Assignment 1

Scanner

과목명	컴파일러
담당교수	박영준 교수님
학생이름	최윤석
학과	컴퓨터 소프트웨어 학과
학번	2015005169
제출일	2021.10.18

HANYANG UNIVERSITY

1. Requirements

■ 목표 : C-minus Scanner의 구현

■ Lexical Convention of C-Minus

- C code를 통한 scan은 scan.c파일을 수정하며 `cminus_cimpl` 실행파일을 통해 동작시킨다.
- LEX(FLEX)를 이용한 scan은 cminus.l파일로 구현하며 `cminus_lex` 실행파일을 통해 동작시킨다.
- Reserved Words(Keywords)로 구분되는 단어는 아래 6개의 단어이다.
  : `int`, `void`, `if`, `else`, `while`, `return`
- Symbols로 구분되는 것은 아래와 같다.
  : `+`, `-`, `*`, `/`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `=`, `;`, `,`, `(`, `)`, `[`, `]`, `{`, `}`
- Identifier와 Number는 아래와 같다.
  : ID = letter (letter | digit)*
  NUM = digit digit*
  letter = a | b | ... | z | A | B | ... | Z |
  digit = 0 | 1 | ... | 9
- White space는 spacebar, enter, tab이 포함되며 위 경우 무시한다.
- Comment는 `//`는 속하지 않고 `/* */`만 해당된다.

2. 컴파일 진행 환경

Ubuntu 20.04.3 LTS에서 컴파일을 진행하였으며,
Makefile은 주어진 아래 Makefile을 수정없이 사용하였습니다.

```
# Makefile for C-Minus Scanner
# ./lex/tiny.l --> ./cminus.l

CC = gcc

CFLAGS = -W -Wall

OBJS = main.o util.o scan.o
OBJS_LEX = main.o util.o lex.yy.o

.PHONY: all clean
all: cminus_cimpl cminus_lex

clean:
  -rm -vf cminus_cimpl cminus_lex *.o lex.yy.c

cminus_cimpl: $(OBJS)
  $(CC) $(CFLAGS) -o $@ $(OBJS)

cminus_lex: $(OBJS_LEX)
```

```
	$(CC) $(CFLAGS) -o $@ $(OBJS_LEX) -lfl

main.o: main.c globals.h util.h scan.h
	$(CC) $(CFLAGS) -c -o $@ $<

scan.o: scan.c globals.h util.h scan.h
	$(CC) $(CFLAGS) -c -o $@ $<

util.o: util.c globals.h util.h
	$(CC) $(CFLAGS) -c -o $@ $<

lex.yy.o: lex.yy.c globals.h util.h scan.h
	$(CC) $(CFLAGS) -c -o $@ $<

lex.yy.c: cminus.l
	flex -o $@ $<
```

3. 구현과정

- makefile과 pdf를 참조하여 코드 작성을 시작하였습니다.
시작은 pdf에 기술되어 있는대로 `main.c`와 `global.h`, `utils.c`를 우선적으로 수정해주었습니다.

<`main.c` 수정> : NO_PARE와 TRACE SCAN을 TRUE로 변경해주었습니다.

```
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
```

```
39 /* allocate and set tracing flags */
40 int EchoSource = FALSE;
41 int TraceScan = TRUE;
42 int TraceParse = FALSE;
RMAL  main.c
```

<`globals.h` 수정>
: 요구사항에서 주어진 keywords가 6개였기 때문에 `MAXRESERVED`값을 6으로 변경하고 `TokenType` 내 `reserved words`와 `special symbols`를 요구사항에 주어진대로 수정하였습니다.

```
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;
```

<`utils.c` 수정>

: output format을 참고하여 `printToken()` 함수 부분을 작성하였습니다.

1. 위 함수는 state가 `DONE`이고 TraceScan값이 `TRUE`일 때(main.c에서 TRUE로 수정함) current token과 tokenString을 매개변수로 받고, 넘겨받은 token에 맞는 출력을 수행하는 함수입니다.

