PROCESADORES DE LENGUAJES Examen (Segunda Convocatoria)

Nombre: _____ Apellidos: _____

Grupo: _____

Sintáctico (Diseño)

```
variable: IDENT CORCHETEABIERTO expresion_entera CORCHETECERRADO
   | IDENT

condicional: IF expresion_booleana bloque
    (((ELIF expresion_booleana bloque)+ ELSE bloque)? | (ELSE bloque)?)

iteracion: WHILE expresion_booleana bloque
   (ELSE bloque)?

expresion_entera: expresion_entera (MAS|MENOS|POR|DIV) expresion_entera
   | PARENTESISABIERTO expresion_entera PARENTESISCERRADO
   | MENOS expresion_entera
   | variable
   | NUM
   | LEN PARENTESISABIERTO expresion_lista PARENTESISCERRADO

expresion_booleana: expresion_booleana (Y|O) expresion_booleana
   | NOT expresion_booleana
   | expresion IGUAL expresion
   | expresion DISTINTO expresion
   | expresion_entera (MAYOR|MENOR|MAYORIGUAL|MENORIGUAL) expresion_entera
   | PARENTESISABIERTO expresion_booleana PARENTESISCERRADO
   | variable
   | TRUE
   | FALSE

expresion_lista: expresion_lista MAS expresion_lista
   | expresion_lista POR expresion_entera
   | PARENTESISABIERTO expresion_lista PARENTESISCERRADO
   | variable
   | funcion
   | lista_por_compresion
   | lista_por_extension

funcion: IDENT PUNTO INSERT PARENTESISABIERTO expresion COMA
                                              expresion_entera PARENTESISCERRADO
   | IDENT DOT APPEND PARENTESISABIERTO expresion PARENTESISCERRADO
   | IDENT DOT POP PARENTESISABIERTO PARENTESISCERRADO
   | IDENT DOT POPLEFT PARENTESISABIERTO PARENTESISCERRADO
   | IDENT DOT COPY PARENTESISABIERTO PARENTESISCERRADO
   | IDENT DOT REVERSE PARENTESISABIERTO PARENTESISCERRADO
   | IDENT DOT COUNT PARENTESISABIERTO  expresion PARENTESISCERRADO

lista_por_compresion:
   CORCHETEABIERTO expresion FOR IDENT IN expresion_lista
      IF expresion_booleana CORCHETECERRADO

lista_por_extension: CORCHETEABIERTO (expresiones)? CORCHETECERRADO
```

PROCESADORES DE LENGUAJES Examen (Segunda Convocatoria)

