Compilador y traductor de pseudocódigo para la lógica de programación (*CompiPrograMación*)

Pseudocode compiler and translator for programming logic

Carlos Alberto Vanegas

Ingeniero de Sistemas, Universidad Incca de Colombia, Especialista en Ingeniería de Software, Universidad Distrital Francisco José de Caldas y Magíster en Ingeniería de Sistemas, Universidad Nacional de Colombia. Docente de tiempo completo de la Universidad Distrital Francisco José de Caldas adscrito a la Facultad Tecnológica.
cavanegas@udistrital.edu.co

Fecha de recepción: abril 15 de 2005
Clasificación del artículo: investigación
Fecha de aceptación: junio 27 de 2005

Palabras clave: compilador, seudocódigo, algoritmo, lógica, programación.
Key words: compiler, pseudocode, algorithm, logic, programming.

Reumen

CompiPrograMación es una herramienta diseñada como recurso didáctico para la enseñanza de la lógica de programación. La herramienta maneja tres módulos. El primero permite la creación de algoritmos en pseudocódigo utilizando el paradigma estructurado; el pseudocódigo puede verificarse mediante un compilador que permitirá detectar posibles errores semánticos o sintácticos; además puede realizarse la traducción del seudocódigo a los lenguajes de programación C++ y Java. El segundo módulo permite hacer un seguimiento del pseudocódigo paso a paso; allí el usuario interactúa con las entradas y salidas del algoritmo. El tercer módulo es un evaluador de expresiones que mostrará al usuario el desarrollo de una expresión aritmética y/o lógica paso a paso.

Abstract

CompiProgramacion is a tool designed like didactic resource for education of programming logic. The tool handles three modules. First it allows to the creation of algorithms in pseudocode using the structured paradigm; the pseudocode can be verified by means of a compiler who will allow to detect possible semantic errors or syntactic; in addition the translation of the pseudocode to the programming languages C++ and Java can be made. The second module allows step by step to make a revision of the pseudocode; there the user interacts with the entrances and exits of the algorithm. The third module is an evaluator of expressions that will step by step show the user the development of an arithmetical and/or logical expression.

1. Introducción

Las técnicas de programación se constituyen en temática obligada de los usuarios o estudiantes de informática en sus primeros años de estudio. Esta circunstancia convierte las asignaturas específicas de programación de los planes de estudio en espacios académicos clave de la formación profesional; el éxito o fracaso en ellas influirá decisivamente en los estudios superiores restantes (Joyanes, 1998: 5). En esta área un curso de lógica de programación es indispensable; se pretende desarrollar la capacidad analítica y creadora del estudiante, para mejorar su destreza en la elaboración de algoritmos que sirvan de base para la codificación de los diversos programas que tendrán que desarrollar.

Para programar es importante seguir un estilo y una metodología apropiados. El propósito no es que el código fuente "quede más bonito", sino mejorar su calidad y eficacia. Hay que entender y asumir la diferencia entre quien consigue que sus programas funcionen y quien elabora sus programas en forma coherente, pero además logra mejorar su ejecución en términos de velocidad o consumo de recursos (Vanegas, 2005: 25).

Programar es algo más que aprender un lenguaje; es también tener el conocimiento previo para analizar y diseñar un algoritmo que dé solución a un problema planteado. Un lenguaje de programación no es más que el vehículo con el cual se da forma a las ideas, luego no es lo primordial; lo es más bien la lógica de programación, que a su vez aportará los parámetros de solución al problema que se desea resolver.

*CompiPrograma*ción es una herramienta que podrá emplearse como recurso didáctico para el proceso de formación en lógica de programación; además, permitirá al usuario, de manera interactiva, la creación de algoritmos y su depuración y seguimiento paso a paso; también le ayudará en la evaluación de expresiones aritmética y/o lógicas.

2. ¿Qué es *CompiPrograma*ción?

