Report

> Project #1. Scanner 2020
>
> 2018000337
> 장호우

Environment

- Ubuntu 20.04.1 LTS
- flex 2.6.4
- gcc version 9.3.0

Part I. Implementation of C-Scanner using C-code

In `globals.h`

```
#define MAXRESERVED 12

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

To define what will be used in actual codes. The keywords reserved words, symbols and other necessary tokens are added to TokenType numbers.

In `scan.c`

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

To make state types are also mapped to StateType.

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

When compiler finished scanning and gets tokens, to check if the token is reserved word.

From line 111 to line 298 of the file as following:

```
  while (state != DONE)
  {
    int c = getNextChar();
    save = TRUE;
    switch (state)
    {
    case START:

      ...

    case INOVER:
      save = FALSE;
      if (c == '*')
      {
        state = INCOMMENT;
        tokenStringIndex--;
      }
      else
      {
        state = DONE;
        ungetNextChar();
        currentToken = OVER;
      }
      break;
    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 = INCOMMENT_;
        else if (c == '/')
          state = START;
        else
          state = INCOMMENT;
        break;

        ...

      if (state == DONE)
      {
        tokenString[tokenStringIndex] = '\0';
        if (currentToken == ID)
          currentToken = reservedLookup(tokenString);
      }
    }
```

The compiler tokenizing given input string streams, keeps calling `getNextChar()` until it meets `DONE` state. But the comments have total three states as following `INOVER`, `INCOMMENT`, `INCOMMENT_`. `INOVER` is state where compiler found '/'. This state figures out if '/' is used for `OVER` or beginning of comment '/*'. If compiler finds '*' after '/', the state changes to `INCOMMENT` state. `INCOMMENT` state is when tokenizer is inside the comment string. it is looking for '*'. It is part of end comment, '*/'. when compiler found '*' in `INCOMMENT` state, state changes to `INCOMMENT_` state. `INCOMMENT_` state is now looking for '/' character to end the comment. if it finds other characters, it goes back to `INCOMMENT` state or stays in `INCOMMENT_` state when the character is '*'.

In `util.c`

```
      case ASSIGN: fprintf(listing,"=\n"); break;
      case EQ: fprintf(listing,"==\n"); break;
      case NE: fprintf(listing,"!=\n"); break;
      case LT: fprintf(listing,"<\n"); break;
      case LE: fprintf(listing, "<=\n"); break;
      case GT: fprintf(listing, ">\n"); break;
```

```
    case GE: fprintf(listing,">=\n"); break;
    case LPAREN: fprintf(listing,"(\n"); break;
    case RPAREN: fprintf(listing,")\n"); break;
    case LBRACE: fprintf(listing,"[\n"); break;
    case RBRACE: fprintf(listing,"]\n"); break;
    case LCURLY: fprintf(listing,"{\n"); break;
    case RCURLY: fprintf(listing,"}\n"); break;
    case COMMA: fprintf(listing,",\n"); break;
    case SEMI: fprintf(listing,";\n"); break;
    case PLUS: fprintf(listing,"+\n"); break;
    case MINUS: fprintf(listing,"-\n"); break;
    case TIMES: fprintf(listing,"*\n"); break;
    case OVER: fprintf(listing,"/\n"); break;
    case ENDFILE: fprintf(listing,"EOF\n"); break;
```

To print tokens.

Part II. Implementation of C-Scanner using lex(flex) by Tiny.lmodification

To use flex instead of scan.c and other files are same as before, such as globals.h, main.c util.c.

In `cminus.l`

```
"/*"            { char c, tmp=NULL;
                    do{
                            c = input();
                            if(c==EOF) break;
                            if(c=='\n') lineno++;
                            if(tmp=='*'&&c=='/' ) break;
                            tmp = c;
                    }while(c);
                }
.               { return ERROR;}
```

Example and Result Screenshot

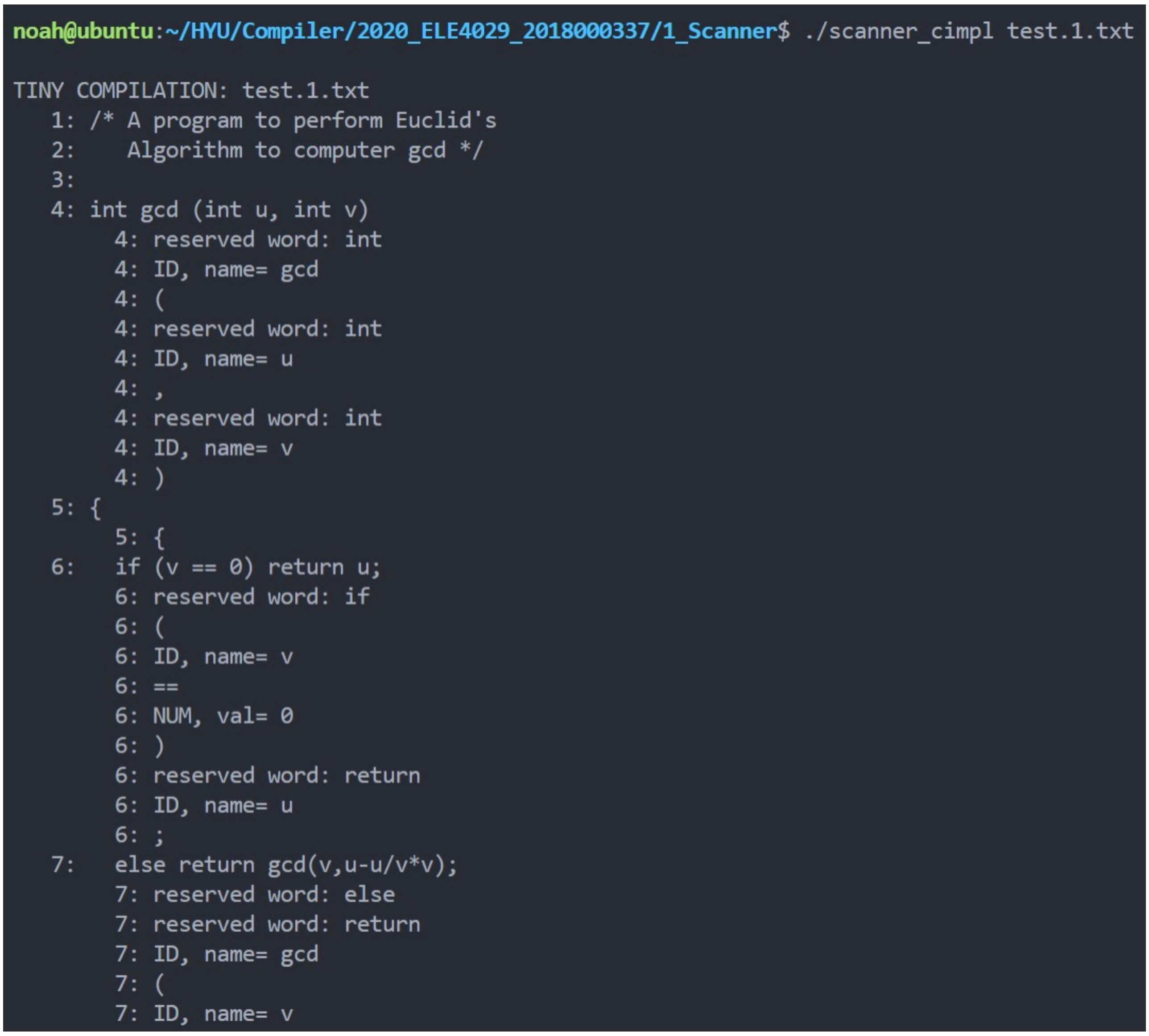
Example: **test.1.txt**

