DEFINICIÓN DEL LENGUAJE DE PROGRAMACIÓN EPLOAM PARA LA EJECUCIÓN DE PSEUDOCÓDIGO Y SU COMPILADOR

Definition of the EPLOAM programming language for pseudocode execution and your compiler

RESUMEN
En la enseñanza de la algoritmia se capacita a los alumnos en la construcción de algoritmos en pseudocódigo y después en su implementación en lenguajes de programación de alto nivel. En el presente artículo se presenta un nuevo lenguaje de programación imperativo y estructurado, con estructuras similares a las de pseudocódigo, se describe su compilador y posterior traductor a programas equivalentes en Java, y se hace referencia a una herramienta adicional que permitirá automatizar el proceso de compilación, traducción y posterior ejecución de los algoritmos escritos.

PALABRAS CLAVES: Algoritmia, compilador, JavaCC, pseudocódigo, traductor.

ORLANDO ARBOLEDA
Ingeniero de Sistemas, M. Sc.
Profesor Auxiliar
Universidad Autónoma de Occidente
oarboleda@uao.edu.co

ABSTRACT
In the algorithms education the students are qualified in pseudocode algorithms construction and later in your high-level programming languages implementation. This paper present one new imperative and structured programming language, with structures similar to those of pseudocode, your compiler and translator to equivalent programs in Java, and one additional tool that will allow to automate the process of compilation, translation and later execution of the written algorithms.

KEYWORDS: *Algorithmics, compiler, JavaCC, pseudocode, translator.*

1. INTRODUCCIÓN

En las instituciones de enseñanza de algoritmia se capacita a los alumnos en la construcción de algoritmos en el pseudocódigo definido por la institución. Siguiendo este método, los algoritmos son escritos en papel o en procesadores de texto, pero no pueden ser vistos en ejecución, creando en los alumnos descontentos. Como última fase de este método de enseñanza de la algoritmia se capacita a los alumnos para construir programas ejecutables en lenguajes de programación de alto nivel que tienen estructuras similares a las existentes en el pseudocódigo adoptado.

Para apoyar la enseñanza de la algoritmia han sido construidos en español lenguajes y herramientas como SL y PSEINT que cuentan con versiones Windows y Linux [9,10]. Desgraciadamente son lenguajes no tipados o levemente tipados y con estructuras diferentes a las existentes en Java.

Entre las universidades colombianas que siguen el método de la enseñanza de la algoritmia descrito está la Universidad Autónoma de Occidente (UAO), que ha evaluado herramientas de visualización como VILLE [8], que no cuenta con todas las estructuras del pseudocódigo seguido.

Actualmente los compiladores pueden ser construidos con metacompiladores, los cuales permiten al diseñador olvidarse de los detalles repetitivos de bajo nivel y concentrarse en los componentes léxicos, sintácticos y semánticos deseados. Un buen metacompilador es JavaCC [1,2,4] que integra en un solo archivo la definición de componentes léxicos y las especificaciones sintácticas utilizando el lenguaje de programación Java.

En el presente artículo a partir del pseudocódigo seguido en la Universidad Autónoma de Occidente, se propone un lenguaje de programación imperativo y estructurado, el cual tendrá las estructuras típicas de un lenguaje de programación de alto nivel, que puede ser utilizado en la enseñanza de los cursos básicos de algoritmia. Se describe su compilador construido con el metacompilador JavaCC, en el cual se implementan las etapas léxicas, sintácticas y semánticas, con recuperación de error, para lograr una traducción a un archivo Java equivalente y sin errores. Finalmente se describe una herramienta que permite automatizar el proceso de traducción del programa inicialmente escrito en el nuevo lenguaje con la posterior compilación y ejecución del archivo Java obtenido.

Fecha de Recepción: 13 de Mayo de 2011
Fecha de Aceptación: 21 de Julio de 2011

2. JAVACC

JavaCC es un metacompilador en Java que facilita la construcción de analizadores léxicos y sintácticos por el método de funciones recursivas y permite notación similar a BNF.

