Compiler Design

Project #1. Scanner

2018008904 이성진

본 프로젝트는 Tiny Compiler를 수정하여 C-minus를 컴파일할 수 있게 하는 scanner를 구현하는 과제입니다.

1. Compilation environment and method

- 프로젝트를 개발하고 컴파일한 환경은 Intel CPU를 사용하는 Windows 10 Edu 64bit에서 WSL 2 버전으로 Ubuntu 20.04를 설치하여 사용했습니다. C compiler는 gcc를 사용했습니다.
- 프로젝트를 빌드하기 위해서는 make 명령을 활용하면 되며, 아래 Makefile의 내용을 바탕으로 세부 옵션을 설정할 수 있습니다.

- **make**: cminus_cimpl과 cminus_lex를 모두 컴파일합니다. 이후 해당 디렉토리에 생성된 cimnus_cimpl과 cminus_lex 바이너리 파일을 C-minus 코드 파일과 함께 실행시키면 scanner가 처리한 결과를 출력하게 됩니다.
- **make cminus_cimpl**: C로 구현한 DFA implementation에 대해서만 컴파일을 수행합니다. cminus_cimpl라는 이름의 바이너리 파일만 생성됩니다.
- **make ciminus_lex**: flex를 이용하여 구현한 implementation에 대해서만 컴파일을 수행합니다. cminus_lex라는 이름의 바이너리 파일만 생성됩니다.
- 생성된 바이너리 파일은 cminus_XXX [C-minus code file]과 같이 command argument를 통한 입력 파일을 지정해 주면 해당 파일을 scan하여 token들을 추출한 결과를 출력해줍니다.

2. Brief explanations about how to implement and how it operates

- 이 프로젝트는 C-minus로 작성된 프로그램의 lexical analysis를 위해 적절한 reserved keyword, symbol, 그리고 token 등을 정의한 다음 DFA를 활용하여 입력받은 문자열을 token 단위로 분리할 수 있게 하는 scanner를 만드는 과제입니다.

2.1. Definitions

Reserved words:
int void if else while return (소문자)
Symbols:
문자 1개: + - * / < > = ; , ( ) [ ] { }
문자 2개: <= >= == !=
Identifier & Number:
digit = 0 | 1 | ... | 9
letter = a | b | ... | z | A | B | ... | Z
NUM = digit+
ID = letter(letter | digit)*
Whitespace: space, newline, tab
Comments:
/* comments */ (C의 주석과 동일한 형식이며, single line comments(//)는 없음)

2.2. Changing Tiny compiler definitions to C-minus definitions

main.c: NO_PARSE와 TraceScan을 TRUE로 설정하여 lexical analysis만 진행하며 각각의 토큰을 출력할 수 있도록 설정했습니다.

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE TRUE
```

```
int TraceScan = TRUE;
```

globals.h: 기존 Tiny를 위해 정의된 reserved words와 symbol들 및 reserved word의 개수인 MAXRESERVED 등을 재정의해 주었습니다.

