1 To LL(1) grammar

1.1 Unproductive characters

Considering that the set of terminals corresponds to the units contained in the Symbol Class

I	V_i
0	ϕ
1	{<Read>}
2	{<Read>, <Print> }
3	{<Read>, <Print>, <While>}
4	{ <Read>, <Print>, <While>, <Comp> }
5	{<Read>, <Print>, <While>, <Comp> <Cond>}
6	{<Read>, <Print>, <While>, <Comp> <Cond>,<If>}
7	{<Read>, <Print>, <While>, <Comp> <Cond>,<If>,<Op>}
8	{<Read>, <Print>, <While>, <Comp> <Cond>,<If>,<Op>,<ExprArith>}
9	{<Read>, <Print>, <While>, <Comp> <Cond> ,<If> ,<Op> ,<ExprArith> ,<Assign>}
10	{<Read>, <Print>, <While>, <Comp> <Cond>, <If>,<Op>, <ExprArith> , <Assign> , <Instruction>}
12	{<Read>, <Print>, <While>, <Comp> <Cond>,<If> ,<Op>, <ExprArith>, <Assign> , <Instruction>,<Code>}
13	{<Read>, <Print>, <While>, <Comp> <Cond>, <If>,<Op>, <ExprArith>, <Assign> , <Instruction> , <Code>, <Program>}

Observation: There are no unproductive rules in the given grammar

1.2 Unreachable characters

I	V_i
0	{<Program>}
1	{<Program> , <Code>}
2	{<Program>, <Instruction>}
3	{<Program>,<Instruction>, <Assign> ,<While> ,<Print>,<Read>}
4 5	{<Program>,<Instruction>, <Assign>,<While>,<Print>,<Read>,<ExprArith>}
6	{<Program>,<Instruction>, <Assign>,<While>,<Print>,<Read>,<ExprArith>,<Op>}
7	{<Program>,<Instruction>, <Assign>,<While>,<Print>,<Read>,<ExpArith>,<Op>, <If>}
8	{<Program>,<Instruction>, <Assign>,<While>,<Print>,<Read>,<ExprArith>,<Op>, <If>}
9	{<Program>,<Instruction>, <Assign>,<While>,<Print>,<Read>,<ExprArith>,<Op>,<If>,<Cond>}
	{<Program>,<Instruction>,<Assign>,<While>,<Print>,<Read>,<ExprArith>,<Op>, <If>,<Code>}

Observation: There are no unreachable rules in the given grammar

To sum up, there are no useless rules

1.3 Ambiguous grammar

In the given grammar, there is ambiguity regarding the addition and multiplication operation which need to be adjusted according to their order

```
<ExprArith> → [VarName]
            → [Number]
            → ( <ExprArith> )
            → - <ExprArith>
            → <ExprArith> <Op> <ExprArith>
 <Op> → +
      → -
      → *
      → /
```

After the adjustment by prioritizing the multiplication and division, the above grammar is obtained

```
<ExprArith> -> <ExprArith> + <Multiplication>
           -> <ExprArith> - <Multiplication>
           -> <Multiplication>
<Multiplication> -> <Multiplication> * <Bracket>
                 -> <Multiplication> / <Bracket >
                 -> <Braquet>
<Bracket> -> (<ExprArith>)
          -> <Var>

<Var> -> [VarName]
      → [Number]
      → - <Var>
```

1.4 Left Factory

Left factor is noticeable in the following rules:

```
<If> → IF (<Cond>) THEN [EndLine] <Code> ENDIF
<If> → IF (<Cond>) THEN [EndLine] <Code> ELSE [EndLine] <Code> ENDIF
```

After the removal of the left factory we get

```
<If> → IF (<Cond>) THEN [EndLine] <Code> <If”>
<If”> -> ENDIF
<If”> → ELSE [EndLine] <Code> ENDIF
```

1.5 Left recursion

Left recursion is noticed in the rule bellow:

```
<ExprArith> -> < ExprArith > + <Multiplication>
           -> < ExprArith > - <Multiplication>
           -> <Multiplication>
<Multiplication> -> <Multiplication> * <Braquet>
                 -> <Multiplication> / <Braquet >
                 -> <Braquet>
```

