

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 existe mai multe simboluri cu același nume

```
program ::= {
    addDomain();           // creates the global domain
}
( defVar | defFunc | block ) * FINISH
{
    stergeDomeniu();       // deletes the global domain
}

defVar ::= VAR ID
{
    const char *name=consumed->text;
    Symbol *s=searchInCurrentDomain(name);
    if(s)tkerr("symbol redefinition: %s",name);
    s=addSymbol(name,KIND_VAR);
    s->local=crtFn!=NULL;
}
COLON baseType
{
    s->type=ret.type;
}
SEMICOLON

baseType ::= TYPE_INT
{
    ret.type=TYPE_INT;
}
| TYPE_REAL
{
    ret.type=TYPE_REAL;
}
| TYPE_STR
{
    ret.type=TYPE_STR;
}

defFunc ::= FUNCTION ID
{
    const char *name=consumed->text;
    Symbol *s=searchInCurrentDomain(name);
    if(s)tkerr("symbol redefinition: %s",name);
    crtFn=addSymbol(name,KIND_FN);
    crtFn->args=NULL;
    addDomain();
}
LPAR funcParams RPAR COLON baseType
{
    crtFn->type=ret.type;
}
defVar* block END
{
```

```

    stergeDomeniu();
    crtFn=NULL;
}

block ::= instr+
funcParams ::= ( funcParam ( COMMA funcParam )* )?
funcParam ::= ID
    {
    const char *name=consumed->text;
    Symbol *s=searchInCurrentDomain(name);
    if(s)tkerr("symbol redefinition: %s",name);
    s=addSymbol(name,KIND_ARG);
    Symbol *sFnParam=addFnArg(crtFn,name);
    }
    COLON baseType
    {
    s->type=ret.type;
    sFnParam->type=ret.type;
    }

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 )?

```