PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DE UNA HERRAMIENTA DE TRADUCCIÓN Y VERIFICACIÓN DE PROGRAMAS DISEÑADOS EN DIAGRAMA DE FLUJO UTILIZANDO COMO PASO INTERMEDIO PSEUDOCÓDIGO

Tesis para optar por el Título de **Ingeniero Informático**, que presente el bachiller:

Jairo Abner Palomino Masco

ASESOR: Ing. Gissella María Bejarano Nicho

Lima, Junio del 2014

Resumen

Este proyecto de fin de carrera parte de la ausencia de herramientas que sirvan de ayuda para aquellos alumnos que quieran aprender a programar. Si bien existen programas que permiten generar diagramas de flujo y luego exportarlos a un lenguaje de programación, estos no poseen un compilador que permita detectar errores de sintaxis, que pueda poseer la solución. Por ello se plantea implementar un proceso de traducción y verificación de programas diseñados en diagrama de flujo utilizando como paso intermedio pseudocódigo, para que el alumno pueda preocuparse más en la lógica de su solución, que en la sintaxis o estructura del lenguaje de programación, a codificar.
Para realizar este proyecto se definió una serie de objetivos específicos. En primer lugar se desarrolló un entorno para dibujar diagramas de flujo, para ello se adaptaron las librerías de Microsoft office que ofrecen las herramientas necesarias para la creación del entorno de trabajo.

Luego, se definió la gramática que almacena la sintaxis del pseudocódigo utilizando la notación Backus-Naur Form (BNF). Después se implementó el método de conversión del formato XML de Microsoft office, representación del diagrama de flujo, a pseudocódigo utilizando la sintaxis definida anteriormente y un editor de texto en el cual se muestre el resultado de la conversión al usuario.

Finalmente se implementó el intérprete que utiliza la gramática para verificar que el código se encuentre léxicamente, sintácticamente y semánticamente correcto. De esta manera los alumnos podrán obtener a partir de un diagrama, el cual al ser gráfico es de fácil entendimiento, el código de su programa sin necesidad de conocer la sintaxis del mismo y validado.

FACULTAD DE
CIENCIAS E INGENIERÍA
ESPECIALIDAD DE
INGENIERÍA INFORMÁTICA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO INFORMÁTICO

TÍTULO:	IMPLEMENTACIÓN DE UNA HERRAMIENTA DE TRADUCCIÓN Y VERIFICACIÓN DE PROGRAMAS DISEÑADOS EN DIAGRAMA DE FLUJO UTILIZANDO COMO PASO INTERMEDIO PSEUDOCÓDIGO
ÁREA:	Ciencias de la computación
PROPONENTE:	Ing. Gissella María Bejarano Nicho
ASESOR:	Ing. Gissella María Bejarano Nicho
ALUMNO:	Jairo Abner Palomino Masco
CÓDIGO:	20080528
TEMA N°:	634
FECHA:	10 de Setiembre de 2014

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Facultad de Ciencias e Ingeniería
06 OCT. 2014
MIGUEL MEJIA PUENTE
DECANO

DESCRIPCIÓN

Este proyecto de fin de carrera parte de la ausencia de herramientas que sirvan de ayuda para aquellos alumnos que quieran aprender a programar. Si bien existen programas que permiten generar diagramas de flujo y luego exportarlos a un lenguaje de programación, estos no poseen un compilador que permita detectar errores de sintaxis, que pueda poseer la solución. Por ello se plantea implementar un proceso de traducción y verificación de programas diseñados en diagrama de flujo a pseudocódigo, para que el alumno pueda preocuparse más en la lógica de su solución, que en la sintaxis o estructura del lenguaje de programación a codificar.

Para realizar este proyecto se definieron una serie de objetivos específicos. En primer lugar se desarrolló un entorno para dibujar diagramas de flujo, para ello se adaptaron las librerías de Microsoft office que ofrecen las herramientas necesarias para la creación del entorno de trabajo.

Luego, se definió la gramática que almacena la sintaxis del pseudocódigo utilizando la notación Backus-Naur Form (BNF). Después, se implementó el método de conversión del formato XML de Microsoft office, representación del diagrama de flujo, a pseudocódigo utilizando la sintaxis definida anteriormente y un editor de texto en el cual se muestre el resultado de la conversión al usuario.