Entre sus características tenemos que las especificaciones léxicas y sintácticas son ubicadas en el mismo archivo, por defecto reconoce gramáticas LL(1) pero puede reconocer gramáticas LL(K) y tiene gestión de errores que incluyen la línea y columna de ocurrencia [2].

En la Figura 1 se puede observar la estructura de un intérprete implementado en JavaCC sin recuperación de error, para sumar números enteros.

```
PARSER_BEGIN(Simple1)
public class Simple1 {
  public static void main(String args[]) throws ParseException {
    Simple1 parser = new Simple1(System.in);
    parser.Sumadora();
  }
}
PARSER_END(Simple1)

void Sumadora() :{}{
  (Expresion())* <FIN>
}

void Expresion() :{ int op1,op2; }{
  op1=entero() <OPERADOR> op2=entero() {System.out.println("="+(op1+op2));}
}

int entero():{}{
     <ENTERO> { return Integer.parseInt(token.image); }
}

SKIP : {
     " " | "\t" | "\r" | "\n"
 }
TOKEN [IGNORE_CASE] :{
       <ENTERO: (<DIGITO>)+>
     | <OPERADOR: "+">
     | <#DIGITO: ["0"-"9"]>
     | <FIN: "fin">
 }
```

Figura 1. Sumador de expresiones enteras escrito en JavaCC.

3. EL LENGUAJE DE PROGRAMACIÓN EPLOAM

El lenguaje de programación propuesto que ha sido denominado **EPLOAM**[1] (the **E**asiest **P**rogramming **L**anguage invented by **O**rlando **A**rboleda **M**olina) está basado en el pseudocódigo utilizado en el curso de Informática 1 impartido en la Universidad Autónoma de Occidente[2], el cual siendo imperativo y estructurado, está compuesto de estructuras nombradas en español, las cuales son equivalentes a las existentes en Java (que es usado para enseñar P.O.O en el curso de Informática 2).

En la notación EBNF de EPLOAM mostrada en la Tabla 1 se puede apreciar que corresponde a un subconjunto en español de las instrucciones y bloques presentes en Java.

[1] Un archivo comprimido conteniendo su descripción y compilador puede ser descargado desde http://ingenieria.uao.edu.co/EPLOAM/

[2] El pseudocódigo fue estandarizado e impartido desde el primer semestre de 2009, en segundo semestre a todos los estudiantes de la facultad de Ingeniería

<EPLOAM>	::=	(<comentario>)? “algoritmo” <nombre Algoritmo> “{” “principal” “(“ “)” “{” <declaraciones> <cuerpo> “}” “}”
<declaraciones>	::=	(<declaracion>)*
<declaracion>	::=	<tipo> <declara variable> (“,” <declara variable>)* “;”
<declara variable>	::=	<variable> (“=” < expresión >)?
<tipo>	::=	“entero” | “real” | “cadena” | “booleano” | “caracter”
<cuerpo>	::=	(<asignacion> | <lectura> | <impresion> | <decision> | <repeticion>)+
<asignacion>	::=	<variable> (<operador autoincremento> | <operador asignación><expresión>) “;”
<operador autoincremento>	::=	“++” | “--”
<operador asignación>	::=	“=” | “+=” | “-=” | “*=” | “/=”
<lectura>	::=	<variable> “=” “leer” “(“ <expresiónCadena> “)” “;”
<impresion>	::=	“imprimir” “(“ <expresiónCadena> “)” “;”
<decision>	::=	<decision simple> | <decision múltiple>
<decision simple>	::=	“si” “(“ <expresiónBooleana> “)” “{” <cuerpo> “}” (“sino” “{” <cuerpo> “}”)?
<decision múltiple>	::=	“segun” “(“ <variable> “)” “{” (“caso” <valor> “:” <cuerpo> (“termina” “;”)?)+ “otroCaso” “:” <cuerpo> “}”
<repetición>	::=	<ciclo Para> | <ciclo Mientras> | <ciclo Haga Mientras>
<ciclo Para>	::=	“para” “(“ <inicialización Para> “;” <expresiónBooleana> “;” <asignación Para> “)” “{” <cuerpo> “}”
<inicialización Para>	::=	“entero” <variable>”=”<expresiónEntera> | “real” <variable>”=”<expresiónReal>
<asignación Para>	::=	<variable> (<operador autoincremento>| <operador asignación> <expresión>)
<ciclo Mientras>	::=	“mientras” “(“<expresiónBooleana> “)” “{” <cuerpo> “}”
<ciclo Haga Mientras>		“haga” “{” <cuerpo> “}” “mientras” “(“ <expresiónBooleana> “)” “;”

