Experiment 7

SDT

Expression Evaluation

.l

```
%{
#include "y.tab.h"
extern int yylval; //global variable; yylval is predefined name
%}

%%
[0-9]+ {yylval = atoi(yytext); //atoi convert intergal string to int
      return N;}
[\t]+ ;
"\n" {return NL;}
. {return yytext[0];}
%%
int yywrap()
{}
```

.y

```
%{
#include <stdio.h>
int yylex(void);
int yyerror(char *s);
%}

%token N NL
```

```
%left '+' '-'
%left '*' '/'

%%
E : T NL {printf("Result = %d\n",$$);   //if does not work => E : T (remove NL)
          return 0;}

T : T '+' T {$$ = $1 + $3;} //T =$1; '+'=$2; T=$3; $$ = LHS Non terminal
| T '-' T {$$ = $1 - $3;}
| T '*' T {$$ = $1 * $3;}
| T '/' T {$$ = $1 / $3;}
| '-' N {$$ = -$2;}
| '(' T ')' {$$ = $2;}
| N {$$ = $1;}


%%
int main()
{
printf("Enter the Expression:");
yyparse();
}

int yyerror(char *s)
{
printf("Invalid\n");
}
```

Output

```
student@hostserver42:~/Desktop/21BCE1070$ lex sample.l
student@hostserver42:~/Desktop/21BCE1070$ cc lex.yy.c y.tab.c
student@hostserver42:~/Desktop/21BCE1070$ lex sample.l
student@hostserver42:~/Desktop/21BCE1070$ yacc -d sample.y
student@hostserver42:~/Desktop/21BCE1070$ cc lex.yy.c y.tab.c
student@hostserver42:~/Desktop/21BCE1070$ ./a.out
Enter the Expression:12*(9-8)-8
Result = 4
student@hostserver42:~/Desktop/21BCE1070$ ./a.out
Enter the Expression:askjas
Invalid
student@hostserver42:~/Desktop/21BCE1070$
```

Binary to Decimal

.l

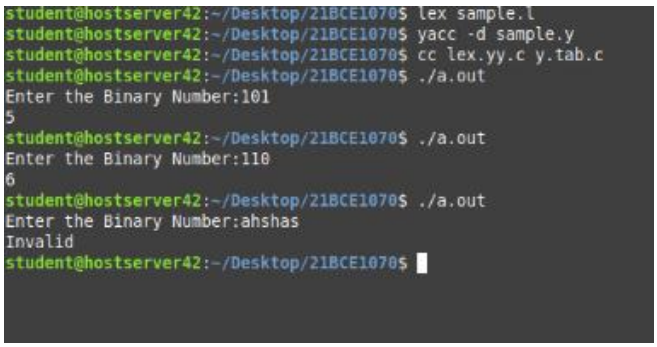
```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0] {yylval = atoi(yytext);
      return N0;}
[1] {yylval = atoi(yytext); return N1;}
[\t]+ ;
"\n" {return NL;}
. {return yytext[0];}
%%
int yywrap()
{}
```

.y

```
%{
#include <stdio.h>
int yylex(void);
int yyerror(char *s);
%}
```

```
%token N1 N0 NL

%left '+' '-'

%left '*' '/'

%%

N: L NL{printf("%d\n", $$);return 0;}

L: L B {$$=$1*2+$2;}

| B {$$=$1;}

B:N0 {$$=$1;}

|N1 {$$=$1;};

%%

int main()

{

printf("Enter the Binary Number:");

yyparse();

}

int yyerror(char *s)

{

printf("Invalid\n");

}
```

Output



Fractional Signed Binary to Decimal

```
.l
%{
#include "y.tab.h"
extern int yylval;
%}

%%
[0] {yylval = atoi(yytext);
      return N0;}
[1] {yylval = atoi(yytext); return N1;}
[\t]+ ;
"\n" {return NL;}
. {return yytext[0];}
%%


int yywrap()
{}


.y
%{
#include <stdio.h>
int yylex(void);
int yyerror(char *s);
%}

%token N1 N0 NL
%left '+' '-'
%left '*' '/'
```

```
%%
E : N {printf("%lf\n", (float)$$);return 0;}
N : S L '.' R {$$=$2+$4;}
S : '+' {$$=1;}
| '-' {$$=0;}

L : L B {$$= 2*$1 + $2;}
L : B {$$ = $1;}

R : R B {$1 = ($1 + $2)/2;}
| B { $$ = $1/2;}

B:N0 {$$=0;}
|N1 {$$=1;};

%%
int main()
{
printf("Enter the Binary Number:");
yyparse();
}

int yyerror(char *s)
{
printf("Invalid\n");
}
```

