Internal_Document

프로젝트 개요

이 프로그램은 **어휘 분석기(Lexical Analyzer)** 와 **구문 분석기(Parser)** 로 구성되어 있으며, 입력 파일에 작성된 문장을 분석하여 **변수 정의, 수식 계산, 구문 오류** 등을 탐지하고 결과를 출력합니다.
또한 **C 언어의 키워드(예약어)를 사전에 전처리하여 식별자와 구분**하도록 개선되었습니다.

1. 주요 파일 구성

파일명	역할
main.cpp	프로그램 진입점. 입력 파일을 받아 어휘 분석 및 구문 분석을 수행
LexicalAnalyzer.h / .cpp	어휘 분석기(Lexical Analyzer). 문자 단위로 입력을 읽고 토큰(Token)을 생성
Parser.h / .cpp	구문 분석기(Parser). 생성된 토큰을 기반으로 문법 규칙에 따라 파싱 트리를 구성하고 오류 처리 수행

---

2. Lexical Analyzer (어휘 분석기)

LexicalAnalyzer.cpp

함수명	설명
lookup(char ch)	연산자 및 구분자(+, -, *, /, ;, (,), :=)를 인식하여 해당 토큰 반환
addChar()	현재 문자(nextChar)를 lexeme 버퍼에 추가
getChar()	입력 파일로부터 다음 문자를 읽어 문자 유형(LETTER, DIGIT, UNKNOWN) 결정

함수명	설명
getNonBlank()	공백 문자(스페이스, 탭, 줄바꿈 등)를 건너뜀
lexical()	전체 어휘 분석의 핵심 함수. 문자 유형에 따라 **식별자, 키워드, 숫자 리터럴, 연산자, 괄호, 세미콜론, EOF** 등을 인식하고 nextToken 을 반환

키워드 전처리 기능 (Keyword Preprocessing)

- 프로그램에는 **C 언어 예약어 목록(C_KEYWORDS)** 이 미리 정의되어 있습니다.

```
static const unordered_set<string> C_KEYWORDS = {
    "auto", "break", "case", "char", "const", "continue", "default", "do",
    "double", "else", "enum", "extern", "float", "for", "goto", "if",
    "inline", "int", "long", "register", "restrict", "return", "short",
    "signed", "sizeof", "static", "struct", "switch", "typedef", "union",
    "unsigned", "void", "volatile", "while", "_Bool", "_Complex", "_Imaginary"
};
```

- 어휘 분석 과정에서 식별자(IDENT)로 인식된 문자열이 이 집합에 포함되어 있으면 KEYWORD 토큰으로 분류합니다.
- 이를 통해 **C 예약어와 사용자 정의 변수명을 구분**할 수 있으며, 구문 분석기에서는 키워드가 식별자 자리에 올 경우 오류로 처리합니다.

예시:

입력: int := 10;
출력: (ERROR) "C 키워드(int)는 식별자로 사용할 수 없습니다."

LexicalAnalyzer.h

- 문자 유형(CharClass)과 토큰 코드(TokenCode)를 enum 으로 정의
- 전역 변수(lexeme, nextChar, nextToken, in_fp) 및 함수 선언 포함

3. Parser (구문 분석기)

함수명	설명
program()	프로그램의 시작 규칙 <program> → <statements> 실행. 심볼테이블 출력 포함
statements()	여러 문장을 파싱하며 세미콜론(;) 단위로 문장 구분. 중복 세미콜론 및 구문 오류 복구 처리
statement()	문장 단위 파싱. 식별자, 대입 연산자(:=), 표현식 <expression>으로 구성
expression(), **term()**, **factor()**	수식의 문법 규칙을 재귀적으로 구현 (+, -, *, / 연산 처리)
term_tail(), **factor_tail()**	다항식 또는 곱셈식의 꼬리 부분을 처리. 중복 연산자, 다른 연산자 혼용 오류 감지
freeTree()	파싱 트리의 메모리를 재귀적으로 해제
resetCounts(), printCounts()	식별자(ID), 상수(CONST), 연산자(OP) 개수를 카운트 및 출력
오류 처리	정의되지 않은 변수, 잘못된 토큰, 괄호 누락, 중복 연산자, 0 으로 나누기 등 상세 오류 메시지를 출력

---

4. Main Function

1. 명령행 인자(argv[1])로 입력 파일을 받음
2. 파일 열기 실패 시 오류 메시지 출력
3. 첫 문자 읽기(getChar()), 어휘 분석 수행(lexical())
4. program() 함수를 통해 전체 구문 분석 수행
5. 파싱 트리 해제(freeTree()), 파일 닫기

5. 프로그램 동작 흐름 요약

입력 파일 열기
    ↓
어휘 분석기(LexicalAnalyzer)
    ↓
    ├── 키워드 전처리 (C 예약어 인식)
    ├── 각 문자 → 토큰으로 변환
    ↓
구문 분석기(Parser)
    └── 토큰들을 문법 규칙에 따라 분석
    ↓
오류 및 결과 메시지 출력
    ↓
심볼 테이블 및 ID/CONST/OP 카운트 결과 표시

---

6. 주요 특징 및 개선점

C 언어 키워드 전처리 및 식별자 구분 기능 추가
실수/정수 리터럴 인식 및 오류 복구
중복 세미콜론, 잘못된 대입문, 미닫힌 괄호 등 세부 오류 처리
파싱 트리 구조화 및 결과 시각적 출력
Result ==> 형태의 심볼 테이블 결과 요약 제공