Tabla 1. Gramática del lenguaje de programación EPLOAM.

Las expresiones validas en EPLOAM e indicadas en la Tabla 2 permiten el uso de los operadores aritméticos, relacionales, de igualdad y lógicos listados en la Tabla 3.

<expresión>	::=	<expr básica> | <expr básica>(<operador> <expr básica>)*
<expr básica>	::=	((“-“)? (<variable> | <constante entero> | <constante real> | <funciones>)) | <constante cadena> | <constante caracter> | <constante booleano >
<funciones>	::=	<nombre función> “(” <expresión> “)”
<nombre función>	::=	“raiz” | “seno” | absoluto | “potencia”
<operador>	::=	<operador aritmético> | <operador relacional> | <operador igualdad> | <operador lógico>

Tabla 2. Definición de expresión en EPLOAM.

4. COMPILADOR DEL LENGUAJE EPLOAM

Para el reconocimiento de programas escritos en EPLOAM se construyó su correspondiente compilador utilizando el metacompilador JavaCC. La finalidad del compilador es realizar el correspondiente análisis léxico, sintáctico y semántico, con recuperación de error en modo de pánico y obtener una traducción identada del programa Java equivalente.

En la Figura 2 se presenta la arquitectura del proceso de compilación en EPLOAM, la cual requiere como entrada un archivo de extensión .epl como el que se aprecia en la Figura 3, la indicación del directorio en el que se encuentra y el directorio en el cual se espera obtener el archivo Java equivalente si el archivo de entrada no tiene errores.

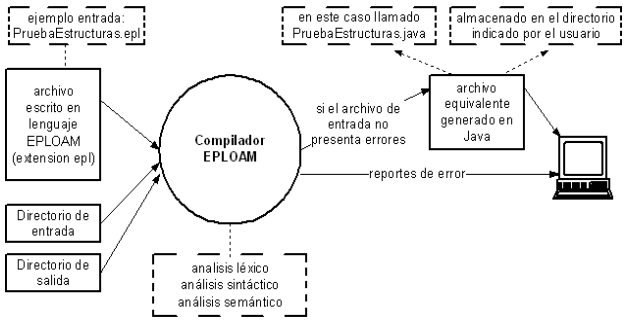
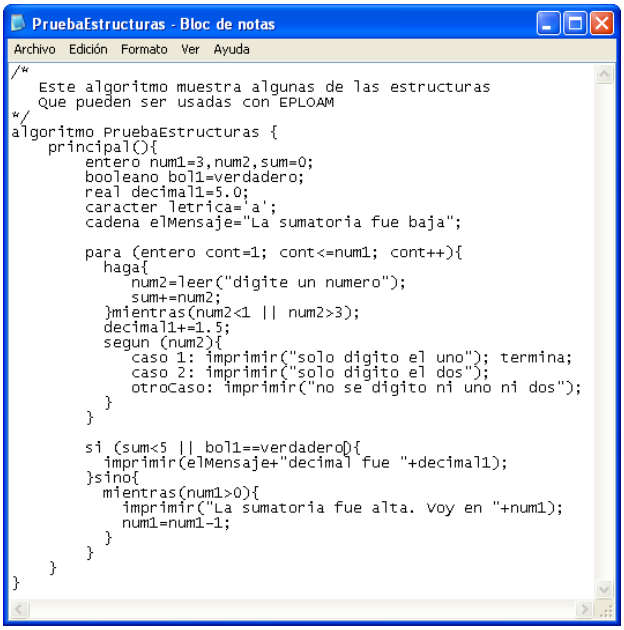


