Compiler project

#1. Scanner

2015004493
김형순

1. 프로젝트 개요
   ① C minus 라는 프로그래밍의 컴파일러를 만들기 위한 첫번째 단계인 scanner 를 만든다. 총 2개의 scanner 를 만들게 된다.
   ② C language 를 사용하여 state machine 을 구현 한 뒤 입력 되는 text stream 에서 미리 정의 되어 있는 토큰을 검출한다.
   ③ Flex 를 사용하여 토큰 검출 규칙을 입력시켜 준 뒤 flex 가 생성해주는 lexer 를 사용한다.

2. 컴파일 환경
   ① GCC 4.8.2 버전을 사용하여 컴파일 하였다.
   ② Ubuntu 14.04 LTS 환경에서 컴파일 및 실행 하였다..

3. 실행 방법
   ① Scanner 실행 파일을 만들기 위해 make all 명령을 입력하여 Makefile 을 이용해 컴파일을 한다.
   ② 성공적으로 컴파일이 되었다면 cminus 와 cminus_flex 라는 2개의 실행파일이 생성된다.
   ③ ./cminus [input file] 혹은 ./cminus_flex [input file] 을 이용해 input file 을 scan 할 수 있다.

4. 구현 방법

1) 공통 사항
   ① globals.h 에 reserved word 와 특수 기호들의 이름을 정의 해준다.

```
typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF,ELSE,WHILE,RETURN,INT,VOID, /*discarded*/ 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;
```

   ② main.c 에서 scan 까지만 할 수 있도록 parse 를 하지 않고, scan 의 결과를 출력 하도록

한다.

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE TRUE
int EchoSource = TRUE;
int TraceScan = TRUE;
```

2) C language 를 이용한 cminus scanner

① scan.c 에서 state machine 을 구현하여 state machine 을 통과 한 결과를 이용해 token 을 판독한다.

② 2개 이상으로 이루어진 특수 기호 토큰 (==, <=, >=, !=) 과 주석 (/* */) 을 판독하기 위해서 state 들을 정의 한다.

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

START : 시작 state DONE: 완료 state

INEQ : '=' 을 하나 검출 했을 때

INCOMMENT: "/*" 을 이미 검출해서 주석을 읽고 있을 때

INNUM: 숫자를 하나 이상 검출 했을 때

INID: 알파벳을 하나 이상 검출 했을 때

INLT: '<' 를 검출 했을 때

INGT: '>' 를 검출 했을 때

INNE: '!' 를 검출 했을 때

INOVER: '/' 를 검출 했을 때

INCOMMENT_ : 주석을 읽고 있을 던 중 '*' 를 검출 했을 때

③ reserved words 들의 형태를 입력해준다.

```
{{"if",IF},{"else",ELSE},{"while",WHILE},{"return", RETURN},{"int",INT},{"void",VOID}, /*discarded*/ {"then",THEN},{"end",END},
 {"repeat",REPEAT},{"until",UNTIL},{"read",READ},
 {"write",WRITE}};
```

④ 시작 state 에서 특수 기호를 검출 했을 때 각각을 다른 토큰으로 인식 하도록 한다.

```
case EOF:
  save = FALSE;
  currentToken = ENDFILE;
  break;
* case '=':
  currentToken = EQ;
  break;
case '<':
  currentToken = LT;
  break;*/
case '+':
  currentToken = PLUS;
  break;
```

```
case '-':
  currentToken = MINUS;
  break;
case '*':
  currentToken = TIMES;
  break;
case '(':
  currentToken = LPAREN;
  break;
case ')':
  currentToken = RPAREN;
  break;
```

```
case '{':
   currentToken = LCURLY;
   break;
case '}':
   currentToken = RCURLY;
   break;
case '[':
   currentToken = LBRACE;
   break;
case ']':
   currentToken = RBRACE;
   break;
```

```
case ';':
  currentToken = SEMI;
  break;
case ',':
  currentToken = COMMA;
  break;
default:
  currentToken = ERROR;
  break;
```

⑤ INEQ, INLT, INGT, INNE 의 state 에서는 새로 검출하는 문자에 따라 새 토큰을 기존 문자와 하나로 합쳐 토큰으로 인식 할지 각각을 다른 토큰으로 인식 할 지 결정한다.

