

# Instituto Tecnológico de Costa Rica



Administración en Tecnología de la Información

Lenguajes de Programación

Prof. Andrei Fuentes

## **Tarea programada # 3**

### **Analizador de código SML**

José Daniel Chinchilla Cerdas – 201211069

Fabiana Díaz Redondo- 201268063

Josué Masis Álvarez – 201240146

Verónica Vargas Mora – 201269405

7 de noviembre del 2013

# Tabla de contenido

Descripción del problema.....3

Diseño del programa .....4

Librerías utilizadas .....15

Análisis de resultados .....16

Manual de usuario .....17

Lecciones aprendidas .....22

## Descripción del problema

En esta tarea programada se realizó un programa que analizará un documento, el cual, contiene código del lenguaje de programación SML, entonces se debe analizar el código para determinar la tabla del ambiente estático y la tabla del ambiente dinámico del código. Para la tabla del ambiente estático, se debió almacenar la información de los tipos de datos de cada una de las expresiones. Y para la tabla del ambiente dinámico, se debió almacenar la información de los valores de cada una de las expresiones.

El lenguaje utilizado para desarrollar esta tarea programada fue Java. Y al final el programa debe imprimir por medio de tablas la información que se encuentra en el ambiente estático y el ambiente dinámico.

## Diseño del programa

A continuación se presenta parte del código utilizado para la realización de esta tarea programada. Comenzamos mostrando el código creado para la utilización de listas, empezamos viendo el código de nodos simples.

```
public class NodosListaSimple {
    // datos amigables para que la Clase Lista Tenga un acceso directo
    Object dato;
    String dato;
    String valor;
    String tipo;
    int posicion;
    NodosListaSimple siguiente;

    NodosListaSimple (String var, String tipo)
    {
        dato = var;
        this.tipo = tipo;
        siguiente = null;
    }

    //Constructor Crea un nodo del Tipo Object y al siguiente nodo de la lista
    NodosListaSimple (String var, String tipo, NodosListaSimple signodo)
    {
        dato = var;
        this.tipo = tipo;
        siguiente = signodo;
    }

    NodosListaSimple (String var)
    {
        dato = var;
        siguiente = null;
    }
}
```

Aquí se muestran los constructores del nodo dependiendo de la aridad del nodo.

```
//Constructor Crea un nodo del Tipo Object y al siguiente nodo de la lista
NodosListaSimple (String var, NodosListaSimple signodo)
{
    dato = var;
    siguiente = signodo;
    //sig = sigui;
}

NodosListaSimple(Object dat, int pos){
    this.dat = dat;
    this.posicion = pos;
    siguiente = null;
}

NodosListaSimple(Object dat, int pos, NodosListaSimple signodo){
    this.dat = dat;
    this.posicion = pos;
    siguiente = signodo;
}

//Constructor Crea un nodo del tipo Object
NodosListaSimple (String dato, String valor, String tipo)
{
    this.dato = dato;
    this.valor = valor;
    this.tipo = tipo;
    siguiente = null;
    //sig = Prime;
}
```

A continuación se mostrara el código de la clase lista simple, esta clase es utilizada para manejar el contenido del archivo.txt a utilizar.

```
public class ListaSimple {
    public NodosListaSimple PrimerNode;
    public NodosListaSimple UltimoNode;
    String Nombre;

    //Constructor construye una lista vacia con un nombre s
    public ListaSimple (String s)
    { Nombre = s;
      PrimerNode = UltimoNode = null;
    }
    //Retorna True si Lista Vacía-a
    public boolean VaciaLista () {return PrimerNode == null;}

    // Imprime el contenido de la lista
    public void Imprimir()
    { if (Vacialista()){
      System.out.println("Lista vacia");
    }
    else{
      System.out.print( "La expresiÃ³n en " + Nombre + " es: ");
      NodosListaSimple Actual = PrimerNode;
      while (Actual != null){
        System.out.print(Actual.valor + " ");
        Actual=Actual.siguiente;
      }

      System.out.println();
      System.out.println();
    }
  }
}
```

Aquí se muestra el método de lista vacía, y uno de los métodos de imprimir la cual permite imprimir el contenido de la lista.

```
//Constructor construye una lista vacia con un nombre de List
public ListaSimple(){ this ("Lista");}

//Inserta un Elemento al Frente de la Lista
//Si esta vacía-a PrimerNode y UltimoNode se refieren al nuevo nodo. Si no PrimerNode se refiere al nuevo nodo.

public void InsertaInicio (String dato, String valor, String tipo){
  if (Vacialista()){
    PrimerNode = UltimoNode = new NodosListaSimple (dato,valor,tipo);
    //return this;
  }
  else{
    PrimerNode = new NodosListaSimple (dato,valor,tipo, PrimerNode);
    //return this;
  }
}

public void InsertaInicio (String valor, String dato, String tipo, ListaSimple adya){
  if (Vacialista()){
    PrimerNode = UltimoNode = new NodosListaSimple (dato,valor,tipo);
  }
  else{
    PrimerNode = new NodosListaSimple (dato,valor,tipo, PrimerNode);
  }
}
```

En esta parte del código se muestra un constructor de lista y se muestra métodos para ingresar elementos al inicio de la lista.

```
public void Insertar (Object a, int posicion){
    if (VacíaLista()){
        PrimerNodo = UltimoNodo = new NodosListaSimple (a,posicion);
        //return this;
    }
    else{
        UltimoNodo=UltimoNodo.siguiente =new NodosListaSimple (a,posicion);
        //return this;
    }
}

public void InsertaFinal(String dat){
    if ( VacíaLista())
        PrimerNodo = UltimoNodo = new NodosListaSimple (dat);
    else
        UltimoNodo=UltimoNodo.siguiente =new NodosListaSimple (dat);
}

public void InsertaFinal(String variable, String valor){
    if ( VacíaLista())
        PrimerNodo = UltimoNodo = new NodosListaSimple (variable, valor);
    else
        UltimoNodo=UltimoNodo.siguiente =new NodosListaSimple (variable, valor);
}
```

En esta parte se muestra el método inserta final y en la imagen que sigue se ve la función que permite eliminar en el medio de una lista.

```
public Object EliminaMedio (int Posicion)
{
    Object ElementoDel = null;
    if ( VacíaLista())
        System.out.println( "Nada");
    else
    {
        ElementoDel = UltimoNodo.valor; // recuperar la información
        NodosListaSimple Aux =null;
        NodosListaSimple Actual = PrimerNodo;
        int i =1;
        while (( i != Posicion) & (Actual.siguiente != null))
        {
            i++;
            Aux =Actual;
            Actual =Actual.siguiente;
        }
        if( i ==Posicion)
        {
            if (Aux == null)
            {
                Actual = PrimerNodo;
                ElementoDel=PrimerNodo;
                PrimerNodo = PrimerNodo.siguiente;
            }
            else
            {
                ElementoDel=Aux.valor;
                Aux.siguiente = Actual.siguiente;
            }
        }
    }
    return ElementoDel;
}
```

Por último se ve el método de largo que permite ver cuantos elementos tiene la lista y el otro método que esta es el que permite buscar un elemento dentro de la lista.

```
public int Largo()
{
    NodosListaSimple aux=PrimerNodo;
    int i=0;
    while (aux !=null)
    {
        i++;
        aux=aux.siguiente;
    }
    return i;
}

public NodosListaSimple BuscarElemento1(String Elem)
{
    NodosListaSimple aux= PrimerNodo;
    while(aux!=null){
        if (aux.dato.equals(Elem)){
            return aux;
        }
        else
            aux=aux.siguiente;
    }
    return aux;
}
```