Figura 2. Arquitectura proceso de compilación en EPLOAM.




```
/*
   Este algoritmo muestra algunas de las estructuras
   Que pueden ser usadas con EPLOAM
*/
algoritmo PruebaEstructuras {
    principal(){
        entero num1=3,num2,sum=0;
        booleano bol1=verdadero;
        real decimal1=5.0;
        caracter letrica='a';
        cadena elMensaje="La sumatoria fue baja";

        para (entero cont=1; cont<=num1; cont++){
          haga{
             num2=leer("digite un numero");
             sum+=num2;
          }mientras(num2<1 || num2>3);
          decimal1+=1.5;
          segun (num2){
             caso 1: imprimir("solo digito el uno"); termina;
             caso 2: imprimir("solo digito el dos");
             otroCaso: imprimir("no se digito ni uno ni dos");
          }
        }

        si (sum<5 || bol1==verdadero){
          imprimir(elMensaje+"decimal fue "+decimal1);
        }sino{
          mientras(num1>0){
            imprimir("La sumatoria fue alta. Voy en "+num1);
            num1=num1-1;
          }
        }
    }
}
```

Figura 3. Ejemplo archivo *PruebaEstructuras.epl.*

4.1. DETALLES DEL COMPILADOR

En las subsecciones siguientes se mostrarán porciones de código JavaCC, a los cuales se les han eliminado las instrucciones que van generando la traducción.

4.1.1. Análisis Léxico

En la Figura 4 puede apreciarse un fragmento JavaCC, con la definición de tipos y los valores constantes permitidos.

```
TOKEN :
{
     <TIPOENTERO: "entero">
   | <TIPOREAL: "real">
   | <TIPOCARACTER: "caracter">
   | <TIPOBOOLEANO: "booleano">
   ……
   |  <ID: <LETRA>("_"|<LETRA>|<DIGITO>)*>
   | <ENTERO: (<DIGITO>)+>
   | <REAL: <ENTERO>"."<ENTERO>>
   | <CADENA: "\""(~["\""])*"\"">
   | <#LETRA: ["A"-"Z","a"-"z"]>
   | <#DIGITO: ["0"-"9"]>
   …….
}
```

Figura 4. Fragmento de la sección de tokens escrita en JavaCC.

El compilador propuesto será sensible a mayúsculas y minúsculas, con el fin de acostumbrar a los alumnos a la sensibilidad presente en los lenguajes de alto nivel.

4.1.2. Análisis Sintáctico

El mapeo de la gramática de EPLOAM mostrada en la Tabla 1 es inmediata en la sintaxis de JavaCC. Esto se puede apreciar en la Figura 5 donde se definen los tipos de instrucciones permitidas.

```
void cuerpo():{}{
   ( LOOKAHEAD(3)
        asignacion()
      | lectura()
      | impresion()
      | decision()
      | repeticion()
   )+
}
```

Figura 5. Implementación en JavaCC de las instrucciones permitidas en EPLOAM.

La definición de <expresión> indicada en la Tabla 2 es ambigua y aunque el compilador propuesto no va a generar código ejecutable sino una traducción correcta en Java, fue replanteada a una definición no ambigua que pudiera ser evaluada teniendo en cuenta la prioridad de los operadores permitidos los cuales son listados en la Tabla 3.

Operador	Descripción	Tipo operadores	Prioridad
- ()	unario y alteración de prioridad		mayor
* / %	multiplicación, división y residuo	aritméticos	
+ -	suma y resta		
< <= > >=	menor, menor-o-igual, mayor y mayor-o-igual	relacionales	
== !=	Igualdad y diferencia	de igualdad	
&&	and lógico	lógicos	menor
||	or logico		

Tabla 3. Prioridades de los operadores existentes en EPLOAM.

