Principles of Programming Languages
Fall, 2023

Programming Assignment #1

< 문 제 >

※ 파일을 통해 입력된 프로그램이 하나의 lookahead symbol 을 활용하는 아래의 LL(1) 문법을 따르는지 여부를 판단하는 파서(Parser)를 Recursive Descent Parsing 기법을 이용하여 작성하고, 파싱된 결과를 이용하여 입력된 프로그램의 결과를 출력하시오.

< 문 법 >

<program> → <statements>

<statements> → <statement> | <statement><semi_colon><statements>

<statement> → <ident><assignment_op><expression>

<expression> → <term><term_tail>

<term_tail> → <add_op><term><term_tail> | ε

<term> → <factor> <factor_tail>

<factor_tail> → <mult_op><factor><factor_tail> | ε

<factor> → <left_paren><expression><right_paren> | <ident> | <const>

<const> → any decimal numbers

<ident> → any names conforming to C identifier rules

<assignment_op> → :=

<semi_colon> → ;

<add_operator> → + | -

<mult_operator> → * | /

<left_paren> → (

<right_paren> →)

< 세부 사항 >

※ 개발 조건

- C / Java / Python 으로 개발된 Command Line Application 만 허용
- 자신의 학번으로 명명된 폴더 내에 작성된 모든 소스 코드 저장
- Java 의 경우, 폴더 최상단에 Main.jar 를 Runnable Jar 파일로 생성한 다음, 'java –jar Main.jar'를 실행
- Python 의 경우, main.py 을 진입점(entry point)으로 하고 'python main.py'를 실행

※ 입력: 임의의 이름이 부여된 텍스트 파일 (Command Line 파라미터로 파일명이 주어짐. (예: "python main.py eval1.txt"), 옵션으로 "-v"를 제공할 수 있음. (예: "java –jar Main.jar –v eval1.txt"))

※ 출력-(a): 주어진 문법에 따라 입력파일에 저장되어 있는 프로그램을 분석한다. 파싱(parsing)되는 과정을 <처리 예>와 같이 출력하고, 문법 오류 없이 파싱된 경우, 파싱 트리를 구축하고, 이 트리를 이용하여 <ident>들의 최종값을 출력한다.
=> 출력 형식:
"프로그램에서 읽은 라인"
"ID: {개수}; CONST: {개수}; OP: {개수};"
"파싱 결과 (OK), (WARNING), (ERROR)"
"Result ==> {변수 1}: {최종값}; {변수 2}: {최종값}; {변수 3}: {결과값};"

※ 출력-(b): 옵션으로 "-v"가 주어진 경우, 주어진 문법에 따라 입력파일에 저장되어 있는 프로그램을 분석하되 **출력-(a)**의 파싱되는 과정은 출력하지 않는다. 대신, 아래 처리 조건의 next_token 변수가 변경될 때마다 그 값을 출력한다.
=> 출력 형식:
"token1"
"token2"
"token3" "..."