A continuación se muestra los métodos utilizados para calcular lets , if's y otras variables necesarias para el programa.

```
public class metodo_if {
    public static ListaSimple listaCodigo = new ListaSimple(); // lista que almacena todo el codigo
    static ListaSimple estatico = new ListaSimple ("Estatico"); // lista que almacena los datos estaticos
    static ListaSimple dinamico = new ListaSimple ("Dinamico"); // lista que almacena los datos dinamicos
    static String resultadoValor;
    static String resultadoTipo;

    /*
    * Funcion principal: lee el archivo que contiene el codigo XML
    * almacena cada dato del codigo en listaCodigo
    * luego llama a la funcion evaluar
    * y por ultimo a la funcion imprimirTablas */
    public static void main(String [] args) throws IOException{
        // Archivo que contiene el codigo XML
        File archivo = (new File("codigo.txt"));
        FileReader fr = new FileReader(archivo);
        BufferedReader br = new BufferedReader(fr);
        String linea = br.readLine();
        int pos=0;
        while(linea !=null){
            StringTokenizer dato = new StringTokenizer(linea," ");
            while(dato.hasMoreTokens()){
                Object a = dato.nextToken();
                listaCodigo.Insertar(a, pos);
                pos++;
            }
            linea = br.readLine();
        }
        NodosListaSimple aux = listaCodigo.PrimerNodo;
        evaluar(aux,listaCodigo);
        imprimirTablas();
    }
}
```

En esta imagen se muestra donde se lee el archivo.txt y se agrega a una lista la cual funciona como base de datos.

```

public static void evaluar(NodosListaSimple aux, ListaSimple listaCodigo) throws IOException{
    while(aux!=null){
        // Verifica los datos que hay despues de un val y los separa por let, if o tipo de dato dado
        if(aux.dat.equals("val")){
            aux = aux.siguiente;
            String variable = aux.dat.toString();
            aux = aux.siguiente;
            if(aux.siguiente.dat.equals("let")){
                aux = aux.siguiente;
                int pos = evaluarLet(aux.posicion); // llama a la funcion evaluarLet, para crear una lista con los datos de Let
                estatico.InsertaFinal(variable, resultadoTipo);
                dinamico.InsertaFinal(variable, resultadoValor);
                if(listaCodigo.Largo()==pos+1){ ;break;}
            }
            else{
                while(aux!=null){
                    if(aux.posicion==pos){aux = aux.siguiente; break;}
                    else aux = aux.siguiente;
                }
            }
            else if(aux.siguiente.dat.equals("if")){
                aux = aux.siguiente;
                int pos = arregloIf(aux.siguiente.posicion); //Llama a la funcion arregloIf para separar los datos por if
                estatico.InsertaFinal(variable, resultadoTipo);
                dinamico.InsertaFinal(variable, resultadoValor);
                if(listaCodigo.Largo()==pos+1){ ;break;}
            }
            else{
                while(aux!=null){
                    if(aux.posicion==pos){aux = aux.siguiente; break;}
                    else aux = aux.siguiente;
                }
            }
        }
        else{

```

Aquí comienza el método evaluar, en esta parte del código se busca la palabra “val” si se encuentra se valida si lo que sigue es “let” se llama el método evaluarLet(), sino verifica si es “if” se llama al método evaluarIF() y si no se hace lo de la siguiente imagen:

```

        String valor = aux.siguiente.dat.toString();
        estatico.InsertaFinal(variable);
        dinamico.InsertaFinal(variable, aux.siguiente.dat.toString());
        try{
            //Verifica si el dato dado es un int
            if (Integer.parseInt(valor) %1 == 0){
                NodosListaSimple aux2 = estatico.PrimerNodo;
                while(aux2!=null){
                    if(aux2.dato.equals(variable)){ aux2.tipo = "int";break;}
                    else aux2 = aux2.siguiente;
                }
                aux = aux.siguiente;
            }
        }
        catch (NumberFormatException e){
            //llama a la funcion verificarOperacion(), para determinar el tipo de dato dado
            int numVeri = verificarOperacion(valor);
            if(numVeri == 1);
            else{
                NodosListaSimple aux2 = estatico.PrimerNodo;
                NodosListaSimple aux1 = dinamico.PrimerNodo;
                while(aux2!=null){
                    if(aux2.dato.equals(variable)){ aux2.tipo = "int"; aux1 = dinamico.PrimerNodo; break;}
                    else{ aux2 = aux2.siguiente;}
                }
                while(aux1!=null){
                    if(aux1.dato.equals(variable)){ aux1.tipo = resultadoValor; break;}
                    else aux1 = aux1.siguiente;
                }
            }
            aux = aux.siguiente;
        }
    }
}

```

En esta parte lo que se hace es insertar la variable tanto en el ambiente dinámico como en el estático y después intenta verificar si la variable es un int si lo es, guarda en el ambiente estático que el tipo es “int” y en el dato la variable. Sino es “int” se llama a la función verificarOperacion() para verificar el tipo de dato de la variable.



```

NodosListaSimple exp = arreglo.PrimerNodo;
//arreglos donde se guardan los datos del if
ArrayList e1= new ArrayList();
ArrayList e2= new ArrayList();
ArrayList e3= new ArrayList();
while(exp!=null){
    //Guarda los datos en e1
    if (exp.dat.equals("if")){ exp = exp.siguiente;
        while(!(exp.dat.equals("then"))){ e1.add(exp.dat); exp = exp.siguiente;}}
    //Guarda los datos en e2
    else if (exp.dat.equals("then")){
        exp = exp.siguiente;
        if(exp.dat.equals("if")){
            while(!(exp.dat.equals("else"))){ e2.add(exp.dat); exp = exp.siguiente;}}
            e2.add(exp.dat); exp = exp.siguiente;
            while(!(exp.dat.equals("else"))){ e2.add(exp.dat); exp = exp.siguiente;}}
        }
        while(!(exp.dat.equals("else"))){ e2.add(exp.dat); exp = exp.siguiente;}}
    else if(exp.dat.equals("else")){ exp = exp.siguiente;
        if(exp.dat.equals("if")){ e3.add(exp.dat);
            while(!(exp.dat.equals("else"))){ e3.add(exp.dat); exp = exp.siguiente;}}e3.add(exp.dat);
        }
        else if(exp.dat.equals("let")){ e3.add(exp.dat);
            while(!(exp.dat.equals("end"))){ e3.add(exp.dat); exp = exp.siguiente;}}
            evaluar(e1,e2,e3); return exp.posicion;
        }
        else{ e3.add(exp.dat); evaluar(e1,e2,e3); return exp.posicion; }
    }
    else exp = exp.siguiente;
}return 0;

```

Este método es evaluarIF() recibe como parámetro un arreglo con todos los datos del if, la función lo que hace es ir recorriendo el arreglo separando por los datos o palabras claves que contiene un if que son "if", "then" y "else", la función cuando encuentra el if toma todos los datos que estén después del if y antes del then como e1, lo que esta después del then y antes del else como e2 y el resto que se encuentre después del else como e3.