매개변수로 받은 token이 reserved word인 IF, ELSE, WHILE, RETURN, INT, VOID의 경우 reserved word : ~~~ 로 출력될 수 있도록 switch문을 수정했습니다.

```
15 void printToken( TokenType token, const char* tokenString )
16 { switch (token)
17   { case IF:
18     case ELSE:
19     case WHILE:
20     case RETURN:
21     case INT:
22     case VOID:
23       fprintf(listing,
24          "reserved word: %s\n",tokenString);
25       break;
```

2. 매개변수로 받는 token이 special symbols로 지정해둔 19개의 symbol중 하나일 경우 각 symbol이 적절하게 출력될 수 있도록 아래와 같이 코드를 수정해 주었습니다.

```
26     case ASSIGN: fprintf(listing,"=\n"); break;
27     case EQ: fprintf(listing,"==\n"); break;
28     case NE: fprintf(listing,"!=\n"); break;
29     case LT: fprintf(listing,"<\n"); break;
30     case LE: fprintf(listing,"<=\n"); break;
31     case GT: fprintf(listing,">\n"); break;
32     case GE: fprintf(listing,">=\n"); break;
33     case PLUS: fprintf(listing,"+\n"); break;
34     case MINUS: fprintf(listing,"-\n"); break;
35     case TIMES: fprintf(listing,"*\n"); break;
36     case OVER: fprintf(listing,"/\n"); break;
37     case LPAREN: fprintf(listing,"(\n"); break;
38     case RPAREN: fprintf(listing,")\n"); break;
39     case LBRACE: fprintf(listing,"[\n"); break;
40     case RBRACE: fprintf(listing,"]\n"); break;
41     case LCURLY: fprintf(listing,"{\n"); break;
42     case RCURLY: fprintf(listing,"}\n"); break;
43     case SEMI: fprintf(listing,";\n"); break;
44     case COMMA: fprintf(listing,",\n"); break;
```

3. `ENDFILE`, `NUM`, `ID`, `ERROR`, `default`의 경우 기존에 존재하는 코드를 그대로 유지하였습니다.

[Scanner - Method 1 : C code를 사용하는 scan]

여기서부터는 C code를 사용하여 scanner의 기능을 구현하기 위해 작성한 부분입니다.

< `scan.c` 수정>

1. scan.c에서 reserved word를 찾는 부분인 `reservedWords[MAXRESERVED] struct`부분을 수정하였습니다.

`if, else, while, return, int, void`를 `IF, ELSE, WHILE, RETURN, INT, VOID`로 아래와 같이 struct로 구성했습니다.

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

2. DFA와 같은 역할을 수행하는 `getToken()` function을 수정하였습니다.

`getToken()`함수는 START state에서 시작하여 state가 DONE으로 변경될 때 까지 글자를 하나씩 읽어 state를 변경합니다. state가 DONE이 될 경우 save해둔 character를 token string에 넣어주고 TraceScan이 켜져있기 때문에 printToken을 호출하는 기능을 수행합니다.

`getToken()`을 수정하기 전 DFA의 state를 정의하기 위해 pdf힌트를 참조하여 아래와 같이 `StateType enumeration`을 정의했습니다.

```
12 /* states in scanner DFA */
13 typedef enum
14    { START,INEQ,INNE,INLT,INGT,INOVER,INCOMMENT,INCOMMENT_,INNUM,INID,DONE }
15    StateType;
16
```

`getToken()`의 while문 내부에 들어가면 `getNextChar()`를 통해 글자를 하나 읽어 `c`에 할당합니다.

1) state가 Start일 경우

숫자인지, 알파벳인지, =, !, <, >, /인지 white space인지, 나머지인지에따라 구분합니다.

=, !, <, >, /를 따로 구분한 이유는 =를 예시로 들면 ==, = 와 같이 경우의 수가 존재할 수 있기 때문이며, 각각 INEQ, INNE, INLT, INGT, INOVER state로 변경될 수 있게 지정해주었습니다.

else부분은 애매모호하지 않은 token들, EndOfFile이거나 +, -, *, (,), [,], {, }들이 포함되어 있으며 각각 자신의 token에 맞게 current Token을 변경해 줍니다. 이런 token들은 결과에 한 줄로 출력을 할 것이기 때문에 state역시 DONE으로 변경해줍니다. 모든 코드에 공통된 사항이라 else구문의 가장 첫줄에 작성하였습니다.

```
86    while (state != DONE)
87    { int c = getNextChar();
88      save = TRUE;
89      switch (state)
90      { case START:
91          if (isdigit(c))
92            state = INNUM;
93          else if (isalpha(c))
94            state = INID;
95          else if (c == '=')
96            state = INEQ;
97          else if (c == '!')
98            state = INNE;
99          else if (c == '<')
00            state = INLT;
01          else if (c == '>')
02            state = INGT;
03          else if (c == '/')
04          {
05            save =FALSE;
06            state = INOVER;
07          }
08          else if ((c == ' ') || (c == '\t') || (c == '\n'))
09            save = FALSE;
10          else
11          { state = DONE;
12            switch (c)
13            { case EOF:
14                save = FALSE;
15                currentToken = ENDFILE;
16                break;
```

2) state가 INEQ일 경우

`=`, `==` 두 가지 경우의 수가 존재하며 두 경우 모두 state를 DONE으로 바꿔야 합니다.