*CompiPrograma*ción es una herramienta de software didáctico desarrollada en el lenguaje de programación Java (Deitel, 2004); tiene como objetivo apoyar el aprendizaje de la lógica de programación utilizando el paradigma estructurado[1] . La herramienta cuenta con una interfaz comunicativa didáctica que permite al estudiante el fácil aprendizaje de la sintaxis y las reglas semánticas del seudocódigo; para ello dispone de un analizador gramatical con recuperación de errores, con submenús y barra de herramientas que permiten al usuario una interacción agradable y de fácil manejo.

La herramienta consta de tres módulos. El primero denominado "compilador y traductor de pseudocódigo"[2] a C++ y Java, permite escribir un pseudocódigo y compilarlo, con el fin de detectar posibles errores semánticos o sintácticos. Cuando la compilación se ha realizado con éxito, el usuario podrá traducir el pseudocódigo a los lenguajes de programación C++ y Java (autónomos y *applets*[3]). Un segundo módulo facilitará el seguimiento de un algoritmo en seudocódigo, ejecutándolo paso a paso y visualizando el efecto de cada línea de código sobre él; este proceso es similar a un depurador de código, pero en este caso la depuración no se realiza para buscar errores, sino para observar la ejecución de entradas y salidas del algoritmo. En un tercer módulo, llamado "evaluador de expresiones", pueden incluirse varias expresiones aritméticas y/o lógicas, con el fin de evaluarlas para determinar los valores que pueden tomar las diferentes expresiones, variables o constantes.

Con *CompiPrograma*ción pueden manejarse los siguientes conceptos del paradigma estructurado:

---

[1] El software fue desarrollado por el autor de este artículo como trabajo de grado para optar el título de Magíster en Ingeniería de Sistemas en la Universidad Nacional de Colombia, Bogotá D.C.

[2] El *pseudocódigo* es un lenguaje para especificar un algoritmo.

[3] El *applet* es un programa en Java, creado para ser ejecutado desde otra aplicación, normalmente desde un navegador de Internet.

- Las estructuras de secuencia
- La estructura de alternativa simple *Si*
- La estructura de alternativa compuesta *Si-Sino*
- La estructura de alternativa múltiple *Seleccionar Caso*
- La estructura de repetición *Mientras*
- La estructura de repetición *Haga / Mientras*
- La estructura de repetición *Desde / Para*
- Estructura de datos (*arreglos*)
- El manejo de funciones de usuario y recursivas
- El manejo de procedimientos
- Manejo básico de archivos

3. Módulo compilador de pseudocódigo y traductor a C++ y Java

Este es el módulo principal de *CompiPrograma-ción*. Cuenta con una interfaz que contiene tres ventanas: la primera permite crear, modificar o eliminar un pseudocódigo, abrirlo desde un archivo y almacenarlo en disco[4]. Para crear el pseudocódigo, es necesario que éste contenga un procedimiento principal, en el cual se escribirán todas las instrucciones del algoritmo. Su esquema es el siguiente:

```
procedimiento principal()
constantes
        <declaración de constantes>
variables
        <declaración de variables>
inicio
        <instrucciones>
fin_procedimiento
```

CompiProgramación cuenta con tipos de datos (reales, enteros) y palabras reservadas; estas últimas no pueden ser nombres de variables, constantes, funciones o procedimientos. Las palabras reservadas de CompiProgramación son:

- ***escribir*:** permite imprimir en pantalla un texto; su formato es:

  escribir (<variable o mensaje>)

  Por ejemplo, si se desea imprimir en pantalla "hoy es día de fiesta", se escribiría lo siguiente:

  escribir("hoy es día de fiesta")

- ***leer*:** permite almacenar en una variable un valor leído desde el teclado; su formato es:

  leer (<nombre de variable>)

  Por ejemplo, si se desea almacenar un valor digitado desde el teclado en una variable X, se escribiría lo siguiente:

  leer(X)