Finalmente se implementó el intérprete que utiliza la gramática para verificar que el código se encuentre léxicamente, sintácticamente y semánticamente correcto. De esta manera el proyecto ayudará al alumno a enfocarse en la lógica de la solución más que en la sintaxis de un lenguaje de programación. Así mismo el programa brindará al alumno las herramientas gráficas necesarias para que desarrolle sus diagramas de flujos y pueda llegar a una solución planteada correctamente en pseudocódigo.

Av. Universitaria 1801
San Miguel, Lima – Perú

Apartado Postal 1761
Lima 100 – Perú

Teléfono:
(511) 626 2000 Anexo 4801

OBJETIVO GENERAL

Implementar un proceso de traducción y verificación de programas diseñados en diagrama de flujo a pseudocódigo.

OBJETIVOS ESPECÍFICOS

1. Desarrollar un entorno para dibujar diagramas de flujo adaptando las librerías de Microsoft office.
2. Diseñar la gramática definida para el pseudocódigo que será utilizado por el intérprete.
3. Implementar un método de conversión del formato XML de Microsoft office, que representa al diagrama de flujo, a pseudocódigo.
4. Implementar un intérprete para verificar la lógica, sintaxis, semántica del pseudocódigo.

ALCANCE

El paradigma que se soportará es el imperativo.

El entorno para dibujar diagramas de flujo consta de las siguientes partes:

- Una barra de herramientas que contiene las siguientes funciones: Nuevo documento, Abrir, Guardar, Guardar Como, Salir, Ejecutar.
- Una paleta de herramientas con las siguientes estructuras: lectura y escritura de datos, operaciones aritméticas simples, estructuras selectivas y estructuras iterativas.
- Una hoja de trabajo donde se arrastren los objetos para crear el diagrama de flujo.

El código obtenido de la traducción del diagrama a pseudocódigo tendrá un editor de texto que visualice al usuario el código obtenido.

El intérprete será llamado desde una consola para verificar la lógica y sintaxis del código generado e imprimir los errores detectados en un archivo de texto.

Máximo: 100 páginas

Av. Universitaria 1801
San Miguel, Lima – Perú

Apartado Postal 1761
Lima 100 – Perú

Teléfono:
(511) 626 2000 Anexo 4801

Dedicatoria

Dedico la presente tesis a las personas que me apoyaron a lo largo de mi vida universitaria y en especial a mi familia que siempre me brindó el apoyo necesario.

Agradecimientos

A mis asesores, Layla Hirsh por apoyarme y guiarme durante la primera parte del curso de tesis y Gissella Bejarano por asesorarme y guiarme hasta el final exitoso del mismo.

También agradecer a mi familia y a todas las personas de las cuales he obtenido conocimiento.

Tabla de contenido

Índice de Gráficos y Tablas

CAPÍTULO 1: PROBLEMÁTICA, OBJETIVOS Y ALCANCES

1.1 Introducción

En el presente capítulo se hablará de la problemática de este proyecto de fin de carrera, que en breve resumen es la falta de unificación de herramientas como la generación de código a partir del diagrama de flujo y un compilador que verifica su sintaxis para ayudar a los alumnos que deseen aprender a programar.

También se abordarán los objetivos planteados para desarrollar este proyecto. Así como las herramientas utilizadas, el alcance, las limitaciones, la justificación, la viabilidad y el plan de ejecución del proyecto.

1.2 Problemática

Según estudios realizados en el período 2009, se ha identificado que existe una gran cantidad de alumnos desaprobados en los cursos introductorios de programación (TAN 2009: 42).

El mayor problema reside en el aprendizaje del alumno de las estructuras lógicas y reglas de sintaxis de un lenguaje de programación en particular. Por ello, para que el alumno se enfoque más en desarrollar la lógica de la solución, en vez de buscar como traducirlo a un lenguaje de programación, existen herramientas que le permiten diseñar su solución tales como el diagrama de flujo y pseudocódigo.