También valida que si lo que esta después del end es if llama esta misma función y si es let llama a evaluarLet() y si no es ninguna sigue con su funcionamiento normal.

```

Iterator<String> exp = a1.iterator();
//este parte permite separar
while(exp.hasNext()){
    String dato1 = exp.next(); String dato2 = exp.next();
    if(dato2.equals("<")|dato2.equals(">")|dato2.equals("=")){
        String dato = exp.next(); int num1,num2;
        try{ num1 = Integer.parseInt(dato1); //Determina si el dato es num o no
            //Cuando determina que los dos datos ingresados son int, llama a la funcion comparaciones
            try{ num2 = Integer.parseInt(dato); comparaciones(num1,num2,dato2,a2,a3);}}
            catch(NumberFormatException e2){
                NodosListaSimple nodoAux1 = estatico.BuscarElemento1(dato), nodoAux2 = dinamico.BuscarElemento1(dato);
                if(nodoAux1 != null){
                    String tipo = nodoAux1.tipo, valor = nodoAux2.tipo;
                    num2 = Integer.parseInt(valor);
                    comparaciones(num1,num2,dato2,a2,a3);}
                else ;}}
            catch(NumberFormatException e1){
                NodosListaSimple nodoAux = estatico.BuscarElemento1(dato1), nodoAux3 = dinamico.BuscarElemento1(dato1);
                if(nodoAux!=null){
                    String tipo = nodoAux.tipo, valor = nodoAux3.tipo;
                    try{ num1 = Integer.parseInt(valor);
                        try{ num2 = Integer.parseInt(dato); comparaciones(num1,num2,dato2,a2,a3);}}
                        catch(NumberFormatException e2){
                            NodosListaSimple nodoAux1 = estatico.BuscarElemento1(dato), nodoAux2 = dinamico.BuscarElemento1(dato);
                            if(nodoAux1 != null){
                                tipo = nodoAux1.tipo;
                                valor = nodoAux2.tipo;
                                num2 = Integer.parseInt(valor);
                                comparaciones(num1,num2,dato2,a2,a3);} else;}}
                            catch(NumberFormatException e2){;}} else;}}break;}}

```

La imagen anterior es del método evaluar, esta función lo que hace es evaluar las expresiones obtenidas en el IF y lo que hace es determinar el valor a retornar para la función if. Si en el if hay una expresión como “x=y” se llama a la función operaciones ()

```

funcion que compara la condicion del if, y determina si evaluar la parte del then o del else
public static void comparaciones(int num1, int num2, String comp, ArrayList e2, ArrayList e3) throws IOException{
    if(comp.equals("<")){
        if(num1 < num2){ evaluarThen(num1,num2,e2);}
        else evaluarElse(num1,num2,e3); }
    else if(comp.equals(">")){
        if(num1>num2) evaluarThen(num1,num2,e2);
        else evaluarElse(num1,num2,e3);}
    else{
        if(num1==num2) evaluarThen(num1,num2,e2);
        else evaluarElse(num1,num2,e3);}
    }
}
/*Funcion evaluarThen(num1,num2,e2)

```

La imagen que sigue es de la función EvaluarThen() esta función lo que hace es revisar si la e2 obtenida del if es un int, un let, un if u otro tipo de elemento.

```

    resultadoTipo = "int";}
    catch(NumberFormatException e){
        NodosListaSimple nodoAux1 = estatico.BuscarElemento1(nombre), nodoAux2 = dinamico.BuscarElemento1(nombre);
        if(nodoAux1 != null){
            String tipo = nodoAux1.tipo, valor = nodoAux2.tipo;
            resultadoValor = valor; resultadoTipo = tipo; } else ; } }
    else{
        try{ int a = Integer.parseInt(nombre); resultadoValor = String.valueOf(a);
            resultadoTipo = "int";}
        catch(NumberFormatException e){/*System.out.println("funcion de josue"); //javax.script*/}}
    else{
        if(nombre.equals("let")) System.out.println("funcion vero");
        else if(nombre.equals("if")){
            int tamanno = e2.size();
            int pos = 1;
            ListaSimple lista = new ListaSimple();
            lista.Insertar(nombre, 0);
            while(pos!=tamanno){lista.Insertar(datos.next(), pos); pos++;}
            EvaluarIf(lista);}
        else{
            try{int numero = Integer.parseInt(nombre);
                resultadoValor = String.valueOf(numero);resultadoTipo = "int";}
            catch (NumberFormatException e){
                int tamanno = nombre.length();
                if(tamanno == 1){
                    NodosListaSimple a = dinamico.BuscarElemento1(nombre), a1 = estatico.BuscarElemento1(nombre);
                    if(a!=null && a1!=null){
                        resultadoValor = a.tipo;
                        resultadoTipo = a1.tipo;}}
                else{int numVeri = verificarOperacion(nombre);}}}}
    }
}

```

Ahora la función a explicar es let esta función lo que recibe es la posición en la lista código del let a evaluar, y en si lo que hace es separar la lista en dos listas, variables y asignaciones. La primera guarda todo lo que este antes del in y la segunda lo que esta después del in hasta el end.

```

    NodosListaSimple aux = listaCodigo.PrimerNodo;
    ListaSimple listaLet = new ListaSimple();
    while(aux!=null){
        if(aux.posicion==pos){
            while(aux!=null){ listaLet.Insertar(aux.dat, aux.posicion); aux = aux.siguiente;}
            break;}else aux = aux.siguiente;}
    NodosListaSimple auxLet = listaLet.PrimerNodo;
    ListaSimple listaVariables = new ListaSimple(), listaAsignaciones = new ListaSimple();
    while(auxLet!=null){
        if (auxLet.dat.equals("let")){
            auxLet = auxLet.siguiente; int posic = 0;
            while(!(auxLet.dat.equals("in"))){ Object a = auxLet.dat;
                if(a.equals("let")){
                    while(!(a.equals("end"))){
                        listaVariables.Insertar(a, auxLet.posicion);
                        auxLet = auxLet.siguiente; a = auxLet.dat;}}
                    listaVariables.Insertar(a,auxLet.posicion); posic++;
                    auxLet = auxLet.siguiente;}}
            else if (auxLet.dat.equals("in")){auxLet = auxLet.siguiente;
                int posic = 0;
                while(!(auxLet.dat.equals("end"))){
                    Object a = auxLet.dat;
                    listaAsignaciones.Insertar(a,auxLet.posicion);
                    posic++;
                    auxLet = auxLet.siguiente;
                } }
            else if (auxLet.dat.equals("end")){
                NodosListaSimple aux1 = listaVariables.PrimerNodo;
                NodosListaSimple aux2 = listaAsignaciones.PrimerNodo;
                evaluar(aux1,listaVariables);
            }
        }
    }

```

A continuación se presenta la función validar\_tipo(), esta función lo que hace es ver el tipo que tiene el dato principalmente "booleans" e "int".

```

public static void validar_tipo(){
    NodosListaSimple veri = dinamico.PrimerNodo;
    NodosListaSimple guar = estatico.PrimerNodo;
    while (veri!=null && guar!=null){
        if (veri.tipo.equals("true") || veri.tipo.equals("false")){
            //guar.Igual = "->";
            guar.tipo = "Boolean";
            guar = guar.siguiente;
            veri = veri.siguiente;
        }//Fin del if
        /*guar = guar.siguiente;
        veri = veri.siguiente;*/
        else{
            try{
                if (Integer.parseInt(veri.tipo) %1 == 0){
                    //guar.Igual = "->";
                    guar.tipo = "int";
                    guar = guar.siguiente;
                    veri = veri.siguiente;
                }//Fin del if
            }//Fin del try
            catch (NumberFormatException e){
                //guar.Igual = "->";
                guar.tipo = definir(veri.tipo);
                guar = guar.siguiente;
                veri = veri.siguiente;
            }//Fin del catch
        }//Fin del else
    }//Fin del while
} //Fin de valida_tipo