Output

Scientific Calculator

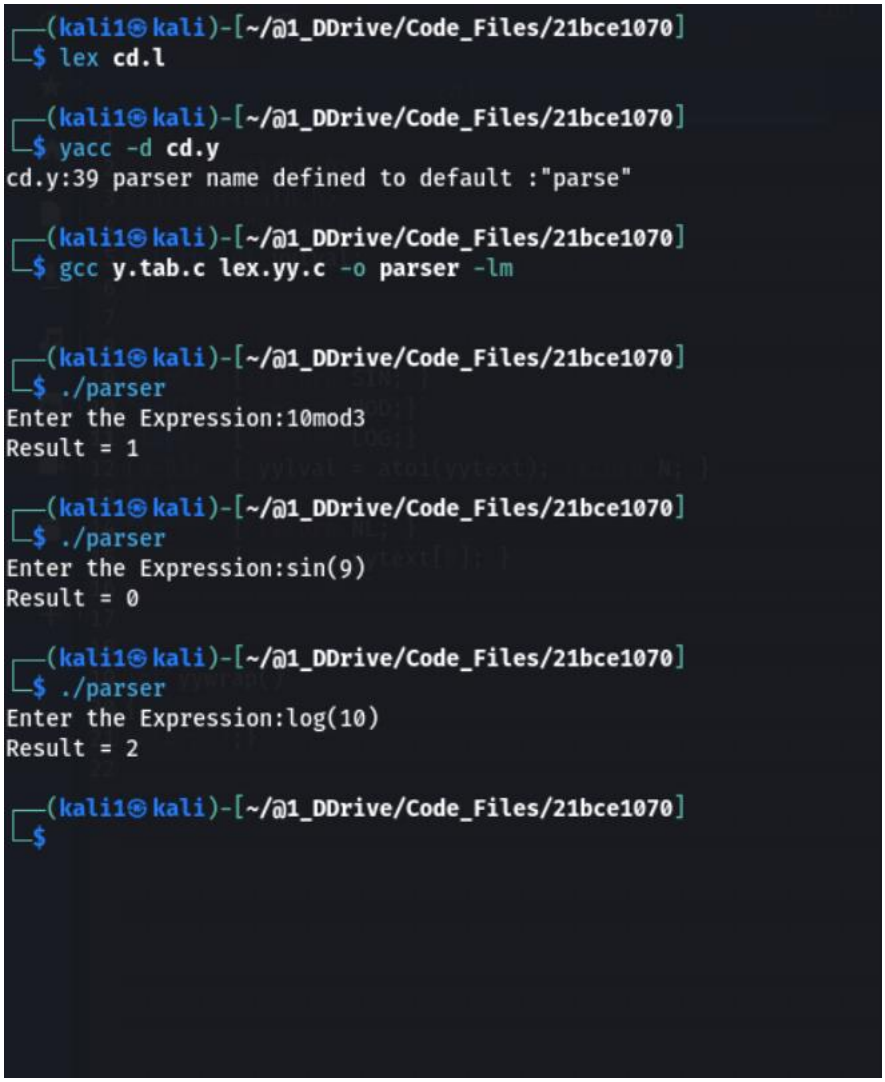
.l

```
%{
#include<stdio.h>
#include<math.h>
#include "y.tab.h"
extern int yylval;
%}

%%
"sin"   { return SIN; }
"mod"   { return MOD;}
"log"   { return LOG;}
[0-9]+  { yylval = atoi(yytext); return N; }
[\t]+   ;
"\n"    { return NL; }
.       { return yytext[0]; }
%%

int yywrap()
```

```
{
return 1;}



.y
%{
#include <stdio.h>
#include <math.h>

int yylex(void);
void yyerror(char *s);
%}

%token N NL SIN LOG MOD

%left '+' '-'
%left '*' '/'

%%
E   : T NL { printf("Result = %d\n", $1); return 0; }
    ;

T   : T '+' F   { $$ = $1 + $3; }
    | T '-' F   { $$ = $1 - $3; }
    | F         { $$ = $1; }
    ;

F   : F '*' G   { $$ = $1 * $3; }
    | F '/' G   { $$ = $1 / $3; }
    | G         { $$ = $1; }
    ;
```

```
G   : G MOD H   { $$ = fmod($1, $3); }
    | H         { $$ = $1; }
    ;

H   : '-' N     { $$ = -$2; }
    | '(' T ')' { $$ = $2; }
    | SIN '(' T ')' { $$ = sin($3); }
    | LOG '(' T ')' {$$ = log($3);}
    | N         { $$ = $1; }
    ;

%%
int main()
{
    printf("Enter the Expression:");
    yyparse();
    return 0;
}

void yyerror(char *s)
{
    printf("Invalid\n");
}
```

