Homework 2: Lex & Yacc

資工二甲 1112914 宋冠穎

系統與工具版本

- Windows 11 版本: Windows 11 家用版 版本: 23H2 安裝於 2024/1/26 OS 組建: 22631.3447 體驗: Windows Feature Experience Pack 1000.22688.1000.0
- flex(lex): flex version 2.5.4
- bison(yacc): bison (GNU Bison) 2.4.1
- gcc (Rev6, Built by MSYS2 project) 13.1.0

限制

- 最大變數數量:1024
- 變數名稱長度限制:16 字元

編譯方式(附檔已編譯完成)

方式 1.命令提示字元

命令提示字元 (command line)，切換到該文件路徑下執行以下指令:

```
flex calc.l
bison -d calc.y
gcc -o calculator calc.tab.c lex.yy.c -lm
```

方式 2.makefile

命令提示字元 (command line)，切換到該文件路徑下執行以下指令:

```
make
```

執行方式(測試程式檔“input.txt”放於 HW2_1112914 目錄下)

方式 1.命令提示字元

命令提示字元 (command line)，切換到該文件路徑下執行以下指令:

```
calculator input.txt
```

方式 2.makefile

命令提示字元 (command line)，切換到該文件路徑下執行以下指令:

```
make run
```

測試輸入輸出

Test Case 1.

- Input

```
var1 = neg(3) -5
var2=10+ var1
var3 = log(100)+10
var4 = abs(var1) - var2* cos(5)
var5 = sin(var2) + 2^2* var3 - (var2 + var1)
var6 = var3 ^var2
```

- Output

```
-8
2
12
7.43268
54.9093
144
```

Test Case 2.

- Input

```
var1 = 3-5
var2=2**2*2
```

- Output

```
-2
Line 2: syntax error with token "*"
```

Test Case 3.

- Input

```
var1=3-5
var2 = 2* var3*2
```

- Output

```
-2
Line 2:var3 is undefined
```

Test Case 4.

- Input

```
var1 = 5+3
var2 = var1 * 2
var3 = log(1000) + var2
var4 = abs(var3) - var1 * cos(2)
var5 = sin(var2) + 3^2 * var3 - (var2 + var1)
var6 = var3^var1
```

- Output

```
8
16
19
22.3292
146.712
1.69836e+10
```

Test Case 5.

- Input

```
var1 = abs(neg(5)) + 3*2
var2 = var1^2 + log(1000)
var3 = sin(var2) + cos(var1)
var4 = var3 / 3 + 2*var2
var5 = var4 % 2 + ++var1
var6 = --var5 + var4*var3
```

- Output

```
11
124
-0.991261
247.67
13.6696
-232.836
```

程式碼

calc.l (lex)

```
%{
#include "calc.tab.h"
#include <string.h>


char* yytext_ptr;
%}

DIGIT [0-9]
NUMBER ({DIGIT}+(\.{DIGIT}*)?|\.{DIGIT}+)

%%
[ \t]                       ; /* ignore whitespace */
\n                          {yylval.string = strdup(yytext);return EOL;}

"="                         {yylval.string = strdup(yytext);return ASSIGN;}
"+"                         {yylval.string = strdup(yytext);return PLUS;}
"-"                         {yylval.string = strdup(yytext);return MINUS;}
"*"                         {yylval.string = strdup(yytext);return TIMES;}
"/"                         {yylval.string = strdup(yytext);return DIVIDE;}
"^"                         {yylval.string = strdup(yytext);return POWER;}
"%"                         {yylval.string = strdup(yytext);return MODULO;}
```

```
"neg("                 {yylval.string = strdup(yytext);return NEG;}
"abs("                 {yylval.string = strdup(yytext);return ABS;}
"cos("                 {yylval.string = strdup(yytext);return COS;}
"sin("                 {yylval.string = strdup(yytext);return SIN;}
"log("                 {yylval.string = strdup(yytext);return LOG;}
"++"                   {yylval.string = strdup(yytext);return INC;}
"--"                   {yylval.string = strdup(yytext);return DEC;}
"("                    {yylval.string = strdup(yytext);return LPAREN;}
")"                    {yylval.string = strdup(yytext);return RPAREN;}
{NUMBER}               { yylval.number = atof(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.string = strdup(yytext); return VARIABL
E; }
.                      {yylval.string = strdup(yytext);return yytext
[0];}
%%

int yywrap() {
    return 1;
}
```

