Quick - analiza de tipuri

// Reguli semantice pentru analiza de tipuri:

1. simbolurile trebuie definite anterior folosirii lor
2. funcțiile se pot doar apela
3. un apel de funcție trebuie să aibă același număr de argumente cu cel de la definirea funcției
4. argumentele de la apelul funcției trebuie să aibă tipurile identice cu cele de la definire
5. o variabilă nu se poate apela ca funcție
6. tipul returnat de return trebuie să fie identic cu cel returnat de funcție
7. return poate exista doar într-o funcție
8. condițiile pentru if și while trebuie să aibă tipul int sau real
9. nu există conversii implicite între tipurile de date, nici măcar între int și real. Din acest motiv operanzii operatorilor aritmetici binari și de comparație (+ - * / < ==) trebuie să aibă tipuri identice. Operanzii operatorilor logici binari (&& ||) pot avea tipuri diferite.
10. operatorii aritmetici și logici (+ - (inclusiv unar) * / && || !) sunt definiți doar pentru tipurile int și real
11. atribuirea și operatorii de comparație (= < ==) sunt definiți pentru toate tipurile
12. operatorii de comparatie și logici (&& || ! < ==) returnează int 0 sau 1 (false/true)

program ::=
{
adaugaFnPredefinite(); // se insereaza dupa codul de la analiza de domeniu
}
(**defVar** | **defFunc** | **block**)* FINISH
defVar ::= VAR ID COLON **baseType** SEMICOLON
baseType ::= TYPE_INT | TYPE_REAL | TYPE_STR
defFunc ::= FUNCTION ID LPAR **funcParams** RPAR COLON **baseType** **defVar*** **block** END
block ::= **instr**+
funcParams ::= (**funcParam** (COMMA **funcParam**)*)?
funcParam ::= ID COLON **baseType**
instr ::= **expr**? SEMICOLON
| IF LPAR **expr**
{
if(ret.tip==TYPE_STR)tkerr("conditia lui if trebuie sa aiba tipul int sau real");
}
RPAR **block** (ELSE **block**)? END
| RETURN **expr**
{
if(!crtFn)tkerr("return poate fi folosit doar intr-o functie");
if(ret.tip!=crtFn->tip)tkerr("tipul lui return este diferit de tipul returnat de functie");
}
SEMICOLON
| WHILE LPAR **expr**
{
if(ret.tip==TYPE_STR)tkerr("conditia lui while trebuie sa aiba tipul int sau real");
}
RPAR **block** END
expr ::= **exprLogic**
exprLogic ::= **exprAssign** ((AND | OR)
{
Ret tipStanga=ret;
if(tipStanga.tip==TYPE_STR)tkerr("operandul stang al lui && sau || nu poate fi de tip str");
}
exprAssign
{
if(ret.tip==TYPE_STR)tkerr("operandul drept al lui && sau || nu poate fi de tip str");

```
        setRet(TYPE_INT,false);
        }
        )*
exprAssign ::= ID
        {
        const char *nume=consumed->s;
        }
        ASSIGN exprComp
        {
        Simbol *s=cautaSimbol(nume);
        if(!s)tkerr("identificator necunoscut: %s",nume);
        if(s->tip==FEL_FN)tkerr("o functie (%s) nu poate fi folosita ca destinatie a unei atribuiri",nume);
        if(s->tip!=ret.tip)tkerr("sursa si destinatia atribuirii au tipuri diferite");
        ret.lval=false;
        }
        | exprComp
exprComp ::= exprAdd ( ( LESS | EQUAL )
        {
        Ret tipStanga=ret;
        }
        exprAdd
        {
        if(tipStanga.tip!=ret.tip)tkerr("tipuri diferite pentru operanzii lui < sau ==");
        setRet(TYPE_INT,false);        // rezultatul comparatiei este int 0 sau 1
        }
        )?
exprAdd ::= exprMul ( ( ADD | SUB )
        {
        Ret tipStanga=ret;
        if(tipStanga.tip==TYPE_STR)tkerr("operanzii lui + sau - nu pot fi de tip str");
        }
        exprMul
        {
        if(tipStanga.tip!=ret.tip)tkerr("tipuri diferite pentru operanzii lui + sau -");
        ret.lval=false;
        }
        )*
exprMul ::= exprPrefix ( ( MUL | DIV )
        {
        Ret tipStanga=ret;
        if(tipStanga.tip==TYPE_STR)tkerr("operanzii lui * sau / nu pot fi de tip str");
        }
        exprPrefix
        {
        if(tipStanga.tip!=ret.tip)tkerr("tipuri diferite pentru operanzii lui * sau /");
        ret.lval=false;
        }
        )*
exprPrefix ::= SUB factor
        {
        if(ret.tip==TYPE_STR)tkerr("expresia lui - unar trebuie sa aiba tipul int sau real");
        ret.lval=false;
        }
        | NOT factor
        {
        if(ret.tip==TYPE_STR)tkerr("expresia lui ! trebuie sa aiba tipul int sau real");
```

```
        setRet(TYPE_INT,false);
        }
        | factor
factor ::= INT
        {
        setRet(TYPE_INT,false);
        }
        | REAL
        {
        setRet(TYPE_REAL,false);
        }
        | STR
        {
        setRet(TYPE_STR,false);
        }
        | LPAR expr RPAR
        | ID
        {
        Simbol *s=cautaSimbol(consumed->s);
        if(!s)tkerr("identificator necunoscut: %s",consumed->s);
        }
        ( LPAR
                {
                if(s->fel!=FEL_FN)tkerr("%s nu poate fi apelata, deoarece nu este o functie",consumed->s);
                Simbol *argDef=s->args;
                }
                ( expr
                        {
                        if(!argDef)tkerr("functia %s este apelata cu prea multe argumente",s->nume);
                        if(argDef->tip!=ret.tip)tkerr("tipul argumentului de la apelul functiei %s este diferit de cel de
                la definirea ei",s->nume);
                        argDef=argDef->urm;
                        }
                        ( COMMA expr
                                {
                                if(!argDef)tkerr("functia %s este apelata cu prea multe argumente",s->nume);
                                if(argDef->tip!=ret.tip)tkerr("tipul argumentului de la apelul functiei %s este diferit de
                        cel de la definirea ei",s->nume);
                                argDef=argDef->urm;
                                }
                                )* )? RPAR
                                        {
                                        if(argDef)tkerr("functia %s este apelata cu prea putine
                                argumente",s->nume);
                                        setRet(s->tip,false);
                                        }
                                        |
                                        {
                                        if(s->fel==FEL_FN)tkerr("functia %s se poate doar apela",s->nume);
                                        setRet(s->tip,true);
                                        }
                                        )
```