 ISBN 978-612-47015-1-1 ISSN 2305-9958

HITO: Pseudocode compiler with graphics library

Instituto de Investigación en Ciencias de la Computación - UNAP-Perú[13]
Universidad Nacional del Altiplano Puno – Perú [3]
Universidad Nacional Micaela Bastidas de Apurímac - Perú [12]
Ecler Mamani[1], Juan C. Muñoz M[2], Nayer Tumi[3]
eclervirtual@iicc-unap.pe[1], juancarlomm@iicc-unap.pe[2] , nayertumi@iicc-unap.pe[3]

Abstract— HITO is a compiler designed for teaching programming logic. It consists on 3 modules. The first one compiles algorithms in pseudo code. The language was defined using BNF notation without checking possible errors by syntactic and semantic analyzers. The second one allows you to translate the pseudo code to C++ programming languages, Java and C#. The third one allows you to draw primitive's graphics. The software was developed with XP methodology. System and usability unitary test were performed. The results of the evaluation regarding the syntax of the algorithms were "Strongly agree" and "agree" valuation given for 9 experts. It was a descriptive level research with application type.

***Palabras Claves*— compilador, pseudocódigo, algoritmo, BNF, sintaxis, programación, traducción, léxico, sintáctico, semántico, lenguaje, grafica.**

I. NOMENCLATURA

BNF:	Backus-Naur Form
DFA:	Deterministic Finite Automaton
CSG:	Context-Sensitive Grammar
XP:	eXtreme Programming
GLP:	General Public Licence

II. INTRODUCCION

El Desarrollo de compilador es una tarea que reconforta a un científico de computación o informático, un compilador es un programa como cualquier otro, pero un poco grande y complejo (1) así también son programas de computador que traducen un programa escrito a otro lenguaje (2). Un estudiante que se inicia en el mundo de programación con toda seguridad se ha realizado la clásicas pregunta: ¿Qué lenguaje de programación utilizar para comenzar a programar?, podría ser ninguna, la experiencia demuestra que mientras no dominemos los fundamentos de programación mediante algoritmos o que son códigos falsos de lenguaje de programación (3) y diagramas para entender la lógica de programación. El propósito de la investigación es el desarrollo del analizador léxico, sintáctico y semántico elementos que componen el compilador de pseudocódigos el cual facilite el aprendizaje de la lógica de programación (4) de modularidad basados en algoritmos y pseudocódigos, el producto final una herramienta funcional para la actividad educativa, el uso de las herramientas cuenta con alta tradición (5) (6) (7) a la hora de educar, y con múltiples beneficios desde la perspectiva de la actividad docente permitiendo enfatizar o ilustrar determinados ejercicios que a veces resultan dificultosos de enfatizar sin tal soporte. Las fases de la construcción del compilador (8) se realizó utilizando la metodología XP, una metodología basada esencialmente en la simplicidad y agilidad (9), permitiendo que cada elemento y componente del compilador sea comprobado en la planeación, diseño y codificación.

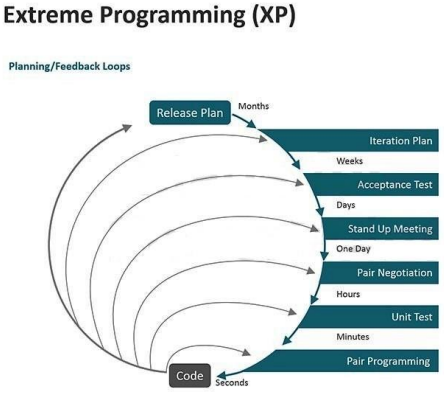

Figura 1. Planificación de la Metodología XP

El compilador es de una sola pasada el cual genera el código máquina a partir de una única lectura del código fuente, teniendo tres dos fases de desarrollo; Front-End y Back-End, destacando que no hay un acuerdo general de la dividir el compilador, en la principal bibliografía consultada (10), sólo se reconocen las etapas Front-end y Back-end, pero, cada vez más es reconocido una nueva etapa intermedia, debido a la importancia que ésta está tomando hoy día el uso de una representación intermedia.