El diagrama de flujo tiene la ventaja de ser simple para el entendimiento debido a que ofrece una solución gráfica, además de una sintaxis definida y limitada, reduciendo el número de errores que puede cometer el usuario (BIKAS 2005: 2). Pero el dibujar los diferentes bloques y sus conectores puede generar dificultades en especial para las personas con pocas habilidades artísticas (MARRER 2009: 51).

Para mejorar e incentivar el uso del diagrama de flujo existen herramientas que facilitan la creación del diagrama de flujo mediante el uso de componentes drag and drop haciendo más interactivo e intuitivo al usuario por ofrecer un ambiente de trabajo visual. También permiten exportarlo a un lenguaje de programación, pero al no contar con un compilador el usuario desconoce que el código generado es el correcto, por ende necesita de una herramienta adicional para comprobar que la sintaxis del código sea la correcta.

Ante la falta de integración que se encuentra se propone implementar un proceso de traducción y verificación de programas diseñados en diagrama de flujo utilizando como paso intermedio pseudocódigo.

Para verificar la utilidad de la herramienta propuesta se realizó una encuesta para los estudiantes del curso de introducción a la computación de la Universidad Católica en la cual se obtuvo los siguientes resultados: el 90% de los encuestados afirmó que tiene problemas en el diseño de la solución (ya sea con el uso de pseudocódigo o diagrama de flujo), el 60% prefiere no utilizar las herramientas de diseño, a menos que se les exija, debido a que no se puede verificar y ejecutar, alrededor del 80% asegura que tener una herramienta que permita que les permita a partir del diagrama de flujo generar código en pseudocódigo sería de

mucha utilidad debido a que ya no necesitan realizar una tarea doble de generar el diagrama de flujo y luego traducirlo a pseudocódigo.

1.3 Objetivo general

Implementar un proceso de traducción y verificación de programas diseñados en diagrama de flujo utilizando como paso intermedio pseudocódigo.

1.4 Objetivos específicos

- Objetivo 1: Desarrollar un entorno para dibujar diagramas de flujo adaptando las librerías de Microsoft office.
- Objetivo 2: Diseñar la Gramática definida para el pseudocódigo que será utilizado por el intérprete.
- Objetivo 3: Implementar un método de conversión del formato XML de Microsoft office, que representa al diagrama de flujo, a pseudocódigo.
- Objetivo 4: Implementar un intérprete para verificar la lógica, sintaxis, semántica del pseudocódigo.

1.5 Resultados esperados

- Resultado para el objetivo 1: Entorno para dibujar diagramas de flujo adaptando las librerías de Microsoft office empleando el entorno gráfico de Windows 7/8.
- Resultado para el objetivo 2: Gramática definida para el pseudocódigo que será utilizado por el intérprete aplicando la notación Backus-Naur (BNF).
- Resultado 1 para el objetivo 3: Método de traducción para convertir el diagrama de flujo en pseudocódigo utilizando la sintaxis del mismo.
- Resultado 2 para el objetivo 3: Implementar un editor de texto que muestre al usuario el pseudocódigo generado.
- Resultado para el objetivo 4: Intérprete para verificar la lógica, sintaxis, semántica del pseudocódigo aplicando las fases del intérprete (análisis léxico, sintáctico y semántico).

1.6 Herramientas, métodos y procedimientos

En esta sección se presentará la planificación para la elaboración de la solución del proyecto. Además se muestran las metodologías aplicadas para lograr los objetivos específicos necesarios para la solución.

1.6.1 Métodos y procedimientos

Resultado Esperado	Resultado para el objetivo 1 y 3

Método, Metodología o Procedimiento	Se utilizará el método de cascada que es una secuencia de actividades donde el punto de vista principal es el progreso del producto que se basa en puntos de revisión definidos mediante entregas calendarizadas. Este método define una serie de actividades que se deben realizar en el orden señalado: especificación de requisitos, análisis, diseño, implementación, pruebas parciales, integración y mantenimiento. Para el proyecto a desarrollar no se incluirá la etapa de mantenimiento debido a que el alcance del proyecto no incluye el seguimiento y mejora de este.
Justificación	Es un modelo bien recibido debido a que las actividades son razonables y lógicas para el tiempo del proyecto (WEITZENFELD 2005: 51).

