미니컴파일러 개발

(2021년 2학기: 오토마타와컴파일러)

분반: 1분반

이 름: 한예진

학 번: 32164881

학 과: 무역학과

제출일: 2021년 12월 11일

Mini-C compiler의 스캐너 개발 프로젝트

1. 스캐너(Scanner) 기능

스캐너(Scanner)는 컴파일러 전단부에서 어휘를 분석하는 도구로, 소스 프로그램(문자열)을 입력으로 받아들여 문법적으로 의미가 있는 최소 단위인 토큰(token)으로 분리하여 토큰 스트림을 출력해낸다.

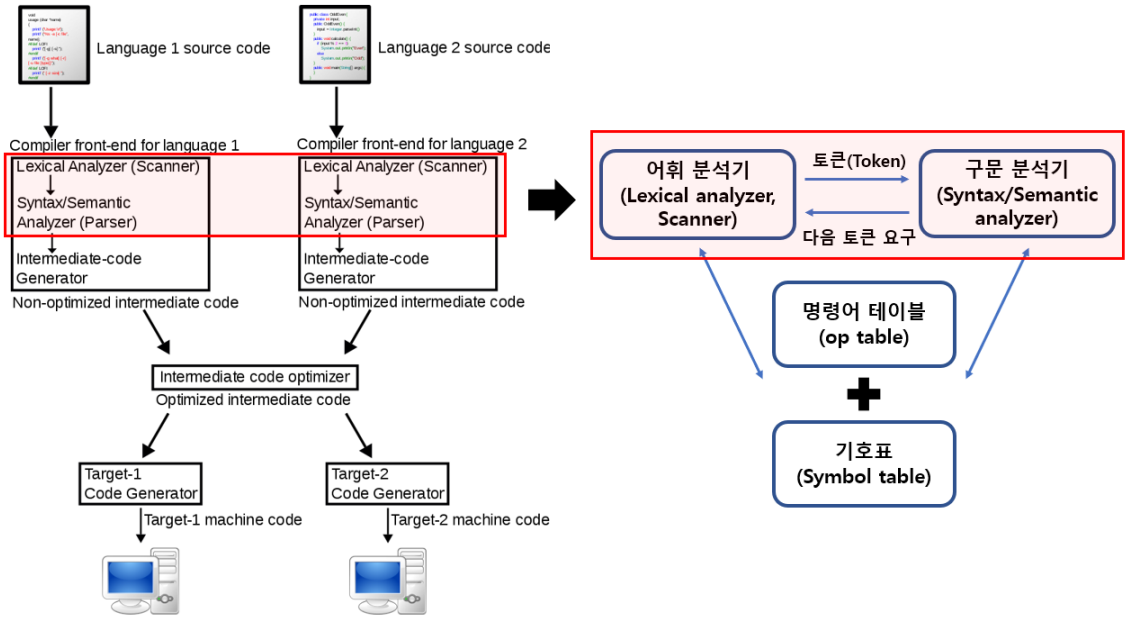


그림 1 컴파일러 동작 과정에서 어휘 분석기와 구문 분석기의 상호 작용 (출처: ko.wikipedia.org/wiki/컴파일러)

그림 1을 보면, 컴파일러의 첫 번째 단계에서 시작되는 메인 부분은 어휘 분석기(Lexical Analyzer)와 구문 분석기(Syntax analyzer)이다. 어휘 분석기인 스캐너는 구문 분석기의 보조 시스템으로, 메인 부분은 구문 분석기이다. 구문 분석기는 입력으로 들어온 소스 코드가 문법에 맞는지 판단하는데, 이를 위해 구문 분석기는 필요할 때마다 어휘 분석기에게 토큰을 구별해서 보내 달라고 요청한다. 요청을 받은 어휘 분석기는 소스 코드의 각 어휘(토큰)들이 유효한지에 대해서 전문적으로 확인하여 토큰을 보낸다.

스캐너는 컴파일 시 번역할 필요가 없는 부분(comment, white space, tab, new line 등)을 제외시키고 macro를 처리하는 전처리(preprocessing) 과정 이후 토큰을 인식한다. 그림 1의 오른쪽을 보면, 어휘 분석기와 구문 분석기는 작업을 수행하는 과정에서 명령어 테이블과 기호표를 참조하는 것을 알 수 있다. 명령어 테이블(op table)에는 for, if, while 등 이미 그 위치와 의미가 정해져 있는

예약어(keyword)가 있다. 한편 기호표는(Symbol table) 변수, 함수, 상수 이름 등 프로그래머가 임의로 정의할 수 있는 어휘인 user identifier(Id)를 저장한다. 표 1은 명령어 테이블과 심볼 테이블을 구성하는 토큰들의 종류를 분류한 표이다.

표 1 일반적인 프로그래밍 언어에서 사용하는 토큰들 (출처: 강의자료 ch04_어휘분석 슬라이드5)

토큰 형태	토큰 타입	Description	example
Special form (by. Language designer), 각각이 토큰이 됨	**지정어/예약어 (Keyword)**	언어에 이미 정의된 단어	for, if, while, int, char 등
	연산자 기호 (Operator symbol)	연산에 쓰이는 기호	+. -. *, /, <, =, ++ 등
	구분자 (Delimiter)	단어와 단어, 문장과 문장을 구분 (특수 기호)	(,), [,], {, }, ;, , 등
General form (by. Programmer), 모두 묶어서 식별자/상수 토큰	**식별자 (Identifier)**	프로그래머가 정의하는 변수	abc, b12, _sum 등
	상수 (Constant)	실수형, 정수형, 문자형 상수	526, 3.0, 0.1234e-10, 'string'등

이러한 명령어 테이블(op table)은 이미 해당 언어 개발자에 의해 채워져 있고 각 명령어 테이블의 토큰들의 의미는 바꿀 수 없지만, 기호표(symbol table)는 컴파일이 수행되기 이전에 비워져 있는 상태이다. 이 비어 있는 기호표는 스캐너가 실행될 때 채워진다. 먼저 소스 프로그램의 토큰이 예약어가 아닌지 검사하고, 예약어가 아니라면 테이블에 토큰(user identifier)을 채워 넣는다. 또한 스캐너와 구문 분석기는 문법적으로 맞지 않는 문장과 어휘에 대한 진단을 내리고 그에 맞는 에러 메시지(syntax error)를 출력하는 역할도 함께 수행한다.

스캐너는 하나의 [1]DFA (Deterministic finite automation)이다. 이에 맞춰 그림 1을 다시 보면, 어휘는 type3의 조건을 만족하는 형식 언어이고, 이 어휘를 판별하는 가상 기계를 프로그램화한 것이 어휘분석기, 즉 스캐너임을 알 수 있다.

[1] 형식 언어 중 Type3에 해당하는 정규 언어를 판독할 수 있는 가상 기계

2. 프로젝트 설명 (각 토큰에 대한 설명 및 DFA)

본 프로젝트에서는 위와 같이 컴파일러의 앞단에서 수행되는 스캐너인 mini-C-Scanner를 개발하였습니다.