A nivel sintáctico no es posible definir que una expresión sea **expresiónCadena** o **expresiónBooleana** o de algún otro tipo requerido, porque estas pueden estar compuestas de variables o constantes de un tipo diferente.
Por ejemplo: si num1 es una variable de tipo entero, la expresión 5+num1>10 es una expresiónBooleana, a pesar que sus componentes son numéricos. Mientras que "el resultado es "+num1 que es una expresiónCadena.

El compilador posee recuperación de error a nivel de pánico. Un ejemplo de su implementación puede ser vista en el bloque catch de la versión simplificada en JavaCC mostrada en la Figura 6 para el reconocimiento de las estructuras de decisión simple.

```
void decisionSimple():{
   char tipoExpr;
}{
   try{
      <SI><PAREN_ABRE>
      tipoExpr=expr() {
          if (tipoExpr!=Simbolo.tipoBooleano) {
              despliegaError("condición no es valida");
           }
      }
      <PAREN_CIERRA>
      <LLAVE_ABRE>
      cuerpo()
      <LLAVE_CIERRA>
      (
         <SINO> <LLAVE_ABRE>
         cuerpo()
         <LLAVE_CIERRA>
      )?
   }catch(ParseException x){
      despliegaError("Error estructura de decisión");
      Token t;
      do {
         t = getNextToken();
      } while (t.kind != LLAVE_CIERRA);
   }
}
```

Figura 6. Implementación de estructura de decisión simple.

4.1.3. Análisis Semántico

Fue necesario definir semánticamente el resultado de la evaluación de una expresión, la cual depende del operador aplicado y del tipo de los operandos. Como el resultado de la evaluación de una expresión compuesta es conmutativa, en la Tabla 4 se resumen las decisiones semánticas realizadas al evaluar una expresión, las cuales permitirán determinar su tipo resultante.

Cualquier otra combinación de tipo de los operandos para cada uno de los operadores mostrados, provocará un tipo resultante no valido, desplegándose el mensaje de error correspondiente.

Como EPLOAM cuenta con definición explicita de tipos, todas las variables indicadas y separadas por comas hasta llegar al fin de la instrucción (;) son definidas del tipo indicado al inicio de la instrucción.

Tipo de los operandos	Operador	Tipo resultante
entero y entero	* / %	entero
entero y real		real
real y real		
entero y entero	-	entero
entero y real		real
real y real		
cadena y cadena	+	cadena
entero y entero		entero
entero y real		real
real y real		
tipo1 y tipo2 igual	== !=	booleano
entero y real	< <= > >=	
booleano y booleano	&& ||	booleano

Tabla 4. Tipos resultantes al evaluar las expresiones.

4.2. INTERACIÓN CON EL COMPILADOR

Para invocar al compilador de EPLOAM es necesario tener correctamente instalado a Java y adicionar en la variable de entorno PATH el directorio **bin** correspondiente.

La versión construida para el sistema operativo Windows es un archivo jar denominado *compiladorEPLOAM.jar*[3] que debe ser ejecutada desde el intérprete de comandos, con los argumentos definidos en la Figura 1 que son respectivamente: el nombre de la clase que contiene el compilador escrito en java (**EPLOAM**), el nombre del archivo a procesar sin incluirle la extensión epl, el nombre del directorio en donde se encuentra el archivo procesado y como último argumento el nombre del directorio en donde se desea almacenar el archivo Java obtenido.

En la Figura 7 se muestran las instrucciones para procesar al archivo denominado ***PruebaEstructuras.epl*** (mostrado en la Figura 3) existente en el subdirectorio ***Entrada*** y generar el archivo Java equivalente mostrado en la Figura 8 en el subdirectorio ***Salida***, ambos existentes en la carpeta C:\Documents and Settings\oarboleda\Escritorio\EPLOAM_V1.0.