```
/* 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 */
}
```

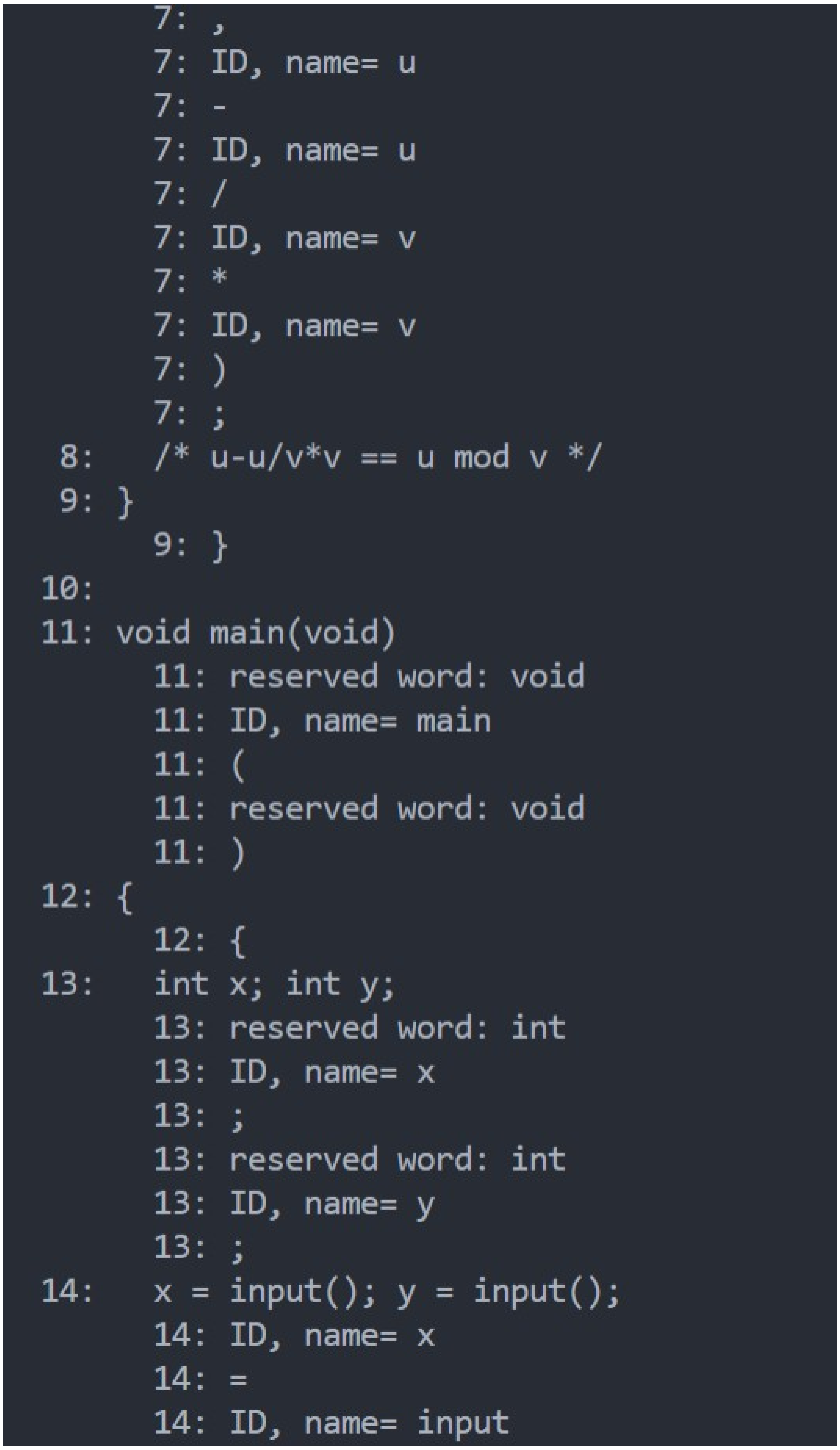
```
void main(void)
{
    int x; int y;
    x = input(); y = input();
    output(gcd(x,y));
}
```

