LL(1) grammar ('' is ε):

```
Prog -> ''
Prog -> Decl Prog
Decl -> TypeDecl
Decl -> TermDecl
TypeDecl -> VAR :: Type ;
Type -> Type0 TypeRest
TypeRest -> ''
TypeRest -> a Type
Type0 -> Integer
Type0 -> Bool
Type0 -> ( Type )
TermDecl -> VAR Args = Exp ;
Args -> ''
Args -> VAR Args
Exp -> Exp0
Exp -> if Exp then Exp else Exp
Exp0 -> Exp1 Rest0
Rest0 -> ''
Rest0 -> == Exp1
Rest0 -> <= Exp1
Exp1 -> Exp2 Rest1
Rest1 -> ''
Rest1 -> + Exp2 Rest1
Rest1 -> - Exp2 Rest1
Exp2 -> Exp3 Rest2
```

>>

FIRST	FOLLOW	Nonterminal	VAR	::
{'',VAR}	{$}	Prog	Prog -> Decl Prog	
{VAR}	{$,VAR}	Decl	Decl -> TypeDecl Decl -> TermDecl	
{VAR}	{$,VAR}	TypeDecl	TypeDecl -> VAR :: Type ;	
{Integer,Bool,(}	{;,)}	Type		
{'',a}	{;,)}	TypeRest		
{Integer,Bool,(}	{;,a,)}	Type0		
{VAR}	{$,VAR}	TermDecl	TermDecl -> VAR Args = Exp ;	
{'',VAR}	{=}	Args	Args -> VAR Args	
{if,VAR,NUM,BOOLEAN,(}	{;,then,else,)}	Exp	Exp -> Exp0	
{VAR,NUM,BOOLEAN,(}	{;,then,else,)}	Exp0	Exp0 -> Exp1 Rest0	
{'',==,<=}	{;,then,else,)}	Rest0		
{VAR,NUM,BOOLEAN,(}	{;,==,<=,then,else,)}	Exp1	Exp1 -> Exp2 Rest1	
{'',+,-}	{;,==,<=,then,else,)}	Rest1		
{VAR,NUM,BOOLEAN,(}	{;,==,<=,+,-,then,else,)}	Exp2	Exp2 -> Exp3 Rest2	
{'',VAR,NUM,BOOLEAN,(}	{;,==,<=,+,-,then,else,)}	Rest2	Rest2 -> Exp3 Rest2	
{VAR,NUM,BOOLEAN,(}	{;,==,<=,+,-,VAR,NUM,BOOLEAN,(,then,else,)}	Exp3	Exp3 -> VAR	

Maximum number of steps: 100

Input (tokens): id + id

GO!

Trace			Tree
Stack	Input	Rule	
$ Prog	id + id $		