ELE4029 Project Report
1. Scanner

박준영

Contents

1 How to build

1.1 Compilation Environment

본 프로젝트는 다음의 환경에서 개발, 테스트 하였다.

- Ubuntu Server 20.04.3 LTS
- gcc version 9.3.0
- GNU Make 4.2.1
- flex 2.6.4

1.2 Compilation Method

우선 아래 명령어를 통해 hconnect에서 repository를 clone한다.

```
git clone https://github.com/JYPark09/ELE4029
```

이후 다음 명령어들을 수행하여 컴파일한다.

```
cd ELE4029/1_Scanner
make
```

컴파일이 완료되면 `cminus_lex`와 `cminus_cimpl` 실행파일이 생성된다.

2 Implementation

2.1 Method 1: C implementation

2.1.1 Tokens

C-MINUS의 언어 스펙에 맞춰 다음의 토큰 타입을 정의하였다.

reserved words : IF, ELSE, WHILE, RETURN, INT, VOID

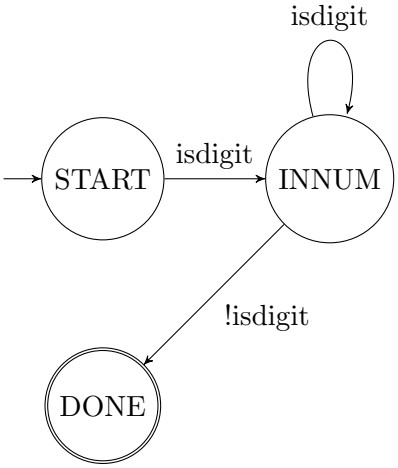
multicharactor tokens : ID, NUM

special symbols : ASSIGN(=), EQ(==), NE(!=), LT(<), LE(<=), GT(>), GE(>=), PLUS(+), MINUS(-), TIMES(*), OVER(/), LPAREN((), RPAREN()), LBRACE([), RBRACE(]), LCURLY({), RCURLY(}), SEMI(;), COMMA(,)

2.1.2 DFA

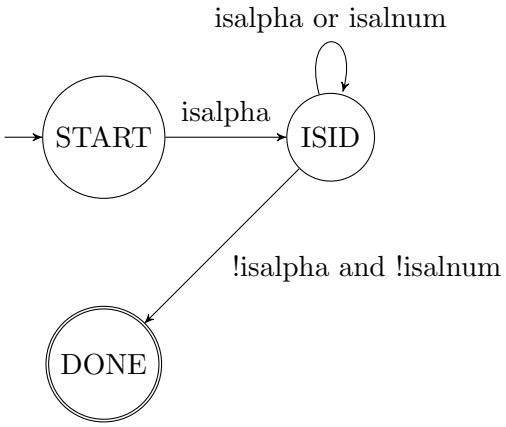
아래 그림들은 각 상황에 따른 DFA의 일부분이다.

숫자(number)인지 여부를 판단할 때



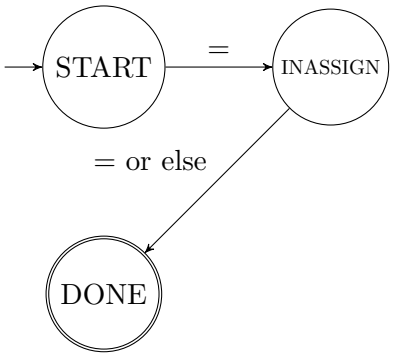
DONE state에 도달하면 현재 토큰의 타입에 NUM을 부여한다.

식별자(identifier)인지 여부를 판단할 때



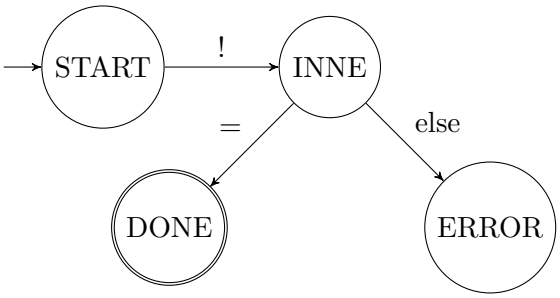
DONE state에 도달하면 현재 토큰의 타입에 ID를 부여한다.

ASSIGN과 EQ 토큰 구분



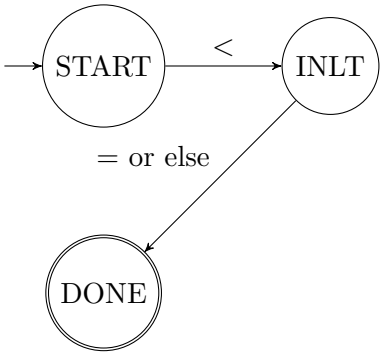
INASSIGN state에서 현재 character가 '='이면 현재 토큰의 타입에 EQ를, 그 이외에는 ASSIGN을 부여한다.

NE 토큰 구분



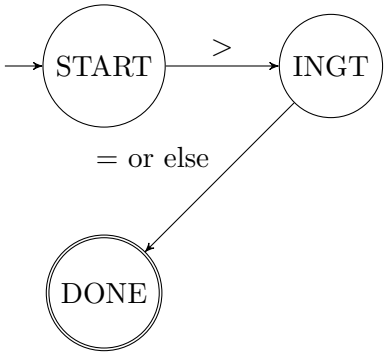
DONE state에 도달하면 현재 토큰의 타입에 NE를 부여한다.