Nombre: _____ Apellidos: _____

Grupo: _____

Semántico (Diseño)

```
(GLOBAL)
/*
memorias globales conteniendo información relativa a las variables
que ocurren en la expresión analizada: (1) mem_lista almacena las
variables de tipo lista que ocurren en la expresión junto a los
valores (constantes) y tipos de sus componentes, (2) mem_entero
almacena las variables de tipo entero con sus valores,
(3) mem_booleano almacena las variables de tipo booleano con sus
valores y (4) mem_no-tipo almacena las variables de tipo no-tipo.
*/
mem_lista, mem_entero, mem_booleano, mem_no-tipo

op_bool_binario: Y | O

op_rel_orden: MAYOR | MENOR | MAYORIGUAL | MENORIGUAL

expresion dev valor,tipo: t1,v1=expresion MAS t2,v2=expresion
   { si t1 y t2 son entero entonces
        tipo=entero y valor=operacionAritm(v1,v2,MAS)
     sino
        si t1 y t2 son lista entonces
            tipo=lista y valor=concatenar v1 y v2
        sino
            tipo=error y valor=desconocido  //restricciones 1,4,5
        fsi
     fsi }
  | t1,v1=expresion POR t2,v2=expresion
     { si t1 y t2 son entero entonces
         tipo=entero y valor=operacionAritm(v1,v2,POR)
       sino
          si t1 es lista y t2 es entero entonces
             tipo=lista y valor=concatenar v1 v2 veces
          sino
             tipo=error y valor=desconocido  //restricción 1,4,5
          fsi
       fsi }
 | t1,v1=expresion MENOS t2,v2=expresion
   { si t1 y t2 son entero entonces
        tipo=entero y valor=operacionAritm(v1,v2,MENOS)
     sino
        tipo=error y valor=desconocido  //restricción 1,4,5
     fsi }
 | t1,v1=expresion DIVISION t2,v2=expresion
   { si t1 y t2 son entero entonces
        tipo=entero y valor=operacionAritm(v1,v2,DIVISION)
     sino
        tipo=error y valor=desconocido  //restricción 1,4,5
     fsi }
 | t1,v1=expresion op_bool_binario t2,v2=expresion
   { si t1 y t2 son booleano entonces
        tipo=booleano y valor=operacion(v1,v2,op_bool_binario)
     sino
        tipo=error y valor=desconocido  //restricción 2
     fsi }
```

```
| t1,v1=expresion op_rel_orden t2,v2=expresion
  { si t1 y t2 son entero entonces
       tipo=booleano y valor=operacion(v1,v2,op_rel_orden)
    sino
       tipo=error y valor=desconocido  //restricción 1
    fsi }
| PARENTESISABIERTO tipo,valor=expresion PARENTESISCERRADO
| MENOS t,v=expresion
  { si t es entero entonces
      tipo=entero y valor = -v
    sino
      tipo=error y valor=desconocido  //restricción 1
    fsi }
| NOT t,v=expresion
  { si t es booleano entonces
        tipo=booleano y valor = negacion de v
    sino
        tipo=error y valor=desconocido  //restricción 2
    fsi }
| LEN PARENTESISABIERTO t,v=expresion PARENTESISCERRADO
  { si t es tipo lista entonces
       tipo = entero y valor = número de componentes de v
    sino
       tipo=error y valor=desconocido  //restricción 3
    fsi }
| IDENTIFICADOR PUNTO COUNT PARENTESISABIERTO t,v=expresion PARENTESISCERRADO
{ si IDENTIFICADOR no ocurre en mem_lista entonces
     tipo=error y valor=desconocido  //restricción 3
   sino
     tipo=entero y valor = consulta en mem_lista la
                cantidad de elementos de IDENTIFICADOR iguales a v
   fsi }
| IDENTIFICADOR CORCHETEABIERTO t,v=expresion CORCHETECERRADO
  {  si IDENTIFICADOR ocurre en mem_lista, t es entero y
       v es la posicion de un elemento existente de IDENTIFICADOR entonces
       v es la posicion de un elemento existente de IDENTIFICADOR entonces
             tipo=tipo del componente en la posicion v de la
             lista IDENTIFICADOR y
             valor=valor del componente en la posicion v de la lista
             IDENTIFICADOR
     sino
        tipo=error y valor=desconocido //restricción 6
     fsi
  }
| IDENTIFICADOR
  { si IDENTIFICADOR ocurre en mem_lista entonces
       tipo=lista y valor = el valor almacenado en mem_lista
            para IDENTIFICADOR
    sino
       si IDENTIFICADOR ocurre en mem_entero entonces
          tipo=entero y valor = el valor almacenado en mem_entero
             para IDENTIFICADOR
       sino
          si IDENTIFICADOR ocurre en mem_booleano entonces
              tipo=booleano y valor = el valor almacenado en
                  mem_booleano para IDENTIFICADOR
          sino
                  tipo=no-tipo y valor=ninguno
          fsi
       fsi
    fsi }
| TRUE  { tipo=booleano, valor=cierto }
| FALSE { tipo=booleano, valor=falso }
| NUMERO{ tipo=entero y valor=interpretación de NUMERO }
| NONE { tipo=no-tipo,valor=ninguno }
;
```

Nombre: _____ Apellidos: _____

Grupo: _____

Intérprete (Implementación)

```
public class Interprete extends AnasintIntBaseVisitor<Object>{


 /*
    programa : (instrucciones)?

     instrucciones:
          instruccion instrucciones
        | instruccion

     instruccion:
          asignacion  | eliminacion  | impresion
     */
    /*
    asignacion:
        vars=variables ASIG exps=expresiones
          { actualizarMemorias(vars,exps) }
     */
    @Override
    public Object visitAsignacion(AnasintInt.AsignacionContext ctx) {
        List<Variable> vars=visitVariables(ctx.variables());
        List<ValorExpresion> exps=visitExpresiones(ctx.expresiones());
        actualizarMemorias(vars,exps);
        return null;
    }
    /*
        variables dev vars {nombre_var,componente}:
              nombre_var,componente=variable COMA vars=variables
              { añadir el elemento (nombre_var,componente) al princiio de la lista
vars}
            | nombre_var,componente=variable
              { la lista vars se inicializa con un primer elemento
                (nombre_var,componente) }
     */
    public List<Variable> visitVariables(AnasintInt.VariablesContext ctx) {
        List<Variable> vars;
        if (ctx.children.size()==3){
            vars=visitVariables(ctx.variables());
            vars.add(0,visitVariable(ctx.variable()));
        }
        else {
            vars = new LinkedList<>();
            vars.add(visitVariable(ctx.variable()));
        }
        return vars;
    }

    /*
    variable dev nombre_var,componente:
        IDENTIFICADOR CORCHETEABIERTO v,t=expresion CORCHETECERRADO
        { nombre_var=IDENTIFICADOR y componente=v }
      | IDENTIFICADOR { nombre_var=IDENTIFICADOR y no tiene componente }
     */
    @Override
    public Variable visitVariable(AnasintInt.VariableContext ctx) {
        Variable var = null;
```

```
        if (ctx.children.size()==4){
            ValorExpresion valor=interpreteExpresion.visitExpresion(ctx.expresion());
            var=new Variable(ctx.IDENTIFICADOR().getText(),
                    (Integer)valor.consultarValor());
        }
        else
            var=new Variable(ctx.IDENTIFICADOR().getText(),
                    null);
        return var;
    }
    /*
    expresiones dev exps {valor,tipo}:
          valor,tipo=expresion COMA exps=expresiones
          { añadir el elemento (valor,tipo) al principio de la lista exps }
        | valor,tipo=expresion
          { la lista exps se inicializa con un primer elemento (valor,tipo) }
     */
    @Override
    public List<ValorExpresion> visitExpresiones(AnasintInt.ExpresionesContext ctx) {
        List<ValorExpresion> exps=null;
        if (ctx.children.size()==3){
            exps=visitExpresiones(ctx.expresiones());
            exps.add(0,interpreteExpresion.visitExpresion(ctx.expresion()));
        }
        else {
            exps=new LinkedList<>();
            exps.add(interpreteExpresion.visitExpresion(ctx.expresion()));
        }
        return exps;
    }

    /*
    eliminacion :
        DEL nombre_var,componente=variable
            { eliminarMemoria(nombre_var,componente) }
     */
    @Override
    public Object visitEliminacion(AnasintInt.EliminacionContext ctx) {
        Variable var = visitVariable(ctx.variable());
        eliminarMemoria(var);
        return null;
    }

    /*
    impresion:
        PRINT PARENTESISABIERTO exps=expresiones PARENTESISCERRADO
            { imprimir(exps) }
     */
    @Override
    public Object visitImpresion(AnasintInt.ImpresionContext ctx) {
        List<ValorExpresion> exps=visitExpresiones(ctx.expresiones());
        imprimir(exps);
        return null;
    }
}
```

