폰트, 텍스트, 그래픽, 그래픽 디자인이(가) 표시된 사진

자동 생성된 설명

Internal Document

시계, 폰트, 그래픽, 상징이(가) 표시된 사진

자동 생성된 설명

* 과목명: 프로그래밍언어론
* 담당교수님: 김진성 교수님
* 학과: 소프트웨어학부
* 보고서작성자: 20195815 이석진

20220702 서정민

# 작성한 소스코드에 대한 설명

1. LexicalAnalyzer.h

LexicalAnalyzer 헤더 파일. 입력 데이터를 읽어서 token을 생성하고 분류하는데 필요한 상수, 전역변수, 함수 선언

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

charClass: 현재 문자 유형을 나타내는 변수

lexeme: 현재 어휘 분석 중인 문자열을 저장할 배열

nextChar: 다음에 분석할 문자를 저장

lexLen: 현재 lexeme의 길이

token: 현재 lexeme의 토큰 값

nextToken: 다음에 읽을 어휘 항목의 토큰 값

in\_fp: 입력 파일에 대한 파일 포인터

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

//문자 유형 상수

LETTER, DIGIT, UNKNOWN 세가지 문자 유형 상수를 정의하여 문자 판별에 사용

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

//토큰 코드 상수 – 각 토큰 유형을 구분

INT\_LIT: 정수 리터럴

IDENT: 식별자

ASSIGN\_OP: 배정 연산자

ADD\_OP, SUB\_OP, MULT\_OP, DIV\_OP: 사칙연산 연산자

LEFT\_PAREN, RIGHT\_PAREN: 괄호

SEMI\_COLON: 세미콜론

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

// 함수 선언

addChar(): lexeme에 현재 문자를 추가함

getChar(): 다음 문자를 읽고 charClass를 업데이트

getNonBlank(): 공백을 건너뜀

lexical(): 어휘 분석 수행, 다음 어휘 항목을 식별하고 토큰을 반환

lookup(char ch): 주어진 문자를 검사하고 해당 토큰을 반환

2. LexicalAnalyzer.cpp

입력된 소스를 읽어서 token을 분석하고, token 유형을 반환하는 LexicalAnalyzer.cpp

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

입력된 특수문자를 확인하고, 해당하는 토큰 코드를 반환. 대입 연산자(:=)의 경우 ‘:’ 를 먼저 읽은 후 다음 문자를 검사. addChar를 호출하여 해당 문자를 lexeme에 추가하고, 적절한 토큰 값을 설정

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

현재 읽고 있는 문자를 lexeme배열에 추가하고, 길이가 너무 길다면 오류 메시지 출력

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

입력 파일에서 문자를 하나씩 읽고, 문자 유형을 LETTER, DIGIT, UNKNOWN으로 분류, EOF를 만나면 CharClass를 EOF으로 설정

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

공백을 건너뛰고 다음 문자를 찾으면, EOF를 만나면 loop종료

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

getNonBlank을 호출하여 공백을 무시하며, 현재 문자 유형(CharClass)에 따라서 처리. LETTER의 경우 알파벳으로 시작하는 식별자를 파싱하고, DIGIT의 경우 숫자로 이루어진 정수 리터럴을 파싱하고, UNKNOWN의 경우 괄호나 연산자 등의 특수 문자를 처리하고, 파일의 끝에 도달한 경우 EOF 토큰을 반환.

3. SyntaxAnalyzer.h

SyntaxAnalyzer 에서 사용되는 구조체와 함수 선언을 정의하는 헤더 파일

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

OK, WARNING, ERROR 등의 상수는 구문 분석 결과를 나타냄

idCount, constCount, opCount는 각각 식별자, 상수, 연산자의 개수를 추적하는 변수

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

심볼 테이블에 저장되는 각 심볼의 이름, 값, 초기화 여부를 저장

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

프로그램에서 사용하는 변수와 관련 정보를 저장. (최대 100개). createSymbol, findSymbol, updateSymbol, print 메서드.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

