

[Compiler] 1. Parser

CSE 2019067101 황인용

Overview

C-Minus Scanner Implementation

1. Implementation Method 1 : C code (scan.c) - “cminus_cimpl”
 - Recognize token by **DFA**
2. Implementation Method 2 : Lex(Flex) (cminus.l) - “cminus_lex”
 - Specify lexical patterns by **Regular expression**

[Compilation method and environment]

VirtualBox Linux : Ubuntu 18.04

Method 1 : C code

목표는 C-Minus의 Lexical convention을 accept하는 DFA를 구현하는 것이다. 직접 구현해야 하는 부분은 multi-character symbol, ID였다.

```
typedef enum
{START, INCOMMENT, INCOMPARE, INNUM, INID, DONE}
StateType; i
```

```
case INCOMPARE :
{
    int before = tokenString[tokenStringIndex-1];
    if(before != "!"){
        state = DONE; // <, >, = 만으로도 DONE State에 도달할 수 있음
        switch(before){
            case '<':
                if(c == '='){
                    currentToken = LE;
                } else {
                    ungetNextChar();
                    save = FALSE;
                    currentToken = LT;
                }
            }
        }
```

```

        }
        break;
    case ...
    }
} else {
    state = DONE;
    if(c == '=') currentToken = NE;
    else {
        ungetNextChar();
        save = FALSE;
        currentToken = ERROR; // ! 만으로는 Symbol을 완성할 수 없으므로 ERROR
    }
}
break;
}

```

- INCOMPARE : START에서 >, <, !, = 를 입력 받아 이동한 State
- INCOMMENT : START에서 / 를 입력 받은 후 lineBuf의 다음 Character가 * 일때 이동한 State
 - 실제 코드를 보면 START에서 lineBuf[linepose]를 통해 다음 Character를 보고, INCOMMENT에서 getNextChar()를 다시 호출하여 다음 Character를 보는 것을 알 수 있는데 이는 Comment 특성 상 실제 코드에 문법적인 것을 의미하지 않기 때문에 그렇게 설계하였다.
- ID의 경우 letter(letter | digit)* 이다. 이 경우 INID에서 if statement의 조건을
 - !isalpha(c) && !isdigit(c)로 구현하였다.

Test Code

1. test.cm

```

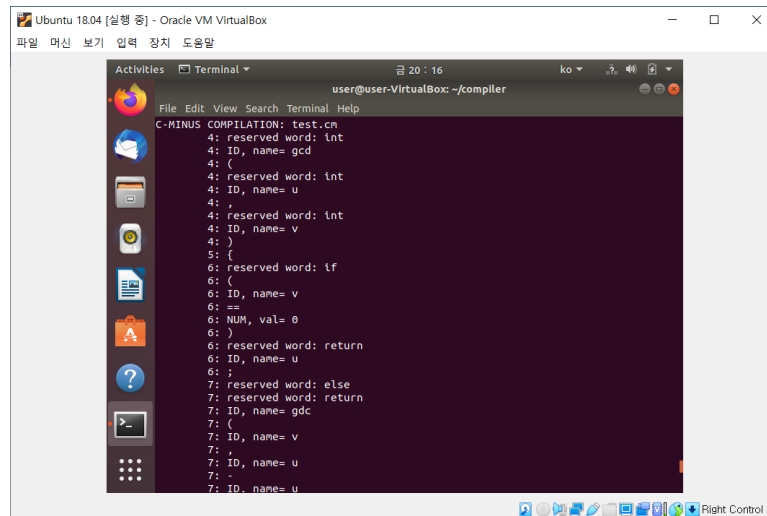
/* A program to perform Euclid's
   Algorithm to computer gcd*/

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();
}

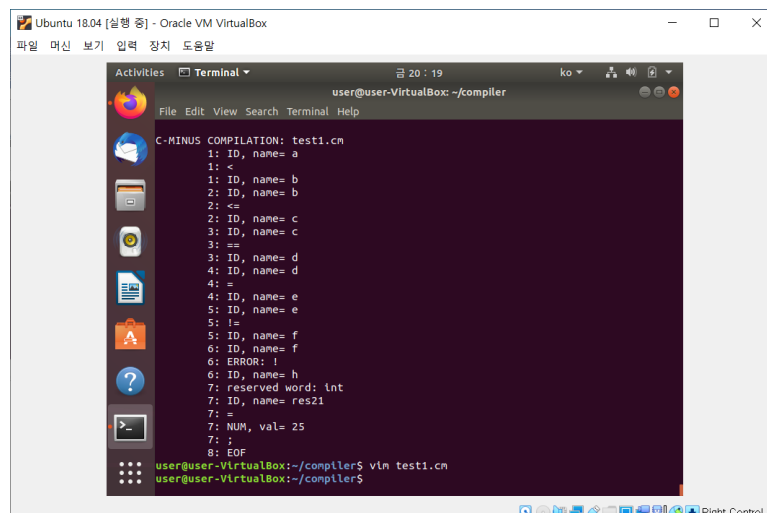
```

```
output(gcd(x, y));
}
```



