Project #1. Scanner

- Compilation Environment
  - macOS Monterery 12.2.1(M1 Pro 16GB)
  - clang : Apple clang version 13.1.6 (clang-1316.0.21.2.5)
  - flex : flext 2.6.4 Apple(flex-34)

- C code Implementation
  - C code scanner를 구현하기위해서 기본코드에서 프로젝트 pdf에 제공된 힌트대로 'main.c', 'globals.h', 'utils.c', 'scan.c'를 수정하였습니다.
  - 'main.c'는 project spec에 주어진대로 flag setting만 수정하였습니다.
    'globals.h'와 'utils.c'는 사진과 같이 C-Minus 토큰들을 추가하고 기존의 Tiny token들은 지우는 정도로만 수정하였습니다.

```c
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;
```

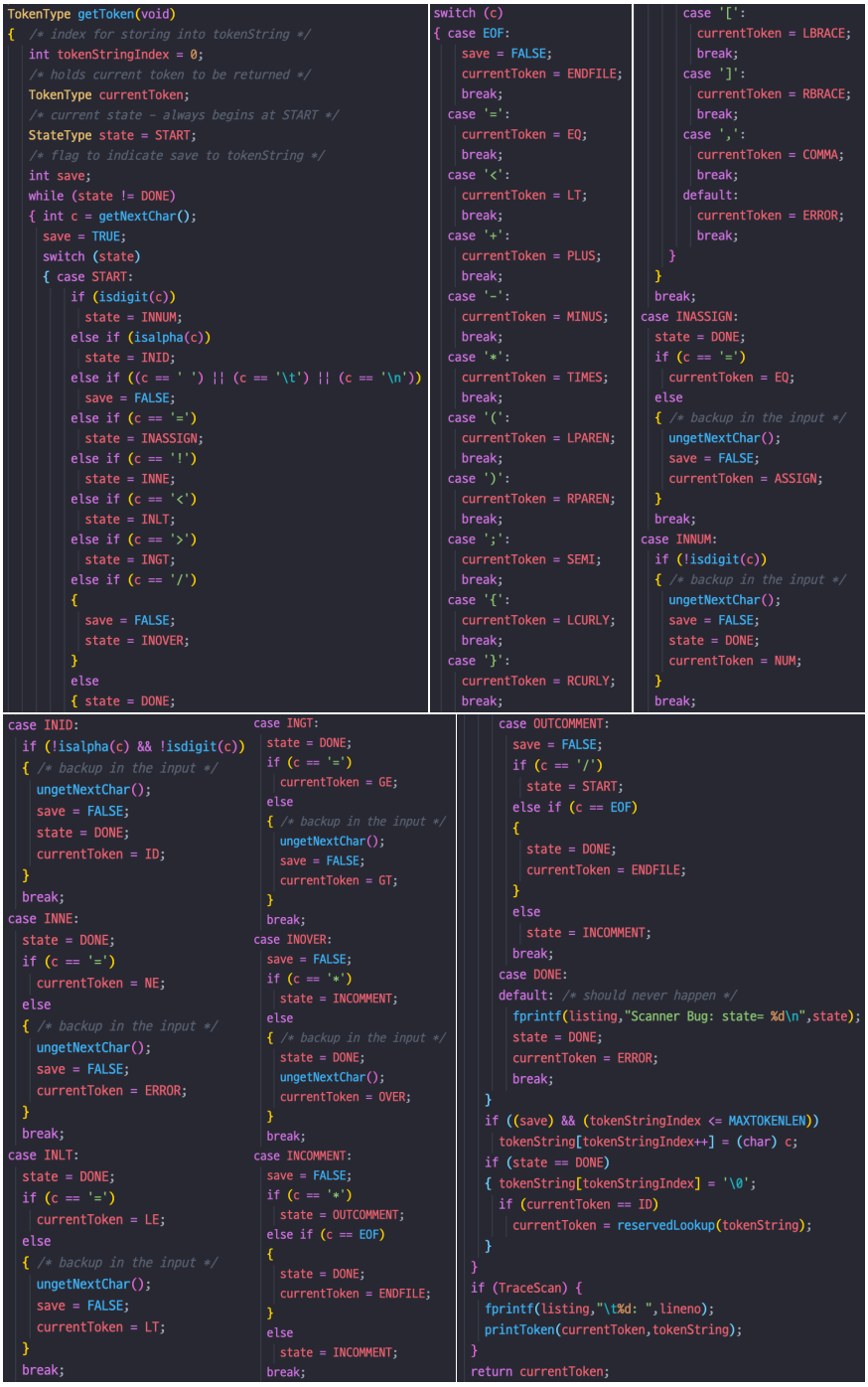
```c
void printToken( TokenType token, const char* tokenString )
{ switch (token)
  { case IF:
    case ELSE:
    case WHILE:
    case RETURN:
    case INT:
    case VOID:
      fprintf(listing,
          "reserved word: %s\n",tokenString);
      break;
    case ASSIGN: fprintf(listing,"=\n"); break;
    case LT: fprintf(listing,"<\n"); break;
    case LE: fprintf(listing,"<=\n"); break;
    case EQ: fprintf(listing,"==\n"); break;
    case NE: 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 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;
    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 ENDFILE: fprintf(listing,"EOF\n"); break;
```