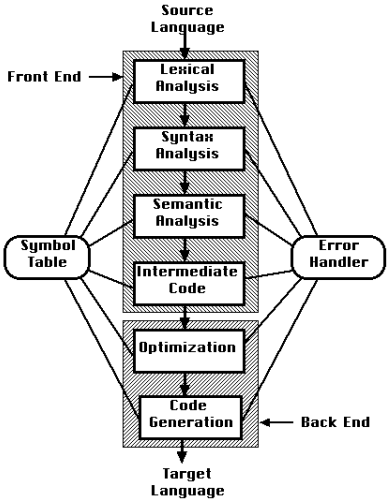

Figura 2. Fases de un compilador

Estas fueron desarrolladas independientes del analizador léxico, sintáctico, semántico, generación de código intermedio y generación de código por separados

 ISBN 978-612-47015-1-1 ISSN 2305-9958

y desarrollados en el lenguaje de programación Visual Studio 2012 a partir de la gramática escrita en la notación BNF.

El módulo de generación de código está dividido en dos, la tabla de símbolos y tabla de código intermedio los cuales permiten la traducción de pseudocódigo a código C#, Java y C++, al mismo tiempo se agregó la librería gráfica System.Drawing, el cual permite generar las gráficas primitivas. Además se consideró las etapas de especificación de requisitos, estos están basados en las funcionalidades básicas del entorno de desarrollo. Se realizó las etapas de diseño, implementación, pruebas e integración en el entorno de desarrollo. Para la evaluación de la evaluación de la sintaxis de control fue realizado mediante el juicio de expertos (11).

III. DESARROLLO DEL COMPILADOR

A. ANÁLISIS DE REQUERIMIENTOS FUNCIONALES

USER STORIES. Se usó para la especificación de requerimientos, tal como lo sugiere la metodología XP (9) , sobre la cual está basada la metodología utilizada en este documento y los que fueron identificados son:

- Al cerrar la aplicación. Pregunta si se desean guardar los cambios, se puede realizar desde alguna opción de la barra de menú.
- Al abrir la aplicación. Muestra la ventana principal, barra de menú y permite crear y editar el pseudocódigo.
- Archivo. Crea, abre y editar, archivos planos de texto con extensión (hito).
- Barra de menú de comandos y operadores. Permite insertar sintaxis de estructuras.
- Vista de creación archivo. Muestra un editor del código con la estructura básica de un código fuente y en diferentes colores del texto.
- Vista de edición. Editar un archivo de Pseudocódigo y muestra las herramientas, definidas a libre disposición del usuario.
- Guardar archivos. Guardar los cambios en el mismo documento o realizar una copia del mismo.
- Imprimir. Impresión de archivos de código fuente y vista previa de impresión.
- Verificar Sintaxis. Muestra los posibles errores sintácticos y semánticos del código fuente.
- Ejecutar: Inicia la compilación en caso de que no exista errores.
- Vista de errores. Muestra el cuadro con los errores.
- Ayuda. Acceso desde la barra de menú e icono de barra de herramientas.
- Traducción del Pseudocódigo. Exporta a C#, C++ y Java, antes debe estar guardado.
- Ejemplos. Muestran ejemplos para utilizar en las herramientas del compilador.

B. ANÁLISIS DE REQUERIMIENTOS NO FUNCIONALES

- Ambiente de ejecución. Sistema operativo Windows XP y versiones superiores.
- Requerimientos Técnicos. Tener instalado el .NET Framework 2.0 o versiones superiores.
- Licenciamiento: GNU- GPL ver. 3, de libre distribución.

C. PROGRAMACIÓN DEL COMPILADOR

Realizado mediante componentes los cuales se observan en la figura 3

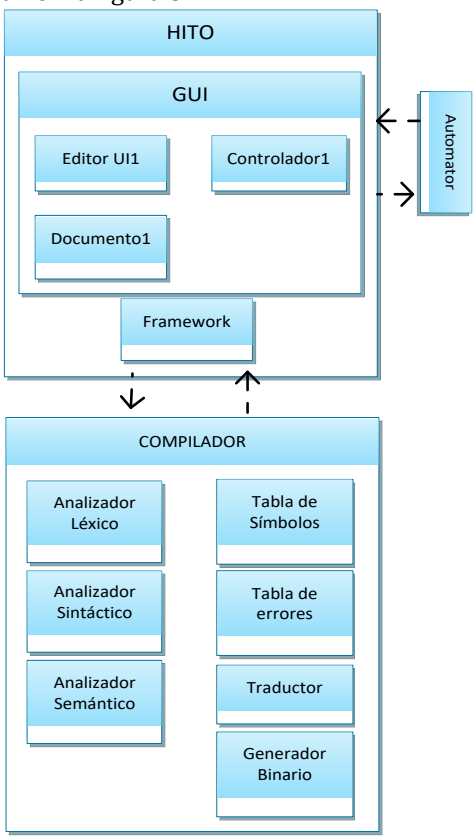

Figura 3. Diagrama de Componentes del compilador

La descripción de los componentes según se detalle: GUI, encargado de administrar y proveer la interacción con el usuario subdividido en:

