컴파일러 과제-2

20192800 권대현

1. 과제 내용

* 이번 컴파일러 과제는 1장에서 설명한 LR Parsing 방식을 이용하여 수식이 문법에 맞는지를 검사하고 문법이 맞으면, 수식의 값을 계산하는 프로그램을 완성하는 것이다.
* 입력으로 주어지는 수식은 +, \*, (, ), 정수, 실수 들로 구성된다.
* 수식이 잘못된 경우 오류 메시지를 출력한다.

1. 해결 방법

* 먼저 프로그램 실행 환경은 cygwin으로 설정하였다.

전자제품, 텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* 기본적인 프로그램 구조는 ‘컴파일러-1장 강의노트.pdf’의 1-29 ~ 1-31, 1-35 ~ 1-36을 참고하여 작성했다.
* 입력으로 정수와 실수가 들어오기 때문에 두 자료형을 처리하기 위한 RET 구조체를 선언하여 자료형의 종류를 담는 enum형 KIND와 값을 담는 union형 VALUE를 포함하게 했다.
  + 이에 따라 기존 코드에서 value배열을 RET형으로, yylval을 RET형으로 수정했다.
  + Value의 kind에 따라 결과 출력의 개행문자를 달리하여 처리했다.
  + 실제 값 계산이 이뤄지는 reduce()의 switch문에서 value의 kind값이 INT이냐 FLT이냐에 따른 형변환과 계산이 이뤄지는 코드를 추가했다. 또한, return되는 value[top]의 kind가 결정되고 계산된 값이 value에 저장된다.
* 실수 계산을 위한 자료형을 float으로 설정했더니, 긴 자리 수 숫자들의 계산에서 결과값에 오차가 생기는 오류가 발생했다.
  + 이는 float과 int간 형변환에서 데이터가 소실되는 결과라고 판단하였고, 이를 줄이기 위해 float보다 데이터가 큰 double형으로 자료형을 변경했더니 해결되었다.

1. 결론

* 수식을 사용자로부터 입력 받은 뒤 EOF를 만나면 입력을 종료하고 문법 검사와 수식의 계산이 이뤄지는 프로그램을 완성했다.
* 수식의 문법이 틀리면 에러를 출력하고 종료를, 수식의 문법이 맞다면 수식의 값을 계산하여 출력하고 종료를 수행한다.
* 프로그램 실행 결과와 원시프로그램은 아래와 같다.
* 프로그램 실행결과
  + 정수형 수식

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* + 실수형 수식

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* + 괄호가 포함된 수식

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* + 문법이 잘못된 수식

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 원시프로그램

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define NUMBER 256

#define PLUS 257

#define STAR 258

#define LPAREN 259

#define RPAREN 260

#define END 261

#define EXPRESSION 0

#define TERM 1

#define FACTOR 2

#define ACC 999

int action[12][6] = {

    {5, 0, 0, 4, 0, 0}, {0, 6, 0, 0, 0, ACC}, {0, -2, 7, 0, -2, -2},

    {0, -4, -4, 0, -4, -4}, {5, 0, 0, 4, 0, 0}, {0, -6, -6, 0, -6, -6},

    {5, 0, 0, 4, 0, 0}, {5, 0, 0, 4, 0, 0}, {0, 6, 0, 0, 11, 0},

    {0, -1, 7, 0, -1, -1}, {0, -3, -3, 0, -3, -3}, {0, -5, -5, 0, -5, -5}

};

int go\_to[12][3] = {

    {1, 2, 3}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {8, 2, 3}, {0, 0, 0},

    {0, 9, 3}, {0, 0, 10}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}

};

typedef enum KIND {

    INT = 0,

    FLT

}KIND;

typedef union VALUE {

    int i;

    double f;

}VALUE;

typedef struct RET {

    KIND kind;

    VALUE value;

}RET;

int prod\_left[7] = { 0, EXPRESSION, EXPRESSION, TERM, TERM, FACTOR, FACTOR };

int prod\_length[7] = { 0, 3, 1, 3, 1, 3, 1 };

int stack[1000]; int top = -1; int sym;

RET value[1000];

int yyparse();

void push(int i);

void shift(int i);

void reduce(int i);

void yyerror(const char\* str);

int yylex();

void lex\_error();

char yytext[32];

