**C-Minus Project1 Report**

컴퓨터소프트웨어학부 박제우

1. 컴파일 환경/컴파일 방법

\* 컴파일환경: Ubuntu 16.04.7 LTS (Oracle VM VirtualBox에서 구동)

\* http://www.cs.sjsu.edu/faculty/louden/cmptext/ 에서의 Tiny compiler + 프로젝트 설명에서 제공한 Makefile을 기초로 작성함

\* make all -> ./cminus\_cimpl (대상파일명) 또는 ./cminus\_lex (대상파일명)으로 실행

2. Scanner 구현/작동방식 설명

(1) 구현하고자 하는 C-Minus에 맞도록 파일 수정

- globals.h

#define MAXRESERVED 6

typedef enum

/\* book-keeping tokens \*/

{ENDFILE,ERROR,

/\* reserved words \*/

IF,ELSE,WHILE,RETURN,INT,VOID,

/\* 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;

- main.c

#define NO\_PARSE TRUE

int TraceScan = TRUE;

(2) C코드를 통한 Scanner 구현(scan.c)

- 기존의 tiny compiler와는 다르게 우리는 <=, >=, ==, != 등의 연산자도 추가로 구현해야 하고,일부 연산자는 사용하지 않기에 그에 따라 가능한 DFA 상태들도 바꿔주어야 한다.

typedef enum

{ START,INASSIGN,INCOMMENT,INNUM,INID,DONE,INEQ,INLT,INGT,INNE,INOVER,INCOMMENT\_ }

StateType;

- 또한 C-Minus에서는 int, void, if, else, while, return의 6가지의 Reserved word만을 사용하기에 그에 따라 아래와 같이 reserved word lookup table의 목록도 바꿔주어야 한다.

static struct

{ char\* str;

TokenType tok;

} reservedWords[MAXRESERVED]

= {{"if",IF},{"else",ELSE},{"while",WHILE}, {"return",RETURN},{"int",INT},

{"void",VOID}};

- 주요 변경사항은 getToken함수로, 구현한 내용을 요약하면 다음과 같다.

① start state에서 ‘<’를 만났을 때 <인지 <=인지 구분 위해 INLT state로 이동, >, =, ! 도 마찬가지로 구현한다.

else if (c == '<')

state = INLT; // <인지 <=인지 구분 필요

② INLT상태는 <인지 <=인지 확정하기 위한 state로, 이 상태에서 =를 만나면 LE로 currentToken이 결정되고, 아니라면 그냥 <이었던 것이므로 읽은 문자를 한 칸 취소하고 LT로 currentToken이 결정된다. 마찬가지로 INGT, INEQ, INNE상태에 대해 처리한다 (INNE의 경우 !입력된 상태에서 =가 입력되면 NE, 아닐경우 잘못입력된 !로 판정해 ERROR가 된다)

case INLT: // <인지 <=인지 구분

state = DONE;

if (c == '=')

currentToken = LE;

else

{

ungetNextChar();

currentToken = LT;

}

break;

③ /과 /\* \*/를 구분하는 경우는 조금 다른데, 일단 /를 만나면 INOVER 상태로 갔다가 \*를 만나면 INCOMMENT상태로 이동한 후 그 뒤 입력을 무시하도록 save = FALSE로 이동한다. 이 상태에서 \*와 /를 연속으로 만나면 INCOMMENT\_, START 상태로 이동해 다시 START state로 돌아가 다음 입력을 계속해서 받게 된다.

④ 또한 기존에 구현된 case INID: 부분도 수정해야 하는데, 우리가 목표로 하는 C-Minus 컴파일러에서는 identifier를 문자 뿐만 아니라 문자 뒤에 숫자가 붙는 경우도 인정하기에 일단 문자를 만나 INID 상태로 이동하였으면 다음 입력으로 숫자가 나와도 문제가 없도록 한다.

case INID:

if (!isalpha(c) && !isdigit(c)) // identifier에서 letter+숫자도 허용위해 수정

- 위와 같이 scan.c를 수정한 후 util.c에서 우리가 목표로 했던 키워드와 토큰들에 맞게 적절히 수정해주면 완료된다.

(3) Lex를 이용한 Scanner 구현(cminus.l)

- tiny compiler파일에 있던 lex/tiny.l 파일을 cminus.l로 파일명을 바꾸고 우리가 목표로 하는 C-Minus에 맞게 수정해주면 된다. 이 때 주의해야할 사항은 총 두 가지로,

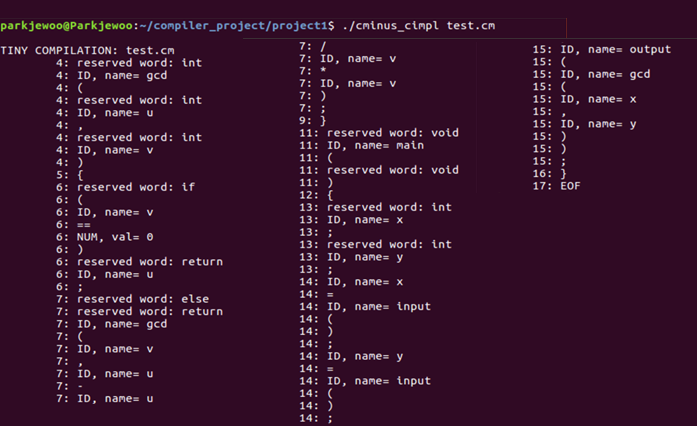
① 기존 파일에선 identifier가 {letter}+로 지정되었으나 C-Minus에서는 identifier로 letter이후 숫자가 섞이는 경우도 허용하기에 {letter}({letter}|{digit})\*로 수정되어야 한다.

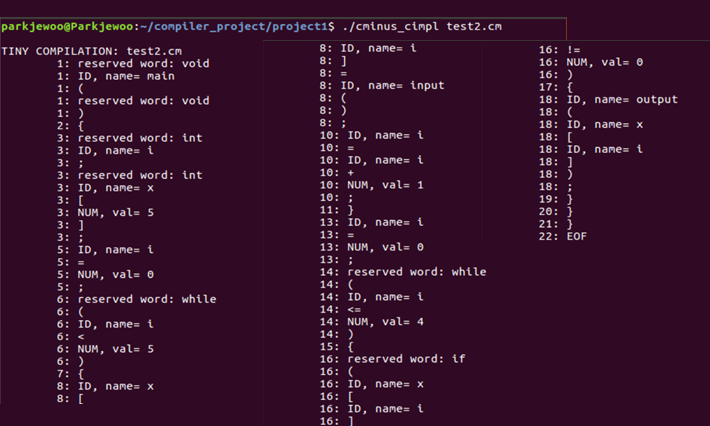
② /\* \*/ 주석에 대한 처리가 필요로 한데, 이 경우 기존 tiny compiler에서 {과 }이 주석역할 했던 것에서 /\* 와 \*/이 주석으로 처리되도록 바꿔주면 된다. 우선 /\*를 만났을 때 계속해서 다음 문자를 받되 \*과 /이 연속해서 나오면 반복문을 탈출하는 식으로 작성하였다.

3. 실행 결과

- 제공된 test.1.txt와 test.2.txt파일명을 test.cm, test2.cm로 변경하고 (반드시 변경할 필요는 없다) make all -> ./cminus\_cimpl test.cm 또는 ./cminus\_lex test.cm과 같이 실행해보니 결과가 잘 나온 것을 확인할 수 있다.

(1) C를 이용한 Scanner





(2) Lex를 이용한 Scanner (실행결과의 뒷부분은 생략함)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명