calc.y (yacc)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


extern int yylex();
extern int yyparse();
extern FILE* yyin;

extern char *yytext;

int yylineno = 0;

void yyerror(const char *s);
double compute(char* var, double val,int assign);//assign==1:assign //a
ssign==0:get value


#define MAX_VARIABLES 1024
#define MAX_VARIABLE_NAME_LENGTH 16

typedef struct {
    char name[MAX_VARIABLE_NAME_LENGTH];
    double value;
} Variable;
Variable variables[MAX_VARIABLES];
int numVariables = 0;

#define STACK_SIZE 100
double stack[STACK_SIZE];
int stackIndex = -1;

void push(double val) {
    if (stackIndex < STACK_SIZE - 1)
        stack[++stackIndex] = val;
    else {
        fprintf(stderr, "Stack overflow\n");
        exit(EXIT_FAILURE);
    }
}

double pop() {
    if (stackIndex >= 0)
        return stack[stackIndex--];
    else {
        fprintf(stderr, "Stack underflow\n");
        exit(EXIT_FAILURE);
    }
}

%}

%union {
    char* string;
    double number;
}


%token <string> VARIABLE
%token <number> NUMBER
%token EOL ASSIGN PLUS MINUS TIMES DIVIDE POWER MODULO NEG ABS COS SIN
LOG INC DEC LPAREN RPAREN

%left PLUS MINUS
%left TIMES DIVIDE MODULO
%nonassoc INC DEC
%right POWER
%nonassoc NEG ABS COS SIN LOG

%type <number> expression
%type <number> term
%type <number> factor
%type <number> unary_op
```

```
%%

input: /* empty */ {;}
    | input statement EOL {;}
    ;

statement:
    | VARIABLE ASSIGN expression {
                                    /*printf("statement:VARIABLE ASSIGN
 expression \n"); */
                                    double tmp=compute($1,$3,1) ;
                                    printf("%g\n",tmp);
                                }
    | expression { /*printf("statement:expression \n");*/ printf("%.5f\
n", $1); }
    ;

expression:
    term{/*printf("expression:term \n");*/ $$ = $1; }
    | expression PLUS term { /*printf("expression:expression PLUS term
\n");*/  $$ = $1 + $3; }
    | expression MINUS term { /*printf("expression MINUS term \n");*/
$$ = $1 - $3; }
    ;

term:
    factor{/*printf("term:factor \n");*/ $$=$1;}
    | term TIMES factor { /*printf("term:term TIMES factor \n");*/  $$
= $1 * $3; }
    | term DIVIDE factor { /*printf("term:term DIVIDE factor \n");*/  $
$ = $1 / $3; }
    | term MODULO factor { /*printf("term:term MODULO factor \n");*/  $
$ = fmod($1,$3); }
    ;

factor:
    unary_op{/*printf("factor:unary_op \n");*/ $$=$1;}
    | factor POWER unary_op {/*printf("factor:factor POWER unary_op \n
");*/  $$ = pow($1, $3); }
    ;

unary_op:
    NUMBER{/*printf("unary_op:NUMBER %.5f\n",$1);*/ $$=$1;}
    | VARIABLE {/*printf("unary_op:VARIABLE \n");*/  $$ = compute($1, 0,
0); }
    | PLUS unary_op { /*printf("unary_op:PLUS unary_op \n");*/  $$ = $2;
 }
    | MINUS unary_op { /*printf("unary_op:MINUS unary_op \n");*/ $$ = -
$2; }
```