Result Screenshot:

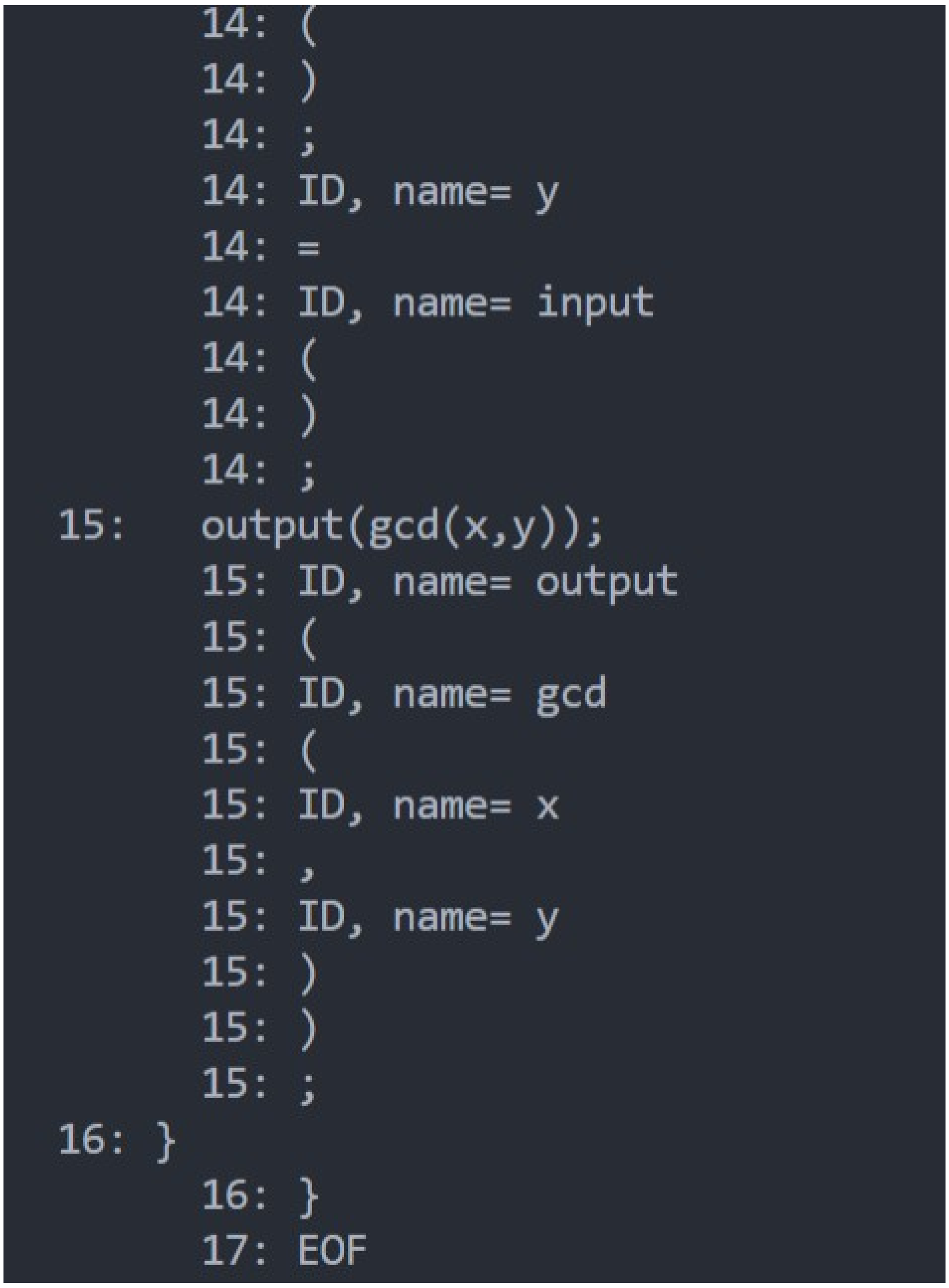
1. For `./scanner_cimpl`



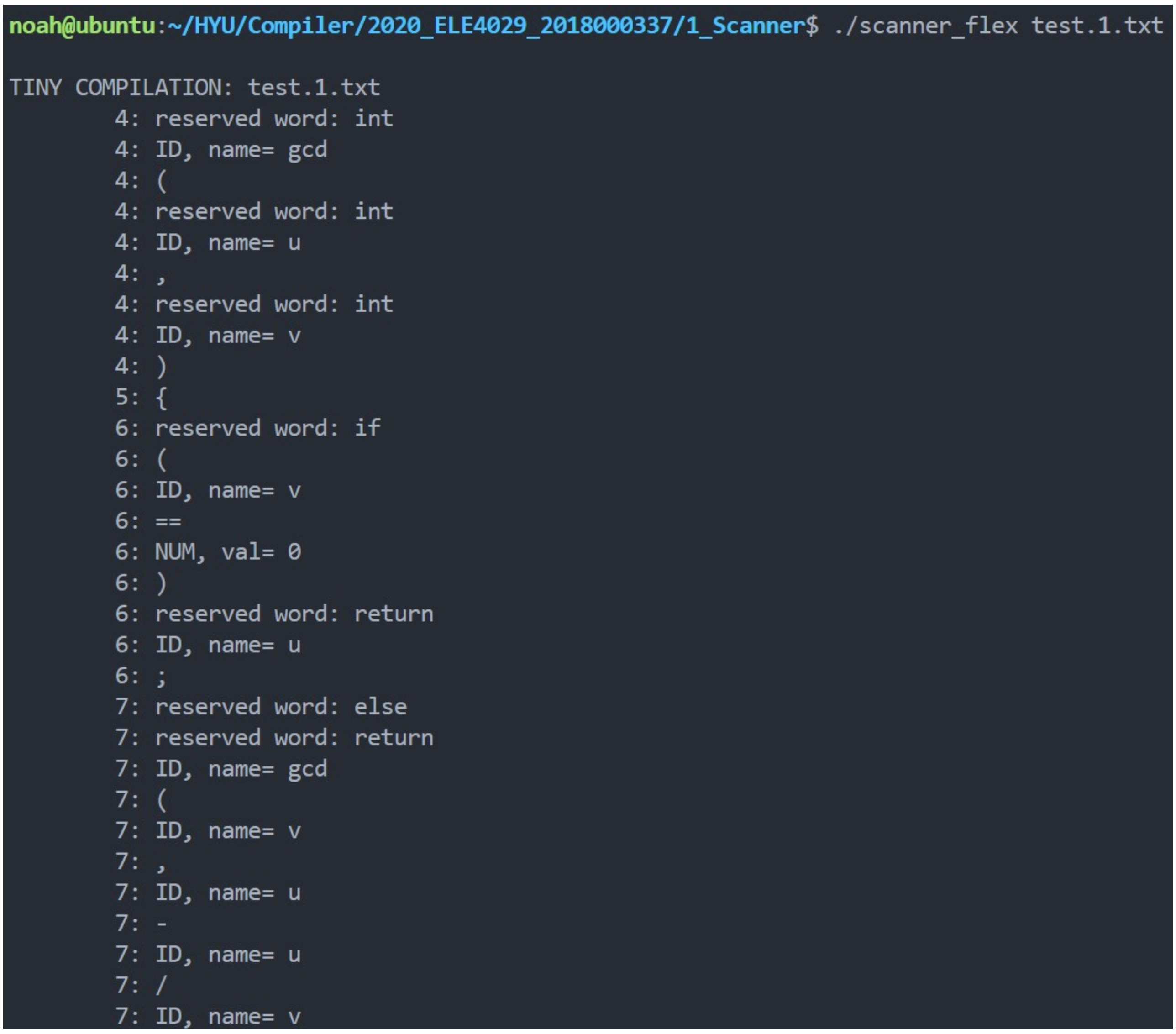
```
noah@ubuntu:~/HYU/Compiler/2020_ELE4029_2018000337/1_Scanner$ ./scanner_cimpl test.1.txt