`=`라면 읽어들인 c가 다시 사용되어야함으로 `ungetNextChar()`를 호출해줍니다. 또한 tokenString에 포함되지 않도록 save를 FALSE로 변경하며 currentToken을 ASSIGN으로 지정해줍니다.

`==`라면 이번에 읽은 c가 '='일 것이기 때문에 currentToken만 EQ로 변경해줍니다.

(INEQ만 사진을 첨부합니다. 나머지 state는 유사하기 때문에 자세한 구현은 코드파일을 참고해주세요)

```
189         case INEQ:
190           state = DONE;
191           if (c == EOF)
192             currentToken = ENDFILE;
193           else if (c=='=')
194             currentToken = EQ;
195           else
196           {
197             ungetNextChar();
198             save = FALSE;
199             currentToken = ASSIGN;
200           }
201           break;
202         case INNE:
203           state = DONE;
```

3) state가 INNE인 경우

!=일 경우와 !가 단독으로 오는 ERROR인 상황이 존재합니다.

!=가 되기 위해서는 현재 c가 '='일 것이며 해당하는 경우 currentToken을 NE로 설정해줍니다.

나머지는 else문에서 ungetNextChar();를 호출해주고 save를 FALSE로 변경하며 currentToken을 ERROR로 설정해줍니다.

4) state가 INLT인 경우
<=일 경우와 <일 경우 두가지 존재합니다.
c가 =인경우 currentToken을 LE로 나머지 경우 LT로 설정해줍니다.
LT는 당연하지만 ungetNextChar();를 통해 읽은 c를 되돌려놓고 save는 FALSE로 바꿔줍니다.

5) state가 INGT인 경우
>=일 경우와 >일 경우 두가지 존재합니다.
c가 =인경우 currentToken을 GE로 나머지는 GT로 설정해줍니다.

6) state가 INOVER일 경우 /(over)연산인 경우와 /* */ (comment)일 경우가 존재합니다.
만약 '*'가 온다면 state를 INCOMMENT로 변경하고 save는 FALSE로 설정합니다.

```
case INOVER:
  save = FALSE;
  if ( c== 'EOF'){
    state = DONE;
    currentToken = ENDFILE;
  }
  else if (c == '=')
    state = INCOMMENT;
```

그 외의 경우 /(over)연산이라는 의미이기 때문에 아래와 같이 처리했습니다.

```
  else
  {
    save = TRUE;
    ungetNextChar();
    c = '/';
    state = DONE;
    currentToken = OVER;
  }
  break;
```

⚠️ **Feed Back**
`c='/';`코드를 추가해준 이유는 startstate에서 /를 통해 INOVERstate로 넘어올 때 save를 하지 않았기 때문이다. 처음 코드를 작성할 때 INOVER로 넘어가는 부분의 save를 TRUE로 설정하였다가, COMMENT이후의 reserved word가 `ID, name = /(reserved word)` 와 같이 출력되어 이를 해결하기 위해 변경했다.
over연산의 경우 /가 출력되야 하기 때문에 따로 save를 TRUE로 변경하고 state를 DONE으로 변경하여 터미널 창에 정상적으로 출력될 수 있게 만들었다.

7) state가 INCOMMENT인 경우
INOVER를 거쳐 INCOMMENT로 왔다는 것은 앞에 `/*`가 나와서 `*/`가 나올 때 까지 모든 글자를 무시해야 한다는 의미입니다.
따라서 save를 FALSE로 설정하고 아래와 같이 경우를 나누어 처리했습니다.
- '*'가 나오는 경우 : INCOMMENT_ state로 간다.
- 나머지 단어가 나올 경우 : 그대로 INCOMMENT state에 머무른다.

8) state가 INCOMMENT_인 경우
INCOMMENT에서 넘어왔을 때 *가 하나 입력된 상태입니다.
여기서 /가 아닌 다른 문자가 올 경우 다시 */를 인식하기 위해 INCOMMENT state로 가야합니다.
다시 '*'가 나올경우 INCOMMENT_ state에 그대로 머무릅니다.
/일 경우 주석이 끝났기 때문에 state를 START로 변경해줍니다.

9) state가 INNUM일 경우
c가 숫자일 동안 계속하여 string을 저장하다가 digit이 아닐경우
ungetNextChar()를 호출하고 save를 FALSE로 바꾸며 NUM을 다 읽었다는 의미이기 때문에 state를 DONE으로 바꾸고 currentToken을 NUM으로 설정합니다.