```
case INEQ:
   state = DONE;
   if (c == '=')
     currentToken = EQ;
   else
   {
     ungetNextChar();
     save = FALSE;
     currentToken = ASSIGN;
   }
   break;
```

```
case INLT:
  state = DONE;
  if (c == '=')
    currentToken = LE;
  else
  {
    ungetNextChar();
    save = FALSE;
    currentToken = LT;
  }
  break;
```

```
case INGT:
  state = DONE;
  if (c == '=')
    currentToken = GE;
  else
  {
    ungetNextChar();
    save = FALSE;
    currentToken = GT;
  }
  break;
```

```
case INNE:
  state = DONE;
  if (c == '=')
    currentToken = NE;
  else
  {
    ungetNextChar();
    save = FALSE;
    currentToken = ERROR;
  }
  break;
```

⑥ INOVER일 때는 새로 검출하는 문자가 '*' 일 경우 INCOMMENT state 로 이동 하여 주석을 입력 받고 있다는 것을 나타낸다.

⑦ INCOMMENT 일 때는 새로 검출하는 문자가 '*' 가 아닌 경우에는 무시하고, '*' 일 경우에는 주석이 종료 될 수 있으므로 INCOMMENT_ state 로 이동한다.

⑧ INCOMMENT_ 일 때는 새로 검출하는 문자가 '/' 일 경우 주석을 종료하고 시작 state 로 이동하고, '*' 인 경우에는 다시 INCOMMENT_ 로, 둘 다 아닌 경우 INCOMMENT로 이동한다.

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

⑨ INNUM, INID 일 때는 계속 숫자인지, 계속 알파벳인지 판단하여 토큰을 구분한다. (스크린 샷 생략)

⑩ scan.c 에서 state machine 으로 검출한 토큰들을 util.c 출력한다.

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

3) Flex 를 이용한 cminus_flex scanner

① cminus.l 파일을 생성하여 토큰 검출 규칙을 입력해준다.

② 각 reserved word 와 특수기호 토큰에 대한 규칙을 입력한다.

```
"while"         {return WHILE;}
"return"        {return RETURN;}
"void"          {return VOID;}
"int"           {return INT;}
```

```
"=="            {return EQ;}
"="             {return ASSIGN;}
"!="            {return NE;}
"<"             {return LT;}
"<="            {return LE;}
">="            {return GE;}
">"             {return GT;}
","             {return COMMA;}
```

```
"("             {return LPAREN;}
")"             {return RPAREN;}
";"             {return SEMI;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
```

③ comment 를 입력 받고 있을 때는 '*' 와 '/' 가 연속해서 나왔을 때만 return 한다.

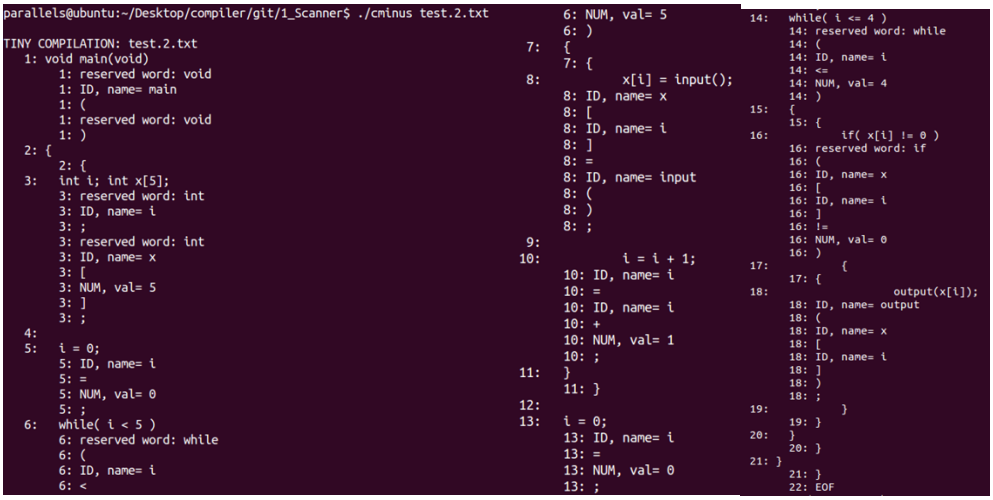
```
"/*"            { char c;
                  int check = 0;
                  do
                  { c = input();
                    if (c == EOF) break;
                    if (c == '\n') lineno++;
                    else if (c == '*')
                    {
                      check = 1;
                    }
                    else if (c == '/' && check == 1){
                      check = 2;
                    }
                    else
                      check = 0;

