Compiler Task #1

> 2017.09.18
>
> 이준수 : 20132429

1. 목적

Recursive Descent Parsing 방법을 이용해서 정수(Integer) 와 실수(Float) 의 덧셈(+) 과 곱셈(*) 과 우선순위('(', ')') 의 문법검사(Syntax Analysis) 와 값의 연산(Calculate) 을 함께하는 분석기(Parser) 를 제작한다.

2. 주요 자료구조 및 함수

Header Files, Enum values

```
/*
Syntax Analytics : only detected error from potential sentense
+
Calculator for "Integer" Operation
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

enum{null, PLUS, STAR, NUMBER, LPAREN, RPAREN, END} t_type;
enum{null2, INTEGER_VAL, FLOAT_VAL} d_type;
```

t_type 은 토큰타입을 의미한다.

d_type 은 데이터타입을 의미한다. 여기서는 정수(INTEGER_VAL, FLOAT_VAL)로 구분한다.

Define valuable

```
// 정수타입이라면 dtype = INTEGER_VAL, 실수타입이라면 dtype = FLOAT_VAL 이 들어갈 것이고,
// 정수타입이라면 i_value 에 값이, 실수타입이라면 f_value 에 값이 들어가 있을 것으로 약속한다
typedef struct{
    int dtype;
    int i_value;
    float f_value;
}RET;

char token = ' ';
char target_string[] = "2.54*(6+4)$";
int cursor = 0; // target string 서 읽어들이는 커서 위치값
int max_len = -1; // Target String 의 길이
int is_debug = 0; // Debug message 출력 여부

int num = 0; // 정수타입 계산값
int float_flag = 0; // 실수 여부 판단('.')
int floating_size = 0; // 실수의 아랫위치값
float f_num = 0.0; // 실수타입 계산값

RET glb_ret;
```

테스트의 용이성을 위해서 본 과제에서는 target_string 문자열을 지정해서 개발했다. 이는 추후 getchar() 로 아에 로직을 변환하거나, 파일의 리다이렉팅 결과값을 argv 형태로 string 에 저장시키는 방법으로 진행을 하려고 한다.

is_debug 는 디버그 메세지를 출력할지 말지를 결정할 변수이다.

floating_flag 는 target_string 에서 '.' 문자가 등장했을 때, 실수형태로 인식하기 위해서 사용한 방법이다. 자세한 로직은 추후에 다루도록 한다.

struct RET 를 정의하였고, 여기에는 dtype : data type 과 정수형변수공간, 실수형변수공간을 지정하였다.

이를 전역변수 RET glb_ret 를 선언한 뒤 관리한다.

Define Functions

```
RET* term();
RET* factor();
RET* expression();
void error(int);
void nice_end();
void do_debug();
void show_RET(RET * ret, char*);
void warning(int i);
```

expression, term, factor 는 각각 RET * 타입을 리턴한다.

나머지는 프로그램의 편의성을 위해서 정의하였다.

warning 함수는 정수와 실수가 연산을 할 경우에 경고메세지를 띄우는 역할을 하게된다.

