프로그래밍 언어

(나)

20181259

조수민

목표

세가지 언어 (C, Java, Python)으로 주어진 bnf에 맞게 Recursive Descent Parser를 구현한다.

주어진 bnf는 다음과 같다

<program> → {<statement>}
<statement> → <var> = <expr> ; | print <var> ;
<expr> → <bexpr> | <aexpr>
<bexpr> → <number> <relop> <number>
<relop> → == | != | < | > | <= | >=
<aexpr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
<factor> → <number>
<number> → <dec> {<dec>}
<dec> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<var> → x | y | z

설계

먼저 3가지 방식을 생각해봤습니다.

1. 전역변수 사용을 최소화 하고 모든 변수를 함수끼리 인자로 주고받음
2. 모든 함수를 void 형식으로 유지하고 각자 전역변수만을 활용함
3. 각 함수가 bnf 형식에 맞는 반환값을 가지고 있음

코드의 효율성을 생각했을 때는 1번이 베스트라고 생각했습니다. 전역변수 및 static을 최소한으로 쓴다면(쓰지 않을 메모리까지 미리 예약해두지 않는다는 면

에서) 메모리 효율이 좋아지기 때문입니다.

최종적으로는 2번에 가까운 형태를 선택하게 되었습니다. 왜냐하면 옳바른 에러 메세지를 출력하는 것 또한 파서의 역할이라고 생각했기 때문입니다. 로컬 변수만으로는 모든 상태 변화를 기록하는 것은 어렵고 특히 재귀구조에서는 더욱 까다롭습니다. 전역변수를 활용하면 히스토리를 관리하기 용이합니다. 이에 맞추어 어느 단계에서 오류가 발생했는지 에러를 추적하고 싶었고, 시간관계상 디테일 하게 구현하진 못했지만, 어느정도 오류가 난 파싱 단계에 맞는 에러 메세지를 출력하게 만들었습니다.

구현

C 언어를 베이스로 구현했기 때문에 c 소스코드로 작성

- 타입
-

```c
typedef enum {
    token_NONE,
    token_NUMBER, token_VAR, token_AEXPROP, token_TERMOP,
    token_ASSIGN, token_RELOP, token_PRINT, token_TERMINATE,
    token_ERROR, token_SEMI
} TokenType;

typedef enum {
    type,NONE, type_NUMBER, type_BOOLEAN, type_ERROR
} VariableType;
```

TokenType은 terminal-symbol과 Error로 나눔

VariableType 은 초기, Number, Boolean, Error

- 변수 구조체

```c
typedef struct _Variable{
    char name[token_LEN];
    double value;
    int isTrue;
    VariableType type;
    char error_message[100];
} Variable;

typedef struct {
    TokenType type;
    char value[token_LEN];
} Token;
```

Variable – statement에서 "=" 기호로 유추되는 변수를 저장하는 구조체, ERROR_MESSAGE 또한 변수로 가지고 있음

Ex) input = "x = 2+3; y = 2==3;"

Variable[0]

Name = "x"

Value = "5"

isTrue = 0 (하지만 타입을 보고 사용하지 않음)

Type = type_NUMBER

Variable[1]

Name = "y"

Value = ""

isTrue = 1
Type = type_BOOLEAN

Token – input을 token 단위로 구분하여 저장하는 구조체
Ex) input = "x = 2+3; print x"
Type = token_VAR value = "x"
Type = token_ASSIGN value = "="
TYPE = token_NUMBER value = 2
.
.
TYPE = token_PRINT value = ""
TYPE = token_VAR value = "x"

간단한 로직 설명

<expr>까지는 void로 구현했고 그보다 밑에서 유추되는 요소들은 int형식 (Boolean 포함) 으로 값을 반환해줍니다. 각 단계에서는 현재 유추되고 있는 variable에 대해서 접근합니다. 현재 variable의 인덱스는 var_pos 전역 변수 입니다. 각 단계에서 에러가 나면 variables[var_pos]의 type을 type_ERROR로 세팅하면서 해당 단계의 에러 메세지를 variable변수에 넣어줍니다. 재귀를 더 깊게 들어가도 먼저 현재 변수의 ERROR타입을 검사하고 ERROR면은 derivation을 진행하지 않습니다. 때문에 처음 에러가 난 단계의 에러 메세지를 담고 있고 이를 출력해줍니다.