```

Los métodos a continuación son los de verificarOperacion() y calcular(); la primera función sirve para identificar si la función a realizar es una multiplicación, suma, resta o división. La segunda función es para formar una expresión a la cual se le puede realizar las operaciones básicas ya sea suma, resta, multiplicación o división.

```
public static int verificarOperacion(String valor) throws IOException{
    ListaSimple listaOperaciones = new ListaSimple();
    int largo = valor.length();
    int pos = 0;
    String unificar = "";
    while(pos!=largo){
        String caracter = String.valueOf(valor.charAt(pos));
        if(caracter.equals("*")|caracter.equals("+")|caracter.equals("-")|caracter.equals("/")){
            listaOperaciones.InsertaFinal(unificar, caracter);
            unificar = "";
            pos++;
        }
        else{unificar = unificar + caracter; pos++;}
    }
    listaOperaciones.InsertaFinal(unificar,null);
    if(listaOperaciones.Largo()==1){letra(unificar);validar_tipo();return 1;}
    else{calcular(listaOperaciones);return 2;}
}

public static void calcular(ListaSimple lista){
    NodosListaSimple aux = lista.PrimerNodo;
    int resultado = 0;
    while(aux!=null){
        String dato1,oper,dato2;
        dato1 = aux.dato;
        oper = aux.tipo;
        dato2 = aux.siguiente.dato;
        resultado = operaciones(dato1,oper,dato2);
        aux.siguiente.dato = String.valueOf(resultado);
        aux = aux.siguiente;
        if(aux.tipo==null) break;
    }
    resultadoValor = String.valueOf(resultado);
    resultadoTipo = "int";
}
```

Como se ve en la imagen anterior en el while del método calcular hay una función llamada operaciones que es la siguiente

```
public static int operaciones(String dato1,String oper,String dato2){
    int num1,num2;
    try{
        num1 = Integer.parseInt(dato1);
        try{num2 = Integer.parseInt(dato2); int result = resolver(num1,num2,oper);return result;}
        catch(NumberFormatException e1){
            NodosListaSimple aux = dinamico.BuscarElemento1(dato2);
            if(aux!=null){
                try{num2 = Integer.parseInt(aux.tipo);int result = resolver(num1,num2,oper);return result;}
                catch(NumberFormatException e3){};
            }
        }
    }
    catch(NumberFormatException e2){
        NodosListaSimple aux = dinamico.BuscarElemento1(dato1);
        if(aux!=null){
            try{
                num1 = Integer.parseInt(aux.tipo);
                try{num2 = Integer.parseInt(dato2);int result = resolver(num1,num2,oper);return result;}
                catch(NumberFormatException e1){
                    aux = dinamico.BuscarElemento1(dato2);
                    if(aux!=null){
                        try{num2 = Integer.parseInt(aux.tipo);int result = resolver(num1,num2,oper);return result;}
                        catch(NumberFormatException e3){};
                    }
                }
            }
            catch(NumberFormatException e){};
        }
    }
    return 0;
}
```

En esta función lo que se hace es tomar los datos y el operando a utilizar, y se convierten los datos en enteros para poder realizar la operación en la función resolver.

```
public static void letra (String info) throws IOException{
    //Se verifica que la lista no se encuentre vacía
    if (dinamico.VaciaLista()){
        System.out.println("Lista vacía");
    } //Fin del if
    else{
        //Se realizan las búsquedas para realizar el cambio del valor en caso que sea necesario
        NodosListaSimple aux = dinamico.PrimerNodo;
        NodosListaSimple guia = dinamico.PrimerNodo;
        while (aux!=null){
            String var = aux.dato;
            if (var.equals(info)){
                while (guia != null){
                    if (guia.tipo.equals(info)){
                        guia.tipo = aux.tipo;
                        break;
                    } //Fin del if
                    else{
                        guia = guia.siguiente;
                    } //Fin del else
                } //Fin del while interno
                aux.tipo = aux.tipo;
                break;
            } //Fin del if
            else{
                aux = aux.siguiente;
            } //Fin del else
        } //Fin del while
    } //Fin del else
} //Fin del metodo letras|
```

La función letras permite verificar si una variable “y” se encuentra en el programa para cambiar la letra por el valor correspondiente de la misma y que el valor de la variable “y=5” se asigne a la variable a la que se está igualando. EJ: X=y, esto es igual a X=5.

```
if (dato.charAt(dato.length()-1)==pc){
    StringTokenizer tokn = new StringTokenizer(dato,";");
    dato = tokn.nextToken();
} try { //Verifico si es un int
    Integer.parseInt(dato);
    return "int";
} catch (NumberFormatException nfe){
    //Verifico si es un bool
    if (dato.equals("true")||dato.equals("false")){ return "boolean";}
    //Verifico si es un char
    else if(dato.charAt(0)=='#'){ return "char";}
    //Verifico si es una tupla
    else if(dato.charAt(0)=='('){
        dato= eliminarChar(dato,'(',')');
        StringTokenizer tupla = new StringTokenizer(dato,"");
        String obtenido= tupla.nextToken();
        if(obtenido.indexOf("[")==0){
            String aux=tupla.nextToken();
            while(aux.indexOf("[")<0){
                obtenido=obtenido+" "+aux;
                aux= tupla.nextToken();
            }
        }
        obtenido=definir(obtenido);
        while(tupla.hasMoreTokens()){
            String aux2=tupla.nextToken();
            if(aux2.indexOf("[")==0){
                String aux=tupla.nextToken();
                while(aux.indexOf("[")<0){
                    aux2=aux2+" "+aux;
                    aux= tupla.nextToken();
                }
            }
        }
    }
}
```

En la imagen anterior se muestra una parte del método definir, en esta primera parte se utiliza más que todo para el ambiente estático puesto que lo que se hace en la función es identificar el tipo de dato de la variable. En esta parte se valida si es “int”, “boolean”, “char” o “tupla”.

```

        String aux2=tupla.nextTokn();
        if(aux2.indexOf("[")==0){
            String aux=tupla.nextTokn();
            while(aux.indexOf("]")<0){
                aux2=aux2+" "+aux;
                aux= tupla.nextTokn();}
            obtenido+=" "+definir(aux2);
        }
        return obtenido;
    }
    //Verifico si es una lista
    else if(dato.charAt(0)=='['){
        dato= eliminarChar(dato,[' ',' ']);
        StringTokenizer lista = new StringTokenizer(dato," ");
        String result = definir(lista.nextTokn());
        result = "list "+result;
        return result;}
    else if(dato.indexOf("let")==0){
        return "let";//Funcion de Vero que retorna un String
    }
    else if(dato.indexOf("if")==0){
        return "if";//Funcion de Jose que retorna un String
    }
    //Verifico si es un string
    else{ //(dato.charAt(0)==' ')
        return "string";}
}
}

```

En esta segunda parte del método definir se termina validando si la variable es de tipo “lista”, “let”, “if” o “string”. Si el tipo es let o if se llama una función externa a esta, sino se asigna directamente el tipo de la variable.

```

public static String eliminarChar(String linea,char caract1, char caract2){
    String nuevo="";
    for(int i=0; i<linea.length(); i++){
        if(linea.charAt(i)!=caract1&& linea.charAt(i)!=caract2)
            nuevo+= linea.charAt(i);
    }
    return nuevo;
}

public static void imprimirTablas(){
    System.out.println(" TABLA ESTATICA ");
    NodosListaSimple aux = estatico.PrimerNodo;
    NodosListaSimple aux1 = dinamico.PrimerNodo;
    while(aux!=null){
        System.out.println(aux.dato+" -> "+aux.tipo);
        aux = aux.siguiente;
    }
    System.out.println(" ");
    System.out.println(" TABLA DINAMICA ");
    while(aux1!=null){
        System.out.println(aux1.dato+" -> "+aux1.tipo);
        aux1 = aux1.siguiente;
    }
}
}

```

Por último se encuentran estas dos funciones eliminarChar() e imprimirTablas(). EliminarChar() es una función que permite eliminar el char y pasar la información o la variable a string, La segunda función imprimirTablas() lo que hace es imprimir la tabla del ambiente estático y el ambiente dinámico para mostrar el contenido de las variables escritas en el “codigo.txt” ingresado por el usuario.