10) state가 INID일 경우
요구사항에서 ID는 letter가 한글자 나온후 (letter | digit)*이라고 하였기 때문에
if (!isalpha(c) && !isdigit(c))로 코드를 작성하여 알파벳이나 숫자가 아닌 문자가 나왔을 경우 ungetNextChar()를 호출하고, save를 FALSE로 바꾸며, state를 DONE으로 변경하고 currentToken을 ID로 설정합니다.

[Scanner - Method 2 : LEX를 사용하는 scan]

tiny.l파일을 참고하여 cminus.l파일을 작성했습니다.
Definition Section은 아래와 같이 구성했습니다.

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

기존과 비교하면 identifier만 변경되었습니다.
letter 한글자가 나온 이후 letter, digit이 0개 이상 순서에 상관없이 나올 수 있기 때문에 위와 같이 작성하였습니다.
Rule section부분은 if, else, while, return, int, void, =, ==, <, <=, >, >=, +, -, *, /, (,), [,], {, }, ;, ,에 해당하는 값이 들어오면 각각 {return IF;}, {return ELSE;}, .. , {return SEMI;}, {return COMMA;}을 수행할 수 있게 코드를 구성했습니다.

number, identifier, newline, whitespace는 그대로 사용했습니다.

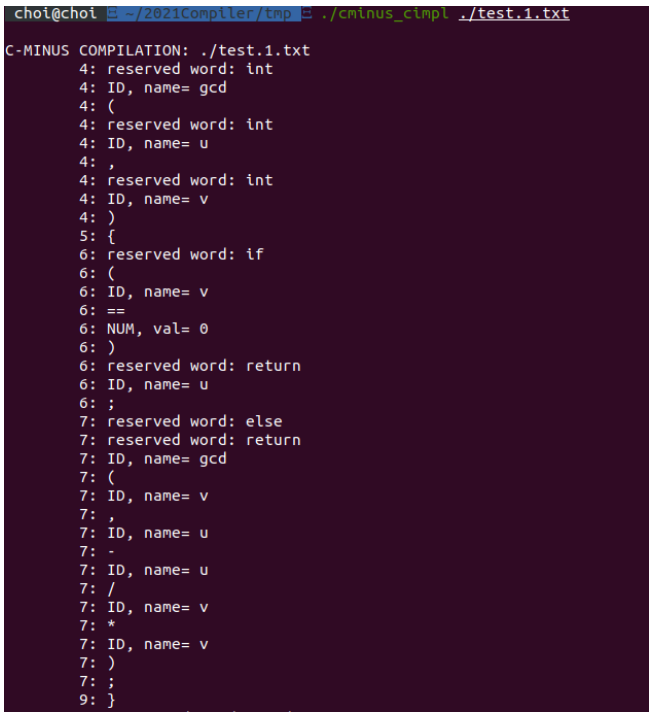
마지막으로 변경한 부분은 `/*` 라는 토큰을 인식했을 때 입니다.
아래와 같이 코드를 구성했습니다.

```
"/*"            { char c;
                  int state =0;
                  do
                  { c = input();
                    if (c == EOF) break;
                    else if(c=='*') state = 1;
                    else if (c=='/' && state==1) state=2;
                    else state=0;
                    if (c == '\n') lineno++;
                  } while (state != 2);
                }
.               {return ERROR;}
```

scan.c와 동일하게 동작하도록 만들기 위해
state라는 값을 추가하고
0은 INCOMMENT 1은 INCOMMENT_ 2는 START와 대응되게 했습니다.
이 cminus.l파일을 통해 토큰을 인식하는 과정을 lex.yy.c로 만들어내고, 이것을 컴파일하여 scanner의 역할을 수행할 수 있게 됩니다.