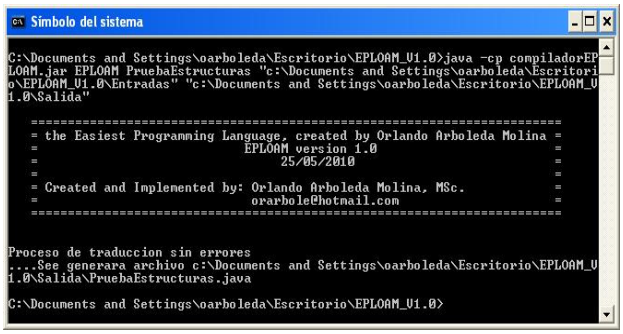


Figura 7. Ejemplo de invocación del compilador de EPLOAM.

[3] Es un archivo jar de solo 102Kilobytes

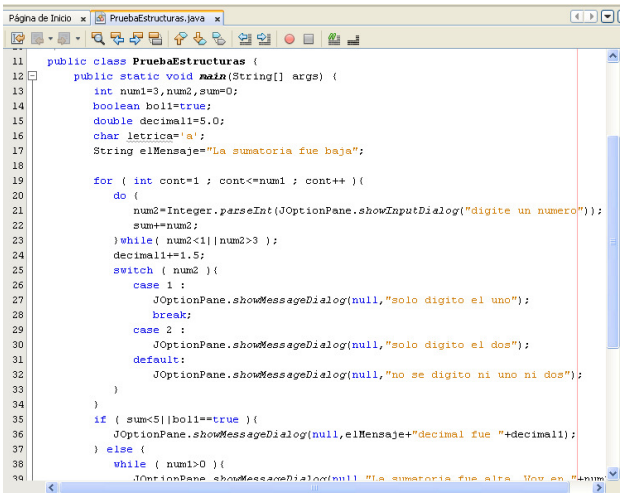


Figura 8. Archivo *PruebaEstructuras.java* generado por el compilador de EPLOAM abierto en Netbeans.

Al procesar un archivo de entrada erróneo, se despliegan cada uno de los errores encontrados y las posibles causas, como se puede observar en las Figuras 9 y 10.

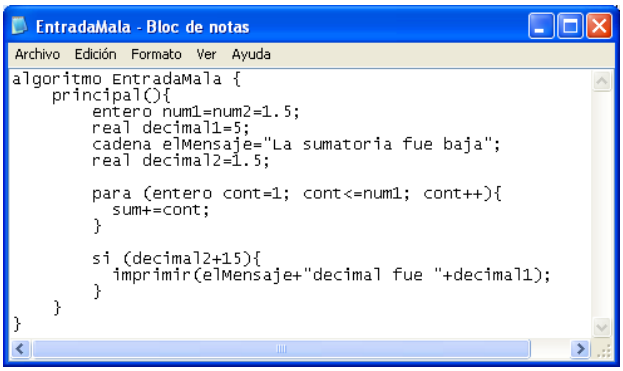


Figura 9. Ejemplo archivo de entrada con errores.

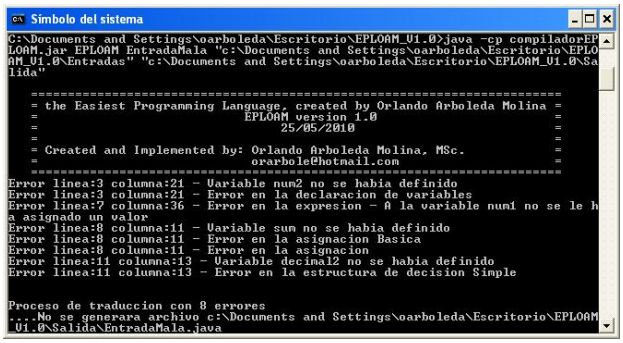


Figura 10. Ejemplo de la salida del procesamiento de un archivo con errores.

