Consider the grammar below with few notations different from what we have learnt so far

Notations

1. Any item enclosed in braces {} are Non-terminals
2. A notation of {} is considered as epsilon ϵ or null production
3. An item followed by parenthesis () is a token type
    a. keyword("program") : here keyword is a token type and program is the keyword
    b. sym(".") : here sym is a token type and . is a symbol or a lexeme
    c. id(_) : here id is a token type and _ is any identifier
    d. stringConst is a string constant surrounded by ""
    e. intConst is an integer constant consists of decimal digits with no decimal point
    f. readConst is a real constant consists of decimal digits with a decimal point
4. :: is a production rule symbol, which we have generally learnt as →

Mini Pascal Language Grammar:

```
{program} :: keyword("program") id(_) sym(";") {var_part} {block} sym(".")

{var_part} :: keyword("var") {var_list}
{var_part} :: {}

{var_list} :: {var_list} id(_) sym(":") {type} sym(";")
{var_list} :: id(_) sym(":") {type} sym(";")

{type} :: keyword("real")
{type} :: keyword("integer")

{block} :: keyword("begin") {stmt_list} keyword("end")

{stmt_list} :: {stmt_list} {stmt} sym(";")
{stmt_list} :: {}

{stmt} :: id(_) sym(":=") {expr}
{stmt} :: keyword("if") {expr} keyword("then") {block}
{stmt} :: keyword("if") {expr} keyword("then") {block} keyword("else") {block}
{stmt} :: keyword("while") {expr} keyword("do") {block}
{stmt} :: keyword("repeat") {block} keyword("until") {expr}
{stmt} :: keyword("readln") sym("(") id(_) sym(")")
{stmt} :: keyword("write") sym("(") {expr} sym(")")
{stmt} :: keyword("writeln") sym("(") {expr} sym(")")
{stmt} :: keyword("write") sym("(") stringConst(_) sym(")")
{stmt} :: keyword("writeln") sym("(") stringConst(_) sym(")")

{expr} :: {expr} sym("+") {term}
{expr} :: {expr} sym("-") {term}
{expr} :: {expr} keyword("or") {term}
{expr} :: {term}

{term} :: {term} sym("*") {factor}
{term} :: {term} sym("/") {factor}
{term} :: {term} keyword("div") {factor}
{term} :: {term} keyword("mod") {factor}
{term} :: {term} keyword("and") {factor}
{term} :: {factor}
```

```
{factor} :: intConst(_)
{factor} :: realConst(_)
{factor} :: id(_)
{factor} :: keyword("true")
{factor} :: keyword("false")
{factor} :: sym("(") {expr} sym(")")
{factor} :: sym("(") {expr} {relop} {expr} sym(")")
{factor} :: keyword("not") sym("(") {expr} sym(")")
{factor} :: keyword("trunc") sym("(") {expr} sym(")")
{factor} :: keyword("real") sym("(") {expr} sym(")")

{relop} :: sym("=")
{relop} :: sym(">")
{relop} :: sym("<")
{relop} :: sym("<=")
{relop} :: sym(">=")
{relop} :: sym("<>")
```