C1) This is a recursive descent parser. Write the grammar from this parser.

```
block()
  match('{')
  stmt()
  match('}')

stmt()
  if( currenttoken == 'id')
    stmt1()
    stmt()
  ε
stmt1()
  match('id')
  match('=')
  expr()
  match(';')

expr()
  match('id')
  exprs()

exprs()
  if( currenttoken == '+')
    match('+')
    exprs()
```

block → { stmt }

ignore this, add

stmt → stmt1 stmt | ε

if clause, not true → ε

stmt1 → id = expr ;

expr → id exprs

exprs → + exprs | ε

C2) Given this grammar, compute First and Follow set, draw the parsing table

```
dcl = ID dcl2
dcl2 =  ( formal ) stmt |  [ NUM ]
formal = ID formals | empty
formals = , formal | empty
```

	First
formals	{,, ε}
formal	{ID, ε}
dcl2	{(, [}
dcl	{ID}

	First	Follow
dcl	{ID}	{$}
dcl2	{(, [}	{$}
formal	{ID, ε}	{)}
formals	{,, ε}	{]}

Should include all Non-Term + Term

① dcl → ID dcl2
② dcl2 → (formal) stmt
③ dcl2 → [NUM]
④ formal → ID formals
⑤ formal → ε
⑥ formals → , formal
⑦ formals → ε

	ID	(	)	[	]
dcl	① dcl → ID dcl2				
dcl2		② dcl2 → (formal) stmt		③ dcl2 → [NUM]	
formal	④ formal → ID formals		⑤ formal → ε		
formals			⑦ formals → ε		⑥ formals → , formal