- ***cambioLinea*:** permite realizar un salto de línea en la pantalla; su formato es:

  escribir (cambioLinea)

  Por ejemplo, si se desea imprimir en pantalla "hola" en una línea y "CompiProgramación" en la siguiente, se escribiría así:

  escribir("hola")
  escribir(cambioLinea)
  escribir("CompiProgramación")

- ***si***: representa una toma de decisión sencilla. Si la condición es verdadera, se realizan las acciones que pertenezcan al *si* y se continúa con el resto del programa; si la condición es falsa no entrará al *si* y, en consecuencia, todas las instrucciones que se encuentran en el *si* no se ejecutarán; su formato es:

  si (<condición>) ***entonces***
        <acción>
  fin_si

  Por ejemplo, si A = 10 y B = 8

  si (A>B) ***entonces***
        escribir("Hola Gente")
  fin_si

[4] Al ingresar a la aplicación, esta contiene un ejercicio ejemplo de pseudocódigo.

Dado que la condición es verdadera (A es mayor que B), entrará al *si* e imprimirá "Hola Gente".

- ***Si - sino***: representa una toma de decisión con dos opciones, si la condición es verdadera se optará por el *si;* en caso contrario se ingresará al *sino*. En esta sentencia de control el programa sólo tomará una de las dos posibilidades y en ningún caso las dos alternativas; su formato es:

si (<condición>) ***entonces***
<accion1>
sino
<accion2>
fin_si

Por ejemplo, si A = 10 y B = 8

si (B>A) ***entonces***
escribir("entre por el si")
sino
escribir("entre por sino")
fin_si

Dado que la condición es falsa (B no es mayor que A), el algoritmo optará por el *sino* e imprimiría "entre por sino".

- ***Seleccionar***: es un proceso de toma de decisión con varias opciones; de acuerdo con el valor de una variable se escogerá una entre varias alternativas; su formato es:

seleccionar (<variable>) ***de***
caso 1<literal>:
<instrucciones>
retornar
caso 2 <literal>:
<instrucciones>
retornar
otrocaso:
<instrucciones>
retornar
fin_seleccionar

Por ejemplo, si A = 2

seleccionar (A) ***de***
caso 1:
escribir("en el caso uno")
retornar
caso 2:
escribir("en el caso dos")
retornar
otrocaso:
escribir("No es un número")
retornar
fin_seleccionar

En este ejemplo, dado que la variable A = 2 se ingresaría al caso 2 y se imprimiría en pantalla "en el caso dos".

- ***para / hasta:*** ejecuta las instrucciones del ciclo un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del ciclo; su formato es:

para (<variable> = <valor inicial> ***hasta*** <valor final> paso <valor_incremento>) ***hacer***
<instrucciones>
fin_para

Ejemplo 1: calcular la suma de los números de 1 a 10

suma=0
para (i = 1 ***hasta*** 10) ***hacer***
suma=suma +i
fin_para
escribir("suma igual a:"+suma)

En este caso, la variable suma obtendría un valor de: 55.

En algunos casos es necesario hacer incrementos o decrementos diferentes de 1 (por omisión, el incremento es de 1), es necesario especificar el incremento o decremento con la opción paso.

Ejemplo 2: hacer la suma de los números pares entre 2 y 20

para (i = 2 ***hasta*** 20 ***paso*** 2) ***hacer***
suma=suma +i
fin_para
escribir("suma igual a:"+suma)

En este caso la variable suma obtendría un valor de:110.