Resultado Esperado	Resultado para el objetivo 2
Método, Metodología o Procedimiento	Se utilizará la notación Backus-Naur para la construcción de la sintaxis del pseudocódigo (BNF), las cuales son las reglas para tener programas formados de manera correcta, mediante la especificación de gramáticas de contexto libre.
Justificación	Se utiliza debido a que ofrece una especificación de sintaxis de un lenguaje de programación precisa y fácil de entender (AHO 2007: 214).

Resultado Esperado	Resultado para el objetivo 4
Método, Metodología o Procedimiento	Se usará las fases del intérprete que se encargan de tomar la representación intermedia del diagrama de flujo y ejecutar el programa en el mismo lenguaje, pero internamente son una serie de pasos internos sucesivos que se deben realizar para poder interpretarlo: análisis léxico, análisis sintáctico, análisis semántico, generación de código intermedio, optimización de código, generación de código.
Justificación	Las fases del intérprete son los procesos que se deben implementar para obtener la funcionalidad deseada.

1.6.2 Herramientas

En esta sección se describirá las herramientas utilizadas durante el desarrollo de los objetivos específicos del proyecto de fin de carrera

1.6.2.1 Visual Studio

Visual Studio provee diferentes herramientas para la creación de aplicaciones que ofrecen mejorar la experiencia de usuario (VISUAL STUDIO s/f). Esta herramienta se utilizará para la creación del entorno para dibujar el diagrama de flujo del objetivo específico 1 y para visualizar el resultado de la traducción a pseudocódigo del objetivo específico 3.

1.6.2.2 Microsoft Visio

Microsoft Visio programa parte de la suite de Microsoft Office que nos facilita la creación de diversos diagramas mediante unos cuantos clics (MICROSOFT VISIO s/f). Esta herramienta será utilizada para la creación de los diagrama de flujo del objetivo específico 1.

1.6.2.3 Byacc

Un generador de parseo que se utiliza para la fase de análisis sintáctico del intérprete, esta fase es esencial dado que en esto se genera la lista de tokens a partir de la gramática definida que es utilizado en las siguientes fases del intérprete (análisis sintáctico y semántico) (BYACC s/f). Esta herramienta se utilizará para el objetivo específico 4.

1.7 Alcance

El paradigma de programación que se soportará es el imperativo, debido a que ofrece instrucciones sencillas de aprender como la asignación y secuencias en donde la idea de guardar o asignar información es muy utilizada en estos días (VUJOŠEVIĆ 2008: 69- 70).

El entorno para dibujar diagramas de flujo consta de las siguientes partes:

- Una barra de herramientas que contiene las siguientes funciones: Nuevo documento, Abrir, Guardar, Guardar Como, Salir, Ejecutar.
- Una paleta de herramientas con las siguientes estructuras: lectura y escritura de datos, operaciones aritméticas simples, estructuras selectivas y estructuras iterativas.
- Una hoja de trabajo donde se arrastren los objetos para crear el diagrama de flujo.

El código obtenido de la traducción del diagrama a pseudocódigo tendrá un editor de texto que visualice al usuario el código obtenido.

El intérprete será llamado desde una consola para verificar la lógica y sintaxis del código generado e imprimir los errores detectados en un archivo de texto.

Una explicación más a detalle de los componentes del entorno para crear diagrama de flujo se desarrolla en el capítulo 1 y para el editor de texto en el capítulo 3. Y una lista detallada de los requerimientos del producto en el Anexo B.

1.7.1 Limitaciones

No se soportará el paradigma orientado a objetos porque la herramienta a desarrollar está dirigida a estudiantes que inician su aprendizaje y no es recomendable para desarrollar las habilidades básicas y necesarias de programación. Además de la dificultad de aprender conceptos como constructores, destructores, herencia, etc (Idem 75) y debido a que el diagrama de flujo no tiene soporte para la declaración de objetos.

El intérprete se encargará de realizar el análisis léxico, sintáctico y semántico del pseudocódigo mas no se implementará la parte de ejecución de código. Para los errores léxicos y semánticos se indicará la línea y elemento detectado. En cambio para los errores sintácticos solo se mostrará un error de sintaxis, debido a que este proceso se realiza utilizando la herramienta Byaac y no existe acceso a mayor información.

