Scanner_report

2018_2 Compiler

2014004693

송보석

1. 컴파일 방법

$make clean

$make all

2. 컴파일 환경

Ubuntu 16.04

gcc version 5.4.0

flex 2.6.0

3. 구현방법 _ 직접 수정

global.h

MAXRESERVED 갯수를 12로 수정한뒤, reserved words와 special symbols에 ELSE, WHILE, RETURN, INT, VOID, NE, LT, LE, GT, GE, LBRACE, RBRACE, LCURLY, RCULRY, COMMA 추가한다.

util.c

global.h에 추가한 토큰들을 출력하기 위해서 해당되는 요소들 추가한다.

scan.c

‘{‘, ‘}’, ‘[’, ‘]’, ‘,’ 가 들어올 경우 currentToken에 해당하는 symbol을 입력한다.

```
else if (c == '=')
  state = INEQ;
else if (c == '!')
  state = INNE;
else if (c == '<')
  state = INLT;
else if (c == '>')
  state = INGT;
```

state가 START일 때 '=' 가 들어올 경우 ASSIGN과 EQ가 가능하므로 처음 받을 때 현재 state를 INEQ로 만든다. '!' 가 들어올 경우 '!=' 인지 확인하기 위해 INNE로 현재 state를 바꾼다. '<', '>' 가 들어올 경우 각각 '<=', '>=' 가 가능하므로 현재 state를 INLT와 INGT로 바꾼다.

```
case INEQ:
  state = DONE;
  if (c == '=')
    currentToken = EQ;
  else
  {
    ungetNextChar();
    currentToken = ASSIGN;
  }
  break;
case INLT:
  state = DONE;
  if (c == '=')
    currentToken = LE;
  else
  {
    ungetNextChar();
    currentToken = LT;
  }
  break;
case INGT:
  state = DONE;
  if (c == '=')
    currentToken = GE;
  else
  { /* backup in the input */
    ungetNextChar();
    currentToken = GT;
  }
  break;
case INNE:
  state = DONE;
  if (c == '=')
    currentToken = NE;
  else
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    currentToken = ERROR;
  }
  break;
```

현재 state가 INEQ인 경우 받아온 토큰이 '=' 라면 결론적으로 '==' 가 들어온 경우이므로 currentToken은 EQ가 된다. 만약 그렇지 않으면 '=' 하나만 들어온 것이므로 ungetNextChar() 로 postion을 뒤로 하나 이동시키고 currentToken을 ASSIGN으로 지정한다.

현재 state가 INNE일 때 받아온 토큰이 '=' 라면 결론적으로 '!=' 가 들어온 경우이므로 currentToken은 NE가 된다. 만약 그렇지 않으면 '!' 는 정상적인 토큰이 아니므로 ERROR가된다.

현재 state가 INLT, INGT인 경우 '=' 가 들어오면 '<=', '>=' 가 되므로 currentToken은 LE, GE가 된다. 만약 그렇지 않으면 '<', '>' 이므로 역시 ungetNextChar()로 position을 뒤로 이동시킨뒤 currentToken이 LT, GT가 된다.

현재 state가 START일 때 '/' 가 들어오는 경우는 뒤에 '*' 가 들어와서 COMMENT가 되는 경우와 나눗셈 OVER이 되는 경우 2가지가 발생한다. 따라서 save는 FALSE로 지정해 일단 저장하지 않는 상태로 만들고 현재 state는 INOVER로 지정한다.

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

state가 INOVER일때 '*' 가 들어올 경우 '/*' 인 경우이기 때문에 state는 INCOMMENT가 된다. 그렇지 않다면 '/' 인 경우이기 때문에 일단 ungetNextChar() 를 두번해서 position을 / 까지 옮겨준 다음 그 다음에도 '/' 가 들어오면 이 때 currentToken 은 OVER 가 된다.

State가 IMCOMMENT인 경우는 save를 FALSE로 한채 주석을 계속 받다가 '*' 가 들어오면 주석이 끝날 가능성이 있는 경우이기 때문에 state를 IMCOMMENT_로 지정해주고 만약 그 다음에 '/' 가 들어오면 주석이 끝났으므로 state를 START로 지정해준다. 만약 그렇지 않으면 아직 주석안이기 때문에 state를 다시 IMCOMMENT로 바꿔준다.

4. 구현방법 _ flex 이용

cminus.l

if, else, while, return ,int, void, =, ==, <=, >=, {, }, [,], , , 에 해당하는 return을 reserved word에 맞게 지정해주었다.

```
"/*"          { char c;
                  char d = '\n';
                do
                { c = input();
                  if (c == EOF) break;
                  if (c == '\n') lineno++;
                  if (c == '/' && d == '*') break;
                  d = c;
                } while (1);
              }
```

주석의 경우 “ /* ” 를 받게 되면 while문 안에서 char형 변수 c, d를 선언 한뒤 c에 input() 함수를 이용해서 문자를 한글자씩 받아오면서 EOF 이면 break, 개행문자가 들어오면 lineno를 하나씩 올려준다. 주석이 끝나는지 판단은 d에 계속 c 를 대입해 주면서 만약 c 가 ‘/‘ 이고 d가 ‘*’ 이면 입력이 “*/“ 로 들어온 상태이기 때문에 주석이 끝났다고 인지하고 반복문을 빠져나간다. flex의 경우 주석은 지워버리기 때문에 return 값은 없다.

makefile

```
OBJS_FLEX  = main.o util.o lex.yy.o parse.o symtab.o analyze.o code.o cgen.o
cminus_flex: $(OBJS_FLEX)
	$(CC) $(CLAGS) main.o util.o lex.yy.o -o cminus_flex -lfl
lex.yy.o: cminus.l scan.h util.h globals.h
	flex cminus.l
	$(CC) $(CFLAGS) -c lex.yy.c -lfl
clean:
	-rm lex.yy.c
	-rm cminus_flex
	-rm $(OBJS_FLEX)
all: tiny tm cminus_flex
```

를 추가해줬다.

cminus.l 를 flex를 이용해 scan.c를 대신할 lex.yy.c 라는 lex c코드를 만들어 준뒤 이를 이용해 lex.yy.o라는 실행파일을 만들어 프로그램을 돌리게 된다.

5. 테스트샘플 및 결과 캡쳐

테스트샘플

```
/* 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));
}
```

결과 _ 직접 수정 / flex 사용

./tiny test.cm

```
TINY COMPILATION: test.cm
   1: /* A program to perform Euclid's
   2: Algorithm to computer gcd */
   3:
   4: int gcd (int u, int v)
        4: ID, name= *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
```

./cminus_flex test.cm

```
TINY 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
        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
```