# Compiler assignment1

학과 : 컴퓨터 소프트웨어학부

학번 : 2017029516

이름 : 김태환

- Compilation environment and method

Oracle VM VirtualBox, ubuntu 18.04.6 에서 작업, gcc를 이용해서 컴파일 하였습니다.

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.6 LTS
Release:        18.04
Codename:       bionic
```

- Brief explanations about how to implement and how it operates

과제 명세에 나온 대로 main.c, globals.h ,util.c의 수정사항들을 차례로 수정하였습니다.

Tiny 에서 쓰였던 명령어들과 symbol들을 C-minus에 맞게 수정하였습니다.

**Method 1**

Method 1에서는 tiny에 맞춰져있는 scan.c 파일을 c-minus spec에 맞게 수정하였습니다.

가장 우선적으로 tiny 코드를 읽어보며 flow를 파악하였고 character가 2개인 심볼(ex.==,!=등)에 대한 처리를 기존의 tiny코드의 처리방식에서 힌트를 얻어 구현하고자 하였습니다.

먼저 start state입니다. Start state에서는 두개의 문자로 이뤄진 심볼들에 대한 처리를 위해 IN~ state로 state를 전환해줍니다. 이때 다른 문자보다 주석에 대한 처리의 flow를 짜는 것 어려웠는데, 우선적으로 INOVER로 넘긴후, INOVER에서 comment로 처리할지를 정하는 방식으로 구현하였습니다.

```
switch (state)
{ case START:
    if (isdigit(c))
      state = INNUM;
    else if (isalpha(c))
      state = INID;
    else if (c == '=')
      state = INASSIGN;
    else if ((c == ' ') || (c == '\t') || (c == '\n'))
      save = FALSE;
    else if (c == '/')
    { save = FALSE;
      state = INOVER;
    }
    else if (c == '!')
        state = INNE;
    else if (c == '<')
        state = INLT;
    else if (c == '>')
        state = INGT;
    else
```

이후, C-minus에 맞게 심볼들을 추가해주었습니다..

INOVER로 넘어오면, 다음 문자가 *일경우 INCOMMENT로 state를 전환시켜주고 아닐경우 OVER로 확정짓습니다. 이때 새로 읽어들인 문자가 있으므로, ungetNextChar()를 호출하고 save하지 않습니다.

INCOMMENT에서는 주석의 끝을 체크할 수 있어야합니다. 따라서 읽어들인 char가 *인지 체크후, *이면 state를 INCOMMENT_으로 넘겨줍니다. INCOMMENT_에서는 읽어들인 문자가 / 일경우 주석이 끝나므로 state를 START로 바꿔줍니다. 이때 **/등의 형태로 문자가 입력될 수 있으므로 만약 읽어들인 문자가 *이면, state를 유지합니다. 그외의 경우엔 다시 INCOMMENT로 돌아갑니다.

INASSIGN,INNE,INLT,INGT의 구현 flow는 거의 동일하기에 INASSIGN만을 설명합니다.

INASSIGN에서 문자를 읽고 =일경우 EQ로 확정, 아닐경우 ASSIGN으로 확정하고 ungetNextChar() 합니다.

그리고 INID에서는 문자+숫자 형태의 입력이 들어올 수 있으므로

```
if (!(isalpha(c)|| isdigit(c)))
```

숫자나 문자를 모두 인식할 수 있도록 조건을 수정해주었습니다.

## Method 2

Method2는 cminus.l에 c-minus의 스펙을 입력해주는 것으로, 가장 우선 tiny 의 스펙을 읽어보며 구조를 파악하였습니다.

이후 C-minus 스펙에 맞게 예약어와 심볼들을 작성해주었습니다. 이때, 두가지 부분에서 trouble shooting 이 존재하였습니다.

첫번째로는 주석의 처리였습니다. 앞선 method1에서는 state를 바꿔가며 주석을 처리할 수 있었지만, method2에서는 char 1개만을 읽으므로 */를 인식하기 위해 char 1개만을 갖

고 처리할 수 없었습니다. 따라서 char 형 변수를 하나 더 선언하여 old라는 변수에 history를 저장하여 */ 를 인식할 수 있도록 처리하였습니다.

```
"/*"            { char c;
                  char old ='a';

                 do
                 { c = input();

                   if (old == '*' && c == '/'){
                   break;
                   }
                   if (c == EOF) break;
                   if (c == '\n') lineno++;
                   old = c;
                 } while (1);
                }
```

두번째는 definition section 의 identifier였습니다. 만약 input으로 dfa2와 같은 문자+숫자 형태가 들어온다면 이를 예약어가 아닌 하나의 ID로 인식해야합니다. 따라서 기존의 tiny 의 identifier 에서 identifier 부분을 수정하여

```
16 digit       [0-9]
17 number      {digit}+
18 letter      [a-zA-Z]
19 identifier  [a-z][a-zA-Z0-9]*
20 newline     \n
21 whitespace  [ \t]+
```

처리할 수 있었습니다.

- Examples and corresponding result screenshots

주어진 test case 1과 2에 대한 결과 입니다.

Test 1.

Cimus_cimpl

```
taehwanklm@taehwanklm-VirtualBox:~/cp22$ ./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
```

Cimus_lex

```
taehwankim@taehwankim-VirtualBox:~/cp22$ ./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
```

Test 2

Cimus_cimpl

```
taehwankim@taehwankim-VirtualBox:~/cp22$ ./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
```

Cminus_lex

```
taehwankim@taehwankim-VirtualBox:~/cp22$ ./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
```

Cminus_lex와 cminus_cimpl의 결과가 같고 주어진 test_result와 같음을 확인할 수 있었습니다.

마지막으로 주석 연속처리,예약어 + 숫자의 처리가 잘이뤄지는지 확인하기 위해 test3를 준비하였습니다.

```
1 /*/*//*/*//*/*/
2 <= => == !=
3 return0
4 int1
5 if2
6
```

```
taehwankim@taehwankim-VirtualBox:~/cp22$ ./cminus_lex test3.txt

C-MINUS COMPILATION: test3.txt
        2: <=
        2: =
        2: >
        2: ==
        2: !=
        3: ID, name= return0
        4: ID, name= int1
        5: ID, name= if2
        7: EOF
taehwankim@taehwankim-VirtualBox:~/cp22$ ./cminus_cimpl test3.txt

C-MINUS COMPILATION: test3.txt
        2: <=
        2: =
        2: >
        2: ==
        2: !=
        3: ID, name= return0
        4: ID, name= int1
        5: ID, name= if2
        7: EOF
```

성공적으로 출력함을 확인할 수 있었습니다.