Grammaire

- PROGRAM ::= program ID ; BLOCK .
- BLOCK ::= CONSTS VARS INSTS
- CONSTS ::= const ID = NUM ; { ID = NUM ; } | ε
- VARS ::= var ID { , ID } ; | ε
- INSTS ::= begin INST { ; INST } end
- INST ::= AFFEC | SI | TANTQUE | ECRIRE | LIRE | FUNC_CALL | FOR_STMT | REPEAT_STMT | ε
- AFFEC ::= ID := EXPR
- SI ::= if COND then INST [else INST]
- TANTQUE ::= while COND do INST
- ECRIRE ::= write (EXPR { , EXPR })
- LIRE ::= read (ID { , ID })
- COND ::= EXPR RELOP EXPR
- RELOP ::= = | <> | < | > | <= | >=
- EXPR ::= TERM { ADDOP TERM }
- ADDOP ::= + | -
- TERM ::= FACT { MULOP FACT }
- MULOP ::= * | /
- FACT ::= ID | NUM | (EXPR)
- CASE_STMT ::= case EXPR of CASE_BRANCHES [else INST] end ;
- CASE_BRANCHES ::= CASE_BRANCH { ; CASE_BRANCH }
- CASE_BRANCH ::= NUM : INST
- FUNC_DECL ::= function ID (PARAMS) : TYPE ; FUNC_BODY
- FUNC_BODY ::= begin INSTS end
- PARAMS ::= PARAM { ; PARAM }
- PARAM ::= ID { , ID } : TYPE
- TYPE ::= Integer | Real
- FUNC_CALL ::= ID (ARG_LIST)
- ARG_LIST ::= EXPR { , EXPR }
- PROC_DECL ::= procedure ID (PARAMS) ; PROC_BODY
- PROC_BODY ::= begin INSTS end
- FOR_STMT ::= for ID := EXPR TO EXPR DO INST
- TO ::= to | downto
- REPEAT_STMT ::= repeat INSTS until COND
- REAL ::= INTEGER FRACTION
- INTEGER ::= [+ | -] DIGIT { DIGIT }
- FRACTION ::= . DIGIT { DIGIT }
- DIGIT ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- ID -> LETTER { LETTER | DIGIT | "_" }
- LETTER -> "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"