## Librerías utilizadas

### ➤ java.io.BufferedReader;

Esta librería lee texto a partir de una secuencia de caracteres de entrada o almacenamiento en búfer caracteres a fin de proporcionar para la lectura eficiente de caracteres, matrices, y líneas. El tamaño de la memoria intermedia puede ser especificado, o el tamaño predeterminado se puede utilizar. El valor por defecto es suficiente para la mayoría de los propósitos de gran tamaño.

### ➤ java.io.File; Permite representación abstracta de las rutas de acceso de archivos y directorios.

### ➤ java.io.FileNotFoundException;

Esto permite generar un error si no se encuentra el archivo esperado dentro de la carpeta o ruta especificada en el código.

### ➤ java.io.FileReader; Clase que permite la lectura de archivos de carácter

### ➤ java.io.FileWriter; Clase que permite la escritura de archivos

### ➤ java.io.IOException;

Señales de que se ha producido una excepción de Entrada/ Salida de algún tipo. Esta clase es la clase general de excepciones producidas por operaciones fallidas o interrumpido de Entrada/ Salida.

### ➤ java.io.PrintWriter;

Permite imprimir representaciones de formato de los objetos en una secuencia de texto de salida. Esta clase implementa todos los métodos de impresión que se encuentran en PrintStream.

### ➤ java.util.ArrayList;

Permite la implementación de todas las operaciones de listas opcionales y permite todos los elementos, incluidos los nulos.

### ➤ java.util.Iterator;

Iterator toma el lugar de empadronamiento en el marco de las colecciones de Java. Los iteradores difieren de las enumeraciones de dos maneras: Iteradores permiten a la persona que llama para quitar elementos de la colección subyacente durante la iteración con una semántica bien definida. Nombres de los métodos han sido mejorados.

### ➤ java.util.StringTokenizer;

La clase tokenizer permite una aplicación para romper una cadena en tokens. El método tokenización es mucho más simple que el utilizado por la clase StreamTokenizer. Los métodos StringTokenizer no distinguen entre los identificadores, números y cadenas entre comillas, ni se reconocen y se saltan los comentarios.

## Análisis de resultados

Para esta parte la dividimos en los puntos requeridos en la especificación de la tarea programada. Los puntos que se debían cumplir son los siguientes:

- ✓ Tabla ambiente estático
- ✓ Tabla ambiente dinámico
- ✓ Funcionalidades:
  - Let
  - If
  - Tuplas
  - Listas
  - Asociación de la variables
  - Int
  - Booleans

Todas las funcionalidades puestas anteriormente funcionan correctamente según las funcionalidades, siendo comprendidas por el grupo de trabajo. Sin embargo aunque se cumple con todos los puntos se tiene una serie de restricciones para el programa que son las siguientes:

1. No se debe poner “;” al final de una expresión
2. Si hay una operación esta debe ir toda pegada, es decir, sin espacios entre las variables o elementos que tengan la operación.
3. Si hay comparaciones, estas deben llevar las variables separadas por espacios.
4. Tampoco se pueden realizar comparaciones con tuplas o listas.



## Manual de usuario

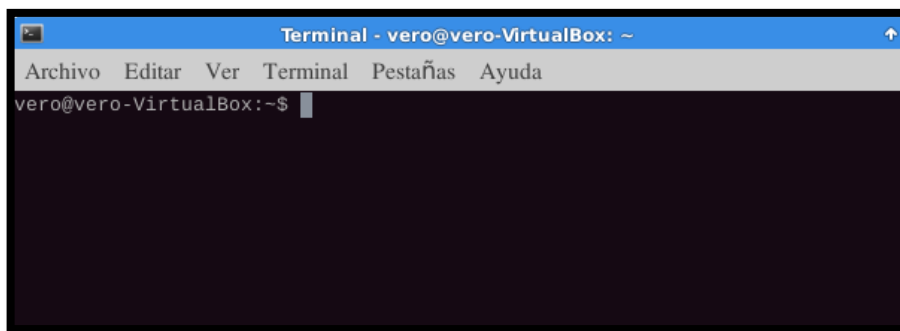
El analizador SML es un programa desarrollado e implementado por José Daniel Chinchilla Cerdas, Fabiana Díaz Redondo, Josué Masis Álvarez y Verónica Vargas Mora; realizado con el fin de leer un archivo con código SML, el cual debe ser 100% funcional, sin errores sintácticos, léxicos y/o semánticos.

Luego de leer el archivo con el código, se realiza un análisis dinámico y estático del mismo, y se muestra en pantalla una tabla con el ambiente dinámico y otra con el ambiente estático. El programa analizador SML, fue desarrollado en el lenguaje de programación JAVA.

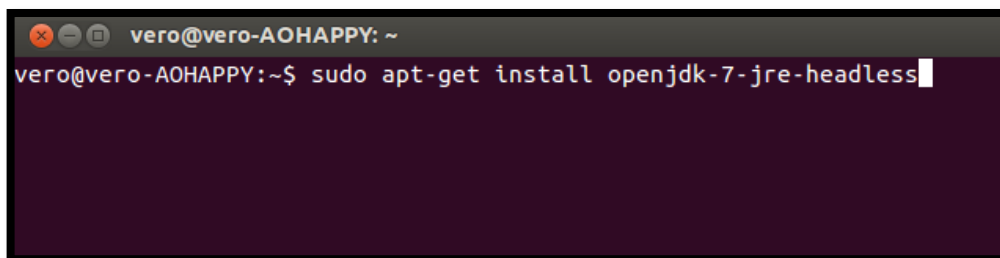
Este manual es un instrumento de ayuda para la operación del sistema.

### Requisitos

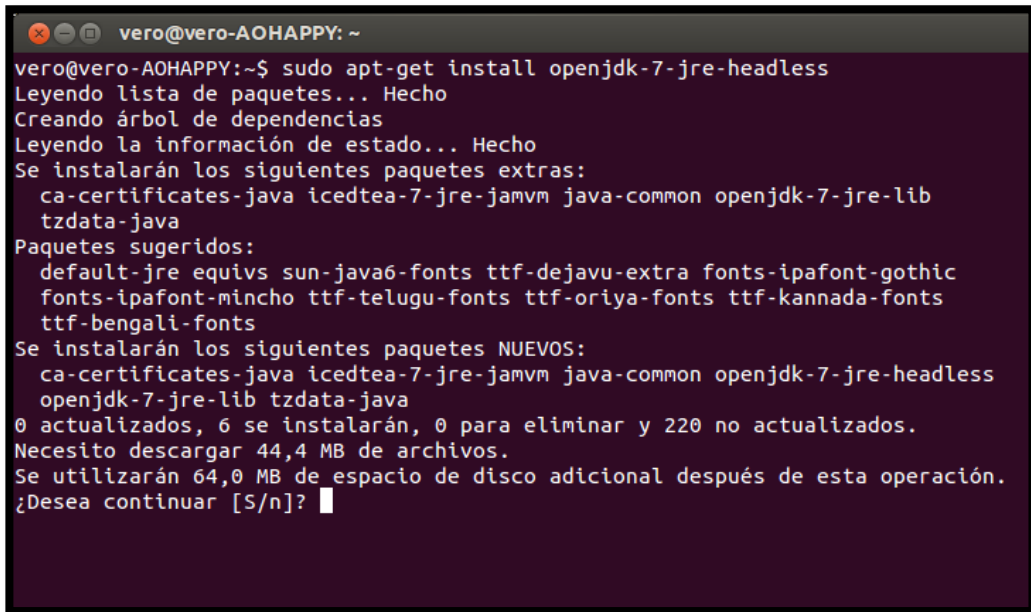
- Instalar JDK.
1. Abrir la terminal del sistema.



2. Escribir el siguiente comando ***"sudo apt-get install openjdk-7-jre-headless"***  
Luego dar *Enter*, el comando anterior inicia la instalación de JDK, requisito tener conexión a Internet.