4.3. CARACTERISTICAS DEL COMPILADOR PARA EL USUARIO FINAL

El compilador puede ser usado por usuarios novatos y avanzados. Fue construido con el propósito de ser sencillo y explícito en el reporte de errores.

La sencillez de uso se obtiene con una estructura de directorios separada para los archivos de entrada y los archivos de salida. A nivel de programación se permite que las variables puedan ser inicializadas con un valor inmediato o con el resultado de la evaluación de una expresión. También puede ser agrupada en una misma instrucción la definición de variables que son del mismo tipo de datos.

El reporte de errores es explícito porque siempre se indican la línea, columna e indicación de la anomalía encontrada, la cual puede ser de carácter sintáctico o semántico.
Resumiendo el compilador provee:

- Una perfecta identación del archivo Java generado.
- Con respecto a la manipulación de archivos: validación de la existencia del archivo a procesar y que su nombre coincida con el nombre del programa.
- Indicación del número de errores encontrados.
- Indicación del nombre del archivo en Java generado.
- Validación de errores sintácticos para cada una de las estructuras definidas en la gramática de EPLOAM.
- Validación de errores semánticos para que el archivo generado sea correcto en Java. Ellas incluyen:
  - Coincidencia de tipos en la declaración de variables.
  - Coincidencia de tipos entre variables y expresiones que se les asignan.
  - Validación que las variables son declaradas una sola vez.
  - Validación que las variables existentes en una expresión hayan sido declaradas y tengan un valor.
  - Validación que las condiciones sean entre tipos compatibles y su resultado sea booleano.
  - Validación que la estructura de decisión múltiple (según caso) solo se realice con variables de tipo entero o de tipo carácter y que las constantes proporcionadas en los casos, sean del mismo tipo de la variable condicionada.
  - Validación que los datos solicitados sean de tipos distintos a booleano.
  - Validación que cualquier valor ingresado sea antecedido por el mensaje que lo solicita.
  - Validación que las expresiones existentes en la estructura de salida sean evaluados como cadena.

5. HERRAMIENTAS ADICIONALES IMPLEMENTADAS

Para automatizar el proceso de traducción del algoritmo en EPLOAM, la compilación y posterior ejecución del archivo Java obtenido se construyó un script denominado **lanzaPseudocodigo.bat,** el cual es esquematizado en la Figura 11.

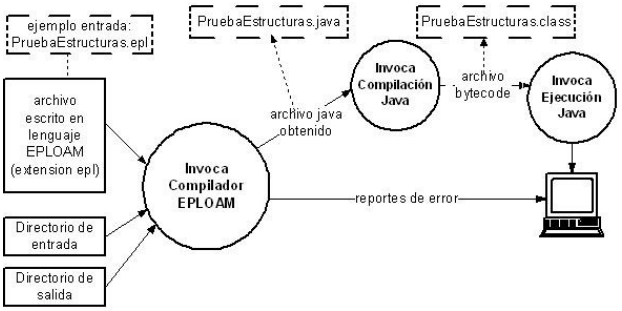


Figura 11. Procesos invocados por el archivo por lotes.

Como se puede apreciar en la Figura 12 el archivo por lotes va indicando cada uno de los procesos que va realizando y si el archivo de entrada es correcto se tendrá la ejecución automática del archivo en pseudocódigo suministrado.

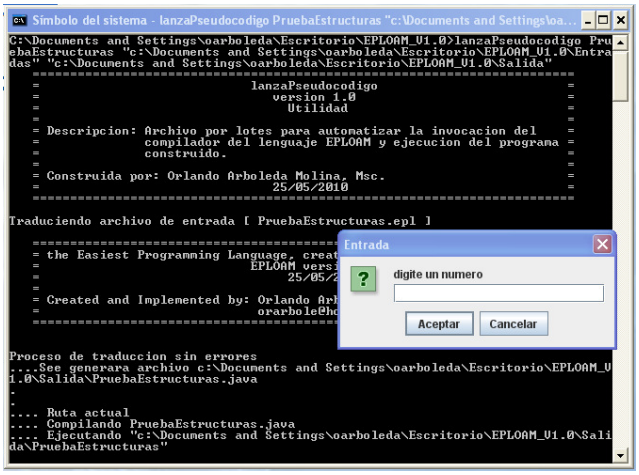


Figura 12. Invocación del archivo por lotes.