get_token Function Implement

```
void get_token(){
    // next token of input --> token
    if (cursor > max_len){
        nice_end();
    }

    int here = target_string[cursor];

    if(here == '+'){
        token = PLUS;
    }else if(here == '*'){
        token = STAR;
    }else if(here == '('){
        token = LPAREN;
    }else if(here == ')'){
        token = RPAREN;
    }else if(here == '$'){
        token = END;
        nice_end();
    }else if((here >= '0') && (here <= '9')){
        float_flag = 0; // 실수 플레그 초기화
        floating_size = 0; // 실수 크기 초기화
        f_num = 0.0; // 실수값 초기화
        token = NUMBER;
        num = here - '0'; // because of ascii type
        while(((target_string[cursor+1] >= '0') && (target_string[cursor+1]
<='9')) || (target_string[cursor+1] == '.')){
            if(target_string[cursor+1] == '.'){
                // printf("up float flag\n");
                float_flag = 1; // 실수 플레그 올림
                f_num = num; // 실수값에 정수부 저장
                cursor++; // '.' : 진행시킴
                continue;
            }
            if(float_flag == 1){
                // 실수플레그가 서있다면
                // printf("in to float_flag\n");
                floating_size += 1; // 실수 크기 하나 증가시키고
                cursor++;
                here = target_string[cursor] - '0';
                f_num += here*pow(0.1, floating_size);
            }else{
```

```
                // 정수 플레그
                num = num*10;
                cursor++;
                here = target_string[cursor];
                num += here - '0';
            }
            // printf(">> f_flag : %d\n", float_flag);
            // printf(">> %d\n", num);
            // printf(">> %f\n", f_num);
        }
        if(float_flag == 1){
            glb_ret.dtype = FLOAT_VAL;
            glb_ret.f_value = f_num;
        }else{
            // integer value;
            glb_ret.dtype = INTEGER_VAL;
            glb_ret.i_value = num;
        }
        show_RET(&glb_ret, "global ret");

    }else{
        error(cursor);
    }
    do_debug();

    cursor += 1;
}
```

expression Function Implements

```
RET* expression(){
    // t + t + t ...
    // 이 작업을 하기 전에 이미 get_token() 수행하여, token 값이 채워져 있다고 가정
    RET * ret = term();
    show_RET(ret, "expression");
    while(token == PLUS){
        get_token();
        RET * tmp = term();
        if(ret->dtype == INTEGER_VAL && tmp->dtype == INTEGER_VAL){
            ret->i_value += tmp->i_value;
        }else if(ret->dtype == INTEGER_VAL && tmp->dtype == FLOAT_VAL){
            warning(cursor);
            ret->dtype = FLOAT_VAL;
            ret->f_value = ret->i_value;
            ret->f_value += tmp->f_value;
        }else if(ret->dtype == FLOAT_VAL && tmp->dtype == INTEGER_VAL){
            warning(cursor);
            ret->f_value += tmp->i_value;
        }else if(ret->dtype == FLOAT_VAL && tmp->dtype == FLOAT_VAL){
            ret->f_value += tmp->f_value;
        }else{
            error(cursor);
        }
        free(tmp); // tmp 포인터 메모리 해제
    }
    return ret;
    // 이 작업이 끝나기 전에 다음 token 값을 읽어서 넘김
}
```

term 의 구조는 expression 과 매우 유사하므로 생략한다.

factor Function Implements

```
RET* factor(){
    // n | (expression)
    RET * ret = (RET*)malloc(sizeof(RET));
    // int n = 0;
    if(token == NUMBER){
        if(glb_ret.dtype == INTEGER_VAL){
            ret->dtype = INTEGER_VAL;
            ret->i_value = glb_ret.i_value;
        }else{
            ret->dtype = FLOAT_VAL;
            ret->f_value = glb_ret.f_value;
        }
        get_token();
    }
    else if(token == LPAREN){
        get_token();
        ret = expression();
        if(token == RPAREN){
            get_token();
        }else{
            error(cursor);
        }
    }else{
        error(cursor);
    }
    return ret;
}
```

get_token 결과 담겨진 token 과 glb_ret 값을 기준으로 n 을 처리하게 된다.

3. 결과 첨부

Successful Case : 2.54*(6+4)$

```
myZZUNG@ijunsuui-MacBook-Pro:~/myworkspace/anything/compiler_lecture$ sh int_float_cal_rdparser.c
target_string : 2.54*(6+4)$
cursor : 0,  max length : 11
This sentense end
>> warning : type mismatch at 11

###Total answer###
message : answer, type : 실수형 , value : 25.400000
```

정수타입 변수와 실수타입 변수의 연산을 진행했을시, warning 메세지가 뜨는 것 까지 완료

Error Case : 2.54*+(6+4)$

```
myZZUNG@ijunsuui-MacBook-Pro:~/myworkspace/anything/compiler_lecture$ sh int_float_cal_rdparser.c
target_string : 2.54*+(6+4)$
cursor : 0,  max length : 12


              2.54*+(6+4)$
                   ^
                   |

you get error! at position : 5  value : +
```