TINY COMPILATION: test.1.txt
   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
```



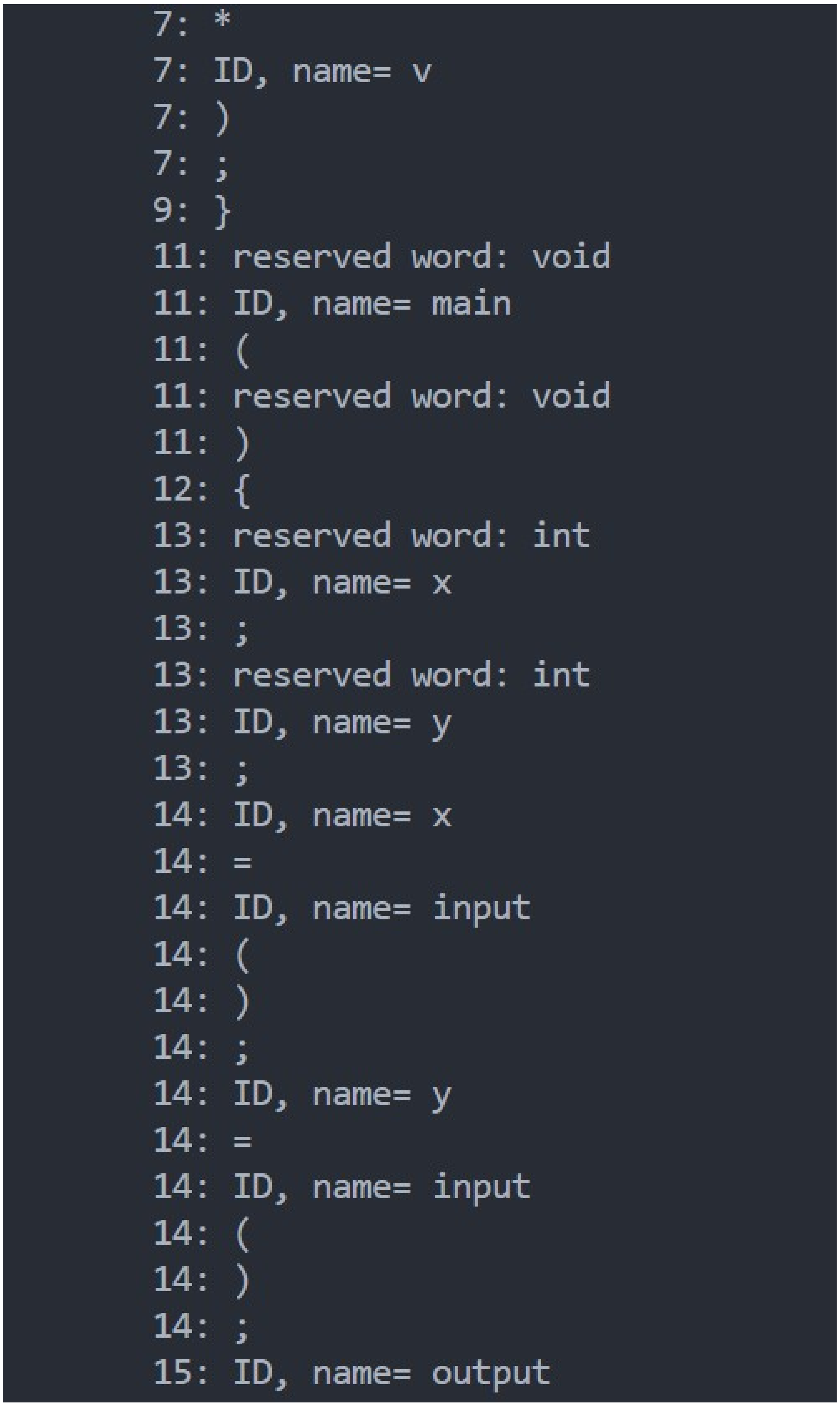
```
        7: ,
        7: ID, name= u
        7: -
        7: ID, name= u
        7: /
        7: ID, name= v
        7: *
        7: ID, name= v
        7: )
        7: ;
  8:    /* u-u/v*v == u mod v */
  9: }
        9: }
 10:
 11: void main(void)
        11: reserved word: void
        11: ID, 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
```

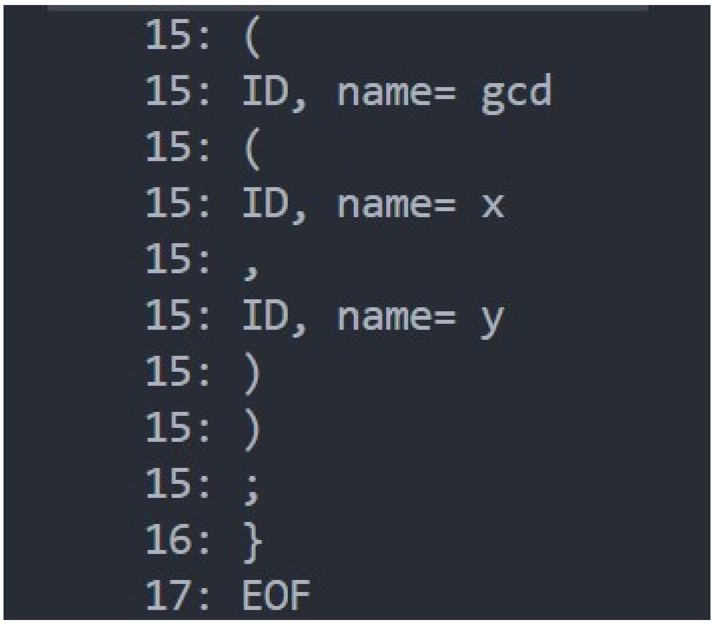
2. For `./scanner_flex`



