오토마타와 컴파일러

Scanner(Lexer)

이름(학번)	황 준 일 (32131766)
담당교수	이 상 범 교수님
제출일	2019. 11. 12

1. Input file

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv[]) {
  // this is line comment
  int n1 = 1, n2 = 2, n3 = 3, n4;
  float n_5, n_6;
  char _c;
  /*
    this is
    block comment
  */
  printf("n1: %d, n2: %d, n3: %d", n1, n2, n3);
  n4 = (n1 + n2*n3)%10 / 2;
  _c = 'A';
  n_5 = 0.01;
  n_6 = 10.01;

  if (n1 == 1) {
    n1 = 2;
  } else n1 = 3

  switch (n1) {
    case 1:
      n1 = 10;
    break;
    case 2:
      n1 = 20;
    break;
    default :
    break;
  }

  return 0;
}
```

2. Table

A. Optable

```
{
  "+": 0,             // 연산자
  "-": 1,             // 연산자
  "*": 2,             // 연산자
  "/": 3,             // 연산자
  "=": 4,             // 연산자
  "%": 5,             // 연산자
  ")": 6,             // 괄호 시작
  "(": 7,             // 괄호 끝
  ";": 8,             // 세미콜론(문장의 끝)
  ",": 9,             // int a, b, c 등에서 사용
  "[": 10,            // 배열 시작
  "]": 11,            // 배열 끝
  "{": 12,            // 블록 시작
  "}": 13,            // 블록 끝
  "<": 14,            // 비교 연산자
  ">": 15,            // 비교 연산
  "==": 16,           // 비교 연산
  "<=": 17,           // 비교 연산
  ">=": 18,           // 비교 연산
  "!=": 19,           // 비교 연산
  ":": 20,            // label 표기 혹은 삼항 연산자
  "int": 21,          // 타입
  "char": 22,         // 타입
  "float": 23,        // 타입
  "void": 24,         // 타입
  "return": 25,       // 키워드
  "#include": 26,     // 키워드
  "if": 27,           // 키워드
  "else": 28,         // 키워드
  "for": 29,          // 키워드
  "while": 30,        // 키워드
  "switch": 31,       // 키워드
  "case": 32,         // 키워드
  "break": 33,        // 키워드
```

```
  "default": 34,        // 키워드
  "number": 35,         // 숫자 상수
  "string": 36,         // 문자 상수
  "line_comment": 37,   // 한줄 주석
  "block_comment": 38,  // 블럭 주석
  " ": 39,              // 공백문자
  "Wt": 40,             // 탭문자
  "WrWn": 41,           // 줄바꿈(윈도우)
  "Wn": 42,             // 줄바꿈(맥, 리눅스)
  "Wr": 43,             // Carriage Return
}
```

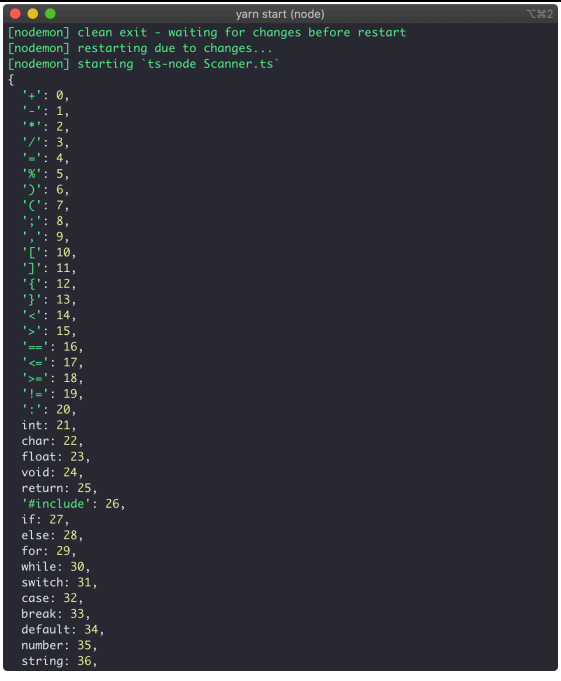
B. Symbol Table

```
{
  "stdio.h": 44,       // stdio.h 라는 문자열. 잘못 읽은 것 같다.
  “stdlib.h”: 45,      // stdio.h 라는 문자열. 마찬가지로 잘못 읽은 것 같다.
  "main": 46,          // main 함수
  "argc": 47,          // main의 매개변수
  "argv": 48,          // main의 매개 변수
  "n1": 49,            // 변수
  "n2": 50,            // 변수
  "n3": 51,            // 변수
  "n4": 52,            // 변수
  "n_5": 53,           // 변수
  "n_6": 54,           // 변수
  "_c": 55,            // 변수
  "printf": 56         // 함수
}
```

C. Token List