LT와 LE 토큰 구분



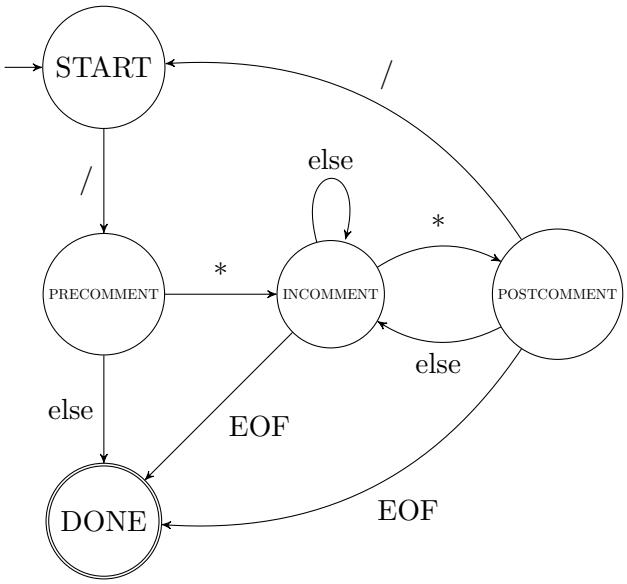
INLT state에서 현재 character가 '='이면 현재 토큰의 타입에 LE를, 그 이외에는 LT를 부여한다.

GT와 GE 토큰 구분



INGT state에서 현재 character가 '='이면 현재 토큰의 타입에 GE를, 그 이외에는 GT를 부여한다.

주석(comment)과 OVER 구분



PRECOMMENT state에서 '*'이 아닌 다른 character가 들어왔을 땐 현재 토큰의 타입에 OVER를 부여한다. INCOMMENT, POSTCOMMENT state에서 EOF를 만나면 현재 토큰의 타입에 ENDFILE을 부여한다.

2.2 Method 2: Flex

기존 코드에서 identifier의 정의를 C-MINUS 스펙에 맞춰 변경하였고, reserved words와 symbol을 추가로 정의, 제거하였다. 주석의 경우 "/*"을 찾아 EOF나 "*/"이 나올 때까지 모든 character를 discard하는 방식으로 구현하였다.

3 Results

테스트에 쓰인 sample code는 testcase 디렉터리에 위치해있다. 아래는 각 sample과 실행 결과이다.

test.0.txt : 산술 연산자 테스트

Sample code

```
int foo1(int u, int v)
{
   < > <= >= != == =
}
```

Result

```
1: reserved word: int
1: ID, name= foo1
1: (
1: reserved word: int
1: ID, name= u
1: ,
1: reserved word: int
1: ID, name= v
1: )
2: {
3: <
3: >
3: <=
3: >=
3: !=
3: ==
3: =
4: }
5: EOF
```

test.1.txt : 기본 제공 샘플
제공된 예시 결과와 동일한 결과가 출력되어 생략함.

test.2.txt : 기본 제공 샘플 (내용 생략)
제공된 예시 결과와 동일한 결과가 출력되어 생략함.

test.3.txt : 중괄호가 닫히지 않았을 때 정상 작동하는지 테스트

Sample code

```
int foo1(int v, int z)
{
    /* ********************** */
    /* Hi//*/

    foo ( v, z);
```

Result

```
1: reserved word: int
1: ID, name= foo1
1: (
1: reserved word: int
1: ID, name= v
1: ,
1: reserved word: int
1: ID, name= z
1: )
2: {
6: ID, name= foo
6: (
6: ID, name= v
6: ,
6: ID, name= z
6: )
6: ;
7: EOF
```

test.4.txt : 주석이 닫히지 않았을 때 정상 작동하는지 테스트

Sample code

```
int foo1(int v, int z)
{
    /* ********************** */
    /* Hi//

    foo ( v, z);
}
```

Result

```
1: reserved word: int
1: ID, name= foo1
1: (
1: reserved word: int
1: ID, name= v
1: ,
1: reserved word: int
1: ID, name= z
1: )
2: {
8: EOF
```

4번째 줄에서 시작한 주석이 닫히지 않아 그 이후 줄의 모든 character는 주석 처리되어 무시되었고, infinity loop 등이 발생하지 않고 정상적으로 종료되는 것을 확인할 수 있다.

test.5.txt : 주석이 주석이 아닌 코드와 같은 line에 있어도 정상 작동하는지 테스트

Sample code

```
int foo1(int v, int z)
{
    /* ********************** */
    /* Hi/*/

    foo ( v, z); /* call foo */
    /* call foo2 */ foo(z, v);
}
```

Result

```
1: reserved word: int
1: ID, name= foo1
1: (
1: reserved word: int
1: ID, name= v
1: ,
1: reserved word: int
1: ID, name= z
1: )
2: {
6: ID, name= foo
6: (
6: ID, name= v
6: ,
6: ID, name= z
6: )
6: ;
7: ID, name= foo
7: (
7: ID, name= z
7: ,
7: ID, name= v
7: )
7: ;
8: }
9: EOF
```