- ***mientras:*** en esta estructura repetitiva., el cuerpo del ciclo se repite mientras se cumple una determinada condición. Si ésta se evalúa falsa, no se toma ninguna acción y el programa prosigue con la siguiente instrucción después del ciclo. Si la expresión es verdadera se ejecutan las instrucciones del ciclo y luego se evalúa de nuevo la expresión. Este proceso se repite una y otra vez mientras la expresión sea verdadera; su formato es:

mientras (<condición>) ***hacer***
<instrucciones>
<incremento> o <decremento>
fin_mientras

Ejemplo: calcular la suma de los números de 1 a 10

suma=0
i=1
mientras i<=10 ***hacer***
suma=suma +i
i=i+1
fin_mientras
escribir("suma igual a:"+suma)

En este caso, la variable suma obtendría un valor de:55.

- ***haga / mientras:*** existen situaciones en las cuales se desea que un ciclo se ejecute al menos una vez antes de comprobar la condición de repetición. En la estructura *mientras*, si el valor de la expresión inicialmente es falso el cuerpo del ciclo no se ejecutará; por esto se necesitan otros tipos de estructuras repetitivas. La estructura haga/mientras se ejecuta por lo menos una vez; su formato es:

haga
<instrucciones>
mientras (<condición>)

Ejemplo : calcular la suma de los números de 1 a 10

suma=0
i=1
haga
suma=suma+i
i=i+1
mientras (i<=10)
escribir("la suma total es:"+suma)

En este caso, la variable suma obtendría un valor de: 55.

- ***funciones:*** una función puede definirse como un conjunto de instrucciones agrupadas bajo un nombre que cumple una tarea específica en un programa. Cuando los programas empiezan a ser más complejos, es necesario dividirlos en partes más pequeñas, denominadas funciones, para que cada una ejecute una tarea específica. Para invocar o llamar la función solo debe especificarse el nombre y la lista de parámetros entre paréntesis; su formato es:

funcion <nombre_funcion> (<arg1:tipo1,...>) : ***<tipo_del_valor_retornado>***
constantes
<declaración de constantes>
variables
<declaración de variables>
inicio
<instrucciones>
retornar <expresión>
fin_funcion

Por ejemplo, capturar dos números e imprimir su suma por medio de una función:

```
funcion suma(arg:entero, arg2:entero ):entero
   variables
         c:entero
   inicio
         c=arg+arg2
   retornar c
fin_funcion
```

Un ejemplo práctico de pseudocódigo, es:

```
procedimiento principal()
variables
   alto: real
   ancho: real
   largo: real
   volumen: real
inicio
   escribir("Escriba el alto: ")
   leer(alto)
   escribir(cambioLinea)
   escribir("Escriba el ancho: ")
   leer(ancho)
   escribir(cambioLinea)
   escribir("Escriba el largo: ")
   leer(largo)
   escribir(cambioLinea)
   volumen = largo * alto * ancho
   escribir("el volumen de la caja es: ")
   escribir(volumen)
fin_procedimiento
```

El pseudocódigo se compila para encontrar los posibles errores semánticos o sintácticos; la segunda ventana permite visualizar los errores generados en esta compilación; por cada error generado se muestra el identificador del error, su descripción y el número de fila y columna donde éste ocurrió. Por ejemplo:

```
Compilación terminada con errores. 3 errores.

Cod 160: Error Sin: Línea 9:6: Llamado a procedimiento: Se esperaba encontrar )

Cod 1033: Error Sem: Línea 17:6: Expresión a la derecha de = puede tener variables no inicializadas

Cod 1050: Error Sem: Línea 19:6: Llamado a procedimiento puede que no se haya inicializado una variable en el parámetro 1.
```

Una tercera ventana permite visualizar la traducción del seudocódigo a los lenguajes de programación C++ y Java (autónomos y *applets*); estas traducciones pueden ser almacenadas en disco. Continuando con el ejemplo anterior, al hacer la traducción a C++ la herramienta genera el siguiente código:

```
#include <iostream.h>
void main()
{
   float alto;
   float ancho;
   float largo;
   float volumen;
   cout<<"Escriba el alto: ";
   cin>>alto;
   cout<<"\n";
   cout<<"Escriba el ancho: ";
   cin>>ancho;
   cout<<"\n";
   cout<<"Escriba el largo: ";
   cin>>largo;
   cout<<"\n";
   volumen=largo*alto*ancho;
   cout<<"el volumen de la caja es: ";
   cout<<volumen;
}
```