RET yylval;

int main()

{

    yyparse();

    if (value[top].kind == INT)

        printf("= %d\n", value[top].value.i);

    else

        printf("= %f\n", value[top].value.f);

    return 0;

}

int yyparse()

{

    int i;

    stack[++top] = 0;

    sym = yylex();

    do

    {

        i = action[stack[top]][sym - 256];

        if (i == ACC)

            printf("success!\n");

        else if (i > 0)

            shift(i);

        else if (i < 0)

            reduce(-i);

        else

            yyerror(NULL);

    } while (i != ACC);

    return 1;

}

void push(int i)

{

    stack[++top] = i;

}

void shift(int i)

{

    push(i);

    value[top] = yylval;

    sym = yylex();

}

void reduce(int i)

{

    int old\_top;

    top -= prod\_length[i];

    old\_top = top;

    push(go\_to[stack[old\_top]][prod\_left[i]]);

    switch (i)

    {

    case 1:

        if (value[old\_top + 1].kind == FLT || value[old\_top + 3].kind == FLT)

        {

        double result = 0;

            if (value[old\_top + 1].kind == FLT)

                result = value[old\_top + 1].value.f;

            else

                result = (double)value[old\_top + 1].value.i;

            if (value[old\_top + 3].kind == FLT)

                result += value[old\_top + 3].value.f;

            else

                result += (double)value[old\_top + 3].value.i;

        if(result - (int)result == 0)

        {

            value[top].kind = INT;

            value[top].value.i = result;

        }

        else

        {

            value[top].kind = FLT;

            value[top].value.f = result;

        }

        }

        else

        {

            value[top].kind = INT;

            value[top].value.i = value[old\_top + 1].value.i + value[old\_top + 3].value.i;

        }

        break;

    case 2:

        value[top] = value[old\_top + 1];

        break;

    case 3:

        if (value[old\_top + 1].kind == FLT || value[old\_top + 3].kind == FLT)

        {

            double result = 0;

            if (value[old\_top + 1].kind == FLT)

                result = value[old\_top + 1].value.f;

            else

                result = (double)value[old\_top + 1].value.i;

            if (value[old\_top + 3].kind == FLT)

                result \*= value[old\_top + 3].value.f;

            else

                result \*= (double)value[old\_top + 3].value.i;

        if(result - (int)result == 0)

        {

            value[top].kind = INT;

            value[top].value.i = result;

        }

        else

        {

            value[top].kind = FLT;

            value[top].value.f = result;

        }

        }

        else

        {

            value[top].kind = INT;

            value[top].value.i = value[old\_top + 1].value.i \* value[old\_top + 3].value.i;

        }

        break;

    case 4:

        value[top] = value[old\_top + 1];

        break;

    case 5:

        value[top] = value[old\_top + 2];

        break;

    case 6:

        value[top] = value[old\_top + 1];

        break;

    default:

        yyerror("parsing table error");

        break;

    }

}

void yyerror(const char\* str)

{

    if (str == NULL)

        printf("syntax error\n");

    else

        printf("%s\n", str);

    exit(1);

}

int yylex()

{

    static char ch = ' ';

    int i = -1;

    while (ch == ' ' || ch == '\t' || ch == '\n')

        ch = getchar();

    if (isdigit(ch))

    {

        do

        {

            yytext[++i] = ch;

            ch = getchar();

        } while (isdigit(ch));

        if (ch == '.')

        {

            do

            {

                yytext[++i] = ch;

                ch = getchar();

            } while (isdigit(ch));

            yytext[i + 1] = '\0';

            yylval.kind = FLT;

            yylval.value.f = atof(yytext);

        }

        else

        {

            yytext[i + 1] = '\0';

            yylval.kind = INT;

            yylval.value.i = atoi(yytext);

        }

        return NUMBER;

    }

    else if (ch == '+')

    {

        ch = getchar();

        return PLUS;

    }

    else if (ch == '\*')

    {

        ch = getchar();

        return STAR;

    }

    else if (ch == '(')

    {

        ch = getchar();

        return LPAREN;

    }

    else if (ch == ')')

    {

        ch = getchar();

        return RPAREN;

    }

    else if (ch == EOF)

        return END;

    else

        lex\_error();

}

void lex\_error()

{

    printf("lex error\n");

    exit(1);

}