```
    | NEG unary_op RPAREN { /*printf("unary_op:NEG unary_op RPAREN \n");
*/ $$ = -$2; }
    | ABS unary_op RPAREN { /*printf("unary_op:ABS unary_op RPAREN \n");
*/ $$ = fabs($2); }
    | COS unary_op RPAREN { /*printf("unary_op:COS unary_op RPAREN \n");
*/ $$ = cos($2); }
    | SIN unary_op RPAREN { /*printf("unary_op:SIN unary_op RPAREN \n");
*/ $$ = sin($2); }
    | LOG unary_op RPAREN { /*printf("unary_op:LOG unary_op RPAREN \n");
*/ $$ = log10($2); }
    | INC VARIABLE { /*printf("unary_op:INC VARIABLE \n");*/ compute($2,
 compute($2, 0,0)+1,1);$$ = compute($2, 0,0); }
    | DEC VARIABLE { /*printf("unary_op:DEC VARIABLE \n");*/ compute($2,
 compute($2, 0,0)-1,1);$$ = compute($2, 0,0); }
    | VARIABLE INC { /*printf("unary_op:INC VARIABLE \n");*/ $$ = compu
te($1, 0,0); compute($1, compute($1, 0,0)+1,1); }
    | VARIABLE DEC { /*printf("unary_op:DEC VARIABLE \n");*/ $$ = compu
te($1, 0,0); compute($1, compute($1, 0,0)-1,1); }
    | LPAREN expression RPAREN { /*printf("unary_op:LPAREN expression R
PAREN \n");*/ $$ = $2; }
    ;

%%


void yyerror(const char *s) {
    //fprintf(stderr, "Error: Line:%d %s\n",yylineno ,s);
    fprintf(stderr, "Line %d: %s with token \"%s\"\n", yylineno, s, yyt
ext);
    exit(EXIT_FAILURE);
}

double compute(char* var, double val,int assign) {
    /*printf("compute(%s,%.5f,%d)\n",var,val,assign);*/
    size_t var_len = strlen(var);
    if(var_len>16){
        printf("Variable %s length exceeds 16 characters.",var);
        exit(EXIT_FAILURE);
    }
    if(assign==0){
        for (int i = 0; i < numVariables; i++) {
            if (strcmp(variables[i].name, var) == 0) {
                if (val != 0)
                    variables[i].value = val;
                return variables[i].value;
            }
        }
        fprintf(stderr, "Line %d:%s is undefined\n", yylineno,var);
        exit(EXIT_FAILURE);
    }
    else{
        for (int i = 0; i < numVariables; i++) {
            if (strcmp(variables[i].name, var) == 0) {
                variables[i].value = val;
                return variables[i].value;
            }
        }
        strcpy(variables[numVariables].name, var);
        variables[numVariables].value = val;

        numVariables=numVariables+1;
        return val;
    }
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <input_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    FILE* inputFile = fopen(argv[1], "r");
    if (!inputFile) {
        fprintf(stderr, "Error: Could not open file %s\n", argv[1]);
        return EXIT_FAILURE;
    }
    // Read the input file line by line
    char line[1024]; // Adjust the buffer size as needed
    while (fgets(line, sizeof(line), inputFile)) {
        yylineno=yylineno+1;
        // Check if the line ends with a newline character
        size_t len = strlen(line);
        if (len == 0 || line[len - 1] != '\n') {
            // Append a newline character if missing
            line[len] = '\n';
            line[len + 1] = '\0';
        }

        // Create a temporary file in memory and write the line content
 into it
        FILE* tempFile = fopen("tempFile.txt","w+");
        if (!tempFile) {
            fprintf(stderr, "Error: Failed to create temporary file\n");
            return EXIT_FAILURE;
        }
        fputs(line, tempFile);
        rewind(tempFile); // Rewind the file pointer to the beginning
        // Pass the temporary file to the parser
        yyin = tempFile;
        yyparse();
        fclose(tempFile);
        /*printf("---------------\n");*/
    }

    /*yyin = inputFile;

    yyparse();*/

    fclose(inputFile);

    return EXIT_SUCCESS;
}
```