2.1 프로젝트 구상목표와 방향

Mini C의 Scanner를 객체지향언어로 구현하기로 하였습니다. 샘플 코드 사이트에는 JAVA로 구현하였지만, 좀 더 익숙한 C++로 구현하기로 하였습니다. 예제를 참고하며 코드를 C++로 작성하고, 개발지침을 따라 코드를 완성해 나가기로 하였습니다.

2.2 프로그램 설명

Mini C 소스코드(*.mc)를 입력받아 Token을 추출해내는 프로그램입니다. ~~~~프로그램이 실행되는 과정 설명하기

2.2.1 토큰에 대한 설명

표 2 토큰의 종류 및 번호

토큰 종류	토큰 번호	참고
tID (식별자)	3	
tConst (상수)	4	리터럴 상수 ex) 숫자 상수'10', 문자 상수'abc'
tInt (정수)	5	
tReal (실수)	6	
EoF	7	파일의 끝을 의미하는 EoF 문자 상수
Plus (더하기 연산자, +)	10	사칙 연산자
Minus (빼기 연산자, -)	11	사칙 연산자
Mul (곱하기 연산자, *)	12	사칙 연산자
Div (나누기 연산자, /)	13	사칙 연산자
Mod (나머지 연산자, %)	14	사칙 연산자
Assign (배정 연산자, =)	15	배정 연산자

Not (Not 연산자, !)	16	논리 연산자
And (And 연산자, &&)	17	논리 연산자
Or (Or 연산자, ||)	18	논리 연산자
Equal (==)	19	관계 연산자
NotEqu (!=)	20	관계 연산자
Less (<)	21	관계 연산자
Great (>)	22	관계 연산자
Lesser (<=)	23	관계 연산자
Greater (>=)	24	관계 연산자
LBracket ([ )	30	특수 기호 연산자
RBracket ( ])	31	특수 기호 연산자
LBrace ({)	32	특수 기호 연산자
RBrace (})	33	특수 기호 연산자
LParen : (	34	특수 기호 연산자
RParen :)	35	특수 기호 연산자
Comma (,)	36	특수 기호 연산자
Semicolon (;)	37	특수 기호 연산자
LApostro (‘)	38	특수 기호 연산자
RApostro (’)	39	특수 기호 연산자
If	40	예약어
While	41	예약어
For	42	예약어
Const	43	예약어
Int	44	예약어
Float	45	예약어
Else	46	예약어
Return	47	예약어

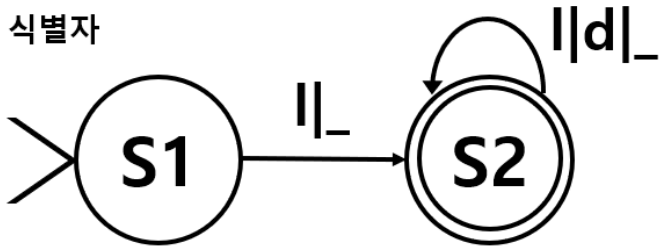
Void	48	예약어
Break	49	예약어
Continue	50	예약어
Char	51	예약어

표 2는 각각의 토큰들을 구분하기 위해 고유의 내부번호를 부여합니다. 본 프로그램의 스캐너는 위 토큰들을 받아들여 사용자 정의어(id)와 예약어(reserved word)를 구별해야 합니다. Id와 예약어는 처음에는 구별이 되지 않고, 두 가지 경우 모두 먼저 예약어 테이블을 참조합니다. 만약 예약어 테이블에 토큰이 있다면 예약어 토큰으로, 토큰이 없다면 id로 간주해야 합니다. 규칙을 만족하는 id는 심볼 테이블(symbol table)에 저장합니다. 심볼 테이블은 C++의 map 자료구조를 이용하여 구현할 것이고, key는 token으로, value는 순번(저장되는 순서)를 저장합니다.

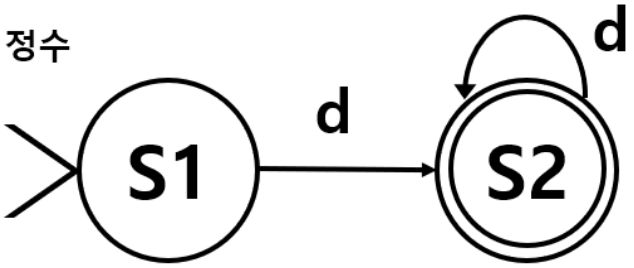
2.2.2 DFA

(상태전이도의 상태는 S1, S2 …로 명명하였으며, 토큰 번호는 표 2에 있습니다)