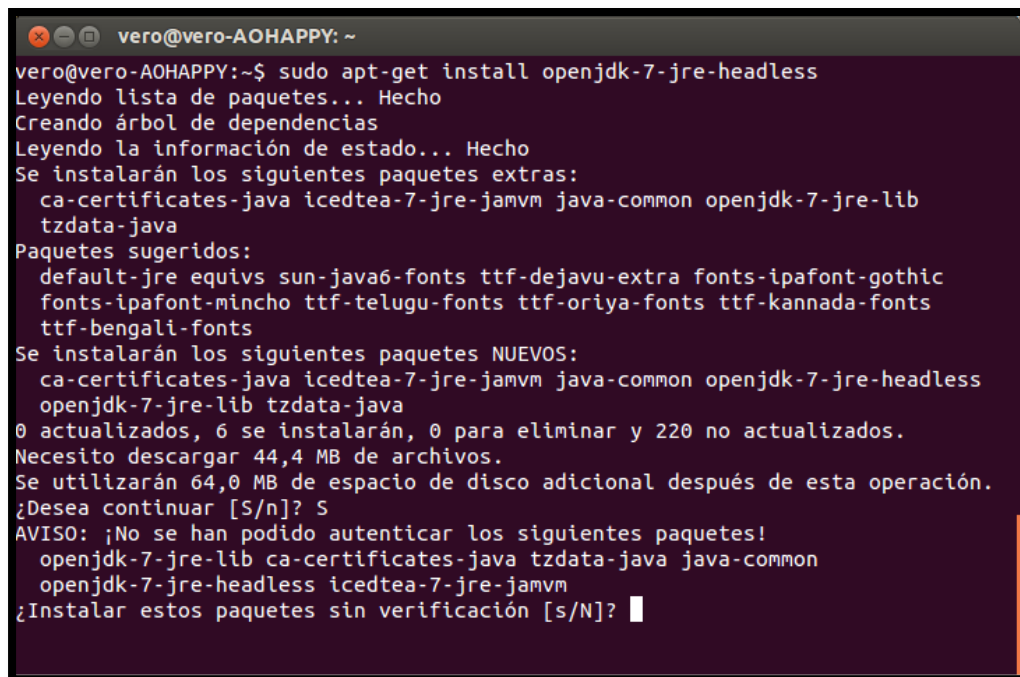


3. Esperar que se descargue el paquete JDK, hasta que aparezca en la consola **¿Desea continuar [S/n]?**, el usuario debe digitar *S* y presionar *Enter*



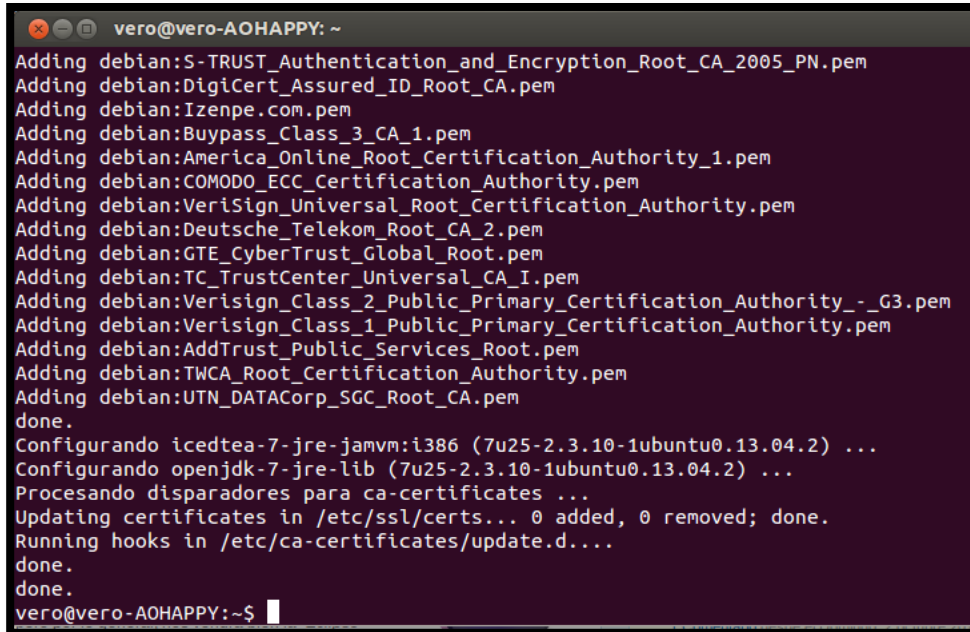
```
vero@vero-AOHAPPY: ~  
vero@vero-AOHAPPY:~$ sudo apt-get install openjdk-7-jre-headless  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes extras:  
  ca-certificates-java icedtea-7-jre-jamvm java-common openjdk-7-jre-lib  
  tzdata-java  
Paquetes sugeridos:  
  default-jre equivs sun-java6-fonts ttf-dejavu-extra fonts-ipafont-gothic  
  fonts-ipafont-mincho ttf-telugu-fonts ttf-oriya-fonts ttf-kannada-fonts  
  ttf-bengali-fonts  
Se instalarán los siguientes paquetes NUEVOS:  
  ca-certificates-java icedtea-7-jre-jamvm java-common openjdk-7-jre-headless  
  openjdk-7-jre-lib tzdata-java  
0 actualizados, 6 se instalarán, 0 para eliminar y 220 no actualizados.  
Necesito descargar 44,4 MB de archivos.  
Se utilizarán 64,0 MB de espacio de disco adicional después de esta operación.  
¿Desea continuar [S/n]? █
```

4. Esperar que se complete la descarga del paquete JDK, hasta que aparezca en la consola **¿Instalar estos paquetes sin verificación [s/N]?** el usuario debe digitar *N* y presionar *Enter*



```
vero@vero-AOHAPPY: ~  
vero@vero-AOHAPPY:~$ sudo apt-get install openjdk-7-jre-headless  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes extras:  
  ca-certificates-java icedtea-7-jre-jamvm java-common openjdk-7-jre-lib  
  tzdata-java  
Paquetes sugeridos:  
  default-jre equivs sun-java6-fonts ttf-dejavu-extra fonts-ipafont-gothic  
  fonts-ipafont-mincho ttf-telugu-fonts ttf-oriya-fonts ttf-kannada-fonts  
  ttf-bengali-fonts  
Se instalarán los siguientes paquetes NUEVOS:  
  ca-certificates-java icedtea-7-jre-jamvm java-common openjdk-7-jre-headless  
  openjdk-7-jre-lib tzdata-java  
0 actualizados, 6 se instalarán, 0 para eliminar y 220 no actualizados.  
Necesito descargar 44,4 MB de archivos.  
Se utilizarán 64,0 MB de espacio de disco adicional después de esta operación.  
¿Desea continuar [S/n]? S  
AVISO: ¡No se han podido autenticar los siguientes paquetes!  
  openjdk-7-jre-lib ca-certificates-java tzdata-java java-common  
  openjdk-7-jre-headless icedtea-7-jre-jamvm  
¿Instalar estos paquetes sin verificación [s/N]? █
```

