SOLUTIONS - CONSTRUCTION OF A RECURSIVE DESCENT PARSER:

Consider the following grammar:

<exp>	::=	<atom> | <list>
<atom>	::=	<digit> | <string>
<list>	::=	(<expr-list>)
<expr-list>	::=	<exp> <expr-list> | ε
<digit>	::=	0 | 1 | 2 | ... | 9
<string>	::=	a | b | c

1.What change would you make to transform it into EBNF?

<exp-list> ::= { <expr> }

2.Write a Recursive descent parser corresponding to this parser.

	FIRST	FOLLOW
<exp>	FIRST (<atom>) ∪ FIRST (<list>) { 0, 1, 2, ..., 9, a, b, c, (}	FIRST (<expr-list>) ∪ FOLLOW (<expr-list>) { $, 0, 1, 2, ..., 9, a, b, c, (,) }
<atom>	FIRST (<digit>) ∪ FIRST (<string>) { 0, 1, 2, ..., 9, a, b, c }	FOLLOW (<expr>) { $, 0, 1, 2, ..., 9, a, b, c, (,) }
<list>	{ (}	FOLLOW (<expr>) { $, 0, 1, 2, ..., 9, a, b, c, (,) }
<exp-list>	FIRST (<exp>) { 0, 1, 2, ..., 9, a, b, c, (, ε }	{) }
<digit>	{ 0, 1, 2, ..., 9 }	FOLLOW (<atom>) { $, 0, 1, 2, ..., 9, a, b, c, (,) }
<string>	{ a, b, c }	FOLLOW (<atom>) { $, 0, 1, 2, ..., 9, a, b, c, (,) }

Is this grammar suitable for recursive descent?

- <exp> FIRST (<atom>) ∩ FIRST (<list>) = ∅
- <atom> FIRST (<digit>) ∩ FIRST (<string>) = ∅
- <exp-list> FIRST (<expr>) ∩ FOLLOW(<exp-list>) = ∅

```
exp ( )
{    if token == a ^ b ^ ...^ 9
          atom ( )
     else if token == (
          list ( )
     else
          error ( )
}

atom ( )
{    if token == 0 ^ 1 ^ ... ^ 9
          digit ( 0
     else if token == a ^ b ^ c
          string ( )
     else
          error ( )
}
```

```
list ( )
{    if token == (
          match ( ( )
          explist ( )
          if token == )
               match ( ) )
          else
               error ( )
}
     else
          error ( )
}
```

```
explist ( )
{    while ( token == 0 ^ ...^ b ^ c )
          exp ( )
}

digit ( )
{    if token == 0 ^ 1 ^ ... ^ 9
          match (token )
     else
          error ( )
}

string ( )
{    if token == a ^ b ^ c
          match ( token )
}
```