Jan Coleen S. Estilo

Exercise 9

Lex Rules

```
NUMBER [0-9]+(\.[0-9]+)?
SPACE [[:space:]]

HEXA H[A-F0-9]+
OCTA O[0-7]+
```

Here, I just created some variables with their corresponding regular expressions to make my rules more readable.

NUMBER pertains to any integer, wherein the decimal point is optional.

SPACE pertains to " ".

HEXA pertains to a character 'H' followed by the correct characters for writing hexadecimal values.

OCTA pertains to a character 'O' followed by the correct characters for writing octal values.

```
%%
"+"         {return ADD;}
"-"         {return SUB;}
"/"         {return DIV;}
"*"         {return MUL;}
"^"         {return POWER;}
"("         {return LEFT;}
")"         {return RIGHT;}
"="         {return ASSIGN;}
{HEXA}      {yylval.number = strtol(&yytext[1], NULL, 16); return NUMBER;}
{OCTA}      {yylval.number = strtol(&yytext[1], NULL, 8); return NUMBER;}
\n          {return EOL;}
[a-z]       {yylval.id = yytext[0]; return IDENTIFIER;}
{NUMBER}    {yylval.number = atof(yytext); return NUMBER;}
{SPACE}     {/*ignore white space*/}
"exit"      {return QUIT;}
.           {printf("\nUnrecognized string %c\n", *yytext);}
%%
```

The "+", "-", "/", and "*" are used for the basic arithmetic operations of the numbers. It returns the token with their corresponding function.

The "^" is used for the exponentiation of the numbers.

The “(“ and “)” are used for the parentheses or the grouping of the expressions. The open parentheses returns the token LEFT and the close parentheses returns the token RIGHT.

The “=” is used for the assignment of the values to the variables.

When we encounter a HEXA/OCTA input, we will convert the string to a long integer, strtol() is used for this. The first argument is the address of the first character to be converted. I put yytext[1] so that we will eliminate the ‘H’ or ‘O’ from the string. The NULL represents the ending part of the string. The third argument is the base of the string to be converted. Afterwards, we return the token NUMBER since it will be just like the token NUMBER.

\n represent the newline, or the EOL token.

The “[a-z]” represent the variable if we ever need to assign a value to any of the single variables. This is our identifier.

In “{NUMBER}”, we first convert the string to a double, which is why we used atof(). Then, we return the token NUMBER.

{SPACE} pertains to a space character.

“exit” pertains to the exit command when we want to exit the program.

“.” Represents to any character other than the ones we specified. It returns an error message whenever we encounter this.

YACC Rules

```
%%

calclist:/*  nothing */
    | assegn EOL
    | calclist assignment EOL
    | calclist exp EOL          {printf("=%f\n", $2); }
    | calclist QUIT             {printf("Exitting...\n"); return 0;}
    ;

assegn : assignment
    | identifiers
    ;

identifiers : assignment { val = $1;}
    | IDENTIFIER ASSIGN identifiers {updateSymbolVal($1,val);}
    ;

assignment : IDENTIFIER ASSIGN exp  {$$=$3; updateSymbolVal($1,$3);}
    ;
```

```
exp:
    term
    | exp ADD exp                {$$ = $1 + $3;}
    | exp SUB exp                {$$ = $1 - $3;}
    | exp MUL exp                {$$ = $1 * $3;}
    | exp DIV exp                {$$ = $1 / $3;}
    | exp POWER exp              {$$ = pow($1, $3);}
    | LEFT exp RIGHT             {$$ = ($2);}
    | SUB exp %prec NEGATIVE     {$$ = -$2;}
    ;

term: NUMBER                     {$$ = $1;}
    | IDENTIFIER                 {$$ = symbolVal($1);}
    ;

%%
```

calclist is the whole expression that we inputted, including the newline.

assegn is the grammar that will be used if we will go to the assignment grammar or the identifiers grammar. This is for if we assign a value to variables or use long assignments (e.g., “a=b=1”)

identifiers is the grammar if we will assign a value to a variable. In the assignment, we put the value to the val variable, which is a defined variable in the prologue section of the bison code. We will go to the grammar IDENTIFIER ASSIGN identifiers if we do a long assignment (a=b=1). We will update the value of the variable name to the value that is stored in val.

assignment is the grammar for the assignment of a value to a variable, that is not a long assignment. Such as “a=1”. The expression becomes the $$ so it can further go deeper to the grammar. And then we update the value of the identifier to the value that will be converted.

exp represents the grammar for the basic arithmetic expressions of the numbers. It includes addition, subtraction, multiplication, division, exponentiation, groupings, and negativity of numbers. %prec is used for if the preceding character is a SUB, or a minus.

term pertains to the numbers inside our expressions or the identifiers.

Sample Runs

Usage of input.txt is optional.