

## Problem 1.

(b) Show that the grammar for CHAIN given above is ambiguous.

Grammar for CHAIN is defined in BNF as below:

```
S -> E                (r1)
S -> EMPTY            (r2)
E -> E # E            (r3)
E -> STRING           (r4)
E -> REVERSE ( STRING ) (r5)
```



There are two parse trees derived from the same grammar. Therefore, this grammar is ambiguous.

```
1  A # B # Reverse(C)
2  . A # B # Reverse(C)    shift
3  A . # B # Reverse(C)    reduce (r1)
4  E . # B # Reverse(C)    shift
5  E # . B # Reverse(C)    shift
6  E # B . # Reverse(C)    reduce (r1)
7  E # E . # Reverse(C)    shift <-
8  E # E # . Reverse(C)    shift
9  E # E # Reverse(C) .    reduce (r5)
10 E # E # E .            reduce (r3)
11 E # E                  reduce (r3)
12 E
```

On Step 7, there is a conflict between whether to shift or reduce by applying rule (r3). Due to this shift-reduce conflict, our grammar is ambiguous.

(c) Find an equivalent grammar (i.e., one that generates the same language) that is not ambiguous.

To solve shift-reduce conflict we would need to rewrite our grammar or define operator precedence to our yacc rules.

The equivalent grammar is as follows:

```
S -> E
```

```

S -> EMPTY
E -> T
E -> E # T | T
T -> STRING
T -> REVERSE ( STRING )

```

```

%token <str>  STRING
%token <str>  REVERSE

/* This defines higher precedence for # operation, which is another way to solve
ambiguity*/
/* %left '#' */

%type <str>  start
%type <str>  exprs
%type <str>  expr
%start  start

%%
/* rules section */

start      :  exprs '\n'          { printf("%s\n", ); }
           |  /* allow "empty" expression */          { }
           ;

exprs      :  expr { }
           |  exprs '#' expr { $$ = strcat(,); }
           ;

expr       :  STRING              { $$ = ; }
           |  REVERSE '(' STRING ')' { $$ = reverse(); }
           ;

%%
/* programs */

```

Note: The %left '#' below defines # operator with higher precedence.