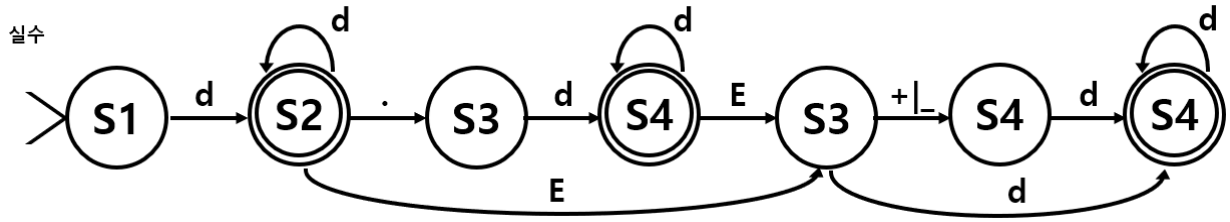
i. 식별자 토큰



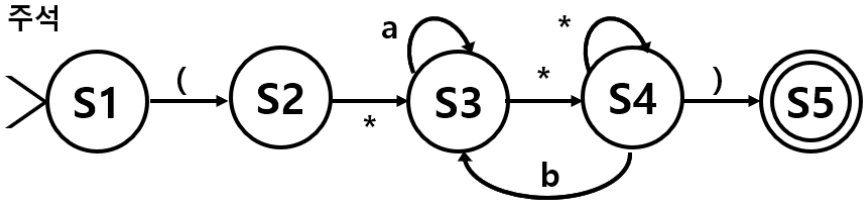
ii. 정수 토큰



iii. 실수 토큰

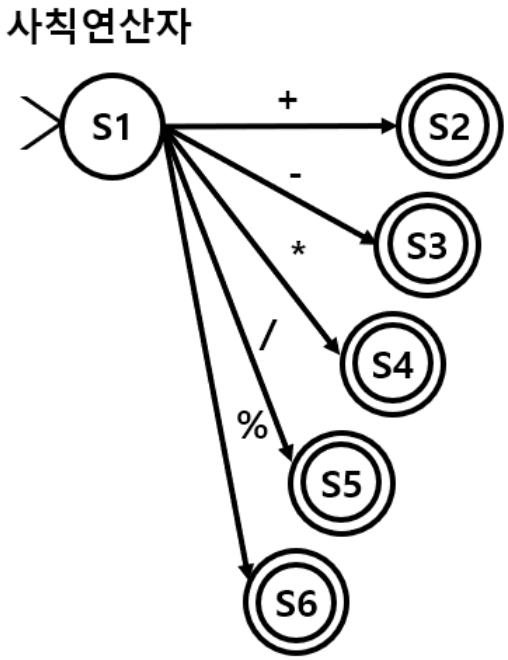


iv. 주석 토큰



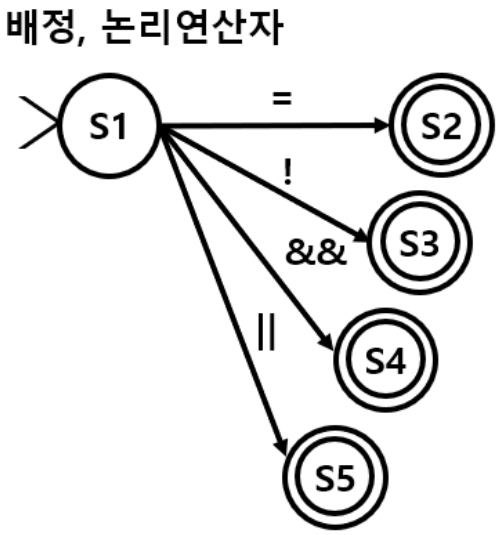
v. 사칙 연산자(그룹) 토큰

더하기 연산자 토큰, 빼기 연산자 토큰, 곱하기 연산자 토큰, 나누기 연산자 토큰, 나머지 연산자 토큰

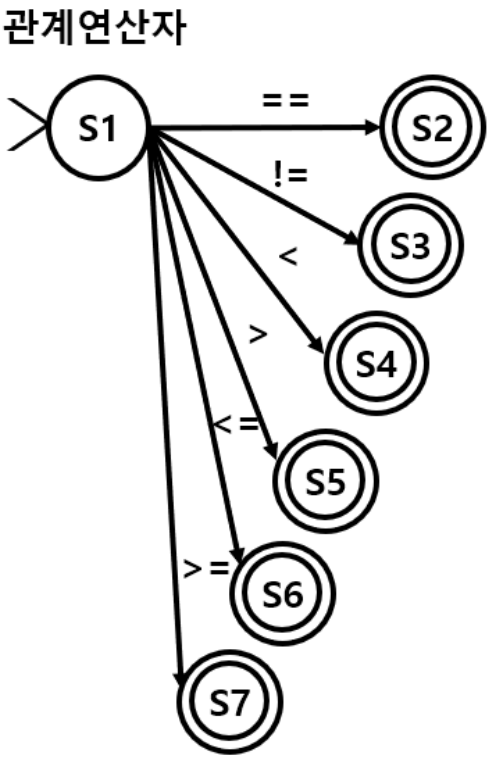


vi. 배정, 논리 연산자(그룹) 토큰

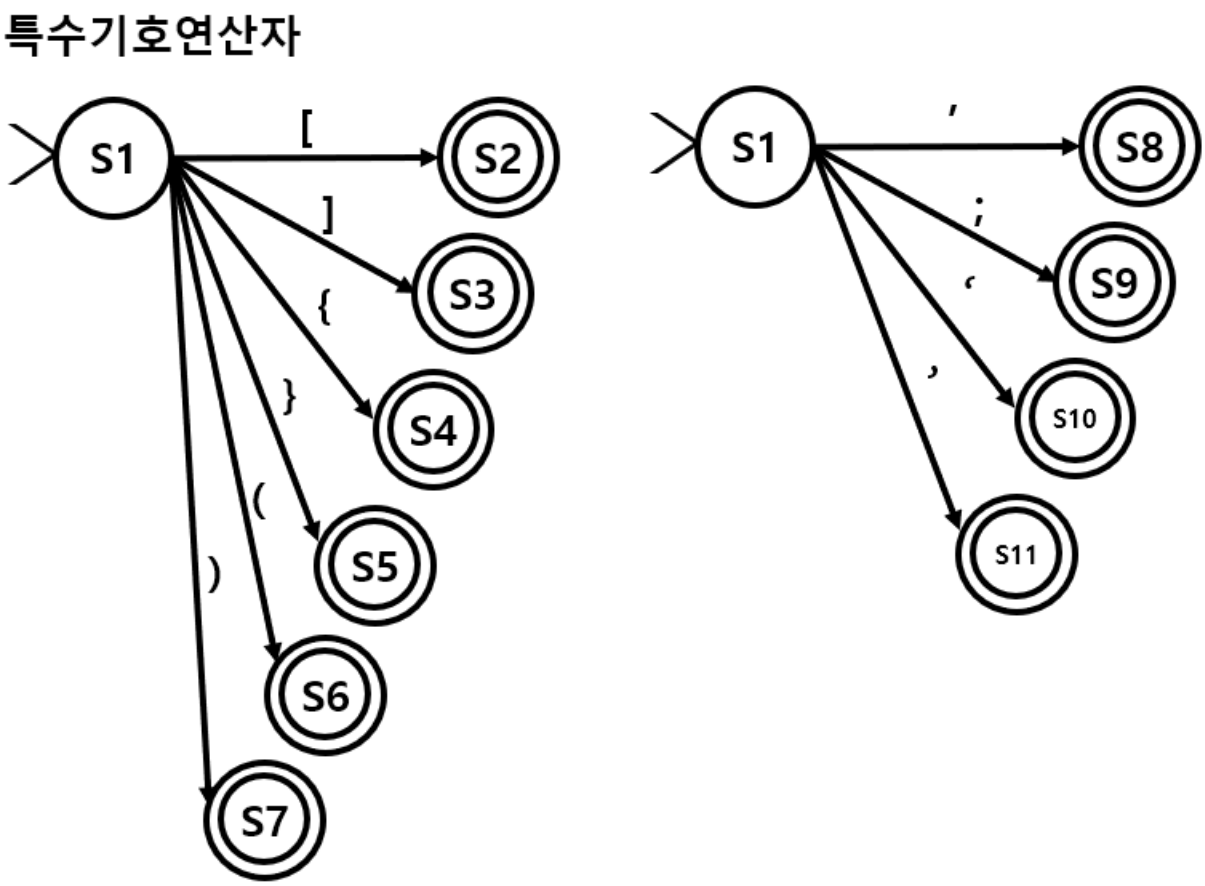
배정 연산자, NOT 연산자, AND 연산자, OR 연산자



vii. 관계 연산자(그룹) 토큰

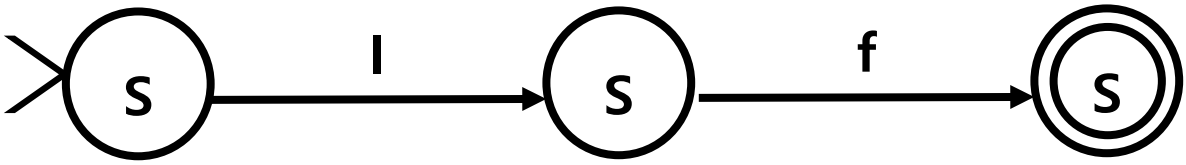


viii. 특수 기호 연산자(그룹) 토큰

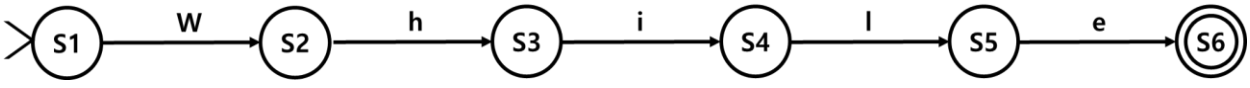


ix. 예약어 토큰

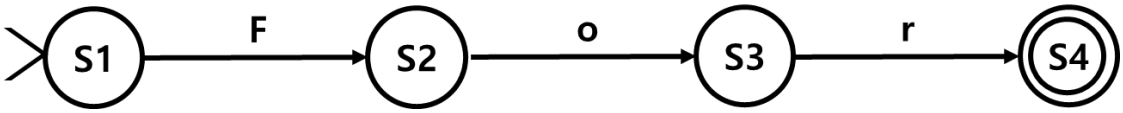
- If 예약어 토큰



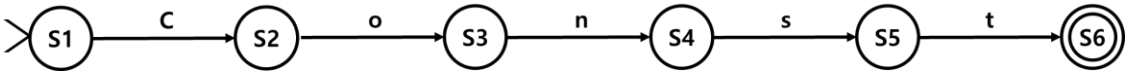
- While 예약어 토큰



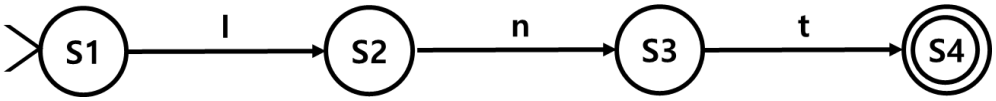
- For 예약어 토큰



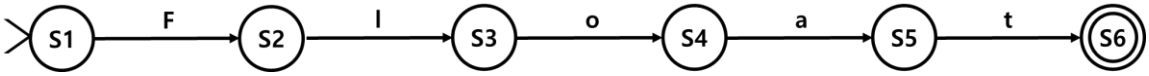
- Const 예약어 토큰



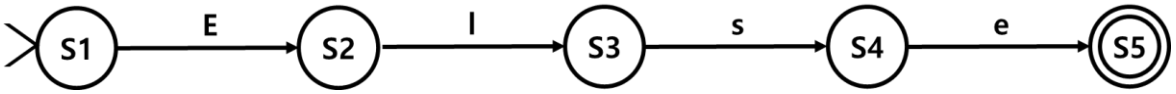
- Int 예약어 토큰



- Float 예약어 토큰

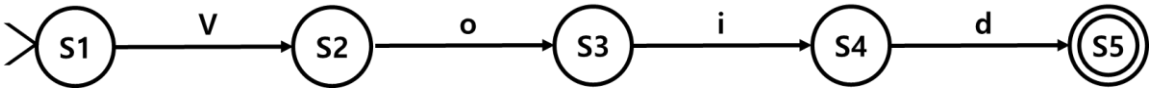


- Else 예약어 토큰

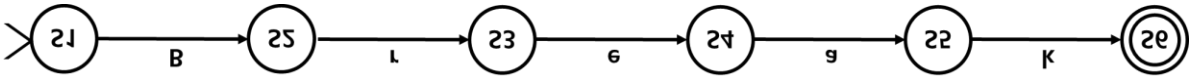


- Return 예약어 토큰

- Void 예약어 토큰

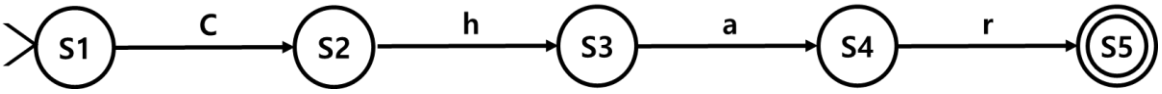


- Break 예약어 토큰



- Continue 예약어 토큰

- Char 예약어 토큰



3. 프로그램 소스 코드

3.1 basic.h

```
#ifndef BASIC_H
#define BASIC_H

#include <iostream>
#include <string>
#include <fstream>
#include <map>
#include <cctype>
#include <regex>
#include "miniCScanner.h"
#include "Token.h"
#include "LexicalError.h"
using namespace std;

#endif // BASIC_H
```

