**2023.2   
Principle of Programming Language  
Programming Assignment #1**

**Internal Documentation**

|  |  |
| --- | --- |
| 20184256 | 박성민 |
| 20226041 | 김규리 |
|  |  |
|  |  |

**본 Parser에서 처리한 오류와 경고**

작성한 Parser에서 처리한 경고는 아래와 같다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**- 유효하지 않은 연산자 토큰 위치** : 가장 앞에 있는 연산자만 남기고 생략한다.  
ex) abc := 3 ++\*\*/- 5;  
-> abc := 3 + 5;

**- UNKNOWN토큰의 입력 :** 적절하지 않은 input인 UNKNOWN 토큰을 생략한다.  
ex) abc := $bcd + 5;  
-> abc := bcd + 5;

**- 좌 괄호보다 우 괄호가 부족함** : 부족한 좌괄호의 개수만큼 우괄호를 Statement 끝에 삽입한다.  
ex) abc := (abc + (bcd + (cde + def;  
-> abc := (abc + (bcd + (cde + edf)));

**- 대입연산자에 등호 없음** : 대입연산자의 등호 자리에 등호를 채워준다.  
ex) abc : 3 + 5;  
-> abc := 3 + 5;

**- 대입연산자에 콜론 없음** : 대입연산자의 콜론 자리에 콜론을 채워준다.  
ex) abc = 3 + 5;  
-> abc := 3 + 5;

**- 세미콜론이 중복으로 입력됨** : 연속으로 입력된 세미콜론을 무시한다.  
ex) abc := 3 + 5;;;;; abc := 5;  
-> abc := 3 + 5; abc := 5;

**- 마지막 statement가 세미콜론으로 끝남** : 필요 없는 세미콜론으로 종료되었음을 알린다.

본 Parser에서 처리한 에러는 아래와 같다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**- 대입연산자 이전에 IDENT가 입력되지 않음**  
ex) := 3 + 5;

**- 잘못된 STATEMENT 구조가 입력됨**  
ex) abc := 3 5 7;

**- 아직 정의되지 않은 IDENTIFIER에 대한 값 접근**  
ex) a := 3; a := b + 5;

**- 0으로 값 나누기를 시도함**  
ex) a := 3 / 0;

**- 우 괄호의 개수가 좌 괄호의 개수보다 많음**  
ex) a := a + b);

**- 연산자에 argument가 부족함**  
ex) a := a + ;

**- 토큰 처리 종료 이후에도 스트림에 토큰이 남아있음**  
 \* 본 에러에 해당하는 케이스는 모두 상기한 에러로 대체되었음

**- 그 외 알 수 없는 에러**  
\* 이외에 발생한 기타 에러는 UNKNOWN\_ERROR로 분류하였음.

**본 Parser에서의 오류, 경고 처리를 위한 설계**

Parser에서 에러나 경고가 발생한 경우에도 적절히 조치를 취하고 그 다음 statement를 처리해주기 위해서 다양한 설계를 적용하였고, 그 중에서 Symbol Table에서 유효하지 않은 변수 값을 가져오려고 하는 경우, 에러가 발생한 Statement에서 값을 대입하는 경우에 대해 처리해주기 위해 OptionalData이라는 Type을 선언하고, 이를 상속받는 타입인 OptionalInt, OptionalDouble Type을 정의하였다. 그리고 각Type에 대한 +, -, \*, / 연산자를 오버로딩하여 Parser에 활용해주었다.

OptionalData는 Nullable, Unknownable Type으로, 아직 선언되지 않은 데이터는 isNull이 참이고, Parsing 과정에서 오류가 발생해 유효하지 않은 값이 대입된 경우에는 isUnknown이 참이 된다. 이후 연산에서 이 flag를 활용해 symbol table의 값을 적절하게 채워넣을 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Optional Dat를 상속받는 Optional Int Type은 아래와 같이 정의되며, 아래에 여러 연산자들을 오버로드 하여 flag를 활용한 연산들을 구현하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

본 프로그램에서는 연산 결과로 정수형(Integer)만 취급하고 있지만, a / b 연산을 a \* (1 / b) 의 형식으로 처리하도록 설계하였기 때문에, Optional Double 타입을 구현하였고, 이를 OptionalInt로 변환할 수 있게 내부 함수도 만들어주었다. Optional Double 타입에 맞게 연산자들도 오버로드하여 해당 타입에 대해 적절히 연산을 수행할 수 있게 구현하였다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

그리고, 두 타입간의 형변환을 통해 쉽게 값을 연산할 수 있도록 Convert 함수도 작성해 적용하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**전반적인 프로그램의 동작 절차**

작성한 분석 프로그램은 main 함수 호출 인자에서 filename을 전달받아 입력 파일 스트림을 생성하면서 시작된다. 만약 파일을 열 수 없다면 에러메시지를 출력하고 프로그램을 종료한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그 후, LexicalAnalyzer 클래스의 객체를 생성하고 멤버 변수 analyzeInputFile 함수에 파일스트림을 인자로 전달해 토큰에 대한 분석을 진행한다. 토큰 분석 결과와 그 과정에서 생성한 symbolTable을 인자로 받아 Parser 객체를 생성해준 뒤, 입력에 대한 구문 분석을 진행한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

모든 처리가 완료되면 파일스트림을 닫고 프로그램을 종료한다.