실행 결과

C

```
>> x = 2 + 3; print x;
x = 5
>> x = 2 + 3 * 3; print x;
x = 11
>> x = 2; x = 3; print x;
x = 3
>> x = 2 + 3 * 2 + 3 * 2; print x;
x = 14
>> x = 2 == 3; print x;
x = FALSE
>> x = 3 >= 3; print x;
x = TRUE
>> x > 3; print x;
Syntax Error!!
        <statement> → <var> = <expr> ; | print <var> ;
 Expected = but current Token is >
>> x = 3 > 3; print x;
x = FALSE
```

```
Syntax Error!!
        <statement> → <var> = <expr> ; | print <var> ;
 Expected = but current Token is >
>> x = 3 > 3; print x;
x = FALSE
>> x = 2 ==
Syntax Error!!
        <bexpr> → <number> <relop> <number> Expected <number> but current Token is empty string
>> x = 2 === ;
Syntax Error!!
        <bexpr> → <number> <relop> <number> Expected <number> but current Token is =
>> x =
Syntax Error!!
         Unexpected Token
>> print x
Syntax Error!!
        <statement> → <var> = <expr> ; | print <var>; Expected ; but current Token is
>> print x;
print variable not found it's not syntax error but it's runtime error
>> print x; x = 2; print x;
print variable not found it's not syntax error but it's runtime error
x = 2
```

Java

```
❯ ls
Main.java Makefile
 sumjo 😎  Java_Assignment
❯ make
mkdir -p ./bin
javac -d ./bin ./Main.java
java -cp ./bin Java_Assignment.Main
>> x = 2; print x;
x = 2
>> x = 3; x = 4; x = 5; print x;
x = 5
>> x = 2 + 2 * 2; print x;
x = 6
>> x = 2 >= 3; print x;
x = FALSE
>> y = 2 ==== 3;
Syntax error!!
        <bexpr> → <number> <relop> <number> Expected <number> but current Token is ==
>> x ( 12 + 3 ) ;
Syntax error!!
Syntax error!!
         Unexpected Token
>> y = 2 ** 3; print y;
Syntax error!!
        <factor> → <number>
 Expected <number> but current Token is *
>> x += 1;
Syntax error!!
        <statement> → <var> = <expr> ; | print <var> ;
 Expected = but current Token is +
>>
```

```
>> x += 1;
Syntax error!!
        <statement> → <var> = <expr> ; | print <var> ;
 Expected = but current Token is +
>> x = 1; print x;y = 2; print y; x = 3; print x; y = 4; print y;
x = 1
y = 2
x = 3
y = 4
>> x = 2 == 3; print x
Syntax error!!
        <statement> → <var> = <expr> ; | print <var>; Expected ; but current Token is empty string
>> ;
Syntax error!!
        <statement> → <var> = <expr> ; | print <var> ;
 Expected <var> or print but current Token is ;
>> qwer
Syntax error!!
```

Python

파이썬은 숙련도가 높지 않기 때문에 Syntax error!!만 출력

```
/usr/local/bin/python3 /Users/josumin/Desktop/pl/Python_Assignment/Rd.py
>> x = 2 + 3; print x;
x = 5

>> x = 2 * 3 + 2; print x;
x = 8

>> y = 3 + 5; print y; x = 2
Syntax error!!
>> y = 5 == 4; print y; print x;
print variable not found it's not syntax error but it's runtime error
y = FALSE

>> x = 2 + 3 + 4 * 0; print x; y = 2 < 5; print y;
x = 5
y = TRUE

>> x = 2 >=== 3; print x;
Syntax error!!
>> x = 2 >==== 3; print x
Syntax error!!
```

```
/usr/local/bin/python3 /Users/josumin/Desktop/pl/Python_Assignment/Rd.py
>> ^[[A
Invalid input. Please try again.
>> ^[[D^[[D^[[D^[[C^[[C
Invalid input. Please try again.
>> a
Syntax error!!
>> x
Syntax error!!
>> b
Syntax error!!
>> x == 2 + 3;
Syntax error!!
>> terminate
```