Output



Relational Expression

.l

%{

#include "y.tab.h"

%}

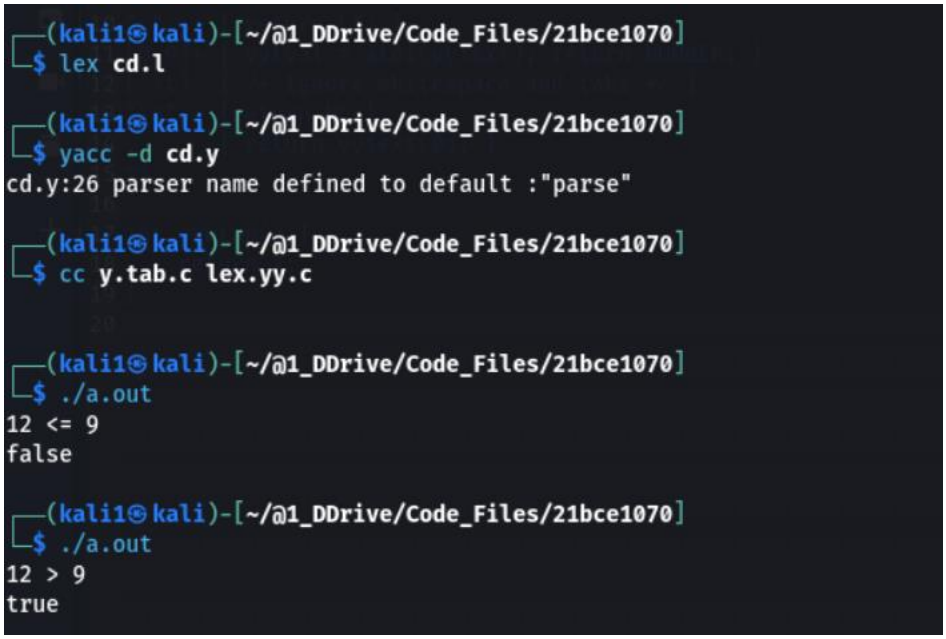
%%

">=" { return GTEQ; }

```
"<="   { return LTEQ; }
"=="   { return EQ; }
">"    { return GT; }
"<"    { return LT; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
[ \t]  { /* Ignore whitespace and tabs */ }
"\n"   {return NL;}
.      { return yytext[0]; }
%%

int yywrap() {
   return 1;
}



.y
%{
#include <stdio.h>
#include <stdbool.h>
int yylex();
void yyerror(const char* s);
bool result;
%}

%token GTEQ LTEQ EQ GT LT NUMBER NL

%left EQ
%left GT LT GTEQ LTEQ

%%
D : expression NL {return 0;}
```

```
expression:
    expression EQ expression       { result = ($1 == $3);}
    | expression GT expression    { result = ($1 > $3); }
    | expression LT expression    { result = ($1 < $3); }
    | expression GTEQ expression  { result = ($1 >= $3); }
    | expression LTEQ expression  { result = ($1 <= $3); }
    | NUMBER                      { result = $1; }
    ;

%%

void yyerror(const char* s) {
    printf("Error: %s\n", s);
}

int main() {
    yyparse();
    printf("%s\n", result ? "true" : "false");
    return 0;
}
```

Output


(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
$ lex cd.l
(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
$ yacc -d cd.y
cd.y:26 parser name defined to default :"parse"
(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
$ cc y.tab.c lex.yy.c
(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
$ ./a.out
12 <= 9
false
(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
$ ./a.out
12 > 9
true