```
/* MAXRESERVED = the number of reserved words */
#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;
```

util.c: globals.h와 마찬가지로 새로 정의되거나 삭제해야 하는 reserved word나 symbol들에 대해서 화면에 올바르게 출력할 수 있도록 printToken 함수 내부를 수정했습니다.

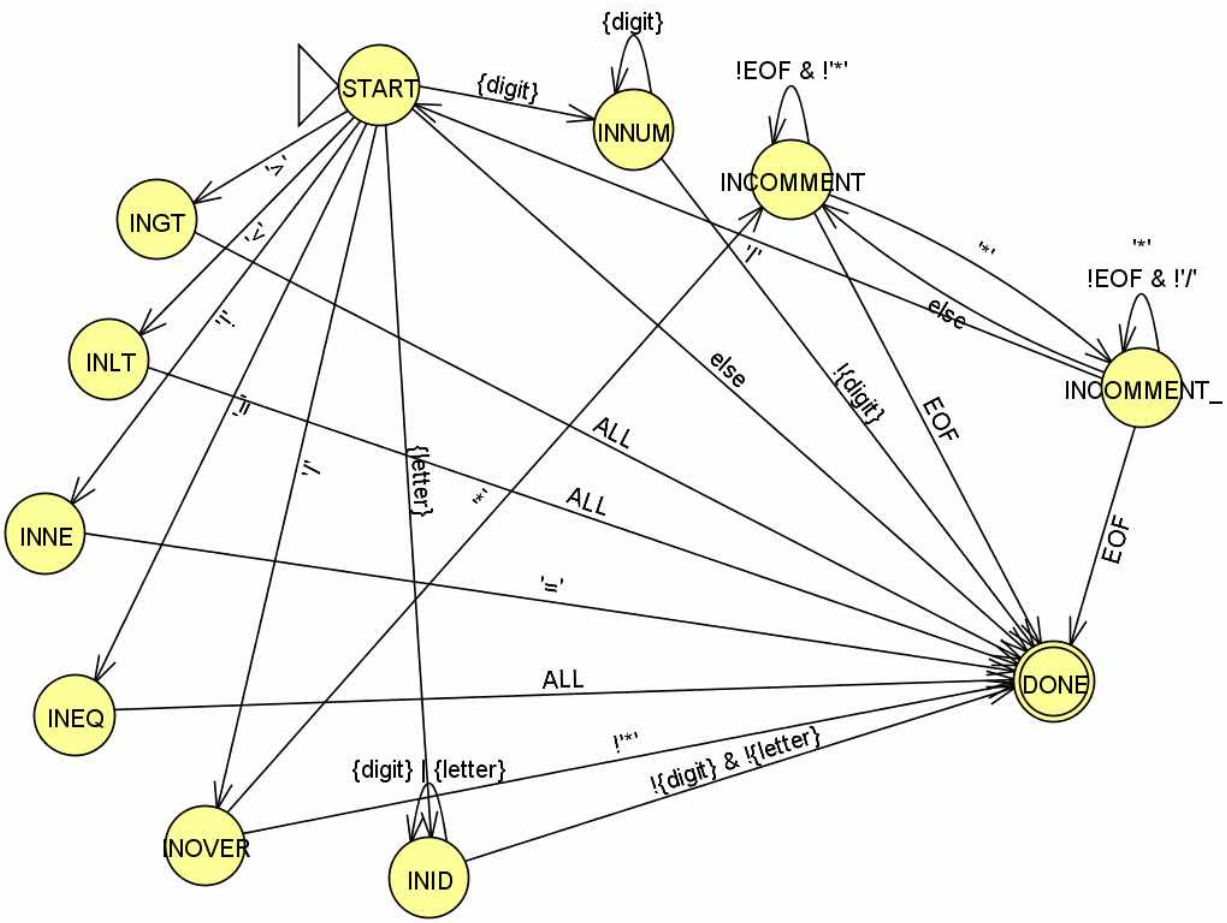
```
void printToken( TokenType token, const char* tokenString )
{ switch (token)
  { case IF:
    case ELSE:
    case WHILE:
    case RETURN:
    case INT:
    case VOID:
      fprintf(listing,
         "reserved word: %s\n",tokenString);
      break;
    case ASSIGN: fprintf(listing,"=\n"); break;
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

2.3. C implementation

- C-minus의 규칙에 맞게 각종 keyword들의 정의를 마친 후에는 scanner를 구현해야 하는데, 먼저 C를 활용하여 DFA를 이용해 각 state를 이동하며 현재 입력된 토큰을 구별하는 getToken 함수를 구현하였습니다.

- getToken 함수는 입력 코드의 문자를 한 개씩 읽으며 DFA에 정의된 규칙을 바탕으로 현재 입력된 문자에 따라 각 state를 이동하며 처음 START state에서 마지막으로 DONE state로 이동하게 되면 입력된 토큰의 종류를 return 하게 됩니다. 중간에 주석이 존재하는 경우, 주석 부분은 생략하고 나머지 부분에 대한 토큰만 분석하게 됩니다.

- C로 구현된 getToken 함수 내부에서 정의된 DFA를 다이어그램으로 나타내면 아래와 같습니다.



2.3. Lex implementation

- C implementation과 같은 결과물을 출력하지만, lexical analysis를 하는 과정을 Lex 프로그램(flex)를 이용하여 대신 처리하여 구현하는 방법입니다. cminus_lex 파일에 C-minus를 컴파일하기 위한 규칙을 정의한 뒤 make cminus_lex를 실행하면 flex 프로그램이 자동으로 규칙을 따르는 getToken 함수를 생성하여 적용하게 됩니다.

- cminus.l 파일 내부에는 Tiny와는 다른 C-minus의 reserved word, token, identifier, number, 그리고 주석을 인식하기 위한 regular expression과 C 코드를 작성했습니다. 주석을 처리하기 위해서 Rule section에 /* 문자열이 입력된 경우 이후에 입력되는 문자열을 지속적으로 확인하면서 */가 나타나는 경우 주석이 종료되는 것을 인식하도록 하는 코드를 입력했습니다.

3. Examples and corresponding result screenshots

make 명령을 이용해 생성된 cminus_cimpl과 cminus_lex를 test용 프로그램을 컴파일하기 위해 실행한 결과 주석에 해당되는 부분은 무시되고 나머지 코드에 대한 lexical analysis가 정상적으로 이루어진 것을 볼 수 있습니다.

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

test.1.txt

```
/1_Scanner$ ./cminus_cimpl ./test.1.txt
```

```
/1_Scanner$ ./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
```

scanner result