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

**Lexical Analyzer의 동작**

설계한 Lexical Analyzer 클래스의 인터페이스는 아래와 같다. 주어진 파일 스트림에 대한 토큰 분류 결과를 담는 \_lexResult, 그 과정에서 생성된 심볼 테이블을 담는 \_symbolTable, 입력 문자열에 대해 토큰을 분류하는 analyzeString 함수와, 로직 처리를 시작하고 결과를 반환하는 함수들로 구성되어 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

주어진 파일의 코드를 토큰으로 분류하기 위해, 아래처럼 토큰을 enum 타입으로 정의하였다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

아래는 입력된 문자열에 대해 특수 기호가 아닌 숫자, 알파벳, \_이 연속으로 입력된 경우, 이를 IDENT, CONST, UNKNOWN으로 분류해주는 함수이다. 숫자로 시작하지 않으면 IDENT로, 숫자로 시작할 때 모든 문자가 숫자이면 CONST로, 그 외에는 UNKNOWN으로 분류한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

아래는 주어진 파일스트림을 읽으면서 토큰을 분류하는 함수이다. 파일에 남은 문자가 없을 때 까지 한 글자씩 반복하며 진행한다. line 변수에는 현재 읽은 문자가 혼자서 토큰이 될 수 없는 경우(IDENT, CONST)를 저장하는데에 사용된다.

받아온 글자가 공백문자인 경우에는 line에 있는 문자열의 토큰을 분류하고 결과에 추가한다.

받아온 글자가 알파벳, 숫자, \_ 인 경우에는 line에 추가해 다음 문자와 함께 처리한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

나머지 기호들에 대해서는 앞서 입력받던 line에 대해 처리를 해주고 나서 입력에 해당하는 토큰으로 분류해 결과에 추가해준다. 주어진 문법에 없는 기호의 경우 UNKNOWN으로 처리하였다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

모든 입력을 다 처리한 뒤에는 마지막으로 line에 남아있는 문자열을 처리한 뒤, END\_OF\_FILE 토큰을 결과에 넣고 분석 결과를 저장한다..

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Parser의 동작**

Parser에서 Program함수를 실행하면서 파싱을 시작한다. Program 함수에서는 값을 초기화하고 statements 함수를 호출한 뒤 파싱이 완료된 뒤 Symbol Table의 데이터를 출력하면서 함수를 종료한다..

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

­­­

statements 함수에서는 statement 함수를 호출해 하나의 statement를 처리한다. 이후 세미콜론이 나올 때 초기화를 진행하고 statement 함수를 다시 호출해 처리한다. 중간에 세미콜론이 반복해서 등장하는 경우에는, 이에 대해 로그를 띄우고 처리한다. 모든 statement에 대한 처리가 끝나면 END\_OF\_FILE 토큰을 확인하고 처리를 종료한다.

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

statement 함수에서는 세미콜론으로 구분되는 하나의 문장 단위를 파싱하는 함수이다. statment함수 내에서, 좌항 IDENT와 :=, 이후 우항의 EXPRESSION 까지 처리를 한다. 이 과정에서 오류나 경고가 발생하는 경우 이를 처리해준다. IDENT는 함수로 호출되어, subprogram으로 처리된다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

대입연산자 :=는 콜론 : 과 등호 = 로 분리하여 입력받아 처리한다. 이 과정에서 예상치 못한 입력이 들어오는 경우에는 경고를 넣으며 적절하게 처리해준다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

대입연산자 이후에 등장하는 토큰들인 EXPRESSION은 expression 함수를 호출해 subprogram에서 처리를 한다. 이 과정에서 발생한 문제에 대해서 expression 함수가 종료된 뒤에 에러와 경고를 기록한다.

위 과정이 종료된 뒤에, 만약 좌항 IDENT의 입력이 정상적으로 들어온 경우에는 정상적으로 계산된 expression에서의 결과를 대입해준다.

**텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명**

expression 함수는 아래처럼 term과 term\_tail 함수를 호출한 뒤, 하위 계산 결과를 더해 반환한다. term\_tail에 아무것도 없는 경우에는 0을 반환해 더해주는 방식으로 처리하였다.



term함수에서는 factor 함수와 factor\_tail 함수의 결과를 곱한 값을 반환한다. 나눗셈의 계산을 수행하기 위해서 factor의 값과 factor\_tail의 값을 Double 타입으로 변환해 곱셈 연산을 처리해준 뒤, 이를 Int 타입으로 다시 캐스팅해 반환한다.