- Documento1 que es la representación lógica de los datos donde el usuario crea y edita, donde el GUI, se encarga de visualizar el pseudocódigo por el usuario.
- El componente controlador administra el intercambio entre el editor gráfico y el documento con el framework, iniciando la primera fase del compilador.
- El framework; destinado cargar dinámicamente las librerías usadas son enlazados por el automator que fue diseñado y codificado como una librería dll con la finalidad de ser reusable.

ISBN 978-612-47015-1-1 ISSN 2305-9958

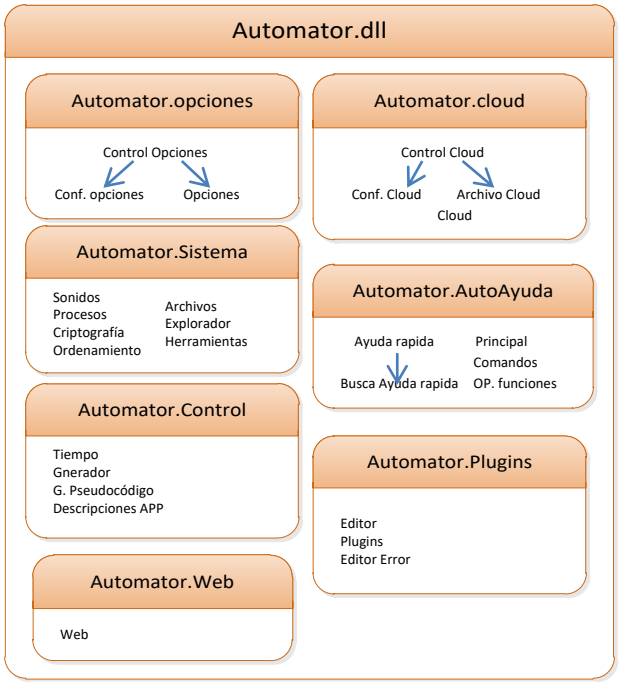

Figura 4. Diagrama del automator.dll

El componente compilador realiza el análisis del proceso de compilación y generando el árbol sintáctico convirtiendo al lenguaje de alto nivel definido. Teniendo así el analizador léxico es el proceso que sirve para identificar que los símbolos de entrada pertenecen al alfabeto de un lenguaje, como se observa en la figura 5.

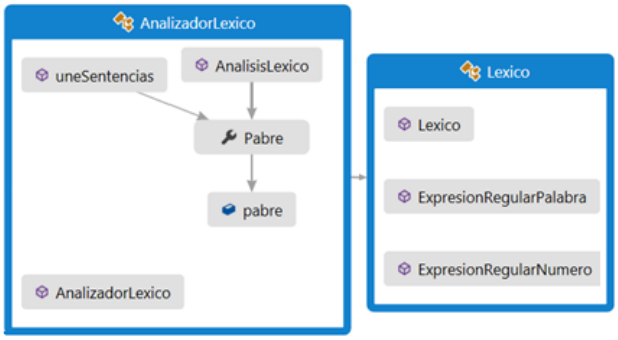

Figura 5. Diagrama del analizador léxico

La definición de sus respectivos lexemas[1] en el código fuente del compilador se encuentran dentro de un arreglo dinámico de caracteres:

```
Char
lexpal[MAXPAL]={"abrirArchivo","absoluto","algoritmo",
"arccos","arcsen","arctan","buscarCadena","cadena",
"cadenaMayusculas", "cadenaMinusculas","caracter",
"cerrarArchivo","comienza","compararCadenas",
"compararCaracteres","concatenar","coseno","div",
"entero","entonces","escribir","escribirln","exp",
"extraerCadena","fin","finalArchivo","finmientras",
"finpara","finsi","funcion","hacer","hasta","inicio",
"leer","leerArchivo","ln","log","longitudCadena",
"mientras","mientrasque","mod","para","paso","pi",
"potencia","principal","procedimiento","raiz","real",
"redondear","ref","repetir","retornar","seno","si",
"sino","tangente","termina","terminaPrograma",
"truncar"};
```

La obtención de Tokens, se implementa en el scanner, basado en DFAs para reconocer expresiones regulares, la figura 6 identifica si es una palabra reservada o identificador, donde; q0: Estado inicial, q1: Estado de aceptación de un identificador y q2: Estado de rechazo.