No se abarcará los tipos de datos compuestos tales como estructura, arreglos, punteros, matrices y demás.

El pseudocódigo solo permitirá el uso de tipo de variable real debido a las dificultades de inferir el tipo de datos en las operaciones.

1.7.2 Riesgos

Riesgo identificado	**Impacto en el proyecto**	**Medidas correctivas para mitigar**
Viaje de asesor por motivos de beca de maestría	Medio	Buscar otros profesores de la carrera con algunos conocimientos relacionados al proyecto.
Falta de un asesor suplente disponible debido a su carga académica y cantidad de asesorados	Alto	Conversar con el profesor del curso sobre el caso para obtener una solución.
Curva de aprendizaje para desarrollar un IDE muy elevado	Alto	Utilizar IDEs existentes para poder crear un plug-in haciendo uso de las funciones que ofrecen.

Retraso en los entrega de los entregables en la fecha prevista	Medio	Dedicar más tiempo a la realización de los entregables para poder terminar en el tiempo establecido.

1.8 Justificación y viabilidad

En la siguiente sección se analiza la viabilidad del proyecto de fin de carrera así como la justificación del mismo.

1.8.1 Justificativa del proyecto de tesis

La herramienta ayudará al estudiante en la etapa de diseño para la resolución de problemas en programación mediante el uso de una interfaz gráfica en la que podrá arrastrar y soltar bloques de comando y sus conexiones respectivas para diseñar su solución. De esta manera el usuario está limitado a escoger las estructuras ofrecidas por el entorno con lo cual se reduce los errores de sintaxis así se puede concentrar más en la lógica de su solución (DRAZEN 2011: 1). Al ofrecer una visualización de la solución con una estructura ordenada descendentemente facilita la comprensión del usuario del programa diagramado.

Para reforzar la viabilidad del proyecto de fin de carrera se realizó una encuesta (Ver Anexo A) a los alumnos del curso de introducción a la computación de la Pontificia Universidad Católica del Perú, en la cual se obtuvo una aceptación del 90% por parte de los alumnos de la existencia de esta herramienta.

1.8.2 Análisis de viabilidad del proyecto de tesis

En cuanto a lo financiero para la realización del proyecto se utilizará software libre o software que cuenta con licenciamiento para estudiantes. Además en la parte de recursos humanos el proyecto propone un alcance a ser realizado por una sola persona y con el uso de una computadora personal. Por lo que no se realizará algún gasto económico para el desarrollo del proyecto.

Este proyecto tiene una duración de 4 meses en los cuales se han dividido las tareas a realizar, por lo cual el proyecto es viable.

1.9 Plan de actividades

En el siguiente diagrama se presenta las actividades programadas para los 4 meses del desarrollo del producto las cuales están programadas desde 17 de Abril de 2014 hasta el 30 de Junio del mismo año.

Entregable	Hito
Entrega de documento de Tesis 1	17/03/2014
Presentación de primer avance	31/03/2014
Corrección de observaciones	02/04/2014
Entorno para dibujar diagramas de flujo	05/04/2014
Presentación de segunda avance	07/04/2014
Analizar estructura del XML de Visio	10/04/2014
Presentación del tercer avance	14/04/2014
Método para la traducción de XML a pseudocódigo	17/04/2014
Presentación de cuarto avance	21/04/2014
Gramática del intérprete	24/04/2014
Presentación del quinto avance	28/04/2014
Desarrollo del intérprete (análisis léxico)	01/05/2014
Presentación del sexto avance	05/05/2014
Desarrollo del intérprete (análisis sintáctico)	08/05/2014
Método de conversión a pseudocódigo, editor de texto para visualizar texto traducido de diagrama de flujo	12/05/2014