어느 위치에서 문제가 발생했는지도 알려주도록 작성

4. 전체 소스 첨부

```
/*
Syntax Analytics : detected error from potential sentense and calculate
answer value
+
Calculator for "Integer" + "Float" Operation
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

enum{null, PLUS, STAR, NUMBER, LPAREN, RPAREN, END} t_type;
enum{null2, INTEGER_VAL, FLOAT_VAL} d_type;

// 정수타입이라면 dtype = INTEGER_VAL, 실수타입이라면 dtype = FLOAT_VAL 이 들어갈 것이
고,
// 정수타입이라면 i_value 에 값이, 실수타입이라면 f_value 에 값이 들어가 있을 것으로 약속한다
typedef struct{
    int dtype;
    int i_value;
    float f_value;
}RET;

char token = ' ';
char target_string[] = "2.54*(6+4)$";
int cursor = 0; // target string 서 읽어들이는 커서 위치값
int max_len = -1; // Target String 의 길이
int is_debug = 0; // Debug message 출력 여부

int num = 0; // 정수타입 계산값
int float_flag = 0; // 실수 여부 판단('.')
int floating_size = 0; // 실수의 아랫위치값
float f_num = 0.0; // 실수타입 계산값

RET glb_ret;
```

```
RET* term();
RET* factor();
RET* expression();
void error(int);
void nice_end();
void do_debug();
void show_RET(RET * ret, char*);
void warning(int i);

void nice_end(){
    printf("This sentense end\n");
    // exit(0);
}

void get_token(){
    // next token of input --> token
    if (cursor > max_len){
        nice_end();
    }

    int here = target_string[cursor];

    if(here == '+'){
        token = PLUS;
    }else if(here == '*'){
        token = STAR;
    }else if(here == '('){
        token = LPAREN;
    }else if(here == ')'){
        token = RPAREN;
    }else if(here == '$'){
        token = END;
        nice_end();
    }else if((here >= '0') && (here <= '9')){
        float_flag = 0; // 실수 플레그 초기화
        floating_size = 0; // 실수 크기 초기화
        f_num = 0.0; // 실수값 초기화
        token = NUMBER;
        num = here - '0'; // because of ascii type
        while(((target_string[cursor+1] >= '0') && (target_string[cursor+1]
<='9')) || (target_string[cursor+1] == '.')){
            if(target_string[cursor+1] == '.'){
                // printf("up float flag\n");
                float_flag = 1; // 실수 플레그 올림
                f_num = num; // 실수값에 정수부 저장
                cursor++; // '.' : 진행시킴
                continue;
```

```
            }
            if(float_flag == 1){
                // 실수플레그가 서있다면
                // printf("in to float_flag\n");
                floating_size += 1; // 실수 크기 하나 증가시키고
                cursor++;
                here = target_string[cursor] - '0';
                f_num += here*pow(0.1, floating_size);
            }else{
                // 정수 플레그
                num = num*10;
                cursor++;
                here = target_string[cursor];
                num += here - '0';
            }
            // printf(">> f_flag : %d\n", float_flag);
            // printf(">> %d\n", num);
            // printf(">> %f\n", f_num);
        }
        if(float_flag == 1){
            glb_ret.dtype = FLOAT_VAL;
            glb_ret.f_value = f_num;
        }else{
            // integer value;
            glb_ret.dtype = INTEGER_VAL;
            glb_ret.i_value = num;
        }
        show_RET(&glb_ret, "global ret");

    }else{
        error(cursor);
    }
    do_debug();

    cursor += 1;

}

RET* expression(){
    // t + t + t ...
    // 이 작업을 하기 전에 이미 get_token() 수행하여, token 값이 채워져 있다고 가정
    RET * ret = term();
    show_RET(ret, "expression");
    while(token == PLUS){
        get_token();
        RET * tmp = term();
        if(ret->dtype == INTEGER_VAL && tmp->dtype == INTEGER_VAL){
            ret->i_value += tmp->i_value;
        }else if(ret->dtype == INTEGER_VAL && tmp->dtype == FLOAT_VAL){