```
noah@ubuntu:~/HYU/Compiler/2020_ELE4029_2018000337/1_Scanner$ ./scanner_flex test.1.txt

TINY 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
```



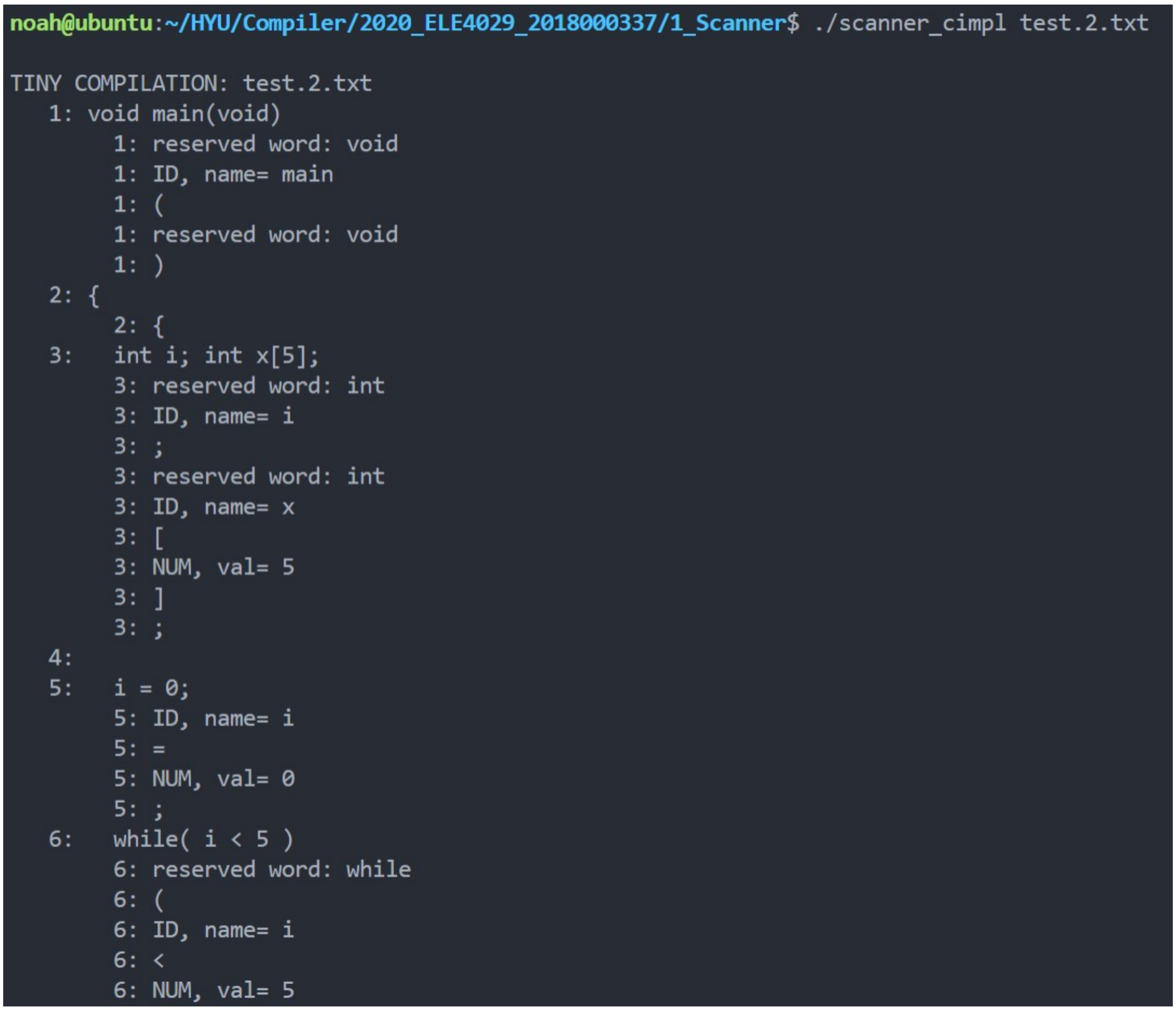
```
15: (
15: ID, name= gcd
15: (
15: ID, name= x
15: ,
15: ID, name= y
15: )
15: )
15: ;
16: }
17: EOF
```

Example: **test.2.txt**

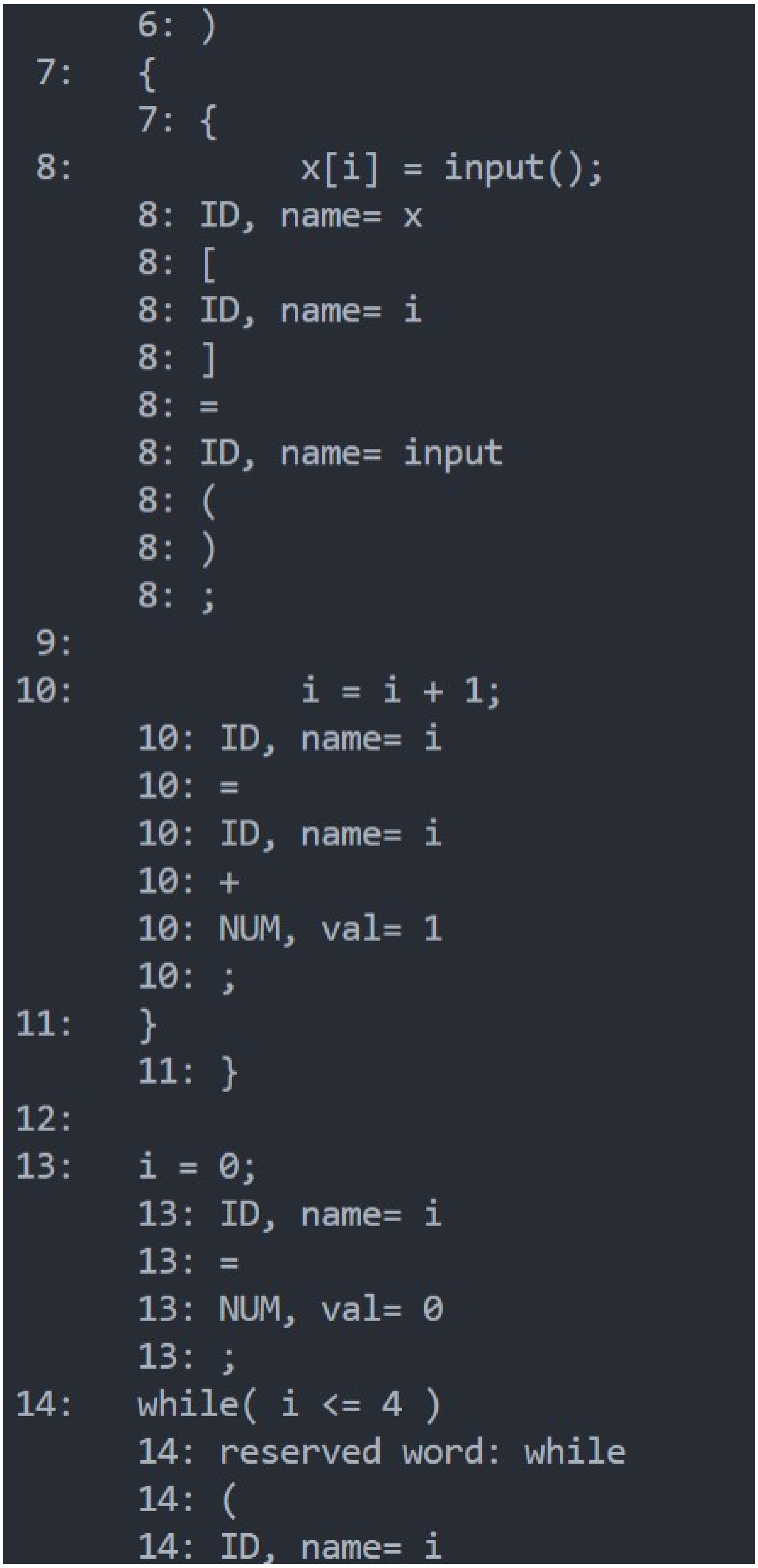
```
void main(void)
{
    int i; int x[5];