Documentación de Objetivo Específico 1	13/05/2014
Entrega de documento parcial de Tesis	15/05/2014
Método de conversión a pseudocódigo, estructuras FOR y WHILE	17/05/2014
Sustentación Parcial de Tesis	19/05/2014
Correcciones de observaciones de la sustentación	22/05/2014
Documentación de Objetivo Específico 2	26/05/2014
Documentación de Objetivo Específico 3	30/05/2014
Documentación de Objetivo Específico 4	06/06/2014
Documentación de Conclusiones y Trabajos futuros	07/06/2014
Entrega de Documento Final de Tesis	09/06/2014
Método de conversión a pseudocódigo, estructura CASE	10/06/2014
Método de conversión a pseudocódigo, desarrollo de la ventana del output	12/06/2014
Plan de Pruebas	14/06/2014
Ejecución del Plan de Pruebas	17/06/2014
Correcciones finales del producto	20/06/2014
Sustentación final de Tesis	23/06/2014

CAPÍTULO 2: MARCO TEÓRICO Y ESTADO DEL ARTE

2.1 Marco teórico

En este capítulo se definirá a más detalle los conceptos relacionados al problema, también se definirá los conceptos necesarios para la solución del mismo.

2.1.1 Conceptos relacionados al problema

En esta sección se detalla los conceptos que se encuentran directamente relacionados con el problema.

Lenguajes de Programación

Un lenguaje de programación es “Conjunto de reglas, símbolos y palabras especiales usados para implementar un programa de computadora” (DALE 2007: 5). Entonces el lenguaje de programación es un lenguaje que utiliza el programador para que el ordenador ejecute las acciones que le ordene.

Durante el proceso de aprendizaje de un lenguaje de programación la persona desarrolla ciertas habilidades necesarias que conlleva la solución de problemas (Idem 6):

- Habilidad para descubrir nuevas formas de originar pensamientos sobre los objetos y procesos.
- Alto interés por la tecnología, en especial referente a las computadoras.
- Una adecuada sistematización de la producción de ideas orientadas a la búsqueda de una respuesta a un problema planteado.
- Buena capacidad de reflexión sobre las propias ideas así como los procesos que deben analizarse para resolver.
- La disposición para aprender reglas y uso de un lenguaje de programación.
- La capacidad de abstracción para poder llevar las ideas que se tiene a un lenguaje de programación.

En cuanto a la programación orientada a objetos, que utiliza los objetos para abstraer o representar un hecho o ente de la vida real con atributos que representan sus características o propiedades, proporciona herramientas para:

- Modelar la realidad desde un modo más cercano a la perspectiva del usuario.
- Interactuar fácilmente con un entorno computacional.
- Construir componentes reutilizables y que sean extensibles.
- Modificar los componentes sin que afecten a los demás.

En cuanto a la programación visual esta proporciona más facilidades a los usuarios al ofrecer un entorno gráfico en donde desarrollar sus programas. Al ofrecer un entorno gráfico se hace más atractivo e intuitivo para la persona al ser algo que se asemeje a la realidad en especial los jóvenes, sienten curiosidad por interactuar con los objetos y encontrar que hacen y cómo funcionan.

2.1.2 Conceptos relacionados a la propuesta de solución

En esta sección se detalla los conceptos que se encuentran directamente relacionados con la solución del problema.

Método para la solución de problemas

Una mejor manera para hallar la solución de un problema es mediante el uso de etapas definidas las cuales son 5 que se detallan a continuación (FLORES 2011: 25-26):

- **Descripción del problema**

  Se debe comprender el problema principal para conocer qué resolver. Por este motivo se debe tener un enunciado claro, concreto y preciso del problema a resolver.

- **Definición de la solución**

  Se debe realizar un estudio profundo sobre el problema para saber exactamente cómo resolverlo y de esta manera poder descomponerlo en partes pequeñas que sean más fáciles de resolver. Luego de esto buscar diferentes maneras de solucionarlo de las cuales se escogerá la óptima.

- **Diseño de la lógica**

  En esta fase se procede al diseño de la solución utilizando alguna herramienta como el diagrama de flujo, o pseudocódigo. A continuación se explicará a más detalle cada una de estas herramientas.

  (1) Diagrama de flujo

  Un diagrama de flujo “es una representación gráfica de los pasos que se necesitan para poder alcanzar la solución de un problema” (CAIRO 2003: 4).

  En el siguiente gráfico se detallan los símbolos que se utilizan en el diagrama de flujo los cuales siguen las recomendaciones de la “International Standard Organization” (ISO) y la “American National Standards Institute” (ANSI).