예를 들어, 16 / 8 의 연산을 수행해야 하는 경우, 16.0 \* (0.125) 을 연산한 결과인 2.0을 Int로 형변환한 2를 반환하게 된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

term\_tail 함수에서는 ADD\_OP 연산자가 +인지 -인지에 따라 하위의 term과 term\_tail subprogram의 연산 결과를 부호에 맞게 변환하여 반환한다. 변환된 값은 해당 함수를 호출한 지점에 반환되어 덧셈을 통해 - 또는 + 연산을 정상적으로 수행하도록 설계하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

factor 함수에서는 처리될 순서의 토큰을 IDENT, CONST, ( EXPRESSION ) 3가지로 분류한다.

( EXPRESSION ) 의 경우, 좌 괄호의 개수를 동일하게 우 괄호를 배치할 수 있도록 처리하기 위해 좌괄호의 개수를 카운트하고, EXPRESSION을 처리한 뒤 정상적으로 우 괄호 토큰이 들어오는지 확인해준다. 만약 우 괄호의 개수가 부족한 경우에는 연산 결과에 영향을 미치지 않기 위해 Statement의 가장 오른쪽에 우 괄호를 부족한 만큼 추가하는 방식으로 처리하였다. 우 괄호의 개수가 좌 괄호보다 많은 경우에는 에러로 처리하였다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

IDENT, CONST, ( EXPRESSION ) 로 분류될 수 없는 토큰이 들어온 경우인 세미콜론, EOF, 우 괄호, 연산자 또는 그 외의 토큰이 들어온 경우에는 각각의 경우에 대해 적합하게 경고 또는 에러를 기록하고 처리한다. 이 경우, 신뢰할 수 없는 연산 값이라 판단하여, statement subprogram에서 대입 연산 시 IDENT에 UNKNOWN으로 값이 대입되도록 설계하였다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

factor\_tail 함수의 경우 MULT\_OP에 대해 처리를 한다. 연산자가 \* 인지 / 인지에 따라 하위 factor와 factor\_tail subprogram의 연산 결과에 대해 그대로 또는 역수로 반환한다. 변환된 값은 해당 함수를 호출한 지점에 반환되어 곱셈을 통해 \* 또는 / 연산을 정상적으로 수행하도록 설계하였다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

ident 함수는 대입연산자의 왼쪽에 위치한 IDENT TOKEN에 대해 처리하는 함수이다.

각 Statement에 대한 ident 토큰의 개수를 카운트한다. IDENT 토큰에서 발생한 문제에 대해서는 Statement에서 처리하도록 설계하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

ident\_val 함수는 대입연산자의 오른쪽에 위치한 IDENT TOKEN에 대해 처리하는 함수이다.

IDENT 토큰을 읽은 경우에 Symbol Table에서 저장된 값을 찾아 반환하는 역할을 수행한다. 만약 아직 선언되지 않은 변수의 경우 null로 되어있어 에러를 기록하고, 대입에서 오류가 있었던 값은 unknown으로 저장되어있고, unknown 값을 반환해 상위 연산의 결과가 연쇄적으로 unknown으로 처리되도록 설계하였다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

add\_op 함수는 입력받은 토큰이 +인지 -인지를 검사한 뒤 반환한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

mult\_op 함수는 입력받은 토큰이 /인지 \*인지를 검사한 뒤 반환한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

const\_val 함수는 입력받은 토큰의 정수 값을 반환한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

printWarningAndErrorList함수에서는 각 Statement를 처리하면서 기록해둔 에러와 경고를 출력한다.

경고로 처리한 케이스는 다음과 같다.

- 유효하지 않은 연산자 토큰 위치  
- UNKNOWN토큰의 입력  
- 좌 괄호보다 우 괄호가 부족함  
- 대입연산자에 등호 없음  
- 대입연산자에 콜론 없음  
- 세미콜론이 중복으로 입력됨  
- 마지막 statement가 세미콜론으로 끝남

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

에러로 처리한 케이스는 다음과 같다.

- 대입연산자 이전에 IDENT가 입력되지 않음  
- 잘못된 STATEMENT 구조가 입력됨  
- 아직 정의되지 않은 IDENTIFIER에 대한 값 접근  
- 토큰 처리 종료 이후에도 스트림에 토큰이 남아있음  
- 0으로 값 나누기를 시도함  
- 우 괄호의 개수가 좌 괄호의 개수보다 많음  
- 연산자에 argument가 부족함  
- 그 외 알 수 없는 에러

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명