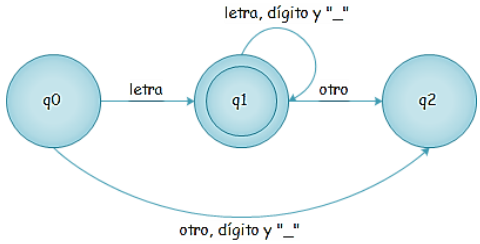

Figura 6. DFA para identificadores y palabras reservadas.

En la fig. 7, observamos DFA para reconocer un numero entero o real, donde q0: Estado inicial, q1: Estado de aceptación de un número entero, q2: Estado de aceptación de un número real y q3: Estado de rechazo.

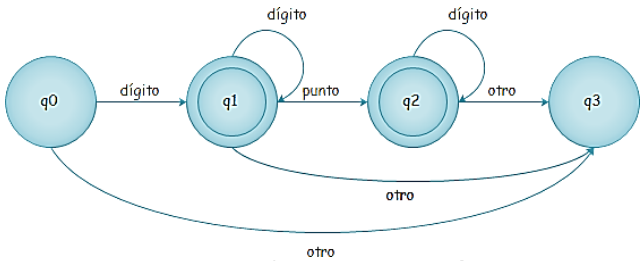

Figura 7. DFA para números enteros y reales.

Para reconocer la cadena de caracteres se encuentra dentro de comillas, observamos en la figura 8, donde q0: Estado inicial, q1: Estado de rechazo, q2: Estado de aceptación de una cadena y q3: Estado de rechazo.

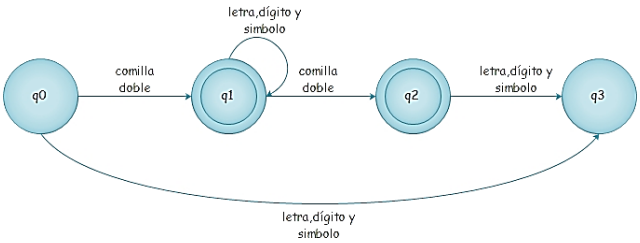

Figura 8. DFA para reconocer cadena de caracteres

El analizador léxico descendente forma parte de una gramática LL1 independiente del contexto, como resultado la generación de árbol sintáctico.

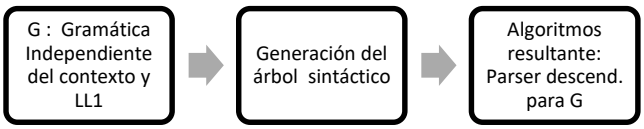

Figura 9. Analizador sintáctico descendente

Segmento de códigos, muestran las expresiones regulares de estructuras de control retornadas por el analizador sintáctico.

```
public static string ExpresionRegularInicioDeAmbito()
{       return @"^Inicio$";     }
public static string ExpresionRegularFinDeAmbito()
  {     return @"^Fin$";    }
public static string ExpresionRegularComienzoDeIf()
{  return @"<<Si\s\(\s+\w+\s(<|>|<:|>:|::|!:)
\s\w+\s\)\s\Entonces$";  }
public static string ExpresionRegularComienzoDeElse()
```

[1] Unidades léxicas que genera el analizador léxico.

