: int
        3: ID, name= x
        3: [
        3: NUM, 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

TINY COMPILATION: test2.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= gcd
        15: (
        15: ID, name= x
        15: ,
        15: ID, name= y
        15: )
        15: )
        15: ;
        16: }
        17: EOF
```