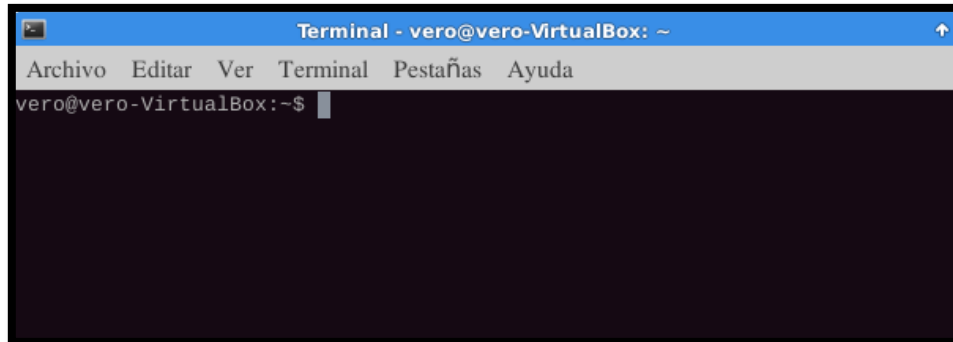
5. Esperar que termine toda la instalación, hasta que en la terminal se muestre el inicio de la misma



```
vero@vero-AOHAPPY: ~  
Adding debian:S-TRUST_Authentication_and_Encryption_Root_CA_2005_PN.pem  
Adding debian:DigiCert_Assured_ID_Root_CA.pem  
Adding debian:Izenpe.com.pem  
Adding debian:Buypass_Class_3_CA_1.pem  
Adding debian:America_Online_Root_Certification_Authority_1.pem  
Adding debian:COMODO_ECC_Certification_Authority.pem  
Adding debian:VeriSign_Universal_Root_Certification_Authority.pem  
Adding debian:Deutsche_Telekom_Root_CA_2.pem  
Adding debian:GTE_CyberTrust_Global_Root.pem  
Adding debian:TC_TrustCenter_Universal_CA_I.pem  
Adding debian:Verisign_Class_2_Public_Primary_Certification_Authority_-_G3.pem  
Adding debian:Verisign_Class_1_Public_Primary_Certification_Authority.pem  
Adding debian:AddTrust_Public_Services_Root.pem  
Adding debian:TWCA_Root_Certification_Authority.pem  
Adding debian:UTN_DATACorp_SGC_Root_CA.pem  
done.  
Configurando icedtea-7-jre-jamvm:i386 (7u25-2.3.10-1ubuntu0.13.04.2) ...  
Configurando openjdk-7-jre-lib (7u25-2.3.10-1ubuntu0.13.04.2) ...  
Procesando disparadores para ca-certificates ...  
Updating certificates in /etc/ssl/certs... 0 added, 0 removed; done.  
Running hooks in /etc/ca-certificates/update.d....  
done.  
done.  
vero@vero-AOHAPPY:~$
```

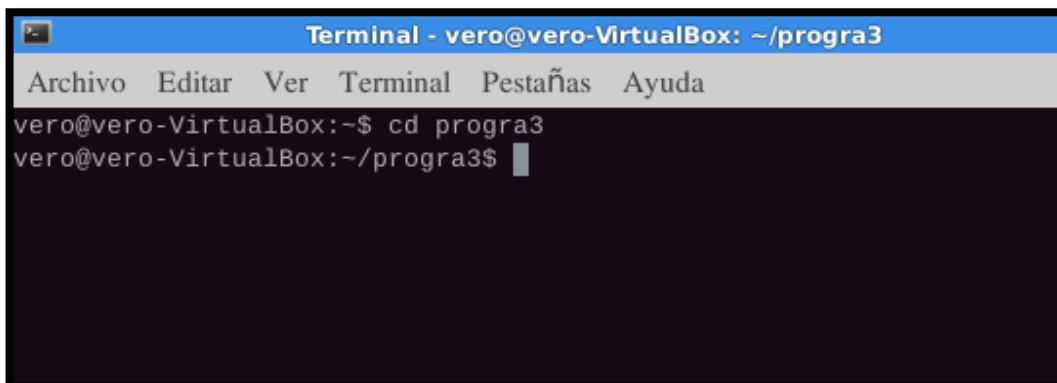
## Como usar el programa

1. Abrir la terminal del sistema



```
Terminal - vero@vero-VirtualBox: ~  
Archivo Editar Ver Terminal Pestañas Ayuda  
vero@vero-VirtualBox:~$
```

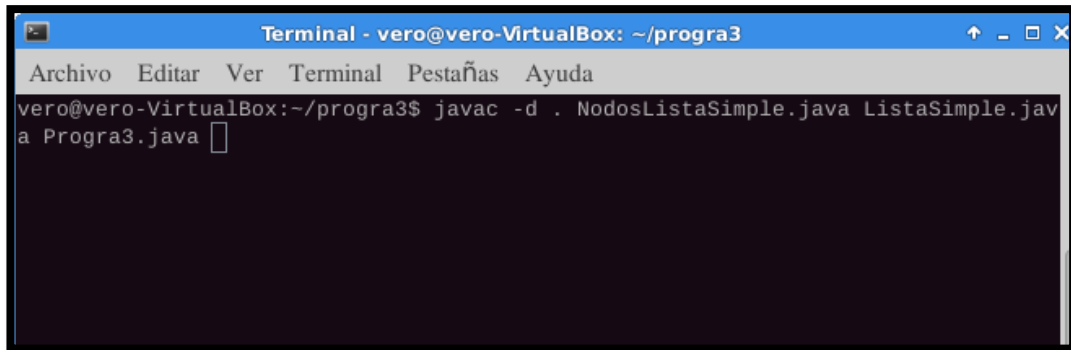
2. Ubicarse en la carpeta que se encuentra el programa



```
Terminal - vero@vero-VirtualBox: ~/progra3  
Archivo Editar Ver Terminal Pestañas Ayuda  
vero@vero-VirtualBox:~$ cd progra3  
vero@vero-VirtualBox:~/progra3$
```

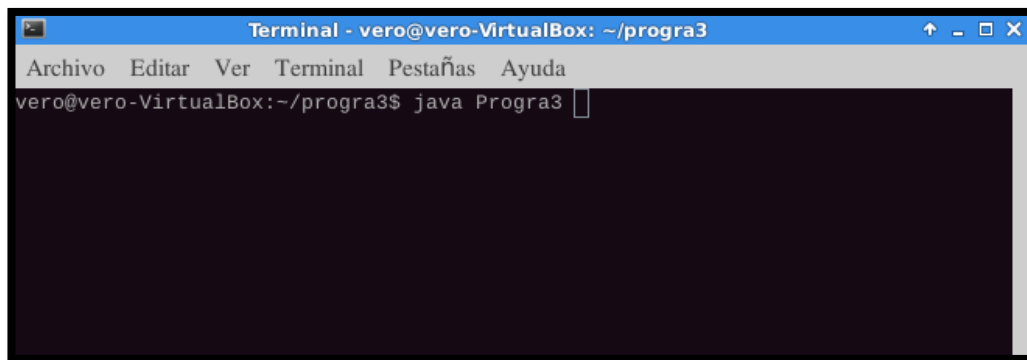
3. Compilar el programa, debe ingresar el siguiente comando

*javac -d . NodosListaSimple.java ListaSimple.java Progra3.java* y presionar *Enter*



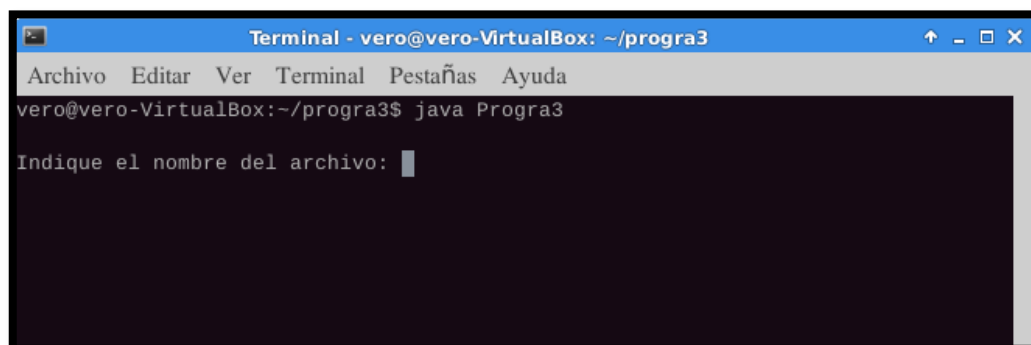
```
Terminal - vero@vero-VirtualBox: ~/progra3
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
vero@vero-VirtualBox:~/progra3$ javac -d . NodosListaSimple.java ListaSimple.java Progra3.java
```