ISBN 978-612-47015-1-1 ISSN 2305-9958

```
{  return @"<<Sino\s*\FinSi$"; }
public static string ExprRegularComienzoDeSwitch()
{ return @"#Segun\s\(\s\w+\s(<|>|<:|>:|::|!:)
\s\w+\s\)\s\Hacer$"; }
```

La tabla de errores desarrollado en dos secciones, el primero contiene un arreglo de cadenas y en cada una corresponda a un posible error generado, la segunda una función encargada de mostrar el error generado mostrando el número de línea del posible el error, observamos el segmento de código del arreglo.

```
public void IniciaListaTablaErrores(){
ListaTablaErrores.Clear();
ClaseTablaErrores objte = new ClaseTablaErrores(0,
"valor incorrecto", "escriba el valor aceptado por el
tipo de variable", "valor diferente al aceptado por
el tipo");
ListaTablaErrores.Add(objte);
ClaseTablaErrores objte1 = new ClaseTablaErrores(1,
"se espera un valor", "escriba un valor para la
variable ", "se espera un valor despues de...");
//Errores sintácticos detectados por el parser.
ClaseTablaErrores objte80 = new ClaseTablaErrores(80,
"warning", "Limitación específica del compilador. El
programa fuente es demasiado largo", "");
ListaTablaErrores.Add(objte80);
```

La tabla de símbolos organizada linealmente en un ArrayList con un apuntador al primer y último registro añadido a la tabla, función encargada de reservar espacio en la memoria para su registro y que sea usado para poder crear un registro vacío.

El analizador semántico, examina el código fuente y verifica las reglas semánticas donde los componentes recorren el árbol sintáctico en la figura 10 observamos los sub componentes:

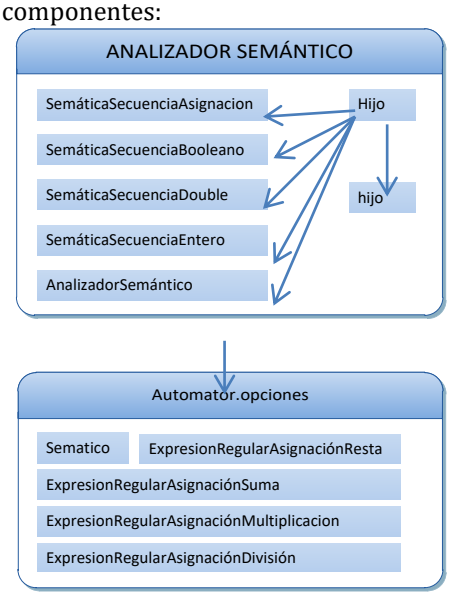

Figura 10. Componentes del analizador sintáctico

El traductor, traduce el código fuente del Pseudocódigo en español a código C#, al finalizar el recorrido del árbol sintáctico, se guarda en un archivo con extensión cs para ser procesado por System.CodeDom.Compiler[2] y ejecutar el archivo con extensión exe. El compilador desarrollado con el fin de ofrecer una capa de abstracción entre el árbol sintáctico y la generación del código destino, el cual se observa en el segmento de código:

```
public ArrayList Compilar(string CodigoFuente,string
Direccion,string NombreArchivo,Boolean Ejecutable)
{  ArrayList ListaErrores = new ArrayList();
   CSharpCodeProvider codeProvider = new
CSharpCodeProvider();
ICodeCompiler icc = codeProvider.CreateCompiler();
string Output = Explorador.DireccionMisDocumentos()+
"\\HITO\\Compilados\\"+NombreArchivo  + ".exe";
System.CodeDom.Compiler.CompilerParameters parameters
= new CompilerParameters();
parameters.GenerateExecutable = true;
parameters.OutputAssembly = Output;
parameters.ReferencedAssemblies.Add("System.dll");
parameters.ReferencedAssemblies.Add("System.Windows.F
orms.dll");
parameters.ReferencedAssemblies.Add("System.Drawing.d
ll");
parameters.ReferencedAssemblies.Add("System.Drawing.D
esign.dll");
CompilerResults results =
icc.CompileAssemblyFromSource(parameters,
CodigoFuente);
   if (results.Errors.Count > 0) {
          foreach (CompilerError CompErr in
results.Errors)
        { ClaseErrores objErrores = new
        ClaseErrores
        (Automator.Control.Generador.GeneraID(),
        CompErr.ErrorText, CompErr.Line, true);
        ListaErrores.Add(objErrores);
         }
   } else
           { if (Ejecutable == true)  {
                  Process.Start(Output);
              }
           }
return ListaErrores;}
```

D. PROCESO COMPILACIÓN DEL PSEUDOCÓDIGO

1. El pseudocódigo en texto plano es pasado al intérprete como una cadena de caracteres.
   - Al recibir la cadena de caracteres, el intérprete pasa al analizador léxico, el cual busca e identifica los lexemas definidos en la tabla de símbolos, utilizando el motor de expresiones regulares.
   - Identificados los lexemas, son almacenados y revisados por un Parser, que actúa como analizador léxico y sintáctico, se encarga de la construcción del árbol sintáctico, identificando los posibles errores que existiesen.
   - Con el árbol sintáctico, el intérprete pasa sobre el analizador semántico y se encarga de buscar y reportar errores.