basic.h 헤더파일에서 프로그램 동작에 필요한 헤더들을 포함하였습니다.

3.2 miniCScanner.h

토큰을 추출해내기 위해 처리를 담당하는 miniScanner 클래스에 대한 헤더파일입니다.

```
#ifndef MINI_C_SCANNER_H
#define MINI_C_SCANNER_H
#include "Token.h"

// 토큰을 가져오기 위해 처리를 담당하는 Scanner 클래스
class miniScanner {
public:
    char EoF = '\255'; // 파일의 끝을 의미하는 EOF 문자 상수
    std::string SPECIAL_CHARS = "!=%&*+-/<>|"; // 두 글자 이상이 하나의 토큰일 수
 있는 특수문자들
    int ID_LENGTH = 12; // 명칭의 길이에 제한을 두기 위한 변수 (컴파일러를 구현할
때에는 명칭 길이에 제한을 두는 것이 좋음)
    miniScanner(std::string filePath); // miniScanner 의 생성자
    Token getToken(); // Core Method, 소스코드를 순차적으로 읽으면서 Token 단위로
String 을 나누고 Token 객체를 만들어 반환
private:
    std::string src; // 소스코드의 전체 내용을 string 으로 저장하기 위한 변수
    int idx; // 소스코드를 읽을 때 커서 역할을 하는 변수
```

```
    enum class State { // 토큰을 추출할 때 어떤 토큰을 인식하고 있는지 나타내기 위
한 State
        Initial, Dec, Oct, Hex, Real, MidReal, RealPlace, IDorKeyword, Operato
r, Zero, PreHex, SingleOperator,
    };
    std::string parseFile(std::string filePath); // 소스코드 경로를 통해 소스코
드 파일을 String으로 읽어 들이는 함수
    bool isEoF(int idx); // 커서가 파일의 끝을 가리키고 있는지 확인, 더이상 읽을
문자가 없는지 확인
    bool isSpecialChar(char c); // 문자가 특수문자(1글자 연산자 제외)인지 확인하
는 함수
    bool isSingleSpecialToken(char c); // 문자가 1글자 토큰인지 확인하는 함수
    Token::SymbolType getSymbolType(State s);     // 추출한 token의 symbol typ
e이 어떤 타입인지 입력받은 state에 따라 분류
    bool exceptComment(); // 현재 소스코드 파일에 대한 커서(idx)로부터 유효한 토
큰이 나올 때까지 주석을 무시하는 함수
};

#endif // MINI_C_SCANNER_H
```