  - 'scan.c'가 가장 많은 수정이 필요했습니다. 실제로 input token을 가지고 DFA simulate를 하는 함수인 getToken 함수가 존재하기 때문입니다. getToken함수는 input을 받아서 state를 이동하며 DFA simulate하고 output token string에 저장하는 함수입니다.

    앞서 다른 부분과 마찬가지로 주어진 spec을 참고해서 INASSIGN, INLT, INGT, INNE, INOVER, INCOMMENT, OUTCOMMENT state를 새로 만들었습니다. 또한 기존에 존재하는 필요없는 tokentype은 지우고 새로 추가된 타입은 추가하였습니다. <=, >=, ==, != 등 2개의 token으로 이루어진 경우에는 추가된 state, save, ungetNextChar()를 활용해서 spec에 예시로 주어진 assign example처럼 DFA transition을 하도록 구현하였습니다.

    테스트 과정을 거치면서 가장 문제가 생긴 부분은 역시 comment를 다루는 부분이였습니

다. comment 처리는 INOVER, INCOMMENT, OUTCOMMENT 3개의 state를 가지고 처리가 되도록 구현하였습니다. ‘/‘가 들어올 경우, INOVER state로 간 후, input token을 하나 더 받아서 ‘*’일 경우 INCOMMENT state로, 아닐 경우엔 단순 나누기 연산자로 판단하고 OVER state로 갑니다. INCOMMENT state에서는 들어오는 input이 모두 주석이기 때문에 tokenstring에 저장하지않고 계속 같은 state에서 loop를 돌다 ‘*’이 들어오면 주석을 닫는 경우인지 판단하기 위해서 OUTCOMMENT state로 이동합니다. 물론 EOF가 들어올 경우 주석여부와 상관없이 종료를 해야 무한루프에 걸리지않기때문에 따로 처리해주었습니다. 마지막으로 OUTCOMMENT state에서는 주석을 닫는 ‘/‘가 들어올 경우 주석처리를 종료하고 start state로 돌아가고 다른 character가 들어올 경우 다시 INCOMMENT state로 돌아가게됩니다. 마찬가지로 EOF가 들어올 경우는 따로 처리해주었습니다. 자세한 구현방식은 아래 코드와 같습니다.



```
TokenType getToken(void)
{  /* index for storing into tokenString */
   int tokenStringIndex = 0;
   /* holds current token to be returned */
   TokenType currentToken;
   /* current state - always begins at START */
   StateType state = START;
   /* flag to indicate save to tokenString */
   int save;
   while (state != DONE)
   { int c = getNextChar();
     save = TRUE;
     switch (state)
     { case START:
         if (isdigit(c))
           state = INNUM;
         else if (isalpha(c))
           state = INID;
         else if ((c == ' ') || (c == '\t') || (c == '\n'))
           save = FALSE;
         else if (c == '=')
           state = INASSIGN;
         else if (c == '!')
           state = INNE;
         else if (c == '<')
           state = INLT;
         else if (c == '>')
           state = INGT;
         else if (c == '/')
         {
           save = FALSE;
           state = INOVER;
         }
         else
         { state = DONE;
```

```
switch (c)
{ 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 = SEMI;
    break;
  case '{':
    currentToken = LCURLY;
    break;
  case '}':
    currentToken = RCURLY;
    break;
```

