

Laboratorio 14

Objetivo

Desarrollar un intérprete para la gramática.

Programa

- `Program ::= VarDecList FunDecList`
- `VarDecList ::= (VarDec)*`
- `FunDecList ::= (FunDec)+`
- `FunDec ::= fun Type id ([ParamDecList]) Body endfun`
- `Body ::= VarDecList StmtList`
- `ParamDecList ::= Type id (, Type id)*`
- `VarDec ::= var Type VarList ;`
- `Type ::= id`
- `VarList ::= id (, id)*`
- `StmtList ::= Stmt (; Stmt)*`
- `Stmt ::= id = CExp |`
 - `print (CExp)`
 - `if CExp then Body [else Body] endif`
 - `while CExp do Body endwhile`
 - `for id in range (CExp , CExp , CExp) Body endfor`
 - `return ([CExp])`
- `CExp ::= Exp [(< | <= | ==) Exp]`
- `Exp ::= Term ((+ | -) Term)*`
- `Term ::= Factor ((* | /) Factor)*`
- `Factor ::= id | Num | Bool | (Exp) | ifexp (CExp , CExp , CExp) | id (ArgList)`
- `ArgList ::= CExp (, CExp)*`
- `Bool ::= true | false`

Ejercicio

Completar el parser avanzado en el laboratorio anterior e implementar de manera correcta los visitors `PrintVisitor` y `EVALVisitor`.

Sugerencias:

- Complete el parser para `ReturnStatement`
- Para el `EVALVisitor` debe tener los atributos

```
class EVALVisitor : public Visitor {
private:
    Environment<int> env;
    unordered_map<string, FunDec*> fdec;
    int retval;
    bool retcall; ... }
```

- El visitor Eval para programa debería ser:

```
void EVALVisitor::visit(Program* p) {
    env.add_level();
    p->vardecs->accept(this);
    p->fundecs->accept(this);
    if (!fdecs.check("main")) {
        exit(0);
    }
    FunDec* main_dec = fdecs.lookup("main");
    retcall = false;
    main_dec->body->accept(this);
    if (!retcall) {
        cout << "Error: Funcion main no ejecuto RETURN" << endl;
        exit(0);
    }
}
```