3.3 miniCScanner.cpp

토큰을 추출해내기 위해 처리를 담당하는 miniScanner.cpp입니다.

```
#include "miniCScanner.h"
#include "basic.h"

// string type을 switch문 case의 표현으로 사용하기 위한 함수
// 출처: http://www.gilgil.net/?listStyle=list&mid=communities_kr&page=4&docum
ent_srl=812969
static constexpr uint32_t const_hash(const char* p) {
    return *p ? static_cast<uint32_t>(*p) + 33 * const_hash(p + 1) : 5381;
}
// miniScanner의 생성자
miniScanner::miniScanner(std::string filePath) {
    src = parseFile(filePath); // 소스 코드의 파일 경로를 src 멤버 변수에 저장
    idx = 0; // 커서를 맨 처음으로 이동시킴
}
```

```
//소스코드 경로를 통해 소스코드 파일을 String 으로 읽어 들이는 Method
std::string miniScanner::parseFile(std::string filePath) {
    string src = ""; // src: 소스코드를 저장해놓을 string 변수
    string readedString = ""; // 소스코드의 한줄을 담아놓을 string 변수

    ifstream fileReader; // 소스코드 파일을 읽기 위한 파일입력스트림 객체 fileRead
er
    fileReader.open(filePath); // fileReader 정보를 이용해 filepath 경로의 파일을
 연결
    if (fileReader.fail()) {
        // 파일을 읽을 수 없을 경우 에러메시지 발생
        cout<< LexicalError::getErrorMessage(LexicalError::ErrorCode::CannotOp
enFile);
        return "";
    }

    try{
        while (getline(fileReader, readedString)) {
            // 파일로부터 한 줄 읽음
            src += readedString + "\n"; // 한 줄 맨 뒤에 개행문자 추가
        }
        src += EoF;
        fileReader.close();

    } catch(std::exception const& e){
        // 파일을 읽을 수 없을 경우 에러메시지 발생
        cout<<"There was an error: "<<e.what()<<endl;
        cout<< LexicalError::getErrorMessage(LexicalError::ErrorCode::CannotOp
enFile);
    }
    return src;
}
// Core Method, 소스코드를 순차적으로 읽으면서 Token 단위로 String 을 나누고 Token
객체를 만들어 반환
Token miniScanner::getToken(){
    Token token = Token();
    Token::SymbolType symType = Token::SymbolType::null; // Symbol Type 을 NULL
로 설정
    std::string tokenString="";

    State state = State::Initial; // 토큰의 상태는 initial 초기 상태

    // 현재 커서로부터 Comment 제거
    if(exceptComment()){
        // Comment 를 지우는 도중에 ERROR 이 발생했을 경우
        cout<< LexicalError::getErrorMessage(LexicalError::ErrorCode::InvalidC
omment);
        return token;
    }
```

```
    while(!isEoF(idx)){ // 소스코드를 전부 읽을 때까지 계속
        char c = src.at(idx); // 커서로부터 글자 하나를 읽고
        idx++; // 커서를 한 칸 이동

        if(isspace(c)){ // white space (needs trimming)
            if(state != State::Initial) break; // 만약 글자들을 인식하고 있었다면
 그대로 종결
            else continue;
        } else if(isSingleSpecialToken(c)){ // single operator : '(', ')', '{'
, '}', '[', ']', ';', EoF
            if(state == State::Initial){ // 처음부터 1 글자짜리
                state = State::SingleOperator;
                tokenString += c;
            } else {--idx;}
            break;
        } else if(isSpecialChar(c)){ // 1 글자짜리 연산자가 아닌 2 글자 이상의 연산
자가 될 수 있는 연산자의 경우

            if(state != State::Initial && state != State::Operator && state !=
 State::RealPlace){
                --idx;
                break;
            }
            else if(state == State::RealPlace && (c == '+' || c == '-
')){ // 실수를 읽는 상태이고 숫자 e+ 또는 숫자 e-인 경우
                state = State::MidReal; // 실수를 읽는 중간의 상태이다
            }
            else {
                state = State::Operator; // 연산자
            }
        } else if(state == State::Initial && c == '0'){ // Zero 를 인식할 경우
            state = State::Zero;
        } else if(isdigit(c)){ // 숫자를 인식한 경우
            if(state == State::Initial) // 아무것도 인식하지 않았을 경우 10 진수로
 취급
                state = State::Dec;
            else if(state == State::Zero) // 숫자 0 을 인식하고 있었을 경우, 8 진
수로 취급
                state = State::Oct;
            else if(state == State::PreHex) // 0x 까지 인식하고 있었을 경우, 16 진
수로 취급
                state = State::Hex;
            else if(state == State::MidReal){ // 소수점.이나 e+, e-
를 인식하고 있었을 경우, 실수로 취급
                state = State::Real;
            }
```

```
            else if(state == State::Operator){ // 연산자가 나온 뒤 숫자가 나올 경우 while 문 탈출
                --idx;
                break;
            }
        } else if(state == State::Zero && c == 'x'){ // 0x 까지 인식했을 경우
            state = State::PreHex; // 16 진수를 읽는 중
        } else if((state == State::Dec) && (c == '.')){ // 숫자. 까지 인식했을 경우
            state = State::MidReal; // 실수를 읽는 중
        } else if((state == State::Dec || state == State::MidReal || state == State::Real) && (c == 'e')){ //숫자 e 까지 인식했을 경우
            state = State::RealPlace;
        } else if(isalpha(c) || c == '_'){ // underscore 혹은 알파벳을 인식했을 경우
            if(state != State::Initial && state != State::IDorKeyword){
                // 명칭 혹은 키워드가 아닌 토큰을 인식하는 중일 경우 while 문 탈출
                if(state == State::Dec){
                    cout<< LexicalError::getErrorMessage(LexicalError::ErrorCode::InvalidChar);
                }
                --idx;
                break;
            }

            state = State::IDorKeyword; // 명칭 혹은 키워드 인식
        }

        tokenString += c; // 토큰 string 에 글자 추가
    }
    symType = getSymbolType(state); // 인식한 state 로부터 토큰이 어떤 값을 의미하는지 대분류
    if( symType == Token::SymbolType::IDorKeyword && tokenString.length() > ID_LENGTH){
        // 명칭의 길이가 제한을 넘어갈 경우 에러로 처리
        // ERROR : 명칭의 길이 초과
        cout<< LexicalError::getErrorMessage(LexicalError::ErrorCode::AboveIDLimit);
        return token;
    }
    //cout << tokenString << endl;
    token.setSymbol(tokenString, symType); // tokenString 과 함께 대분류한 타입을 전달하여 token 을 세팅
    return token; // 인식한 token 을 반환
}
```

3.4 Token.h

Scanner에서 인식된 토큰 String을 입력받아 정확한 Symbol을 배정하고 객체 자체가 Token이 되는 클래스의 헤더파일입니다.

```
#ifndef TOKEN_H
#define TOKEN_H

#include "basic.h"

static constexpr uint32_t const_hash(const char* p);
// MiniScanner 에서 사용할 Token 의 클래스
class Token {
public:
    enum class SymbolType { // Mini_C_Scanner 객체에서 인식한 토큰의 Symbol 타입 (대분류)
        Operator, IDorKeyword, Digit, RealDigit, null
    };
    enum class TokenSymbol { // 토큰의 키워드나 명칭, 연산자를 식별하기 위한 Symbol (소분류)
        null,
        tID=3, tConst=4,tInt=5,tReal=6,EoF=7,
        /* 연산자 (Operator) */
        Plus=10, Minus=11, Mul=12, Div=13, Mod=14, // 사칙 연산자
        Assign=15, // 배정 연산자
        Not=16, And=17, Or=18, // 논리 연산자
        Equal=19, NotEqu=20, Less=21, Great=22, Greater=23, Lesser=24, // 관계 연산자
        /* 특수 기호 */
        LBracket=30, RBracket=31, LBrace=32, RBrace=33, LParen=34, RParen=35, // 괄호
        Comma=36, Semicolon=37, LApostro=38, RApostro=39, // 기타 특수기호
        /* 예약어 (Keyword) */
        If=40, While, For, Const, Int, Float, Else, Return,
        Void, Break, Continue, Char=51,
    };
    Token(); // 생성자함수, 각각의 멤버변수들을 NULL 로 초기화한다
    void setSymbol(std::string token, Token::SymbolType type); // 토큰의 symbol 과 value 를 설정하는 함수
    int getSymbolOrdinal(); // Symbol 에 해당하는 토큰 심볼을 숫자로 표현하는 함수
    std::string getSymbolValue(); // 토큰이 명칭이나 숫자일 경우 그 토큰 값을 얻는 함수
```

```
    std::string toString(); //출력하기 편하게 하기 위해 정의하는 toString 함수
private:
    Token::TokenSymbol symbol; // 토큰이 가진 Symbol ( Symbol 에 상응하는 정수를 추출해낼 수 있음 )
    bool ID_or_Not; // ID 일 경우 true
    std::string val; // 명칭 혹은 숫자일 경우 그 값을 저장
    std::string tokenString; // 인식한 토큰의 원시 String
    Token::TokenSymbol getIDorKeywordSymbol(std::string token); // 입력받은 토큰이 명칭인지 예약어인지 구분하는 구분하는 함수
    Token::TokenSymbol getOperatorSymbol(std::string token); // 입력받은 토큰이 연산자라면 어떤 연산자인지 구분하는 함수
    int parseInt(std::string s); // 10 진수, 8 진수, 16 진수에 상관 없이 string 을 정수형으로 추출하는 함수
};
#endif //TOKEN_H
```