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

        i = i + 1;
    }

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

Result Screenshot:

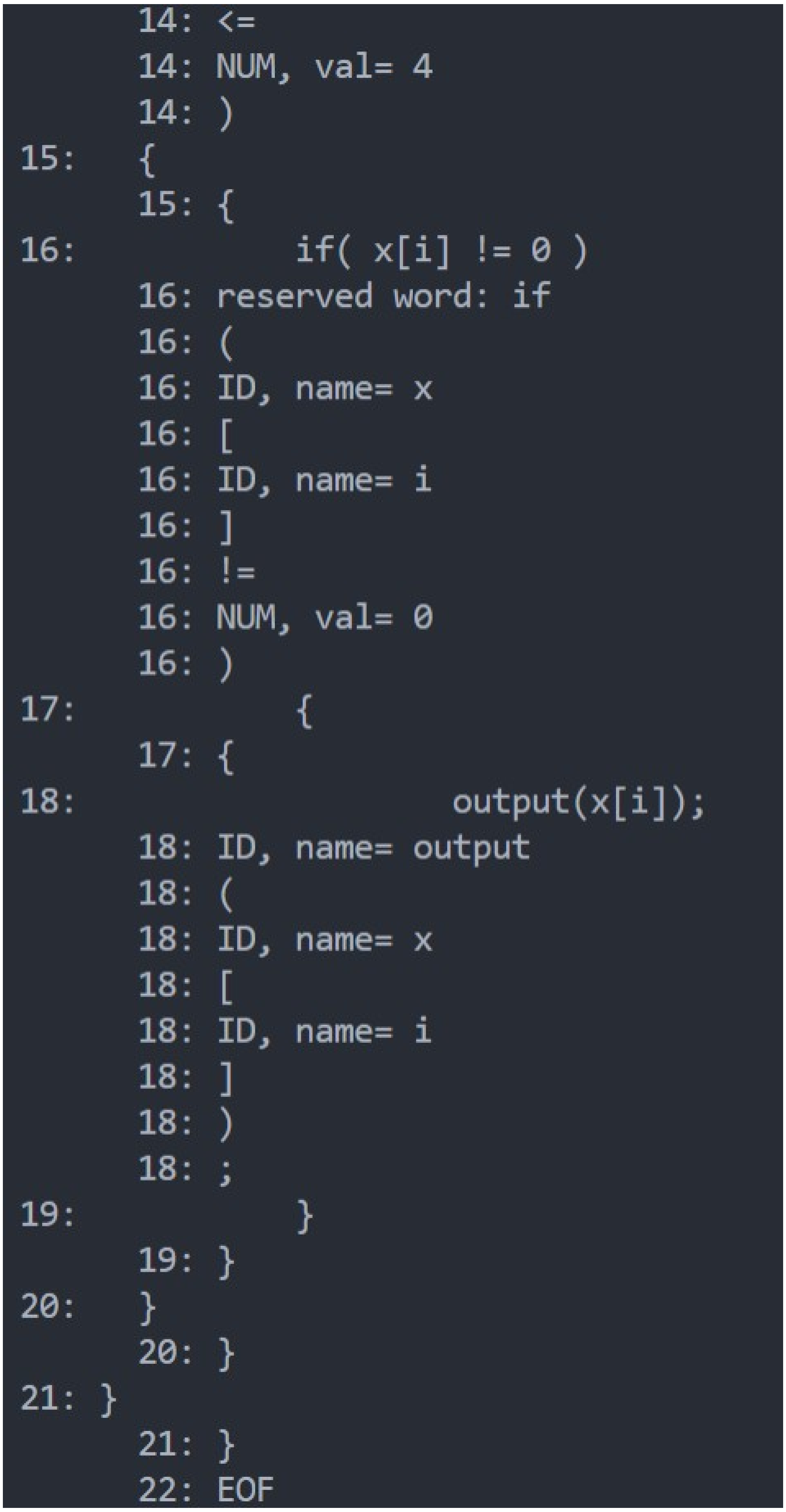
2. For `./scanner_cimpl`



```
noah@ubuntu:~/HYU/Compiler/2020_ELE4029_2018000337/1_Scanner$ ./scanner_cimpl test.2.txt

TINY COMPILATION: test.2.txt
   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
```

