1. Diseñe una gramática para un traductor ascendente.

begin → begin expr ';'
begin → epsilon
expr → ID EQUAL ar_expr
expr → PRINT ar_expr
ar_expr → ar_expr OR conc
ar_expr → conc
conc → conc AND disy
conc → disy
disy → NOT disy
disy → ID
disy → TRUE
disy → FALSE
disy → '(' ar_expr ')'

2. Adapte la gramática del apartado anterior para un traductor descendente.

begin → expr ';' begin
begin → epsilon
expr → ID EQUAL ar_expr
expr → PRINT ar_expr
ar_expr → conc ar_expr'
ar_expr ' → OR conc ar_expr'
ar_expr ' → epsilon
conc → disy conc'
conc' → AND disy conc'
conc' → epsilon
disy → NOT disy
disy → ID
disy → TRUE
disy → FALSE
disy → '(' ar_expr ')'

3. Añada el esquema de traducción a la gramática del apartado 2.

begin → expr ';' begin
begin → epsilon
expr → ID EQUAL ar_expr {ID.lexval = ar_expr.s}
expr → PRINT ar_expr { std::cout << "Resultado es "<< ar_expr.s ; }
ar_expr → conc {ar_expr'.h = conc.s} ar_expr' {ar_expr.s = ar_expr'.s}
ar_expr ' → OR conc {ar_expr1'.h = conc.s} ar_expr1' {ar_expr'.s = ar_expr1'.s}
ar_expr ' → epsilon {ar_expr'.s = ar_expr'.h}
conc → disy { conc'.h = disy.s } conc' {conc.s = conc'.s}
conc' → AND disy { conc1'.h = disy.s } conc1' {conc'.s = conc1'.s}
conc' → epsilon {conc'.s = conc'.h}
disy → NOT disy1 {disy.s = ! disy1.s}
disy → ID {disy.s = ID.lexval}
disy → TRUE {disy.s = 1}
disy → FALSE {disy.s = 0}
disy → '(' ar_expr {disy.s = ar_expr.s} ')'