```

```
            warning(cursor);
            ret->dtype = FLOAT_VAL;
            ret->f_value = ret->i_value;
            ret->f_value += tmp->f_value;
        }else if(ret->dtype == FLOAT_VAL && tmp->dtype == INTEGER_VAL){
            warning(cursor);
            ret->f_value += tmp->i_value;
        }else if(ret->dtype == FLOAT_VAL && tmp->dtype == FLOAT_VAL){
            ret->f_value += tmp->f_value;
        }else{
            error(cursor);
        }
        free(tmp);

    }
    return ret;

    // 이 작업이 끝나기 전에 다음 token 값을 읽어서 넘김
}

RET* term(){
    // f * f * f ...
    RET * ret = factor();
    show_RET(ret, "term");
    while(token == STAR){
        get_token();
        RET * tmp = factor();
        if(ret->dtype == INTEGER_VAL && tmp->dtype == INTEGER_VAL){
            ret->i_value *= tmp->i_value;
        }else if(ret->dtype == INTEGER_VAL && tmp->dtype == FLOAT_VAL){
            warning(cursor);
            ret->dtype = FLOAT_VAL;
            ret->f_value = ret->i_value;
            ret->f_value *= tmp->f_value;
        }else if(ret->dtype == FLOAT_VAL && tmp->dtype == INTEGER_VAL){
            warning(cursor);
            ret->f_value *= tmp->i_value;
        }else if(ret->dtype == FLOAT_VAL && tmp->dtype == FLOAT_VAL){
            ret->f_value *= tmp->f_value;
        }else{
            error(cursor);
        }
        free(tmp);
    }
    return ret;
}

RET* factor(){
    // n | (expression)
```

```
    RET * ret = (RET*)malloc(sizeof(RET));
    // int n = 0;
    if(token == NUMBER){
        if(glb_ret.dtype == INTEGER_VAL){
            ret->dtype = INTEGER_VAL;
            ret->i_value = glb_ret.i_value;
        }else{
            ret->dtype = FLOAT_VAL;
            ret->f_value = glb_ret.f_value;
        }
        get_token();
    }
    else if(token == LPAREN){
        get_token();
        ret = expression();
        if(token == RPAREN){
            get_token();
        }else{
            error(cursor);
        }
    }else{
        error(cursor);
    }
    return ret;
}

void error(int i){

    printf("\n\n");
    printf("\t\t%s\n", target_string);
    int t = 0;
    printf("\t\t");
    for(t=0; t<i-1 ;t++){
        printf("%c", ' ');
    }
    printf("%c\n", '^');
    printf("\t\t");
    for(t=0; t<i-1 ;t++){
        printf("%c", ' ');
    }
    printf("%c\n", '|');
    printf("\n");
    printf("you get error! at position : %d\tvalue : %c\n", i-1,
target_string[i-1]);
    printf("\n");
    exit(0);
}

void warning(int i){
```

```
    printf(">> warning : %s at %d\n", "type mismatch", i);
}

void do_debug(){
    if(is_debug == 1){
        printf("max_len : %d\tcursor : %d\ttoken : %d\ttoken mean : %c\n",
max_len, cursor, token, target_string[cursor]);
    }
}

void show_RET(RET * ret, char * str){
    if(is_debug == 1){
        if(ret->dtype == INTEGER_VAL){
            printf("message : %s, type : %s, value : %d\n", str, "정수형",
ret->i_value);
        }else{
            printf("message : %s, type : %s, value : %f\n", str, "실수
형",ret->f_value);
        }
    }

}

int main(){
    max_len = strlen(target_string);
    printf("target_string : %s\n", target_string);
    printf("cursor : %d,  max length : %u\n", cursor, max_len);

    get_token(); // read start symbol
    // int answer = 0;

    RET * answer = expression(); // start symbol
    if(token != END){
        error(cursor);
    }else{
        // printf("Total Answer is %d\n", answer);
        printf("\n###Total answer###\n");
        is_debug = 1;
        show_RET(answer, "answer");
    }

    return 0;
}
```