After the removal of left recursion:

```
< ExprArith > -> <ExprArith’> < ExprArith’’>
< ExprArith’’> -> + <Multiplication>< ExprArith’’>
               -> - <Multiplication>< ExprArith’’>
               ->  ε
<ExprArith’> -> <Multiplication>


<Multiplication> -> <Multiplication’> <Multiplication’’>
<Multiplication’’> -> * <Braquet> <Multiplication’’>
                   -> / <Braquet> <Multiplication’’>
                   ->ε
 <Multiplication’>-> <Bracket>
```

In the end, the follow grammar is obtained

```
[1] <S>  -> <Program>$
[2] <Program> → BEGINPROG [ProgName] [EndLine] <Code> ENDPROG
[3] <Code> → <Instruction> [EndLine] <Code>
[4]        → ε
[5] <Instruction> → <Assign>
[6]               → <If>
[7]               → <While>
[8]               → <Print>
[9]               → <Read>
[10] <Assign> → [VarName] := <ExprArith>
[11] <ExprArith > -> <ExprArith’> < ExprArith’’>
[12] <ExprArith’’> -> + <Multiplication>< ExprArith’’>
[13]               -> - <Multiplication>< ExprArith’’>
[14]               ->  ε
[15] <ExprArith’> -> <Multiplication>
[16] <Multiplication> -> <Multiplication’> <Multiplication’’>
[17] <Multiplication’’> -> * <Braquet> <Multiplication’’>
[18]                   -> / <Braquet> <Multiplication’’>
[19]                   ->ε
[20] <Multiplication’> -> <Bracket>
[21] <Bracket> -> (ExprArith)
[22]          -> <Var>
[23] <Var> -> [VarName]
[24]       → [Number]
[25]       → - <Var>
[26] <If> → IF (<Cond>) THEN [EndLine] <Code> <If”>
[27] <If”> -> ENDIF
[28] <If”> → ELSE [EndLine] <Code> ENDIF
[29] <Cond> → <ExprArith> <Comp> <ExprArith>
[30] <Comp> → =
[31]        → >
[32] <While> → WHILE (<Cond>) DO [EndLine] <Code> ENDWHILE
[33] <Print> → PRINT([VarName])
[34] <Read> → READ([VarName])
```

1.6 First^1 and Follow^1

Symbol	First^1	Follow^1
<S>	BEGINPROG	
<Program>	BEGINPROG	$
<Code>	VarName IF WHILE PRINT READ ε	ENDPROG ENDIF ELSE ENDWHILE
<Instruction>	VarName IF WHILE PRINT READ	ENDLINE
<Assign>	VarName	ENDLINE
<ExprArith>	(VarName Number -	ENDLINE)
<ExprArith’’>	+ - ε	ENDLINE)
<ExprArith’>	(VarName Number -	+ -
<Multiplication>	(VarName Number -	+ -
<Multiplication’’>	/ * ε	+ -
<Multiplication’>	(VarName Number -	/ *
<Bracket>	(VarName Number -	/ *
<Var>	VarName Number -	/ *
<If>	IF	ENDLINE
<If”>	ENDIF , ELSE	ENDLINE
<Cond>	(VarName Number -	)
<Comp>	= >	(VarName Number -
<While>	WHILE	ENDLINE
<Print>	PRINT	ENDLINE
<Read>	READ	ENDLINE

1.7 Action Table Generation

	BEGINPROG	PROGNAME	(	)	+	-	*	/	=	>	IF	ELSE	ENDIF	WHILE	ENDWHILE	VARNAME	Number	PRINT	READ	ENDPROG	ENDLINE
<S>	1																				
<Program>	2																				
<CODE>											3	4	4	3	4	3		3	3	4	
<Instruction>											6			7		5		8	9		
<Assign>																10					
<ExprArith>			11			11										11	11				
<ExprArith’’>				14	12	13															14
<ExprArith’>			15			15										15	15				
<Multiplication>			16			16										16	16				
<Multiplication’’>					19	19	17	18													
<Multiplication’>			20			20										20	20				
<Bracket>			21			22										22	22				
<Var>						25										23	24				
<If>											26										
<If’’>											27	28									
<Cond>			29			29										29	29				
<Comp>									30	31											
<While>														32							
<Print>																		33			
<Read>																			34		