                  } while (check!=2);
                }
```

4) Makefile 을 이용해 위 의 코드들을 컴파일 하여 cminus 와 cminus_flex 를 만들었다.

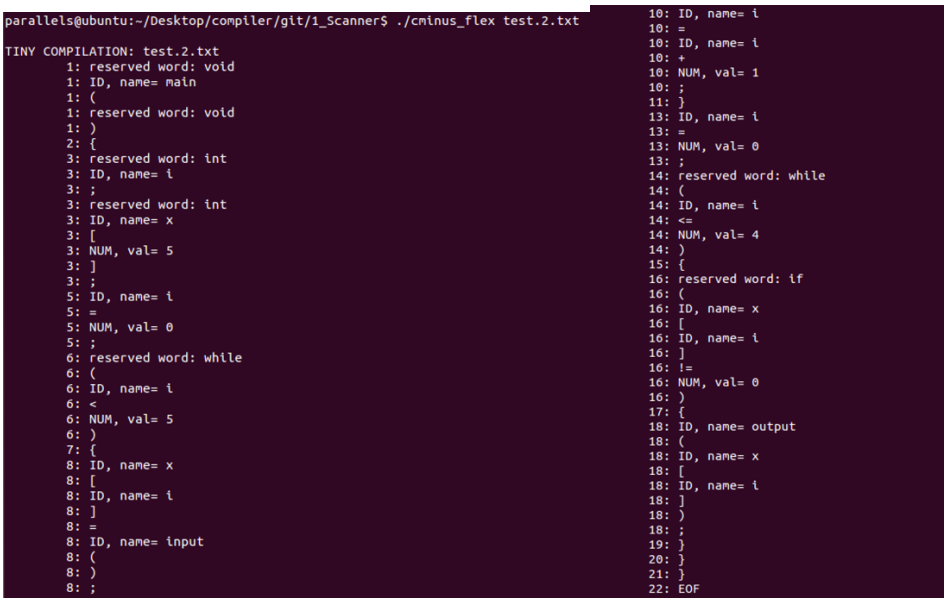
5. 예시 결과 확인

blackboard 에 올라온 test.2.txt 를 사용하여 테스트 해보았습니다.

cminus 로 scan



ciminus_flex 로 scan



주석의 nested 와 주석이 닫힐 때 **/ 가 입력 되는 등 예외를 잘 처리 하는지 test.c 파일을 만들어 확인 해보았습니다.

test.c 파일

cminus scanner 를 사용하여 scan

```
parallels@ubuntu:~/Desktop/compiler/git/1_Scanner$ ./cminus test.c

TINY COMPILATION: test.c
   1: 1+1
        1: NUM, val= 1
        1: +
        1: NUM, val= 1
   2: 1<2
        2: NUM, val= 1
        2: <
        2: NUM, val= 2
   3: ==
        3: ==
   4: =
        4: =
   5: hello
        5: ID, name= hello
   6: /*asdfsadfsadfas*/
   7: 1111
        7: NUM, val= 1111
   8: /*adfdklasjsda/*sadfas*/
   9:
  10: /*sdfsadfsadf **/ hello
       10: ID, name= hello
  11: hello
        11: ID, name= hello
        12: EOF
```

cminus_flex scanner 를 사용하여 scan

```
parallels@ubuntu:~/Desktop/compiler/git/1_Scanner$ ./cminus_flex test.c

TINY COMPILATION: test.c
        1: NUM, val= 1
        1: +
        1: NUM, val= 1
        2: NUM, val= 1
        2: <
        2: NUM, val= 2
        3: ==
        4: =
        5: ID, name= hello
        7: NUM, val= 1111
        10: ID, name= hello
        11: ID, name= hello
        12: EOF
```