3.5 Token.cpp

Scanner에서 인식된 토큰 String을 입력받아 정확한 Symbol을 배정하고 객체 자체가 Token이 되는 클래스인 Token.cpp파일입니다.

```
#include "basic.h"
#include "Token.h"

std::map<std::string, int> ID_Symbol_Table; // 사용자 정의어(id)를 만나면 저장할 symbol table
int ID_number = 1;

// string type 을 switch 문 case 의 표현으로 사용하기 위한 함수
// 출처: http://www.gilgil.net/?listStyle=list&mid=communities_kr&page=4&document_srl=812969
static constexpr uint32_t const_hash(const char* p) {
    return *p ? static_cast<uint32_t>(*p) + 33 * const_hash(p + 1) : 5381;
}

// 생성자함수, 각각의 멤버변수들을 NULL 로 초기화한다
Token::Token() {
    symbol = TokenSymbol::null;
    val = "0";
    tokenString = "NULL";
    ID_or_Not = false;
}
```

```
// 입력받은 토큰이 명칭인지 예약어인지 구분하는 구분하는 함수
Token::TokenSymbol Token::getIDorKeywordSymbol(std::string token){ // Symbol 을
 구분할 토큰 string
    // string type 을 swith 문 case 의 표현으로 사용하기 위해 상수값으로 변환
    uint32_t hash = const_hash(token.c_str());
    // Keyword 라면 어떤 Keyword 인지 Symbol 을 할당
    switch (hash) {
        //구분된 토큰 Symbol(예약어) 리턴
        /* 예약어 */
        case const_hash("if"):          return Token::TokenSymbol::If;
        case const_hash("while"):       return Token::TokenSymbol::While;
        case const_hash("for"):         return Token::TokenSymbol::For;
        case const_hash("const"):       return Token::TokenSymbol::Const;
        case const_hash("int"):         return Token::TokenSymbol::Int;
        case const_hash("float"):       return Token::TokenSymbol::Float;
        case const_hash("else"):        return Token::TokenSymbol::Else;

        case const_hash("return"):      return Token::TokenSymbol::Return;
        case const_hash("void"):        return Token::TokenSymbol::Void;
        case const_hash("break"):       return Token::TokenSymbol::Break;
        case const_hash("continue"):    return Token::TokenSymbol::Continue;
        case const_hash("char"):        return Token::TokenSymbol::Char;
            // ID
        default:
            return Token::TokenSymbol::tID; // 예약어 테이블에 없으면 id 로 간주
함
        }
}

// 입력받은 토큰이 연산자라면 어떤 연산자인지 구분하는 함수
Token::TokenSymbol Token::getOperatorSymbol(std::string token){ // Symbol 을 구
분할 토큰 string
    // string type 을 swith 문 case 의 표현으로 사용하기 위해 상수값으로 변환
    uint32_t hash = const_hash(token.c_str());
    // Keyword 라면 어떤 Keyword 인지 Symbol 을 할당
    switch (hash) {
        // 구분된 토큰 Symbol(연산자) 리턴
        /* 사칙 연산자 */
        case const_hash("+"):   return Token::TokenSymbol::Plus;
        case const_hash("-"):   return Token::TokenSymbol::Minus;
        case const_hash("*"):   return Token::TokenSymbol::Mul;
        case const_hash("/"):   return Token::TokenSymbol::Div;
```

```
        case const_hash("%"):   return Token::TokenSymbol::Mod;
        /* 배정 연산자 */
        case const_hash("="):   return Token::TokenSymbol::Assign;
        /* 논리 연산자*/
        case const_hash("!"):   return Token::TokenSymbol::Not;
        case const_hash("&&"):  return Token::TokenSymbol::And;
        case const_hash("||"):  return Token::TokenSymbol::Or;
        /* 관계 연산자 */
        case const_hash("=="):  return Token::TokenSymbol::Equal;
        case const_hash("!="):  return Token::TokenSymbol::NotEqu;
        case const_hash("<"):   return Token::TokenSymbol::Less;
        case const_hash(">"):   return Token::TokenSymbol::Great;
        case const_hash("<="):  return Token::TokenSymbol::Lesser;
        case const_hash(">="):  return Token::TokenSymbol::Greater;
        /* 특수 기호 */
        case const_hash("["):   return Token::TokenSymbol::LBracket;
        case const_hash("]"):   return Token::TokenSymbol::RBracket;
        case const_hash("{"):   return Token::TokenSymbol::LBrace;
        case const_hash("}"):   return Token::TokenSymbol::RBrace;
        case const_hash("("):   return Token::TokenSymbol::LParen;
        case const_hash(")"):   return Token::TokenSymbol::RParen;
        case const_hash(","):   return Token::TokenSymbol::Comma;
        case const_hash(";"):   return Token::TokenSymbol::Semicolon;
        case const_hash("‘"):   return Token::TokenSymbol::LApostro;
        case const_hash("’"):   return Token::TokenSymbol::RApostro;
        case const_hash("\255"): return Token::TokenSymbol::EoF;
        case const_hash("&"):
            cout<<LexicalError::getErrorMessage(LexicalError::ErrorCode::Singl
eAmpersand);
            break;
        case const_hash("|"):
            cout<<LexicalError::getErrorMessage(LexicalError::ErrorCode::Singl
eBar);
            break;
        default: // 인식하지 못한 TokenSymbol
            cout<<token<<endl;
            cout<<"getOperationSymbol()"<<endl;
            cout<<LexicalError::getErrorMessage(LexicalError::ErrorCode::Inval
idChar);
            break;
    }
    return TokenSymbol::null;
}
```

```
// 토큰의 symbol 과 value 를 설정하는 함수
void Token::setSymbol(std::string token, Token::SymbolType type){
    tokenString=token; // 객체에서 잘라낸 토큰 string
    switch(type){ // Mini C Scanner 객체에서 분류한 타입(IDorKeyword, Operator,
Digit 로 대분류)
        case SymbolType::IDorKeyword: // 객체 타입이 ID 나 Keyword
            // 문자열을 만나면 먼저 예약어 테이블을 참조
            symbol= getIDorKeywordSymbol(token);
            // 만약 예약어 토큰이 아니라 사용자 정의어라면
            if(symbol == Token::TokenSymbol::tID)
            {
                //ID 처리
                if(!std::regex_match(token ,std::regex("[A-Za-z0-
9_]+"))){ // 특수문자를 포함하면
                     // 규칙을 만족하지 않으므로 에러 메시지 발생
                    cout<< LexicalError::getErrorMessage(LexicalError::ErrorCo
de::InvalidChar);
                }
                else{// 특수 문자를 포함하지 않으면
                    //pass
                    if (ID_Symbol_Table.find(token) == ID_Symbol_Table.end())
{ // symbol table 에 토큰이 저장되어 있지 않으면
                        ID_Symbol_Table[token] = ID_number; // symbol table 에
순번(저장되는 순서)을 저장
                        ID_number++; // 순번 1 증가
                    }
                    ID_or_Not = true; // ID 이므로 true 로 바꿈
                }
            }

            // 만약 예약어 토큰이 아니라 keyword 라면
            val = token; // 출력을 위해 토큰 저장
            break;
        case SymbolType::Digit: // 객체 타입이 숫자
            if(token.substr(0, 2) == "‘" && token.substr(token.length() - 2, t
oken.length()) == "’"){ // 리터럴 상수형이면 ex: ‘1234’
                symbol=Token::TokenSymbol::tConst; // 상수 토큰
                val = token;
            }else if(token.substr(0, 2) == "\"" && token.substr(token.length()
 - 2, token.length()) == "\""){ // 리터럴 상수형이면 ex: "1234"
                symbol=Token::TokenSymbol::tConst; // 상수 토큰
                val = token;
            }
```

```
            else{
                symbol=Token::TokenSymbol::tInt;
                val=to_string(parseInt(token));
            }
            break;
        case SymbolType::RealDigit:
            symbol=Token::TokenSymbol::tReal;
            val=token;
            break;
        case SymbolType::Operator:
            symbol=getOperatorSymbol(token);
            break;
        case SymbolType::null:
        default:
            break;
    }
}

// 10진수, 8진수, 16진수에 상관 없이 string을 정수형으로 추출하는 함수
int Token::parseInt(std::string s){ // 정수로 변환할 string을 인자로 받아옴 (ex.
 526, 0526, 0xFF)
    int radix=10; // default 진법은 10진수
    if(s.rfind("0x",0)==0){ //16진수일 경우
        radix=16; // 진법을 16진수로 설정
        s=s.substr(2); // prefix인 0x 제거
    }
    else if(s.rfind("0",0)==0 && s.length()>1){ // 8진수일 경우
        radix=8;
    }
    //cout << "parseInt : " << s << endl;
    return stoi(s, nullptr, radix); // 위에서 설정한 진법대로 진법 변환
}
// Symbol에 해당하는 토큰 심볼을 숫자로 표현하는 함수
int Token::getSymbolOrdinal(){
    int ret = static_cast<int>(symbol);
    //cout<<static_cast<int>(symbol)<<endl;
    if(ret == '\0')
        return -1; // NULL이 -1이기 때문에 -1해야 한다.
    return ret; // 해당 토큰 심볼의 숫자 리턴
}
```

```
// 토큰이 명칭이나 숫자일 경우 그 토큰 값을 얻는 함수
std::string Token::getSymbolValue(){
    if(ID_or_Not)
        return to_string(ID_Symbol_Table[val]);
    //cout<<val<<endl;
    return val; // 토큰 값 리턴 (없을 경우 0 반환)
}

//출력하기 편하게 하기 위해 정의하는 toString 함수
std::string Token::toString(){
    string symbolOrdinal = to_string(getSymbolOrdinal());
    //cout<<getSymbolOrdinal()<<endl;
    return tokenString + "\t : (" + symbolOrdinal + ", " + getSymbolValue() +
")";
}
```