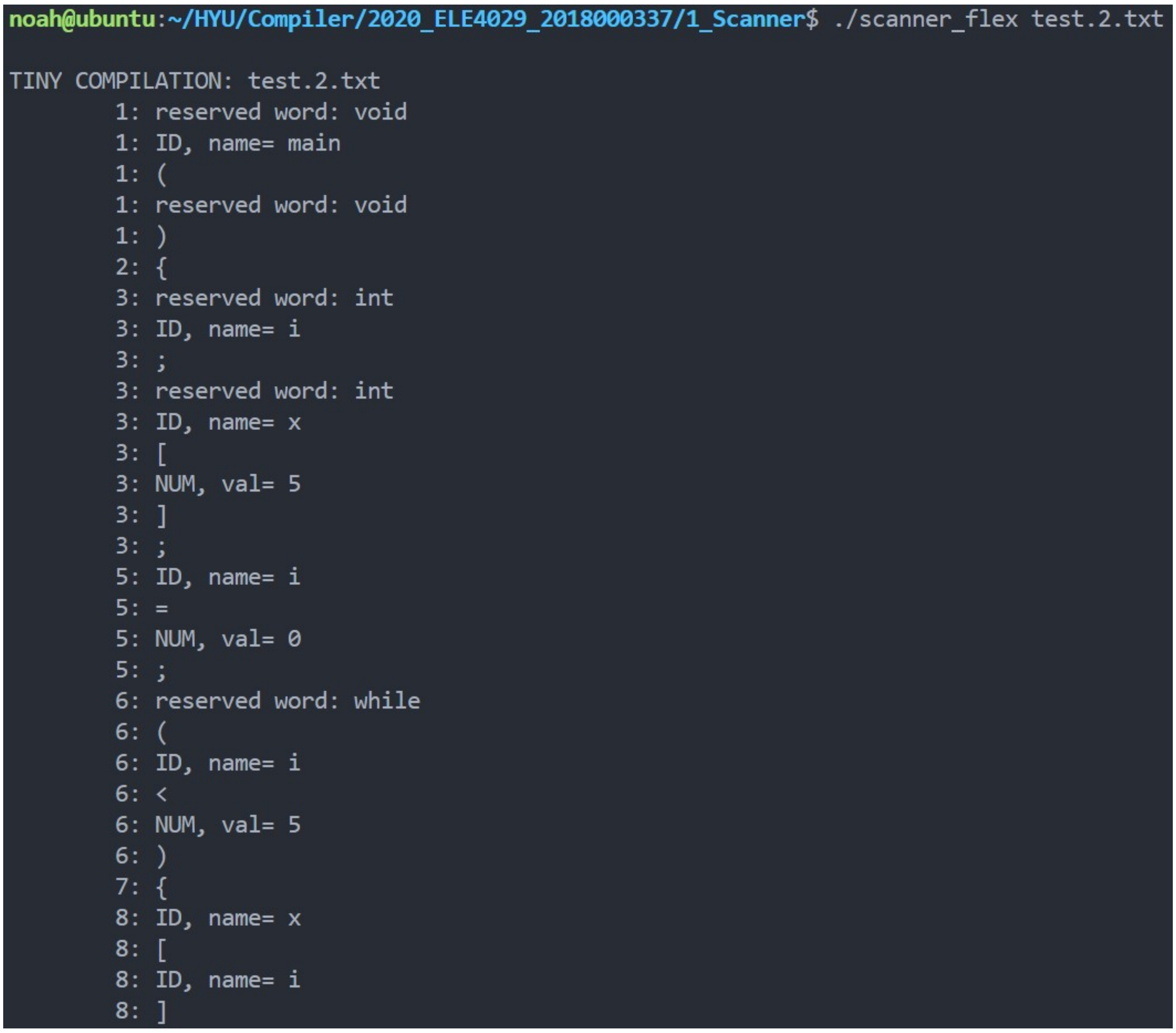


```
      6: )
 7:   {
      7: {
 8:           x[i] = input();
      8: ID, name= x
      8: [
      8: ID, name= i
      8: ]
      8: =
      8: ID, name= input
      8: (
      8: )
      8: ;
 9:
10:           i = i + 1;
      10: ID, name= i
      10: =
      10: ID, name= i
      10: +
      10: 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
```