2. Procesado el texto, y transformado al árbol sintáctico, se envía al compilador generando los objetos ejecutables.
   - El Compilador utiliza un traductor quien se encarga de transformar el árbol sintáctico en código de algún lenguaje de alto nivel (C#), y guardado en un archivo de texto.
   - El archivo de texto es enviado al compilador del lenguaje de alto nivel mediante la librería ICodeCompiler Interface (System.CodeDom. Compiler) para generar el archivo ejecutable.

ISBN 978-612-47015-1-1 ISSN 2305-9958

E. PRUEBAS

1. **Unitarias**, se evalúo por cada componente, verificando y validando el correcto funcionamiento del mismo, en caso de que el componente no pasara la prueba, se hizo una reiteración, para corregirla y seguir con el proceso de revalidación y verificación.
2. **Sistema**, para asegurar los requerimientos funcionales y no funcionales extraídos de los User Stories.
3. **Usabilidad**, se realizaron 5 versiones en base a los cuestionarios de los usuarios.

IV. RESULTADOS

La evaluación del compilador fue con el método Delphi, creado para investigar el impacto de la tecnología en la guerra (11) (12), mediante el cuestionario de juicio de expertos(13) en referencia a la sintaxis de los algoritmos que son compilados, el criterio para la selección de expertos fue basada en la experiencia (14) teniendo 9 expertos, los datos se muestran en la tabla I.

TABLA I.- VALIDACIÓN DEL COMPILADOR POR JUICIO DE EXPERTOS

Item	VALORACIÓN				Total
	1	2	3	4	
Estruc. de control	0	0	5	4	9
Modularidad	0	1	4	4	9
Recursividad	0	0	3	6	9
Total	1	3	15	18	37

1 = Muy en desacuerdo 2=En desacuerdo
3 = De acuerdo 4= Muy de acuerdo

Los datos de la tabla I, muestra el consolidado final de la valoración por los expertos como valores máximos a **de acuerdo** y **muy de acuerdo** en la sintaxis funcional, visualizando en el gráfico 11.

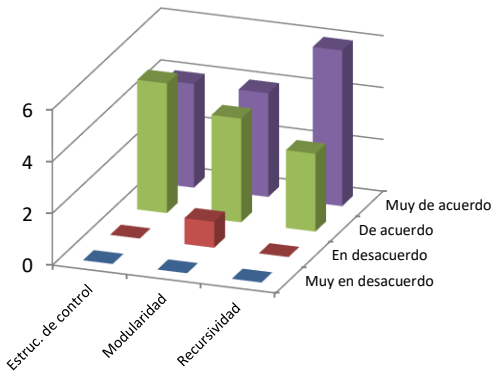

Figura 11. Valoración obtenida por juicio de expertos

La Figura 12, muestra los archivos ejecutables generados por el compilador.

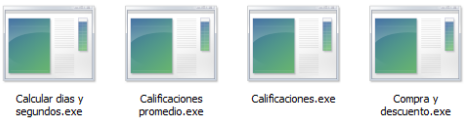

Figura 12. Archivos compilados

El editor de pseudocódigo fue desarrollo con la librería DotNetBar WinForms con menús, controles y barras de herramientas de estilo VS.NET 2005.