3.6 LexicalError.h

에러들의 종류를 분류하고 출력하기 위한 클래스 헤더파일입니다.

```
#ifndef LEXICALERROR_H
#define LEXICALERROR_H

class LexicalError{
public:
    enum class ErrorCode{
        CannotOpenFile, AboveIDLimit, SingleAmpersand, SingleBar, InvalidChar,
InvalidComment
    };
    static std::string getErrorMessage(ErrorCode code);

};
#endif //LEXICALERROR_H
```

3.7 LexicalError.cpp

에러들의 종류를 분류하고 출력하기 위한 클래스에 대한 LexicalError.cpp 파일입니다.

```
#include "basic.h"
#include "LexicalError.h"

std::string LexicalError::getErrorMessage(ErrorCode code){
std::string msg;
int errNum;
std::string num;
    switch(code){
        case ErrorCode::CannotOpenFile:
            errNum=0;
            num = to_string(errNum);
            msg = "Lexical Error(code: " + num + ")\n";
            msg += "cannot open the file. please check the file path.";
            break;
        case ErrorCode::AboveIDLimit:
            errNum=1;
            num = to_string(errNum);
            msg = "Lexical Error(code: " + num + ")\n";
            msg += "an identifier length must be less than 12.";
            break;
        case ErrorCode::SingleAmpersand:
            errNum=2;
            num = to_string(errNum);
            msg = "Lexical Error(code: " + num + ")\n";
            msg += "next character must be &.";
            break;
        case ErrorCode::SingleBar:
            errNum=3;
            num = to_string(errNum);
            msg = "Lexical Error(code: " + num + ")\n";
            msg += "next character must be |.";
            break;
        case ErrorCode::InvalidChar:
            errNum=4;
            num = to_string(errNum);
            msg = "Lexical Error(code: " + num + ")\n";
            msg += "invalid character!!!";
            break;
```

```
        case ErrorCode::InvalidComment:
            errNum=5;
            num = to_string(errNum);
            msg = "Lexical Error(code: " + num + ")\n";
            msg += "invalid block comment!!!";
            break;
        default:
            msg += "Unknown Error";
            break;

    }
    return msg;
}
```

3.8 main.cpp

프로그램을 실행하기 위한 main() 함수가 포함된 main.cpp파일입니다.

```
#include "basic.h"

int main(int argc, char *argv[]){
    // argv[1]로 입력 파일 경로를 받음
    if(argv[1] == NULL){ // 스캐너가 분석할 파일의 경로를 받지 못했을 경우
        cout<<"Please enter the file path (by debug argument)";
        return 0;
    }

    miniScanner sc = miniScanner(argv[1]); // MiniScanner 객체
    Token tok=Token(); // MiniScanner 에서 얻어낸 token 을 저장하기 위한 Token 변
수
    while((tok=sc.getToken()).getSymbolOrdinal() != -1){
        // MiniScanner 에서 다 읽을 때까지 token 을 얻어서 출력
        cout<<tok.toString()<<endl;
    }
    return 0;
}
```