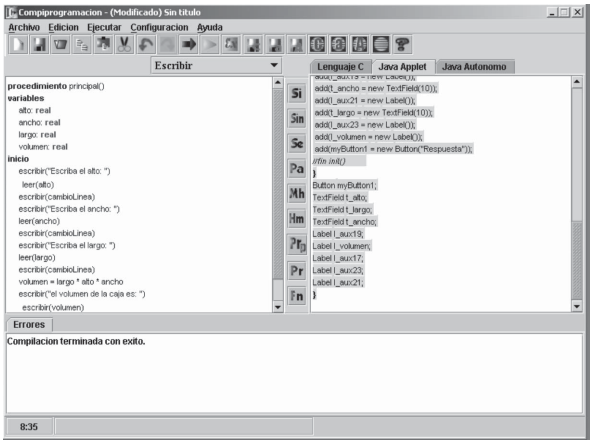
Al hacer la traducción a lenguaje Java en un subprograma *applet*, se genera el siguiente código:

```
import java.awt.*;
import java.applet.*;
public class MyClase extends Applet{
public boolean action(Event e, Object o) {
  float alto=0;
  float ancho=0;
  float largo=0;
  float volumen=0;
if(e.target==myButton1){
  alto = Float.parseFloat(t_alto.getText());
  ancho = Float.parseFloat(t_ancho.getText());
  largo = Float.parseFloat(t_largo.getText());
  l_aux17.setText(“”+“Escriba el alto: ”);
  l_aux19.setText(“”+“Escriba el ancho: ”);
  l_aux21.setText(“”+“Escriba el largo: ”);
  volumen=largo*alto*ancho;
  l_aux23.setText(“”+“el volumen de la caja es: ”);
  l_volumen.setText(“”+volumen);}
return true;}
    public void init(){
      add(l_aux17 = new Label());
      add(t_alto = new TextField(10));
      add(l_aux19 = new Label());
      add(t_ancho = new TextField(10));
      add(l_aux21 = new Label());
      add(t_largo = new TextField(10));
      add(l_aux23 = new Label());
      add(l_volumen = new Label());
      add(myButton1 = new Button(“Respuesta”));}
    Button myButton1;
    TextField t_alto;
    TextField t_largo;
    TextField t_ancho;
    Label l_aux19;
    Label l_volumen;
    Label l_aux17;
    Label l_aux23;
    Label l_aux21;}
```

La interfaz también cuenta con submenús y una serie de iconos que permiten que la interacción usuario-sistema se efectúe de forma sencilla y dinámica.

Al ejecutar los anteriores pasos se obtendrá la siguiente pantalla:

Figura 1. Compilador y traductor de pseudocódigo a C++ y Java



El compilador y traductor de pseudocódigo a C++ y Java está compuesto por analizadores gramaticales (léxico, sintáctico y semántico) con recuperación de errores. La herramienta toma como entrada un algoritmo en pseudocódigo, éste es compilado y evaluado por los analizadores gramaticales con el fin de detectar los posibles errores escritos en el pseudocódigo.

3.1 Analizador léxico

Es la parte del compilador[5] que verifica el programa fuente, caracter a caracter y, a partir de éste, construye unas entidades primarias llamadas *tokens*[6].

[5] Un compilador es un programa que recibe como entrada un programa escrito en un lenguaje de nivel medio o superior (el programa fuente) y lo transforma a su equivalente en lenguaje ensamblador (el programa objeto).

[6] El *token* es una palabra del lenguaje conformada por símbolos terminales de la gramática. En lenguaje Java un *token* puede ser variable, pero tendrá diferentes interpretaciones dependiendo del contexto; en cambio, el *token int* denota un tipo de dato entero.

