Language Translator

Name :- Kadhiwala Karan K.

Roll No :- IT-036

ID :- 17ITUBS034

Name :- Kachhadia Parth A.

Roll No :- IT-035

ID :- 16ITUON002

Language Description

- This toy language can perform very basic mathematical operations.
- Can define single letters as variable . Eg: ‘a;’ or ‘A;’
- Can print values of variables on our command. Eg: “print a;”
- Exits from the terminal on writing “exit;” command on the terminal and pressing enter key.
- It can also be used by users who don’t knows coding ,like the user don’t need to type any programming language syntax or functions.
- The main aim behind this toy language was declaration of variables and assigning them values which can be obtained when we use print statement along with the variable.
- The Yacc and Lex file are as follows.

Lex File

```
%{
        #include "y.tab.h"
%}

%%
"print"         {return print;}
"exit"          {return exit_command;}
[a-zA-Z]        {yylval.id = yytext[0]; return identifier;}
[0-9]+          {yylval.num = atoi(yytext); return number;}
[ \t\n]         ;
[//*-+=;()]     {return yytext[0];}
.               {yyerror("invalid character");}
%%


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

Yacc File

```
%{
        void yyerror(char *s);
        #include<stdio.h>
        #include<stdlib.h>
        int symbols[52];
        int symbolVal(char symbol);
        void updateSymbolVal(char symbol,int val);
%}

%union {int num; char id;}
%start line
%token print
%token exit_command
%token <num> number
%token <id> identifier
%type <num> line exp term
%type <id> assignment


%%

line    : assignment ';'        {;}
        | exit_command ';'      {exit(EXIT_SUCCESS);}
        | print exp ';'         {printf("Printing: %d\n",$2);}
        | line assignment ';'   {;}
        | line print exp ';'    {printf("Printing:: %d\n",$3);}
        | line exit_command ';' {exit(EXIT_SUCCESS);}
        ;

assignment : identifier '=' exp { updateSymbolVal($1,$3);}
           ;

exp     : term  {$$ = $1;}
        | exp '+' term { $$ = $1 + $3; }
        | exp '-' term { $$ = $1 - $3; }
        | exp '*' term { $$ = $1 * $3; }
        | exp '/' term { $$ = $1 / $3; }
        | '(' exp ')' { $$ = $2; }
        ;

term    : number        {$$ = $1;}
        | identifier    {$$ = symbolVal($1);}
        ;
%%
```

```
int computeSymbolIndex(char token)
{
        int idx=-1;
        if(islower(token)){
                idx = token - 'a' + 26;
        }else if(isupper(token)){
                idx = token - 'A';
        }
        return idx;
}

int symbolVal(char symbol)
{
        int bucket = computeSymbolIndex(symbol);
        return symbols[bucket];
}

void updateSymbolVal(char symbol,int val)
{
        int bucket = computeSymbolIndex(symbol);
        symbols[bucket] = val;
}

int main(void) {
        int i;
        for(i=0 ; i<52 ; i++){
                symbols[i] = 0;
        }

        return yyparse();
}

void yyerror(char *s){fprintf(stderr, "%s\n",s);}
```

Results

```
dhairya@karan:~/Desktop/LTPro/test$ ./a.out
b=5;
c=5;
a=b+c*10;
print a;
Printing:: 100
```

```
dhairya@karan:~/Desktop/LTPro/test$ ./a.out
b = 5;
c = 5;
a = (b*10) + c;
print a;
Printing:: 55
b = 5;
c = 5;
a = b + ( c*10);
syntax error
dhairya@karan:~/Desktop/LTPro/test$
```

Results(Continue)

```
dhairya@karan:~/Desktop/LTPro/test$ ./a.out
print 2;
Printing: 2
exit;
dhairya@karan:~/Desktop/LTPro/test$
```

```
dhairya@karan:~/Desktop/LTPro/test$ ./a.out
a = 2*3+4/2;
print a;
Printing:: 5
b=5;
print b;
Printing:: 5
print 2*5+4;
Printing:: 14
print x;
Printing:: 0
exit;
dhairya@karan:~/Desktop/LTPro/test$
```

Some Limitations

- The grammer written in this toy language is without operator precedence which can be modified.
- Right now this language simply computes expression from left to right without any type of precedence.
- Complex mathematical computations can also be added.