프로그래밍 언어

(나)

20181259

조수민

목표

다음의 EBNF로 문법이 정의되는 언어를 위한 Parser를 구현하시오.
[C, C++, Java, Python 중에서 하나의 프로그래밍 언어를 선택하여 구현하시오.]주어진 bnf는 다음과 같다

<program> → {<declaration>} {<statement>}
<declaration> → <type> <var> ;
<statement> → <var> = <aexpr> ; | print <bexpr> ; | print <aexpr> ; |
do ‘ { ’ {<statement>} ‘ } ’ while (<bexpr>) ;
<bexpr> → <relop> <aexpr> <aexpr>
<relop> → == | != | < | > | <= | >=
<aexpr> → <term> {(+ | - | * | /) <term>}
<term> → <number> | <var> | (<aexpr>)
<type> → int
<number> → <dec>{<dec>}
<dec> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<var> → <alphabet>{<alphabet>}
<alphabet> → a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r |
s | t | u | v | w | x | y | z

설계

이전 구현과제 1에서 설계한 방식을 기반으로 설계했다.

구현과제 1과 다른 특징이 몇개 있는데,

1. 연산자의 우선순위가 존재하지 않음
2. Bexpr의 형식이 다름
3. Do while 문의 존재

까다로웠던 건 statement에 do while문이었다.

Current token을 확인하는 인덱스 pos를 do while문에 활용했다.

처음 do while문에 진입했을 때 pos를 temp_pos에 저장해놓고, while문의 boolean값을 확인 후 true면 pos값을 temp_pos - 1로 바꾸고 탈출했다.

때문에 다시 재귀로 돌아가 get_token을 호출하면 이전의 do 토큰부터 호출이 되는 구조이다.

False일때는 pos가 이동한 채로 함수를 탈출하기 때문에 while문에서 탈출하는 것을 볼 수 있다.

구현

C 언어로 구현

- 타입

```
typedef enum {
    TOKEN_NONE,
    TOKEN_NUMBER, TOKEN_VAR, TOKEN_ASSIGN, TOKEN_RELOP, TOKEN_PRINT,
    TOKEN_TERMINATE, TOKEN_ERROR, TOKEN_SEMI, TOKEN_DO, TOKEN_WHILE,
    TOKEN_LBRACE, TOKEN_RBRACE, TOKEN_LPAREN, TOKEN_RPAREN, TOKEN_INT, TOKEN_OP
} 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

```
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;
```

Token – input을 token 단위로 구분하여 저장하는 구조체

Ex) input = "int x ; x = 2 ;"

Type = token_INT value = "int"

Type = token_VAR value = "x"

TYPE = token_SEMI value = 2

TYPE = token_VAR value = x

TYPE = token_ASSIGN value = "="

TYPE = token_NUMBER value = 2

.

Variable – statement에서 "=" 기호로 유추되는 변수를 저장하는 구조체, ERROR_MESSAGE 또한 변수로 가지고 있지만 해당 과제에서는 사용하지 않음

Ex) input = "int x ; x = 3 ;"

Variable[0]

Name = "x"

Value = "3"

isTrue = 0 (하지만 타입을 보고 사용하지 않음)

Type = type_NUMBER

로직 설명

Program, declaration, statement 처럼 변수를 초기화 하거나 값을 세팅하는 문법은 void
이 외에 반환값이 있는 ex) a_expr, term, number 함수는 int로 구현했다.

```
Token get_token();
void program();
void statement();
void declaration();
void expr();
int a_expr();
int b_expr();
int term();
int number();
```

Token및 input과 관련된 변수는 모두 전역변수이고, pos를 통해 token을 확인하고 var_pos를 통해 변수를 초기화 한다.

```
Token cur_token;
char **input_split;
int pos = 0;

Variable variables[MAX_VAR];
int var_pos = 0;
int var_count = 0;
```

각 단계에서 에러 메세지를 출력하는 기능이 있지만, 정확도가 떨어지고 오히려 헷갈리기에 해당 과제에서는 제외했다.
출력(print)를 처리할 때는 buffer에 먼저 기록 한다. 재귀가 종료되고 Syntax_error_flag가 0이라면 buffer의 내용을 출력하고 1이면 syntax error를 출력하고 static 변수들을 다시 초기화 한다.

```
void program() {
    cur_token = get_token();
    while (syntax_error_flag != 1 && cur_token.type == TOKEN_INT) {
        declaration();
    }

    while (syntax_error_flag != 1 && cur_token.type != TOKEN_TERMINATE) {
        statement();
    }

    if(syntax_error_flag == 1) {
        printf("Syntax Error!!\n");
    } else {
        if(strlen(result_buffer) > 0)
            printf("%s\n", result_buffer);
    }
    return ;
}
```

```
while (1) {
    memset(temp, 0, sizeof(temp));
    init_static();
    write(1, ">> ", 3);
    if (!fgets(temp, INPUT_LEN, stdin)) {
        printf("EOF detected or error reading input. Exiting.\n");
        break;
    }

    if (strcmp(temp, "terminate\n") == 0) {
        break;
    }

    input_split = split(temp, " \n");

    program();
}
```

실행 결과

```
>> int k ; int j ; k = 3 ; j = 20 ; do { print k + ( j - 1 ) * 10 ; k = k - 1 ; } while ( > k + 10 10 ) ; print == k 0 ;
220 210 200 TRUE
>> int i ; int j ; i = 0 ; j = 0 ; do { j = j + i ; i = i + 1 ; } while ( < i * ( 5 - 4 ) 5 ) ; print i ; print j ;
5 10
>> int k ; int j ; k = 3 ; j = 20 ; print k + j ;
23
>> int x ; x = 10 + 5 - ( 2 + 3 - 5 * 10 ) ; print x ;
15
>> int x ; x = 10 + 5 * 2 ; print x ; print > 10 x ;
30 FALSE
>> int variable ; variable = 365 ;
```

```
>> int k = 3 ;
Syntax Error!!
>> int k ; j = 3 ;
Syntax Error!!
>> int k ; k = 3 ; print j ;
Syntax Error!!
>> int k;
Syntax Error!!
>> int k ; k = 20 - 30 ; print k ;
-10
```