※ 처리 조건

- 각 문장들이 파싱된 이후, 입력된 문장과 그 문장에 포함된 식별자(IDENT)와 숫자(CONST), 그리고 연산자(OP)의 개수를 출력한다.
- 파싱된 문장이 문법에 적합하면 '<Yes>', 적합하지 않으면 적절한 에러(Error) 메시지나 경고(Warning) 메시지를 출력한다. 위에서 주어진 문법에 의거 오류가 발견된 경우, 오류를 가능한 한 복구한 다음, 파싱을 계속한다. 예를 들어, x = a + + b 일 경우, "+" 연산자가 한 개가 더 존재하므로 "+" 기호를 제거한 다음 적절한 '경고(Warning) 메시지'를 출력한 후, 파싱을 계속한다. 오류 복구가 불가능한 경우는 '에러(Error) 메시지'를 출력하고 파싱을 계속하되, 이 경우 해당 식별자(<IDENT>)의 값은 'Unknown'으로 결정된다.
    - 문장이 포함하고 있는 모든 오류에 대해 경고 또는 에러 메시지를 출력해야 한다.
    - 가능한 한 오류를 복구해야 하고, 불가능한 경우만 에러 메시지를 출력한다.
    - 에러나 오류 메시지 내용은 각자 적절히 정의한다.
    - 처리된 오류 각각에 대해 추가 점수가 부여된다.
- 프로그램에 속한 일부 문장이 문법에 적합하지 않더라도 오류 복구를 통해 프로그램이 끝까지 전부 파싱되어야 한다.
- 파싱 트리 생성 후, 모든 <ident> 값이 출력되어야 한다. 단, <ident>의 값이 정의되지 않은 경우, "Unknown"으로 표시한다.
- <ident>의 현재 값을 저장하기 위해 심볼 테이블(symbol table)을 구축해야 한다.
- 입력 스트림에서 ASCII 코드값이 32 이하인 것은 모두 white-space 로 간주되며, white-space 는 각 token 을 구별하는 용도 이외에는 모두 무시된다.
- 어휘분석기(lexical analyzer)의 소스 코드는 정수 변수 next_token, 문자열 변수 token_string, 함수 lexical()을 포함하여야 한다. 함수 lexical()은 입력 스트림을 분석하여 하나의 lexeme 을 찾아낸 뒤, 그것의 token type 을 next_token 에 저장하고, lexeme 문자열을 token_string 에 저장하는 함수이다.
- 기타 구현 시 요구되는 세부 사항은 직접 결정하고, Internal 및 External Document 에 기술한다.

<처리 예>

※ 입력 #1

```
operand1 := 3 ;
operand2 := operand1 + 2 ;
target := operand1 + operand2 * 3
```

※ 출력 #1

```
operand1 := 3;
ID: 1; CONST: 1; OP: 0;
(OK)
operand2 := operand1 + 2;
ID: 2; CONST: 1; OP: 1;
(OK)
target := operand1 + operand2 * 3
ID: 3; CONST: 1; OP: 2;
(OK)
Result ==> operand1: 3; operand2: 5; target: 18;
```

※ 입력 **#2**

```
operand2 := operand1 + 2 ;
target := operand1 + operand2 * 3
```

※ 출력 **#2**

```
operand2 := operand1 + 2;
ID: 2; CONST: 1; OP: 1;
(Error) “정의되지 않은 변수(operand1)가 참조됨”
target := operand1 + operand2 * 3
ID: 3; CONST: 1; OP: 2
(OK)
Result ==> operand1: Unknown; operand2: Unknown; target: Unknown
```

※ 입력 **#3**

```
operand1 := 1;
operand2 := (operand1 * 3) + 2 ; target := operand1 + operand2 * 3
```

※ 출력 **#3**

```
operand1 := 1;
ID: 1; CONST: 1; OP: 0
(OK)
operand2 := (operand1 * 3) + 2;
ID: 2; CONST: 2; OP:2;
(OK)
target := operand1 + operand2 * 3;
ID: 3; CONST: 1; OP: 2;
(OK)
Result ==> operand1: 1; operand2: 5; target: 16
```

※ 입력 **#4**

```
operand1 := 3 ;
operand2 := operand1 + + 2 ; target := operand1 + operand2 * 3
```

※ 출력 **#4**

```
operand1 := 3;
ID: 1; CONST: 1; OP: 0;
(OK)
operand2 := operand1 + 2;
ID: 2; CONST: 1; OP: 1;
(Warning) "중복 연산자(+) 제거"
target := operand1 + operand2 * 3
ID: 3; CONST: 1; OP: 2;
(OK)
Result ==> operand1: 3; operand2: 5; target: 18
```

<제출 관련 사항>

- 제출물
  - Internal/External document
  - 프로그램 소스 코드 및 실행 파일
- 제출 방법
  - 모든 제출물을 압축하여 학번_이름.zip 형식으로 e-class 과제함에 제출
- 제출 마감 일시
  - 2023 년 11 월 20 일 23:59:59
  - Late penalty 는 course introduction 을 따름