파싱트리의 각 노드를 나타내는 구조체. 토큰 유형, 값, 값 정의 여부, 상태, 상태 메시지, 자식노드의 정보를 포함

addChild 메서드는 자식 노드를 추가하며 상태와 메시지를 업데이트

setValue 메서드는 노드의 값을 설정

setStatus 메서드는 노드의 상태와 메시지를 설정

printMessage 메서드는 노드의 메시지를 출력.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

freeTree - 트리의 루트부터 메모리를 해제

program, statements, statement: 프로그램 구조와 문(statement) 수준에서 파싱을 담당

expression, term, factor, term\_tail, factor\_tail: 표현식과 그 하위 구성 요소를 파싱

각 함수에서 Non=Terminal과 Terminal 처리

각 tail 함수의 argument로서 int leftValue가 존재

이를 통해 term이나 factor 의 값을 leftValue로 전달하고, 연산자가 있을 경우 계산 한 뒤의 값을 node에 추가하여 return

없을 경우 leftValue를 그대로 return

각 구문 분석 노드는 status가 ERROR일 경우 값을 정의하지 않음(value = 0, isDefined = false)

4. SyntaxAnalyzer.cpp

program 함수부터 아래 recursive하게 내려가며 (top-down) 끝에 도달하면 값과 오류 발생 여부를 검증

오류의 경우 가장 먼저 발생한 오류 (leftmost 이므로 가장 왼쪽 아래 노드의 오류)를 먼저 출력하며, status가 OK인 경우에는 WARNING, ERROR로 덧씌워지고, OK, WARNING인 경우에는 ERROR로 덧씌움

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

resetCounts() – idCount, constCount, opCount를 0으로 초기화

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

printCounts() - 현재까지 추적된 식별자, 상수, 연산자 사용 개수 출력

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

freeTree – 재귀적으로 모든 트리의 메모리를 해제

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<program> → <statements>

시작점 노드를 만듦

모든 Non-Terminal 노드는 ParseTreeNode(0)으로 초기화

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<statements> → <statement> { ; <statement> }

세미콜론이 없을 때 까지 statementNode 생성

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<statement> → <ident> <assignment\_op> <expression>

식별자 (정수 리터럴), 대입 연산자(:=) 존재 여부, 중복 여부 확인

경우에 따라 오류 상태 변경

statement 함수가 끝날 때 마다 printCounts 호출하여 ID: x CONST: x OP: x 형식으로 각 counts 출력

node->printMessage 호출하여 [(Error) “배정 연산자(:=)가 필요합니다.] 형태로 현재 노드의 status message 출력

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

<expression> → <term> <term\_tail>

먼저 term 노드 생성, term 노드의 값을 leftValue로 넣어 term\_tail 노드 생성

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

<term> → <factor> <factor\_tail>

먼저 factor 노드 생성, factor 노드의 값을 leftValue로 넣어 factor\_tail 노드 생성

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<term\_tail> → <add\_op> <term> <term\_tail> | ε

BNF에 따라 nextToken이 + 또는 - 일 경우 add\_op, term, term\_tail 노드 생성

아니라면 leftValue를 그대로 할당하여 반환

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<factor> → <left\_paren> <expression> <right\_paren> | <ident> | <const>

BNF에 따라 nextToken이 ( 일 경우 left\_paren, expression, right\_paren 노드 생성

아닐 경우 식별자인지 정수 리터럴인지 확인하여 생성

그 외에는 undefined 상태로 현재 노드 반환

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<factor\_tail> → <mult\_op> <factor> <factor\_tail> | ε

BNF에 따라 nextToken이 \* 또는 / 일 경우 mult\_op, factor, factor\_tail 노드 생성

아니라면 leftValue를 그대로 할당하여 반환

5. main.cpp

파일을 열고, 파일이 없다면 ‘파일 이름이 필요합니다.’ 메시지 출력 후 프로그램 종료.

어휘 분석과 구문 분석을 수행하고 결과를 출력한 뒤에 파일을 닫음.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명