```
26 14 44 15 42 26 14 45 15 42 42 21 46 39 7  21 47 9  39 22 39 48 10 11 6  39 12 42 39 39 37 39 39 21 49 39 4  39
35 9  39 50 39 4  39 35 9  39 51 39 4  39 35 9  39 52 8  42 39 39 23 53 9  39 54 8  42 39 39 22 55 8  42 39 39 38
39 39 56 7  36 9  39 49 9  39 50 9  39 51 6  8  42 39 39 52 39 4  39 7  49 39 0  39 50 2  51 6  5  35 39 3  39 35
8  42 39 39 55 39 4  39 36 8  42 39 39 53 39 4  39 35 8  42 39 39 54 39 4  39 35 8  42 42 39 39 27 7  49 39 4  4
39 35 6  39 12 42 39 39 39 39 49 39 4  39 35 8  42 39 39 13 39 28 49 39 4  39 35 42 42 39 39 31 7  49 6  39 12 42
39 39 39 39 32 35 20 42 39 39 39 39 39 39 49 39 4  39 35  8 42 39 39 39 39 33 42 39 39 39 39 32 35 20 42 39 39 39
39 39 39 49 39 4  39 35 8  42 39 39 39 39 33 42 39 39 39 39 34 20 42 39 39 39 39 33 42 39 39 13 42 42 39 39 25 35
8  42 13
```

C. 실행결과확인



3. Source Code

```
import fs from 'fs';

class Scanner {
  private opTable = {
    '+': 0,              // 연산자
    '-': 1,              // 연산자
    '*': 2,              // 연산자
    '/': 3,              // 연산자
    '=': 4,              // 연산자
    '%': 5,              // 연산자
    ')': 6,              // 괄호 시작
    '(': 7,              // 괄호 끝
    ';': 8,              // 세미콜론(문장의 끝)
    ',': 9,              // int a, b, c 등에서 사용
    '[': 10,             // 배열 시작
    ']': 11,             // 배열 끝
    '{': 12,             // 블록 시작
    '}': 13,             // 블록 끝
    '<': 14,             // 비교 연산자
    '>': 15,             // 비교 연산
    '==': 16,            // 비교 연산
    '<=': 17,            // 비교 연산
    '>=': 18,            // 비교 연산
    '!=': 19,            // 비교 연산
    ':': 20,             // label 표기 혹은 삼항 연산자
    'int': 21,           // 타입
    'char': 22,          // 타입
    'float': 23,         // 타입
    'void': 24,          // 타입
    'return': 25,        // 키워드
    '#include': 26,      // 키워드
    'if': 27,            // 키워드
    'else': 28,          // 키워드
    'for': 29,           // 키워드
    'while': 30,         // 키워드
    'switch': 31,        // 키워드
    'case': 32,          // 키워드
    'break': 33,         // 키워드
    'default': 34,       // 키워드
    'number': 35,        // 숫자 상수
    'string': 36,        // 문자 상수
    'line_comment': 37,  // 한줄 주석
    'block_comment': 38, // 블럭 주석
    ' ': 39,             // 공백문자
    'Wt': 40,            // 탭문자
    "WrWn": 41,          // 줄바꿈(윈도우)
    "Wn": 42,            // 줄바꿈(맥, 리눅스)
    "Wr": 43,            // Carriage Return
  }

  // symbol table 정의
  private symbolTable = {}

  // symbolTable의 key의 값. 추가할 때 마다 increment
  private lastSymbolNumber: number = 44;

  // 생성자에서 소스코드를 받은 후 처리
  constructor (code: string) {
    let i: number = 0,               // 반복용 i
        last: number = code.length,  // code의 길이 만큼 반복
        token = ''                   // token을 기록함
    const tokenList: number[] = [],  // tokenList를 기록함
          { opTable, symbolTable } = this // this에 있는 opTable, symbolTable을 받아옴

    // 반복문을 실행하면서 토큰 기록
    while (i < last) {
      // i의 위치에 있는 문자를 가져옴
      const now = code[i]

      // 만약 i에 따옴표가 들어가면, 이어서 나오는 따옴표를 찾은 후 문자열 토큰으로 기록함
      if (['"', "'"].indexOf(now) !== -1) {
        const next: number = code.indexOf(now, i + 1) + 1
        tokenList.push(opTable['string'])
        i = next; // 다음 i의 위치는 따옴표가 끝나는 위치로 변경
```

```
        continue  // 현재 로직 탈출 후 다시 실행
      }

      // 주석 처리
      // i의 위치가 마지막이 아닐 때 진입
      // now와 다음 문자 조합이 주석의 시작일 때 터리
      if (i < last - 1 && ['/*', '//'].indexOf(`${now}${code[i + 1]}`) !== -1) {
        const commentStart: string = `${now}${code[i + 1]}`
        const chk: boolean = commentStart === '/*'
        const commentType: string = chk ? 'block_comment' : 'line_comment'
        const next: number = chk ? code.indexOf('*/', i) + 2 : code.indexOf("Wn", i)
        tokenList.push(opTable[commentType])
        i = next + 1;
        continue;
      }