2. test1.cm

```
a < b
b <= c
c == d
d = e
e != f
f ! h
int res21 = 25;
```



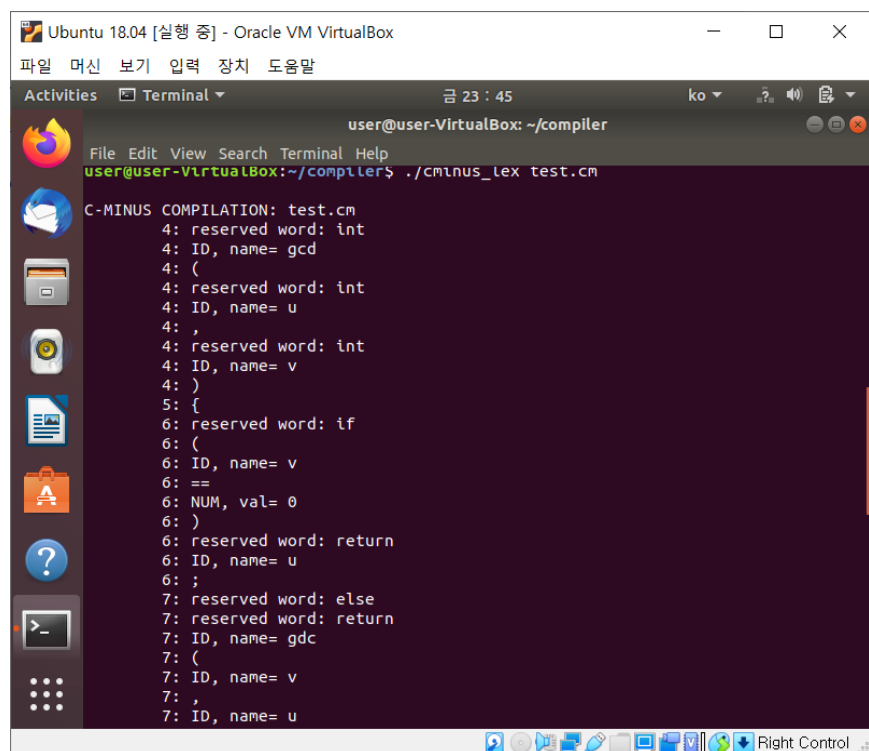
Method 2 : Lex (Flex)

Tiny.I 을 수정하여 C-Minus Convention에 대한 Lexical Analyzer를 구현하는 것이 목표이다. Rule Section에서 각 Convention에 맞는 Token을 Return하고 /* 일 경우 */가 입력될 때까지 무시하도록 하였다.

```
"/" { char c;
    do
    { c = input();
      if(c == EOF) break;
      if(c == '\n') lineno++;
    } while (c != "*" || input() != '/');
  }
```

Test Code

1. test.cm



```
user@user-VirtualBox: ~/compiler
user@user-VirtualBox:~/compiler$ ./cminus_lex test.cm
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
```

2. test1.cm

```
user@user-VirtualBox: ~/compiler
user@user-VirtualBox:~/compiler$ ./cminus_lex test1.cm
C-MINUS COMPILATION: test1.cm
1: ID, name= a
1: <
1: ID, name= b
2: ID, name= b
2: <=
2: ID, name= c
3: ID, name= c
3: ==
3: ID, name= d
4: ID, name= d
4: =
4: ID, name= e
5: ID, name= e
5: !=
5: ID, name= f
6: ID, name= f
6: ERROR: !
6: ID, name= h
7: reserved word: int
7: ID, name= res21
7: =
7: NUM, val= 25
7: ;
8: EOF
user@user-VirtualBox:~/compiler$
```