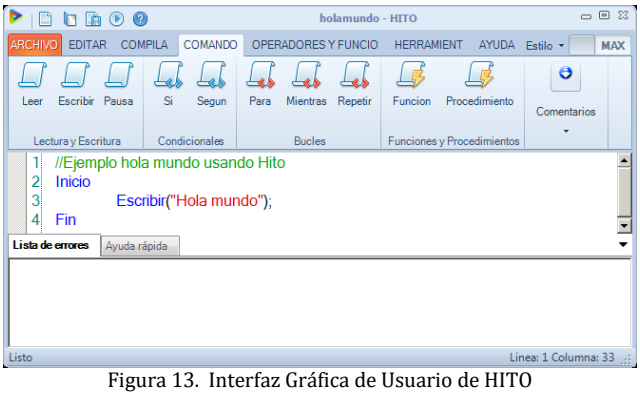

Figura 13. Interfaz Gráfica de Usuario de HITO

Pseudocódigo de funciones que calcula el factorial de un número usando recursividad.

```
//Ejemplo de factorial usando  funciones
Inicio
        entero numero,resultado;
        Escribir("FACTORIAL DE UN NUMERO\n");
        Escribir("Ingrese un numero:");
        Leer(numero);
        resultado=Factorial(numero);
        Escribir("El factorial es :  "+resultado);
Fin
//función factorial
Funcion entero Factorial(entero num)
        Si(num==0) Entonces
                retornar 1;
        Sino
                retornar num*Factorial(num-1);
        FinSi
FinFuncion
```

Ejemplo de gráfico que se muestra en la fig. 14 generado mediante pseudocódigo.

```
Inicio
 entero i;
 Para( i=0;i<=5;i++)Hacer
    DibujarRectangulo(0+i*10,0+i*10,200,200,3,2);
 FinPara
Fin
```

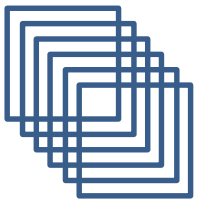

Figura 14. Grafico Generado a partir del pseudocódigo.

En referencia de la fig. 13, se muestra el pseudocódigo exportado a C# en la figura 15.

ISBN 978-612-47015-1-1 ISSN 2305-9958

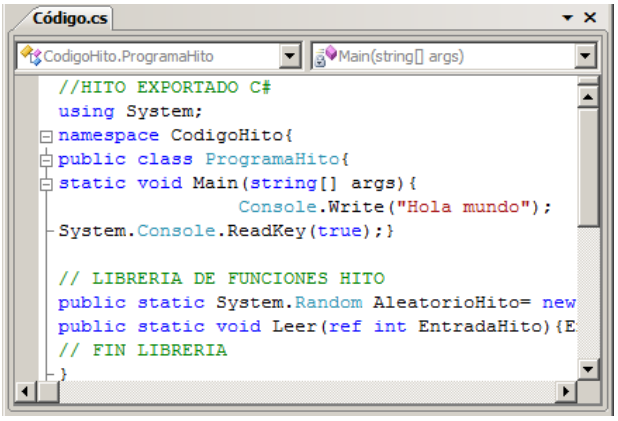

Figura 15. Pseudocódigo generado en C#:

V. CONCLUSIONES

1. Se desarrolló una herramienta educativa para la enseñanza de la programación, un compilador funcional de Pseudocódigos y todos sus componentes.
2. El compilador fue desarrollado con metodología ágil XP.
3. Las pruebas realizadas al sw fueron unitarias, sistema y de usabilidad.
4. El GUI del compilador se desarrolló con la librería DotNetBar Win Forms.
5. Se incorporó al compilador la librería gráfica System.Drawing, el cual permite generar las gráficas primitivas.
6. Los resultados de la evaluación con respecto a la estructura de los algoritmos son; "Muy de acuerdo" y "de acuerdo" valoración otorgada mediante resolución de nueve expertos docentes que regentan los cursos de fundamentos de programación y algorítmica.
7. Hito se encuentra para su descarga en: https://sourceforge.net/projects/hitojcmm/

VI. RECOMENDACIONES Y TRABAJO A FUTURO

Los lenguajes de programación C y C++ son dos de los lenguajes más populares y utilizados en la actualidad por lo cual se recomienda desarrollar este compilador para la plataforma Linux, así también mejorar e incrementar más funciones la librería gráfica.

Así mismo se recomienda usar esta herramienta en los centros educativos de nivel secundario y superior.