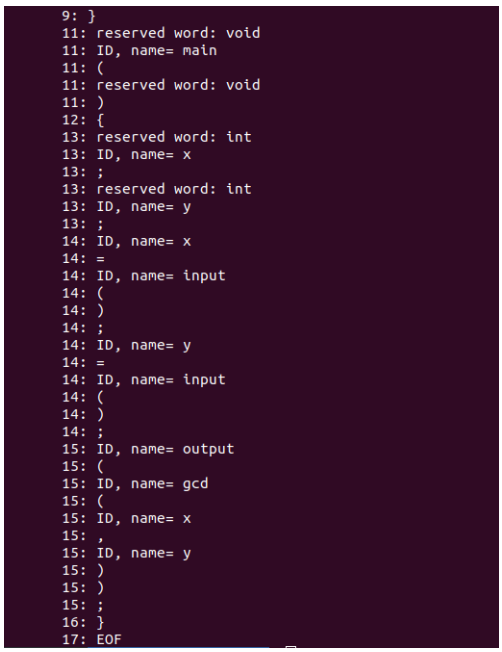
4. 수행결과

`./ciminus_cimpl ./test.1.txt` 수행결과



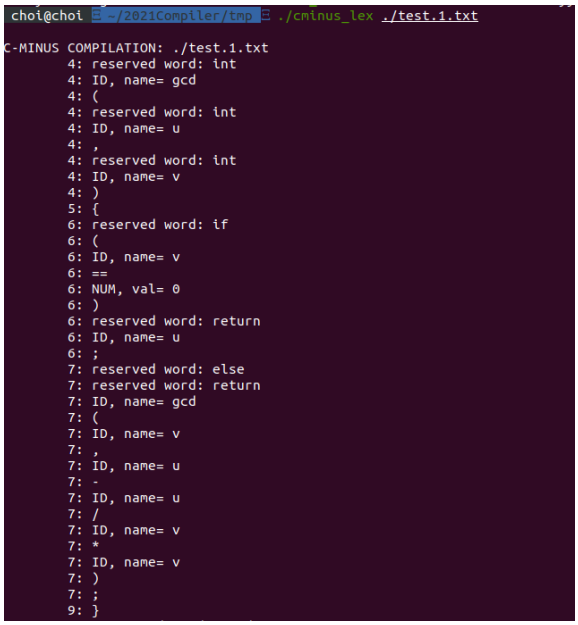
```
choi@choi ~/2021Compiler/tmp ./cminus_cimpl ./test.1.txt

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: }
```



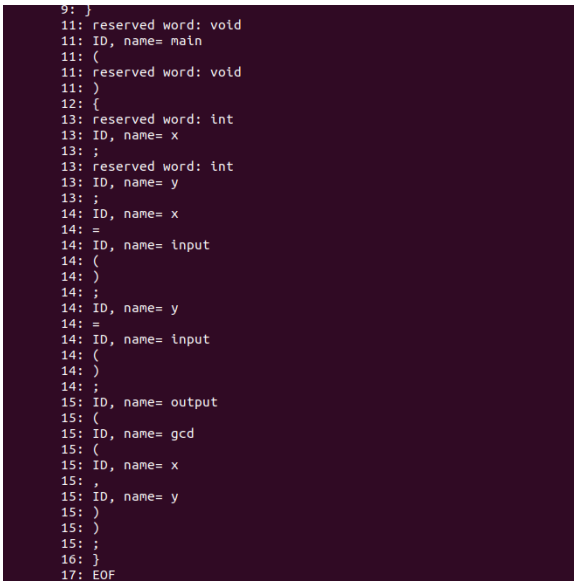
```
        9: }
       11: reserved word: void
       11: ID, name= main
       11: (
       11: reserved word: void
       11: )
       12: {
       13: reserved word: int
       13: ID, name= x
       13: ;
       13: reserved word: int
       13: ID, name= y
       13: ;
       14: ID, name= x
       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
```

`./ciminus_lex ./test.1.txt` 수행결과



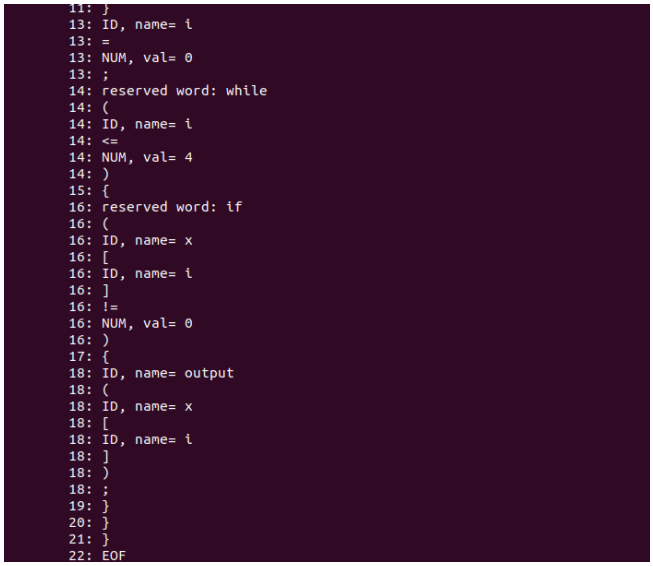
```
choi@choi ~/2021Compiler/tmp ./cminus_lex ./test.1.txt

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: }
```



```
        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
```

`./ciminus_cimpl ./test.2.txt` 수행결과



`./ciminus_lex ./test.2.txt` 수행결과