En otras palabras, el analizador lexicográfico transforma el programa fuente en unidades lexicográficas (Vanegas, 2005: 14), (TEUFEL, 1995).

3.2 Analizador sintáctico

Comprueba que las sentencias que componen el texto fuente sean correctas en el lenguaje correspondiente, creando una representación interna que corresponde a la sentencia analizada. De esta manera se garantiza que sólo serán procesadas las sentencias que pertenezcan al lenguaje fuente. Así como en las demás etapas, durante el análisis sintáctico, se van mostrando los errores que se encuentran (Vanegas, 2005: 14; Teufel, 1995).

3.3 Analizador semántico

Se ocupa de analizar si la sentencia tiene algún significado. Pueden encontrarse sentencias sintácticamente correctas, pero que no pueden ejecutarse porque carecen de sentido. En general, el análisis semántico se hace simultáneamente con el sintáctico, introduciendo en este último rutinas semánticas (Vanegas, 2005: 15; Teufel, 1995).

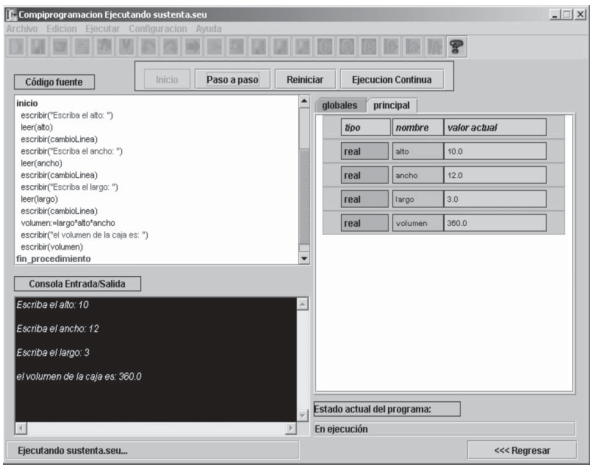
4. Módulo ejecución pseudocódigo paso a paso

Para acceder a este módulo debe existir un pseudocódigo previamente compilado (sin errores sintácticos ni semánticos) y almacenado en disco.

Este módulo de ejecución paso a paso contiene la misma interfaz del módulo "compilador y traductor de pseudocódigo a C++ y Java". En él se encontrarán tres ventanas: la primera permite al usuario hacer un seguimiento del algoritmo paso a paso, visualizando línea por línea el pseudocódigo; en la segunda ventana se observarán los diferentes valores que toman las variables y constantes contenidas en el algoritmo; una tercera ventana permite visualizar la simulación de una consola con las entradas y salidas del programa. El módulo también contiene las opciones que le permiten realizar el seguimiento del algoritmo, a saber: inicio, paso a paso, reiniciar y ejecución continúa.

La pantalla del módulo paso a paso es la siguiente:

Figura 2. Ejecución seudocódigo paso a paso



5. Módulo evaluador de expresiones

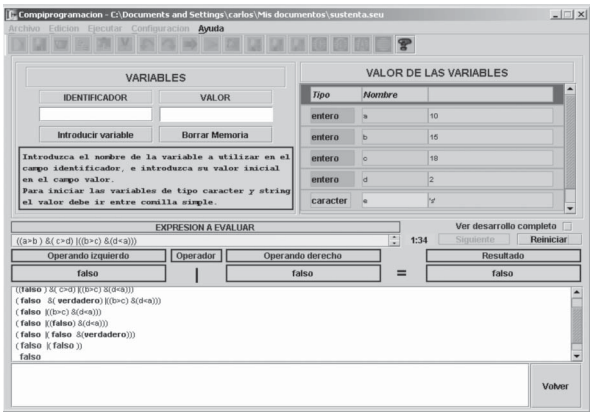
Para acceder a este módulo no es necesario que exista un pseudocódigo, pero sí es necesario acceder el módulo "compilador y traductor de pseudocódigo a C++ y Java". Allí el usuario puede hacer la evaluación de una expresión aritmética y/o lógica y visualizar paso a paso el procedimiento de evaluación de la expresión.

