EXAMPLE 28 *A Yacc specification to accept* $L = \{a^n b^n : n > 0\}$.

Solution.

```
/* anbn0.l */
%%
    a    return (A) ;
    b    return (B) ;
    .    return (yytext[0]) ;
    \n return ('\n') ;
%%
int yywrap( )    {    return 1 ;    }
/* anbn0.y */
%token A B
%%
    start :   anbn '\n' (return 0 ;)
    anbn :    A B
              |    A anbn B
              ;
%%
#include "lex.yy.c"
main( ) {
              return yyparse( ) ;
         }
int yyerror(char *s)   {fprintf(stderr, "%s\n", s); }
```

If the input stream does not match start, the default message of "syntax error" is printed and program terminates.

However, customized error messages can be generated.

```
/* anbn1.y */
%token A B
%%
start : anbn '\n' {    printf("is in anbn\n") ;
                       return 0 ; }
anbn :  A B
        | A anbn B
        ;
%%
#include "lex.yy.c"
yyerror(char *s)  {    printf("%s, it is not in anbn\n", s) ;    }
main( ) {
              return yyparse( ) ;
         }
```

Output :

```
$anbn
aabb
is in anbn
$anbn
acadbefbg
Syntax error, it is not in anbn
$
```

EXAMPLE 29 *Program to test the validity of a simple expression involving operators +, –, * and /.*

Solution.

Lex Specification

```
.....
%{
    #include "y.tab.h"
%}
%%
    [_a-zA-Z][_a-zA-Z0-9]* return ID ;
    [0-9]+
    return NUM ;
    return yytext[0] ;
    \n return 0 ;
%%
.....
```

Yacc Specification

```
%{
    #include
%}
%token NUM ID
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS
%%
    exp : exp '+' exp
    | exp '- ' exp
    | exp '/' exp
    | exp '*' exp
    | '-' exp %prec UMINUS
    | '(' exp ')'
    | NUM
    | ID
    ;
%%
```

```
int main( )
{   printf("\n Enter an expression :") ;
    yyparse( ) ;
    printf("\n valid expression") ;
    return 0 ;
}
void yyerror( )   {
    printf("\n Invalid expression") ;
    exit(0) ;
}
```

Output

```
lex |1.|
yacc -d y1.y
cc lex.yy.c y.tab.c -||
./a.out
Enter an expression : a + (b - c)/a
valid expression
```

EXAMPLE 30 *Evaluate the arithmetic expression +, –, *, /*

Solution.

Lex Specification

```
.....
%{
      #include "y.tab.h"
%}
%%
([0 - 9] + [0 - 9] *\.[0 - 9] +)+    {yylval.fval = atof(yytext) ; return NUM ;}
return yytext[0] ;
\n return 0 ;
%%
```

Yacc specification

```
.....
%{
      #include
 %}
 %union  {    float fval ;    }
 %token NUM
 %type e
 %type start
 %left '+' '-'
```

```
%left '*' '/'
%nonassoc UMINUS
%%
start:e{  printf("=%2.2f", $$) ; }
;
e:e '+' e{$$ = $1 + $3 ; }
| e '-' e{ $$ = $1 - $3 ; }
| e '*' e{ $$ = $1 * $3 ; }
| e '/' e{ if($3 == 0)}
yyerror("divided by zero") ;
else
$$ = $1/$3 ;
}
| '-' e %prec UMINUS{$$ =-$2 ; }
| '('e')' { $$ = $2 ; }
| NUM{$$ = $1 ; }
;
%%
int main( )
{     printf("\n enter arithmetic expression\n") ;
      yyparse( ) ;
      prinft("\n valid expression\n") ;
      return 0 ;
}

void yyerror( )
{     printf("\n invalid expression\n") ;
      exit(0) ;
}
```

```
lex |file5.l
yacc -d yfile.y
cc lex.yy.c y.tab.c -||
./a.out
Enter arithmetic expression
1 + 5 * 4
= 21.00
valid expression
```

EXERCISES

1. Explain the basic parsing techniques. What is top-down parsing ?
2. Differentiate between top-down parser and bottom-up parser.
3. What do you mean by left recursion and how it is eliminated ?