PROCESADORES DE LENGUAJES Examen (Segunda Convocatoria)

Nombre: _____ Apellidos: _____

Compilador (Decisiones Diseño)

(Decisión 1)

El compilador puede saber cuándo debe o no declarar una variable si lleva una contabilidad de las variables declaradas. Una simple memoria para registrar variables declaradas sería suficiente para evitar volverlas a declarar.

(Decisión 2)

La asignación paralela de PY-Lite no existe en Java. Una manera de simularla es hacer uso de asignaciones auxiliares sobre variables nuevas y después asignar estas variables nuevas a las variables asignadas originalmente.

Ejemplo:
x,y=y,x se traduce haciendo uso de asignaciones auxiliares:

```
_aux1=y;
_aux2=x;
x=_aux1;
y=_aux2;
```

La traducción secuencial en Java no vale porque no preserva la semántica de PY-Lite:

```
x=y;
y=x;
```

(Decisión 3)

PY-Lite tiene un tipado dinámico: las variables pueden cambiar de tipo a lo largo del programa. Sin embargo, Java tiene un tipado estático: las variables no pueden cambiar de tipo durante la ejecución del programa. Una manera de simular en Java el tipado dinámico es declarar las variables de tipo Object y usar castings en las expresiones para cambiar de tipo a las variables.
Los castings considerados son Integer para el tipo entero, Boolean para el tipo booleano, List<Object> para el tipo lista.
Los cambios de tipo de las variables tendrán lugar al procesar asignaciones y se registrarán en una memoria interna del compilador:

memoria:

variable	tipo
i	entero
b	booleano
l	lista

(Decisión 4)

La lista por extensión de PY-Lite no existe en Java. Una forma de simularla es mediante una variable auxiliar nueva de tipo List<Object> sobre la que se añade cada uno de los elementos de la lista original.

Ejemplo:
[8,[1,5],3] se traduce como:

```
List<Object> _aux1=new LinkedList<>();
_aux1.add(1);
_aux1.add(5);
List<Object> _aux2=new LinkedList<>();
_aux2.add(8);
_aux2.add(_aux1);
_aux2.add(3);
```

(Decisión 5)

```
import java.io.*;
import java.util.*;
import java.util.stream.Collectors;
public class comp1
{
   public static void main(String[] args) {
      // x,y=[8,[1,5],3],6
      // comienzo declaración de variables
      Object x;
      Object y;
      // fin declaración de variables
      // comienzo declaración de variables auxiliares
      List<Object> _aux1=new LinkedList<>();
      _aux1.add(1);
      _aux1.add(5);
      List<Object> _aux2=new LinkedList<>();
      _aux2.add(8);
      _aux2.add(_aux1);
      _aux2.add(3);
      // fin declaración de variables auxiliares
      // comienzo asignaciones
      List<Object> _aux3=(List<Object>)_aux2;
      Object _aux4=6;
      x=_aux3;
      y=_aux4;
      // fin asignaciones
      // x,y=y,x
      // comienzo declaración de variables
      // fin declaración de variables
      // comienzo declaración de variables auxiliares
      // fin declaración de variables auxiliares
      // comienzo asignaciones
      Object _aux5=(Integer)y;
      List<Object> _aux6=(List<Object>)(List<Object>)x;
      x=_aux5;
      y=_aux6;
      // fin asignaciones
      // comienzo declaración de variables auxiliares
      // fin declaración de variables auxiliares
      System.out.println((Integer)x+","+(List<Object>)y);
   }

}
```