6. CONCLUSIONES Y RECOMENDACIONES

Se ha presentó un lenguaje de programación denominado EPLOAM que tiene las instrucciones típicas de pseudocódigo usado en la Universidad Autónoma de Occidente las cuales son similares a las existentes en los lenguajes de programación de alto nivel imperativos más utilizados actualmente.

Se diseño e implementó un compilador del lenguaje definido más una herramienta adicional para automatizar los pasos requeridos para la ejecución de los algoritmos construidos.

El compilador implementado es dirigido a los programadores novatos, pero es completamente funcional para ser utilizado por programadores más avanzados que deseen construir programas en Java o verlos en ejecución.

Las instrucciones proporcionadas por EPLOAM permiten que los alumnos se familiaricen con la sintaxis y semántica que encontrarían al tomar un curso de programación orientada a objetos en el lenguaje de programación Java, lo que no sucede con los otros lenguajes y herramientas existentes en español.

Los archivos producidos son identados, independiente del formato de los archivos de entrada, por lo tanto pueden ser usados para asimilar la sintaxis y semántica de los programas escritos en Java.

La propuesta presentada es funcional y puede ser usada como acompañamiento a un curso inicial de algoritmia.

Es necesario seguir trabajando en la realización de adiciones. En primer lugar, la adición de un entorno de desarrollo gráfico que permita: resaltado de palabras claves, numeración de líneas e invocación de los procesos de compilación y ejecución. Además, ampliar el lenguaje para permitir la inclusión de comentarios en el cuerpo de los archivos escritos en EPLOAM.

7. BIBLIOGRAFÍA

[1] S. Gálvez, M. M. A. Mora, *Java a tope: Compiladores-Traductores y compiladores con Lex/Yacc, JFlex/Cup y JavaCC*, Universidad de Málaga, 2005, pp. 1-229 [Online] Available: http://www.lcc.uma.es/~galvez/Compiladores.html

[2] JavaCC. *Project home*, [Online] Available: https://javacc.dev.java.net/

[3] A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman, *Compiladores. Principios, Técnicas y herramientas*, 2nd ed., Ed. Pearson Addison-Wesley, 2008.

[4] G. Succi, R. W. Wong, *The Application of JavaCC to Develop a C/C++ Preprocessor*. ACM SIGAPP Applied Computing Review, Vol. 7, Issue 3, Special issue on new applications of parsing tools, 1999, pp. 11-18, ISSN:1559-6915.

[5] A. Apple, *A modern compiler implementation in Java*, Cambridge: University Press, 1998.

[6] P. B. Mann, *A translational BNF grammar notation (TBNF)*. ACM SIGPLAN Notices, Vol. 41, Issue 4, April 2006, pp. 16-23, ISSN: 0362-1340.

[7] L. M. Garshol, *BNF and EBNF: What are they and how do they work?*, [Online] Available: http://www.garshol.priv.no/download/text/bnf.html

[8] T. Rajala, M. Laakso, E. Kaila, T. Salakoski, *VILLE – a language-independent program visualization tool*. In Proc. Seventh Baltic Sea Conference on Computing Education Research, Finland, CRPIT 88, 2007, pp. 151-159.

[9] PSEINT, *PIPEH Pseudo Interprete*, [Online] Available: http://pseint.sourceforge.net/.

[10] Lenguaje SL. *Bienvenidos a la página de SL*, Centro Nacional de Computación-CNC, Paraguay, [Online] Available: http://www.cnc.una.py/sl/.