      // 나머지는 if가 아닌 switch로 처리
      // 코드를 작성 후 생각해본 결과, 위의 if문도 switch로 묶어도 무관할듯함.
      // case에서 체크하는 조건이 true일 때 진입
      switch (true) {
        case opTable[token] !== undefined : // token이 opTable에 존재할 경우,
          tokenList.push(opTable[token])    // tokenList에 기록
          token = ''                        // token 초기화
        break;
        case opTable[now] !== undefined :   // now의 값이 opTable에 존재할 경우,
          if (token.length > 0) {           // 여태까지 기록한 token이 0보다 클때
            switch (true) {
              // token이 숫자 형태일 때 처리
              case /^([0-9]+)$/.test(token) :
              case /^([0-9]+W.[0-9]+)$/.test(token) :
                tokenList.push(opTable['number'])
                token = ''
              break;
              // token이 숫자도 아니라면, 사용자 정의어로 생각하고 처리함
              default :
                // symbolTable에 기록 되지 않은 정의어라면, symbol table에 기록함
                if (symbolTable[token] === undefined) {
                  symbolTable[token] = this.lastSymbolNumber++
                }
                // 그리고 tokenList에 이어 붙임
                tokenList.push(symbolTable[token]);
              break;
            }
          }
          // now를 tokenList에 이어 붙인 후 token 초기화
          tokenList.push(opTable[now]);
          token = '';
        break;
        default :
          token += now
        break;
      }
      // 다음 문자열로 넘어감
      i++;
    }

    // 반복문 종료
    console.log(opTable)              // optable의 값 조회
    console.log(symbolTable)          // symbolTable의 값 조회
    console.log(tokenList.join(' '))  // tokenList의 값 조회
  }
}

// test.c를 읽어들인 후 Scanner에게 넘김
fs.readFile('./test.c', 'utf-8', (err, buffer) => {
  new Scanner(buffer);
});
```

4. 프로그램에 대한 설명

이 프로그램은 if, while, for, switch, function, variable, number, string 정도의 토큰들을 구분할 수 있습니다. 소스코드는 typescript로 작성하였으며, DFA를 기반으로 작성할까 고민하다가 시간이 없어서 간단한 정규식과 약간의 규칙(문자열의 시작과 끝, 주석의 시작과 끝 등)을 이용하여 만들었습니다. 이 수업 이전에는 html parser, json parser 같은 것들을 만들었던 경험이 있습니다. 그런데 이렇게 scanner를 구

현 했던 적은 처음이라 굉장히 혼란스러웠습니다. 문법을 검사하는 것이 아닌 token만 읽어들여서 테이블로 만든다는 개념 자체가 신기했고, 또 곰곰히 생각해보면 굉장히 효과적이라는 것 또한 알 수 있었습니다. 과제 제출 이후에 scanner를 더 정교하게 만들어볼 생각입니다.

5. Bonus – 계산기

```
const computer = v => {
  const str = v.trim().replace(/\s+/g, '') // 문자열에서 공백을 제외함
  const len = str.length; // 문자열의 길이를 가져옴
  const nums: number[] = [] // 숫자가 저장됨
  const opers: any[] = [] // 연산자가 저장됨
  let number = ''
  for (let i = 0; i < len; i++) { // 반복문을 돌면서 연산자/숫자를 분류
    const now = str[i]
    if (['+', '-', '*', '/'].indexOf(now) !== -1) {
      nums.push(~~number) // 현재 문자가 연사자일 경우, 현재까지 기록한 숫자를 push
      opers.push(now) // 그 후 연산자를 push
      number = '' // 숫자를 초기화
    } else {
      number += now // 연사자가 아닐경우 문자열에 숫자를 계속 쌓음
      if (i === len - 1) { // 현재 위치가 마지막일 경우, numbs에 push
        nums.push(~~number)
      }
    }
  }
  let now = nums.shift()
  while (opers[0] !== undefined) {
    const oper = {
      '+': (n1, n2) => n1 + n2, // 더하기 연산 함수
      '-': (n1, n2) => n1 - n2, // 빼기 연산 함수
      '*': (n1, n2) => n1 * n2, // 곱하기 연산 함수
      '/': (n1, n2) => n1 / n2, // 나누기 연산 함수
    }[opers.shift()]; // 위의 테이블에서 현재 연산자에 해당하는 함수를 바로 가져옴
    now = oper(now, nums.shift()) // now에 연산을 한 후 저장
  }
  return now // now에 현재 연산 결과가 기록되어있는 상태. 이것을 반환하면 됨
}

console.log(computer('1 + 2 - 3'))
console.log(computer('1 * 2 - 3'))
console.log(computer('1 * 2 + 3'))
console.log(computer('1 * 2 / 3'))
```

```
[nodemon] starting `ts-node Computer.ts`
0
-1
5
0.6666666666666666
```

원래 scanner를 이용해 만들어야 하지만, 시간이 없어서 네이티브 하게 만들어봤습니다.