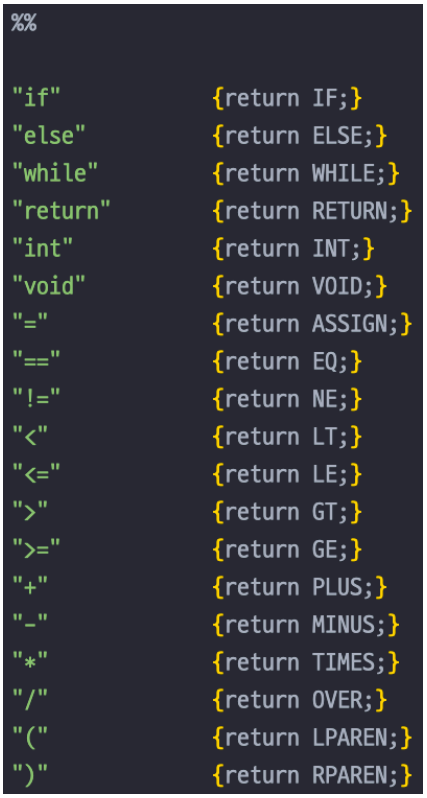
```
      case '[':
        currentToken = LBRACE;
        break;
      case ']':
        currentToken = RBRACE;
        break;
      case ',':
        currentToken = COMMA;
        break;
      default:
        currentToken = ERROR;
        break;
    }
  }
  break;
case INASSIGN:
  state = DONE;
  if (c == '=')
    currentToken = EQ;
  else
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    currentToken = ASSIGN;
  }
  break;
case INNUM:
  if (!isdigit(c))
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    state = DONE;
    currentToken = NUM;
  }
  break;
```

```
case INID:
  if (!isalpha(c) && !isdigit(c))
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    state = DONE;
    currentToken = ID;
  }
  break;
case INNE:
  state = DONE;
  if (c == '=')
    currentToken = NE;
  else
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    currentToken = ERROR;
  }
  break;
case INLT:
  state = DONE;
  if (c == '=')
    currentToken = LE;
  else
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    currentToken = LT;
  }
  break;
```

```
case INGT:
  state = DONE;
  if (c == '=')
    currentToken = GE;
  else
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    currentToken = GT;
  }
  break;
case INOVER:
  save = FALSE;
  if (c == '*')
    state = INCOMMENT;
  else
  { /* backup in the input */
    state = DONE;
    ungetNextChar();
    currentToken = OVER;
  }
  break;
case INCOMMENT:
  save = FALSE;
  if (c == '*')
    state = OUTCOMMENT;
  else if (c == EOF)
  {
    state = DONE;
    currentToken = ENDFILE;
  }
  else
    state = INCOMMENT;
  break;
```