3.9 Makefile

빌드를 위한 Makefile입니다.

```
CC=g++
CFLAGS=-g -Wall
OBJS= miniCScanner.o LexicalError.o Token.o main.o
TARGET=Main

$(TARGET): $(OBJS)
    $(CC) -o $@ $(OBJS)

miniCScanner.o: miniCScanner.h miniCScanner.cpp
LexicalError.o: LexicalError.h LexicalError.cpp
Token: Token.h Token.cpp
main.o: Token.h miniCScanner.h LexicalError.h basic.h main.cpp

all: $(TARGET)

clean:
    rm -f *.o
    rm -f $(TARGET)
```

4. 실행 결과

#1 sample Program (제시된 프로그램)

_sample1.mc

```
1   int a, b, sum;
2   float x1, y1, zoom;
3   char ch1;
4   if (a>b) then sum = a+b
5   else sum = a+10;
6   while (a ==b) {
7     zoom = (sum + x1)/10;
8     ch1 = ‘123’;
9   }
```

int a, b, sum;	**float int a, b, sum;**
PS D:\C_C++_Projects\helloworld> ./Main.exe _sample1.mc int : (44, int) a : (3, 1) , : (36, 0) b : (3, 2) , : (36, 0) sum : (3, 3)	float : (45, float) x1 : (3, 4) , : (36, 0) y1 : (3, 5) , : (36, 0) zoom : (3, 6) ; : (37, 0)
char char ch1;	**char if (a>b) then sum = a+b**
char : (51, char) ch1 : (3, 7) ; : (37, 0)	if : (40, if) (: (34, 0) a : (3, 1) > : (22, 0) b : (3, 2)) : (35, 0) then : (3, 8) sum : (3, 3) = : (15, 0) a : (3, 1) + : (10, 0) b : (3, 2)
else sum = a+10;	**while (a ==b) {**
else : (46, else) sum : (3, 3) = : (15, 0) a : (3, 1) + : (10, 0) 10 : (5, 10) ; : (37, 0)	while : (41, while) (: (34, 0) a : (3, 1) == : (19, 0) b : (3, 2)) : (35, 0) { : (32, 0)
zoom = (sum + x1)/10;	**ch1 = ‘123’;** **}**
zoom : (3, 6) = : (15, 0) (: (34, 0) sum : (3, 3) + : (10, 0) x1 : (3, 4)) : (35, 0) / : (13, 0) 10 : (5, 10) ; : (37, 0)	ch1 : (3, 7) = : (15, 0) ‘123’ : (4, ‘123’) ; : (37, 0) } : (33, 0) : (7, 0)

#2 sample Program (제시된 프로그램)

_sample2.mc

```
1    int a1, 2b;
2    float x2, y2;
3    a1 := 100;
4    x2 = 12.23e+10;
```

int a1, 2b;	float x2, y2;
PS D:\C_C++_Projects\helloworld> ./Main.exe _sample2.mc int : (44, int) a1 : (3, 1) , : (36, 0) 2b → Lexical Error(code: 4) invalid character!!! ; : (37, 0)	float : (45, float) x2 : (3, 2) , : (36, 0) y2 : (3, 3) ; : (37, 0)
a1 := 100;	**x2 = 12.23e+10;**
a1 : (3, 1) := → Lexical Error(code: 4) invalid character!!! 100 : (5, 100) ; : (37, 0)	x2 : (3, 2) = : (15, 0) 12.23e+10 : (6, 12.23e+10) ; : (37, 0) : (7, 0)

#3 sample Program (임의 작성)

_sample3.mc

```
1    (*
2      factorial program by recursive call
3    *)
4
5    void main()
6    {
7      int _input, fac$#1, f;
8      read(n);
9      write(n);
10     f = factorial(n);
11     write(f);
12   }
13   int factorial(int n)
14   {
15     if (n == 1) return 1;
16      else return n*factorial(n-1);
17   }
```

(* factorial program by recursive call *)	void main()
** 주석처리되어 터미널에 출력되지 않는 부분 **	PS D:\C_C++_Projects\helloworld> ./Main.exe _sample3.mc void : (48, void) main : (3, 1) (: (34, 0)) : (35, 0)
{ int _input, fac$#1, f;	read(n); write(n);
{ : (32, 0) int : (44, int) _input : (3, 2) , : (36, 0) fac$#1 → Lexical Error(code: 4) invalid character!!! , : (36, 0) f : (3, 3) ; : (37, 0)	read : (3, 4) (: (34, 0) n : (3, 5)) : (35, 0) ; : (37, 0) write : (3, 6) (: (34, 0) n : (3, 5)) : (35, 0) ; : (37, 0)
f = factorial(n); write(f);	} int factorial(int n)
f : (3, 3) = : (15, 0) factorial : (3, 7) (: (34, 0) n : (3, 5)) : (35, 0) ; : (37, 0) write : (3, 6) (: (34, 0) f : (3, 3)) : (35, 0) ; : (37, 0)	} : (33, 0) int : (44, int) factorial : (3, 7) (: (34, 0) int : (44, int) n : (3, 5)) : (35, 0)
{ if (n == 1) return 1;	else return n*factorial(n-1); }
{ : (32, 0) if : (40, if) (: (34, 0) n : (3, 5) == : (19, 0) 1 : (5, 1)) : (35, 0) return : (47, return) 1 : (5, 1) ; : (37, 0)	else : (46, else) return : (47, return) n : (3, 5) * : (12, 0) factorial : (3, 7) (: (34, 0) n : (3, 5) - : (11, 0) 1 : (5, 1)) : (35, 0) ; : (37, 0) } : (33, 0) : (7, 0)

5. 개발 후기

수업시간에 배운 스캐너와 컴파일러의 내용을 토대로 미니 C 스캐너 구현을 시작하였습니다. 처음에 교수님께서 올려주신 사이트를 참고하였고, Java로 짜여진 코드여서 전체적인 코드 분석을 하였습니다. 저는 C++이 익숙하여서 C++로 코드를 다시 구현하였고, 교수님께서 말씀하신 기능들을 추가하였습니다.

기능을 추가하는 과정을 제 생각과는 달리 오래 걸렸습니다. 원하는 결과가 나오지 않아 코드를 다시 뒤지면서 중간중간 문제가 있는 부분을 찾아나갔습니다. 개발을 하면서 크게 시간을 들였던 부분은 1) 참고사이트와 개발 요구조건을 토대로(일단 ID구분과 실수 제외) C++을 구현하고 결과가 제대로 나오는가? 2) ID를 저장하기 위해 map자료구조를 사용해서 구현을 제대로 했는가? 3) 실수를 추가하여 구현한 뒤 결과를 출력했을 때 제대로 구분지어 출력하는가? 였습니다. 특히 실수를 구현하는 부분은 계속해서 로직을 생각한 뒤 구현해 나갔지만, 제대로 실수를 구분하기까지는 시간이 조금 걸렸습니다.

비록 기한을 넘겨서 제출하였지만, 끝까지 완성하는데 의의를 두고자 열심히 과제를 수행하였습니다. 또한 기한을 맞추지 못해 교수님께 정말 죄송합니다. 이번 과제를 계기로 개발에 있어서는 시간 계획을 잘 세우는 것도 중요하다는 것을 깨달았습니다. 남은 기말고사도 최선을 다해 준비하겠습니다.