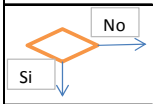
Símbolo	Explicación
	Simbolo que representa el inicio o fin de un diagrama de flujo
	Símbolo para poder leer datos de entrada.
	Símbolo que representa un proceso, en su interior se expresan asignaciones, operaciones matemáticas, etc
	Símbolo que representa una decisión. En su interior se almacena una decisión que dependiendo el resultado de la evaluación sigue por alguna de las ramas.
	Símbolo que representa una selección multiple. En su interior se almacena un selector dependiendo del valor se sigue algunos de las caminos.
	Símbolo que representa la impresión del resultado
	Símbolos que sirven para expresar la dirección del flujo del programa
	Símbolo para expresar la conexión dentro de una misma página.
	Símbolo para expresar la conexión entre diferentes páginas.
	Símbolo que representa un módulo de un problema. En realidad expresa que si quieres seguir con el flujo del diagrama primero debemos de ejecutar el subprograma que contiene en su interior.

Gráfico 1: Elementos del diagrama de flujo

A continuación mostramos las reglas que se deben seguir a la hora de la construcción de diagramas de flujo (CAIRO 2003: 5-7):

1. Todo diagrama debe tener un inicio y fin.

2. Las rectas utilizadas para indicar la dirección del diagrama deben ser rectas, verticales y horizontales mas no deben cruzarse.

3. Todas las líneas que se utilizan para indicar la dirección del diagrama de flujo deben estar conectadas a algún símbolo de los definidos en el cuadro superior.

4. El diagrama debe ser dibujado de arriba hacia abajo o de izquierda a derecha.

5. La notación que se utilice en el diagrama de flujo debe ser independiente de cualquier lenguaje de programación.

6. Cuando se realiza una tarea compleja se debe poner comentarios para que ayuden a entender lo que se realiza.

7. Si el diagrama de flujo es demasiado grande y requiere más de una hoja para su construcción se debe usar los conectores adecuados y enumerar las páginas debidamente.

8. El texto dentro de cada bloque no debe ser más de una línea.

(2) Pseudocódigo

"Es un lenguaje de especificación (descripción) de algoritmos" (JOYANES 2008: 70). Podemos decir que el pseudocódigo es como un primer borrador antes de hacer su traducción a un lenguaje de programación el cual tiene las siguientes características:

- Es una forma sencilla de representar, utilizar y manipular.
- Facilita el proceso de traducción a un lenguaje de programación.
- Es independiente del lenguaje de programación a codificar.
- Permite concentrarse en la lógica y estructuras de la solución en vez de las reglas del propio lenguaje de programación.

En el siguiente gráfico se muestra las estructuras que soporta el pseudocódigo así como la declaración de cada uno:

Estructura	**Declaración**
Selectiva Simple	SI ... ENTONCES; FIN SI
Selectiva doble	SI ... ENTONCES; SINO ENTONCES; FIN SI
Selectiva múltiple	SEGÚN … HACER; FIN SEGÚN
Bucle mientras	MIENTRAS … HACER; FIN MIENTRAS
Bucle para	PARA … HASTA … HACER; FIN PARA
Asignación	:=
Procedimiento	PROCEDIMIENTO …; FIN PROCEDIMIENTO
Función	FUNCIÓN …; FIN FUNCIÓN
Declaración de programa	PROGRAMA ….; INICIO ... FIN

Tabla 1: Elementos del pseudocódigo

- **Desarrollo de la codificación**

  Una vez obtenido el diseño de la lógica de la solución escogida se procede al desarrollo de la codificación del problema en algún lenguaje de programación.

  El proceso de codificación consiste en traducir las variables y los pasos de algoritmo definidos en cada método en sentencias del lenguaje escogido. Estas se almacenan en un archivo y son las instrucciones que la computadora podrá ejecutar.

- **Depuración y pruebas**

  Al obtener el código de la solución escogida se debe ejecutar varias veces para encontrar errores de diferentes tipos (generalmente lógicos y de sintaxis). Para la corrección se debe modificar, anular y crear nuevas instrucciones hasta obtener el resultado deseado.