VII. REFERENCIAS

1. **ANDREU, SANCHEZ DUEÑAS Y VALVERDE.** *COMPILADORES E INTÉRPRETES UN ENFOQUE PRACMÁTICO*. MADRID : LAVEL, 1989.
2. **Torczon, Keith D. CooperLinda.** Construindo Compiladores (Segunda edição). Brazil : Elsevier, 2014.
3. **Ecler Mamani Vilca y J. Reynaldo Paredes.** *Fundamentos de Programación con C++ y Phyton.* [ed.] Vladimiro Ibañez. I. Puno : América, 2012.
4. **Velloso, Fernando.** *Informática: Conceitos Básicos (Nono Edição).* São Paulo – SP – Brasil : Elsevier Editora Ltda., 2014. ISBN 978-85-352-7790-6.
5. *"Guidelines for a graduate curriculum on embedded software and systems" ACM Transactions on Embedded Computing Systems* . **Caspi, P., Folher, G., Garcia-Valls, M., Kopetz, H., Lakhnech, Laroussinie, F., Lavagno, L., Lipari, G., Maraninchi, F., Peti, P., Puente,.** 5, New York, NY, USA : ACM, 2005, Vol. 4. 10.1145/1086519.1086526.
6. **Bouyssounouse, Bruno, Sifakis, Joseph.** *Embedded Systems Design The ARTIST Roadmap for Research and Development.* USA : Springer, 2005. ISBN 978-3-540-31973-3.
7. *"Guest Editorial Special Section on Information Technologies Within Engineering Education", Industrial Informatics.* **Gomes, Rodriguez-Andina and.** 1, USA : IEEE Transactions, 2013, Vol. 9.
8. *Modern compiler implementation in Java.* **W. Appel Andrew, J. Palsberg.** Princeton University, New Jersey : Princeton , 2002. 978-0521820608.
9. **Joskowicz, José.** *Reglas y Prácticas en eXtreme Programming.* 2006.
10. *Compiladores : principios, técnicas y herramientas. s.l.* **Aho, Alfred V., Sethi, Ravi y Ullman, Jeffrey D.** 0201629038, s.l. : Addison-Wesley Iberoamericana, 1990.
11. *Qualitative Research: Consensus methods for medical and health services research.* **Jones J, Hunter D.** 1995, BMJ, Vol. 311.
12. *El procesamiento de la información en investigaciones educacionales.* **Cruz M, Campano A.** 11 de 2008, Education Cabana, Vol. 25.
13. *Validation by expert judgements: two cases of qualitative research in Applied Linguistics.* **Rojas, Pilar Robles Garrote y Manuela del Carmen.** 18, Febrero de 2015, Nebrija.
14. **Fernández, Sandra Hurtado de Mendoza.** *histodidactica.* [En línea] 2016. [Citado el: 21 de mayo de 2016.] http://www.ub.edu/histodidactica/index.php?option=com_content&view=article&id=21:criterio-de-expertos-su-procesamiento-a-traves-del-metodo-delphy&catid=11:metodologia-y-epistemologia&Itemid=103.

VIII. BIOGRAFIAS

M.Sc. Ing. Ecler Mamani Vilca, Graduado en la Universidad Nacional del Altiplano 2002 Ingeniero Informático 2001– Maestro en Educación 2008 - Perú, experiencia en desarrollo de aplicaciones multimedia, innovador tecnológico e investigador en informática, docente de la Universidad Nacional Micaela Bastidas de Apurímac – Perú en la Escuela Profesional de Ing. Informática y Sistemas. Miembro activo del Instituto de Investigación en Ciencias de la Computación – UNAP, Perú

Juan Carlos Muñoz Miranda, en la Universidad Nacional Micaela Bastidas de Apurímac – Ingeniería Informática y Sistemas, experiencia en desarrollo de aplicaciones web, móvil y escritorio, imparto formación presencial, cursos online y sesiones de tele presenciales, diseño 3D para múltiples ámbitos, desde la generación de presentaciones 2D - 3D, hasta el diseño CAD de producto industrial y desarrollo de video juegos 3D. Docente del centro de informática e internet – UNAMBA, Perú.

M.Sc. Ing. Ernesto Nayer Tumi Figueroa, Estudios de Doctorado en la Universidad de Chile 2003, Graduado en la Universidad Nacional del Altiplano 1994 Ingeniero Estadístico 1994– Magister en Informática 2008 - Perú, Docente de la Universidad Nacional del Altiplano Puno, enseña los cursos de Algoritmos, Sistemas Operativos, Biocomputación, Estructura de Datos, en la Escuela Profesional de Ing. Estadística e Informática. Director del Instituto de Investigación en Ciencias de la Computación de la Escuela de Post Grado– UNAP, Perú,