El módulo cuenta con cuatro ventanas. La primera permite el ingreso de variables; el usuario puede escribir el nombre y valor de cada una de las que intervienen en la evaluación de la expresión; en una segunda ventana se visualiza el valor de cada variable mostrando la clase de dato, nombre y valor de la variable. La tercera ventana permite visualizar el proceso de evaluación paso a paso y en la cuarta se visualizan los posibles errores contenidos en la expresión.

El evaluador de expresiones está compuesto por los analizadores gramaticales que son utilizados en el módulo "compilador y traductor de pseudocódigo a C++ y Java".

La pantalla del módulo de evaluación de expresiones se presenta en la figura 3:

Figura 3. Evaluador de expresiones



6. Conclusiones

- *CompiProgramación* es una herramienta que sirve de apoyo didáctico a los interesados en la lógica de programación, con el propósito de analizar y diseñar algoritmos que den solución a un problema planteado.
- La herramienta cuenta con una interfaz comunicativa diseñada con propósitos pedagógicos, que facilita al usuario una interacción agradable con el sistema y es de fácil manejo. El usuario siempre estará en contacto con la aplicación, permitiéndole manejar los diferentes módulos de la aplicación con sencillez.
- *CompiProgramación* contiene una ayuda didáctica que permite al usuario la operación sencilla para la creación de pseudocódigo y la traducción a los lenguajes de programación C++ y Java; además, puede hacerse un seguimiento paso a paso del pseudocódigo para determinar las entradas y salidas del algoritmo.
- *CompiProgramación* contiene una ayuda en ambiente Windows que permite conocer todo acerca de la aplicación, gramática del seudocódigo, relación de errores sintácticos y semánticos identificables y funciones preexistentes.

Referencias bibliográficas

[1] DEITEL, H y DEITEL, P. (2004) *Cómo programar el Java*. 5 edición. México D.F.: Prentice Hall.

[2] JOYANES A., L. (2003). *Fundamentos de programación, algoritmos y estructura de datos*, 1ª ed. México D.F.: Prentice Hall.

[3] TEUFEL B., SCHMIDT S. y TEUFEL T. (1995) *Compiladores, conceptos fundamentales*. Wilmington, Ed. Addison-Wesley.

[4] VANEGAS, Carlos A. (2005) *Compilador y traductor de pseudocódigo para la lógica de programación*. Trabajo de grado para optar al título de Magíster en Ingeniería de Sistemas. Bogotá: Universidad Nacional de Colombia.

Bibliografía recomendada

[1] AHO J., ULLMAN J. D. (1998). *Compiladores: principios, técnicas y herramientas,* 1ª ed., Ed. Addison-Wesley.

[2] BECERRA S., César. (1993) *Lenguaje C, el nuevo concepto*. Tomo I. Bogotá: Kimpres Ltda.

[3] BOOCH, G. (1996) *Análisis y diseño orientado a objetos con aplicaciones*. 2ª. ed. Ed. Addison – Wesley.

[4] CEBALLOS, M.A. (2002) *Trabajo de lenguajes y autómatas,* ITESI.

[5] JOYANES A., L. (1998) *Fundamentos de programación, algoritmos y estructura de datos*, 1ª ed. México D.F.: Prentice Hall.

[6] LEMONE, K. (1996) *Fundamentos de compiladores, cómo traducir al lenguaje de computadora*. 1ª. ed. México D.F.: Continental.

[7] PRESSMAN, R. (1994) *Ingeniería del software, un enfoque práctico*. 3ª. ed. Madrid: McGraw Hill.

[8] The on-line Java 2 SDK Documentation.2001. En: http://java.sun.com/j2se/1.3/docs/index.html