4. Ejecutar el programa, el usuario debe ingresar el siguiente comando *java Progra3* y presionar *Enter*



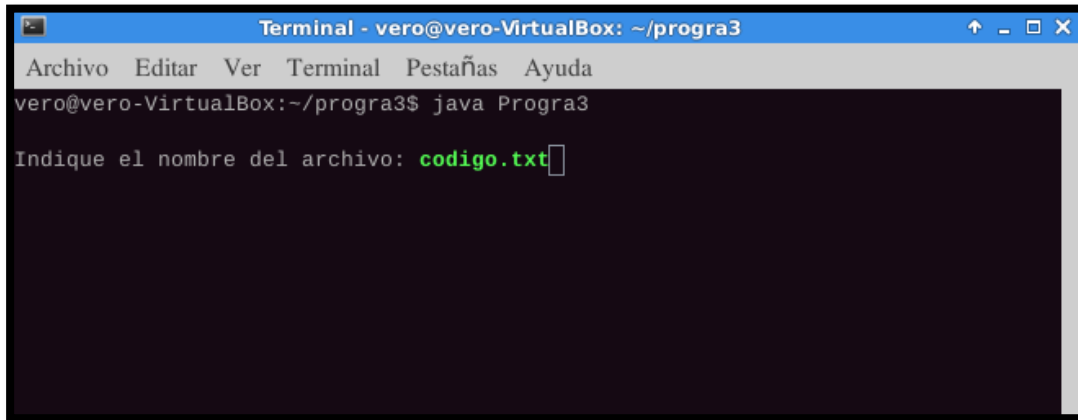
```
Terminal - vero@vero-VirtualBox: ~/progra3
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
vero@vero-VirtualBox:~/progra3$ java Progra3
```

5. Una vez ejecutado el programa, el programa inicia y muestra la siguiente pantalla.



```
Terminal - vero@vero-VirtualBox: ~/progra3
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
vero@vero-VirtualBox:~/progra3$ java Progra3
Indique el nombre del archivo: 
```

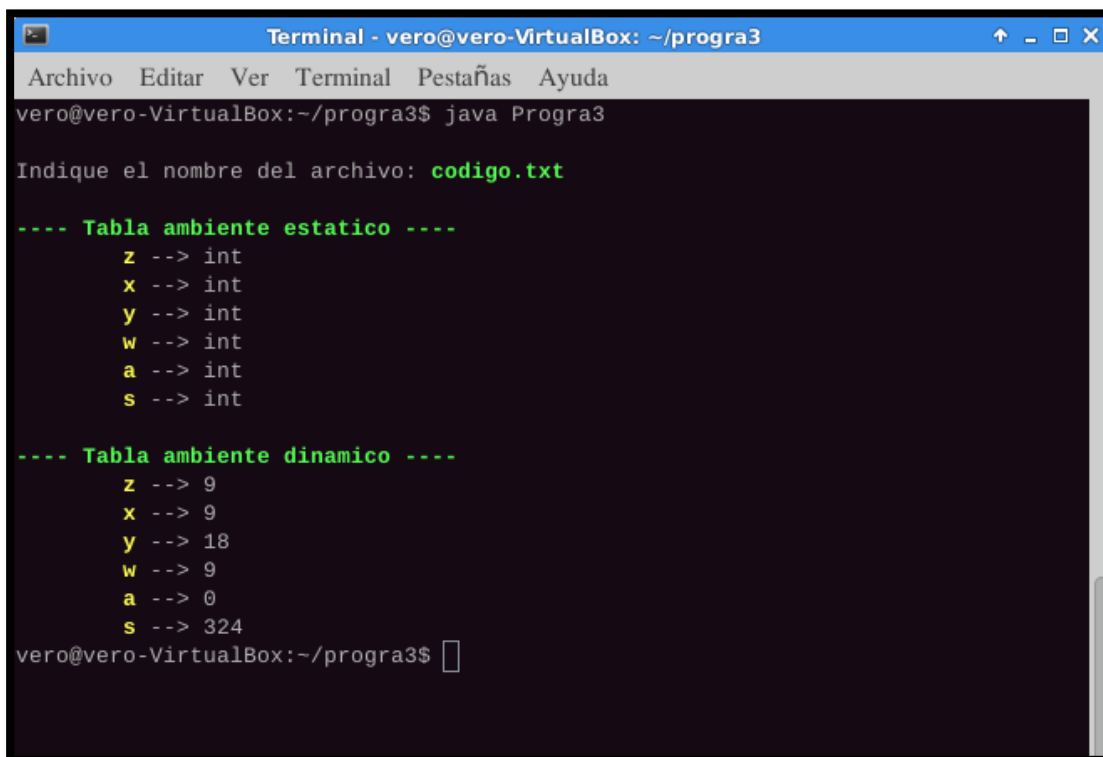
6. El usuario debe ingresar el nombre del archivo que contiene el código SML, es importante destacar que estos archivos deben estar en la misma ubicación, que se encuentran los archivos .java del programa.



```
Terminal - vero@vero-VirtualBox: ~/progra3
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
vero@vero-VirtualBox:~/progra3$ java Progra3

Indique el nombre del archivo: codigo.txt
```

7. Una vez indicado el nombre del archivo, que contiene el código SML. El programa realiza el análisis correspondiente y muestra en pantalla las tablas del ambiente dinámico y estático.



```
Terminal - vero@vero-VirtualBox: ~/progra3
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
vero@vero-VirtualBox:~/progra3$ java Progra3

Indique el nombre del archivo: codigo.txt

---- Tabla ambiente estatico ----
  z --> int
  x --> int
  y --> int
  w --> int
  a --> int
  s --> int

---- Tabla ambiente dinamico ----
  z --> 9
  x --> 9
  y --> 18
  w --> 9
  a --> 0
  s --> 324
vero@vero-VirtualBox:~/progra3$
```

## Lecciones aprendidas

- **Josué**

Mediante la realización de la presente tarea programada, se aprendió como manejar el scope de una variable independientemente del lenguaje que se esté utilizando, así mismo aprendí como se maneja de forma interna un programa en SML con el ambiente dinámico y con el ambiente estático que utiliza este lenguaje de programación"

- **Fabiana:**

Con el trabajo que desarrolle en la presente tarea programada adquirí conocimiento acerca de cómo se determina el tipo de una variable a partir del análisis del valor que se le asigna a esta."

- **José Daniel:**

Con la realización de esta tarea programada se aprendió mas acerca del funcionamiento interno de un programa en código del lenguaje SML, tanto de los ambientes estático y dinámico; también se aprendió más a fondo acerca del funcionamiento de las funciones let e if."

- **Verónica:**

Durante el desarrollo e implementación de esta tarea programada, la cual consiste en un analizador de ambiente estático y dinámico de un código en SML, desde el lenguaje de programación JAVA aprendí varios aspectos tanto del lenguaje SML como del paradigma funcional, las cuales son:

Logre tener una mejor claridad de como escribir códigos de SML, los alcances de cada expresión, variable o valor, también como se maneja el shadowing de las variables, y diferentes maneras para evaluar una expresión. Además, conocer más sobre el paradigma funcional, y algunas de las funcionalidades de este tipo de lenguaje, ya que, para lograr hacer el análisis en ambiente estático y dinámico tuvimos que investigar sobre dicho paradigma y más específicamente sobre código SML."