Quick - analiza de domeniu

// Reguli semantice pentru analiza de domeniu:

1. domeniul global începe de la începutul programului și durează până la sfârșitul acestuia
2. la '(' de la argumentele unei funcții începe domeniul local al acelei funcții, care durează până la **end**-ul funcției
3. într-un domeniu nu au voie să există mai multe simboluri cu același nume

```
program ::= {
        adaugaDomeniu();                // adauga domeniul global in TS
        }
        ( defVar | defFunc | block )* FINISH
        {
        stergeDomeniu();                // sterge domeniul global din TS
        }
defVar ::= VAR ID
        {
        const char *nume=consumed->s;
        Simbol *s=cautaInDomeniulCurent(nume);
        if(s)tkerr("redefinire simbol: %s",nume);
        s=adaugaSimbol(nume,FEL_VAR);
        s->local=crtFn!=NULL;
        }
        COLON baseType
        {
        s->tip=ret.tip;
        }
        SEMICOLON
baseType ::= TYPE_INT
        {
        ret.tip=TYPE_INT;
        }
        | TYPE_REAL
        {
        ret.tip=TYPE_REAL;
        }
        | TYPE_STR
        {
        ret.tip=TYPE_STR;
        }
defFunc ::= FUNCTION ID
        {
        const char *nume=consumed->s;
        Simbol *s=cautaInDomeniulCurent(nume);
        if(s)tkerr("redefinire simbol: %s",nume);
        crtFn=adaugaSimbol(nume,FEL_FN);
        crtFn->args=NULL;
        adaugaDomeniu();
        }
        LPAR funcParams RPAR COLON baseType
        {
        crtFn->tip=ret.tip;
        }
        defVar* block END
        {
```

```
        stergeDomeniu();
        crtFn=NULL;
        }
block ::= instr+
funcParams ::= ( funcParam ( COMMA funcParam )* )?
funcParam ::= ID
        {
        const char *nume=consumed->s;
        Simbol *s=cautaInDomeniulCurent(nume);
        if(s)tkerr("redefinire simbol: %s",nume);
        s=adaugaSimbol(nume,FEL_ARG);
        Simbol *argFn=adaugaArgFn(crtFn,nume);
        }
        COLON baseType
        {
        s->tip=ret.tip;
        argFn->tip=ret.tip;
        }
instr ::= expr? SEMICOLON
        | IF LPAR expr RPAR block ( ELSE block )? END
        | RETURN expr SEMICOLON
        | WHILE LPAR expr RPAR block END
expr ::= exprLogic
exprLogic ::= exprAssign ( ( AND | OR ) exprAssign )*
exprAssign ::= ( ID ASSIGN )? exprComp
exprComp ::= exprAdd ( ( LESS | EQUAL ) exprAdd )?
exprAdd ::= exprMul ( ( ADD | SUB ) exprMul )*
exprMul ::= exprPrefix ( ( MUL | DIV ) exprPrefix )*
exprPrefix ::= ( SUB | NOT )? factor
factor ::= INT
        | REAL
        | STR
        | LPAR expr RPAR
        | ID ( LPAR ( expr ( COMMA expr )* )? RPAR )?
```