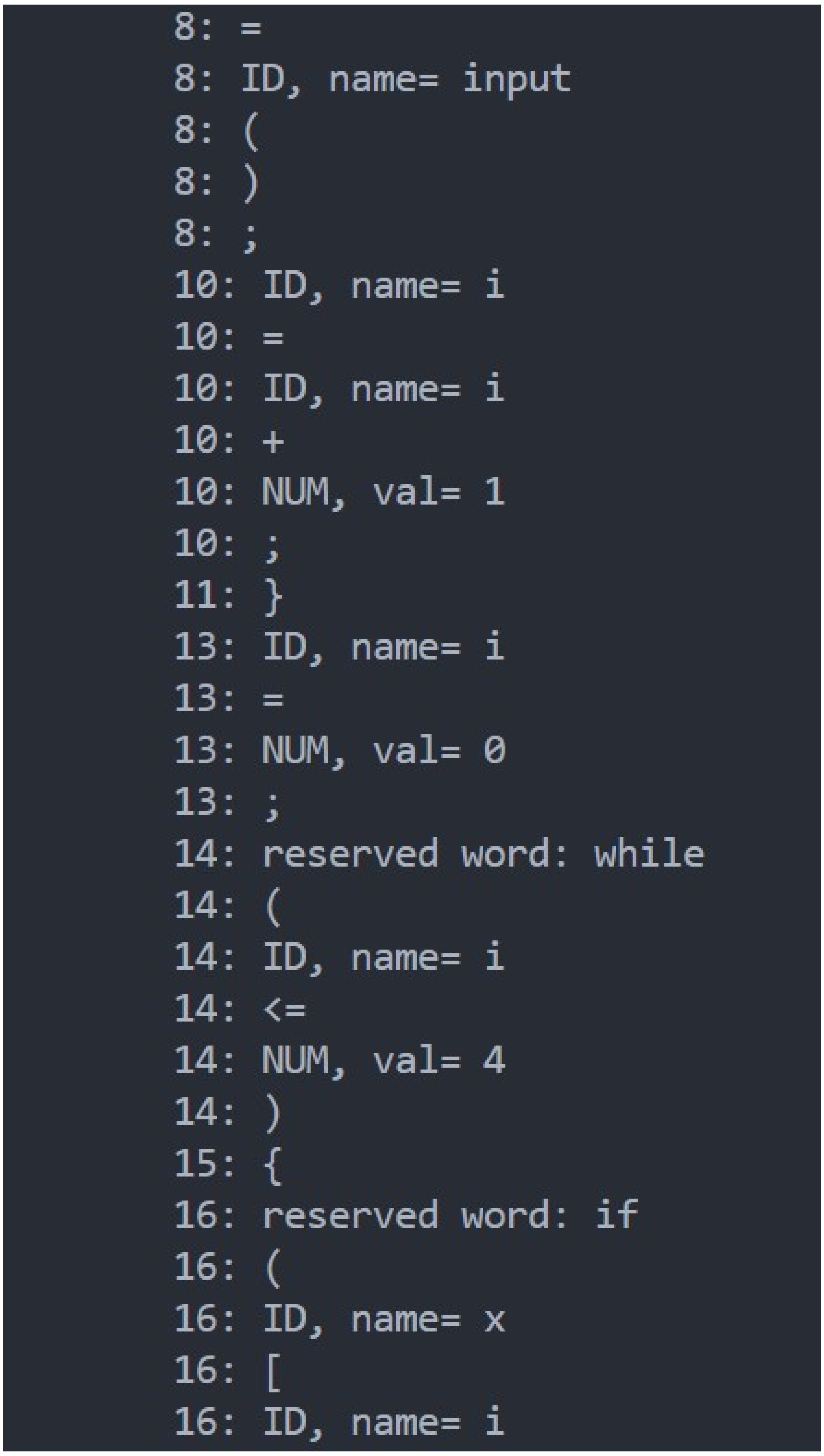


```
        14: <=
        14: NUM, val= 4
        14: )
15:     {
        15: {
16:             if( x[i] != 0 )
        16: reserved word: if
        16: (
        16: ID, name= x
        16: [
        16: ID, name= i
        16: ]
        16: !=
        16: NUM, val= 0
        16: )
17:             {
        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
```

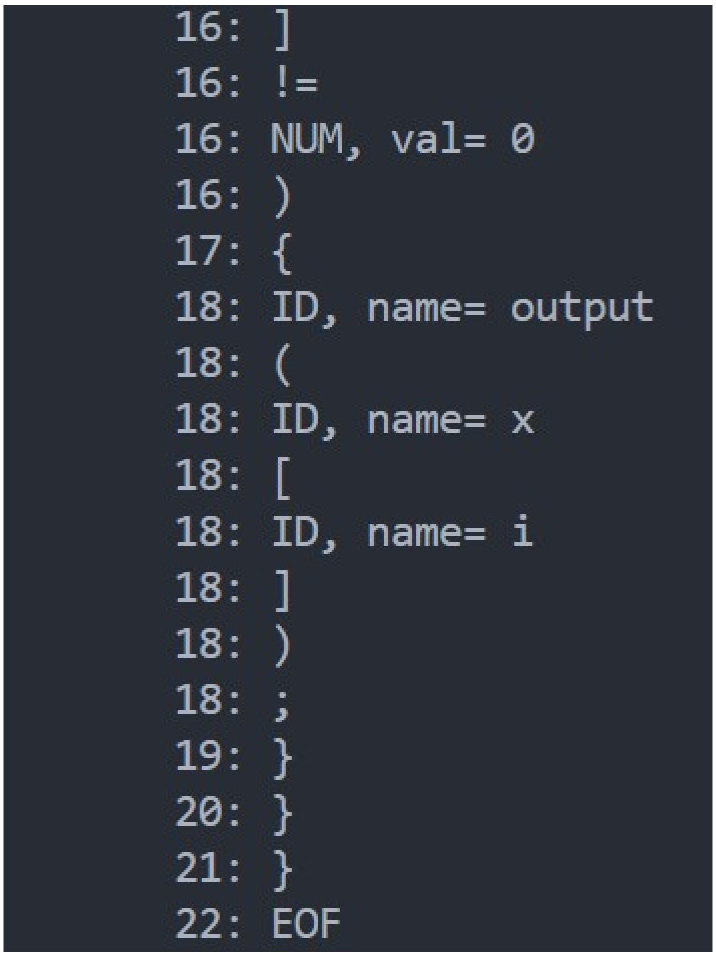
3. For `./scanner_flex`



```
noah@ubuntu:~/HYU/Compiler/2020_ELE4029_2018000337/1_Scanner$ ./scanner_flex test.2.txt

TINY 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
```