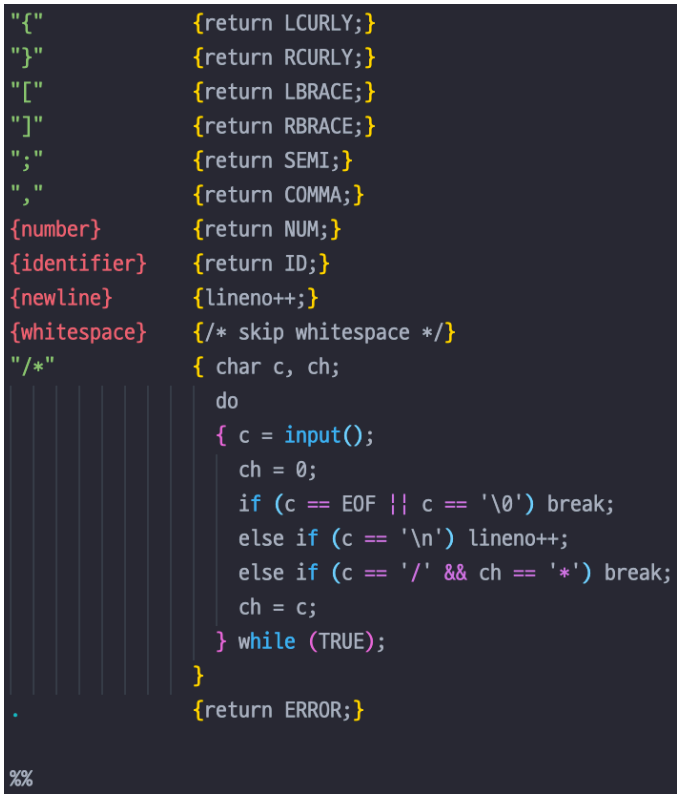
```
      case OUTCOMMENT:
        save = FALSE;
        if (c == '/')
          state = START;
        else if (c == EOF)
        {
          state = DONE;
          currentToken = ENDFILE;
        }
        else
          state = INCOMMENT;
        break;
      case DONE:
      default: /* should never happen */
        fprintf(listing,"Scanner Bug: state= %d\n",state);
        state = DONE;
        currentToken = ERROR;
        break;
    }
    if ((save) && (tokenStringIndex <= MAXTOKENLEN))
      tokenString[tokenStringIndex++] = (char) c;
    if (state == DONE)
    { tokenString[tokenStringIndex] = '\0';
      if (currentToken == ID)
        currentToken = reservedLookup(tokenString);
    }
  }
  if (TraceScan) {
    fprintf(listing,"\t%d: ",lineno);
    printToken(currentToken,tokenString);
  }
  return currentToken;
```

- Lex / Flex Implementation
  - Lex scanner는 기본 코드인 tiny.l을 조금 수정해 cminus.l을 구현하였습니다.
  - C-Minus에 맞게 먼저 keyword, token들을 추가해주었습니다. 그리고 역시 마찬가지로 가장 신경을 쓴 부분은 comment 처리였습니다. C implemenation에 비해선 상대적으로 간단하게 구현할 수 있었습니다. 먼저 주석처리 시작부분은 단순히 “/*”으로 처리할 수 있었고 주석 마무리는 마지막 input token 2개를 계속해서 저장해서 “*/“와 비교를 하는 방식으로 처리하는 방식으로 비교적 간단히 구현할 수 있었습니다. 자세한 구현방식은 아래 코드와 같습니다.



```
%%

"if"            {return IF;}
"else"          {return ELSE;}
"while"         {return WHILE;}
"return"        {return RETURN;}
"int"           {return INT;}
"void"          {return VOID;}
"="             {return ASSIGN;}
"=="            {return EQ;}
"!="            {return NE;}
"<"             {return LT;}
"<="            {return LE;}
">"             {return GT;}
">="            {return GE;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
```



```
"{"             {return LCURLY;}
"}"             {return RCURLY;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
";"             {return SEMI;}
","             {return COMMA;}
{number}        {return NUM;}
{identifier}    {return ID;}
{newline}       {lineno++;}
{whitespace}    {/* skip whitespace */}
"/*"            { char c, ch;
                  do
                  { c = input();
                    ch = 0;
                    if (c == EOF || c == '\0') break;
                    else if (c == '\n') lineno++;
                    else if (c == '/' && ch == '*') break;
                    ch = c;
                  } while (TRUE);
                }
.               {return ERROR;}

%%
```

- Sample Test Result

- 제시된 결과와 같은 것을 확인했고 몇가지 예외사항(comment, brace) 테스트도 이상없음을 확인했습니다.

Testcase	C code	Lex / Flex	Given Result

text.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
		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
			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

text.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
		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
			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