Entorno de Desarrollo Integrado (IDE)

La IDE es un entorno de programación que consiste de un editor de texto, un compilador, un depurador y un constructor de interfaz gráfica (GUI), los cuales han sido empaquetadas en un programa de aplicación (ESLAVA 2012: 10). Las IDEs pueden ser un programa independiente o ser una extensión de otro. Además estas deben facilitar el aprendizaje de programar efectivamente y eficientemente y ayudar a los estudiantes a entender las estrategias de resolución de problemas visualmente (ZHIXIONG 2005: 104).

Generalmente las IDE's ofrecen un marco de trabajo amigable para los diferentes lenguaje de programación como: Java, C++, C#, etc. Así tenemos el caso de Netbeans que es una IDE para los lenguajes de Java, C++, PHP, el cual posee extensiones para tener soporte de otros lenguajes.

Compilador

El compilador se encarga de leer código en un lenguaje (lenguaje fuente) y lo traduce en un lenguaje equivalente (lenguaje objetivo), durante el proceso se encarga de reportar los errores que detecte durante el proceso de traducción (AHO 2007: 1-2).

Una característica que diferencia al intérprete del compilador es, que ejecuta el programa en base a los datos de entrada que da el usuario, en lugar de devolver un programa traducido.

El resultado del compilador es, por lo general, mucho más rápido que del interprete, pero la ventaja del intérprete es que puede detectar muchos más errores ya que ejecuta el programa sentencia por sentencia.

Si se desea obtener como resultado un programa ejecutable se deberá de recurrir a otros procesos además del compilador. El preprocesador une el programa en una fuente de sentencias de lenguaje, ya que puede estar separado en archivos diferentes. El ensamblador traduce el código en lenguaje ensamblador, generado del compilador, a lenguaje máquina. El linkeador junta los diferentes archivos compilados del programa y las librerías utilizadas para ejecutar el programa. Y el cargador pone todos los archivos ejecutables en memoria para ejecutarse.

La estructura del compilador se divide en 2 procesos generales: análisis y síntesis (AHO 2007: 4).

La parte de análisis se encarga de dividir el programa en pequeñas partes consistentes e impone una estructura gramatical que se utiliza para generar una representación intermedia del programa. También se encarga de detectar errores sintácticos y semánticos en el programa y de reportárselos al usuario. Y durante este proceso recolecta información del programa y lo guarda en una estructura llamada tabla de símbolos, el resultado que arroja es la tabla de símbolos y la representación intermedia se envía a la parte de síntesis (Idem 4).

La parte de síntesis es la que se encarga de construir el programa destino a partir de la representación intermedia y la tabla de símbolos. Podemos decir que el análisis es el front-end del compilador y la síntesis el back-end (Idem 4).

2.2 Estado del arte

El propósito del estado del arte es la búsqueda de productos similares que se encuentren disponibles para el público y que cubran la solución del problema de manera parcial o total.

Para ello se buscó productos o trabajos de investigación similares al producto que se desea desarrollar. En el caso de los productos se buscó las características que ofrecían a través de su portal web. Para los trabajos de investigación se indagó en la biblioteca en la universidad sobre proyectos pasados en ciencias de la computación y teoría de compiladores.

2.2.1 Formas similares de resolver el problema

Se utiliza el método de verificación por seguimiento con tablas variables, en el cual se tiene una tabla de seguimiento básica donde se aprecia las variables y los cambios que se produzcan durante la ejecución del programa (COOPER 2011: 475-476). Pero en muchas ocasiones este método es obviado y se ejecute la solución planteada sin una verificación, la cual provoca que la detección de errores se realice en la etapa de codificación y pruebas.

2.2.2 Productos comerciales para resolver el problema

- **Microsoft Visio**

Un programa de la línea de Microsoft Office, con una interfaz sencilla para el usuario, el cual ofrece la posibilidad de crear una gran variedad de diagramas, los cuales se construyen arrastrando los objetos que se desea de la barra de formas, entre ellos tenemos al diagrama de flujo, organigrama, base de datos, etc. Ofrece la posibilidad de poder exportar los diagrama a XML, PDF, JPEG, PNG.

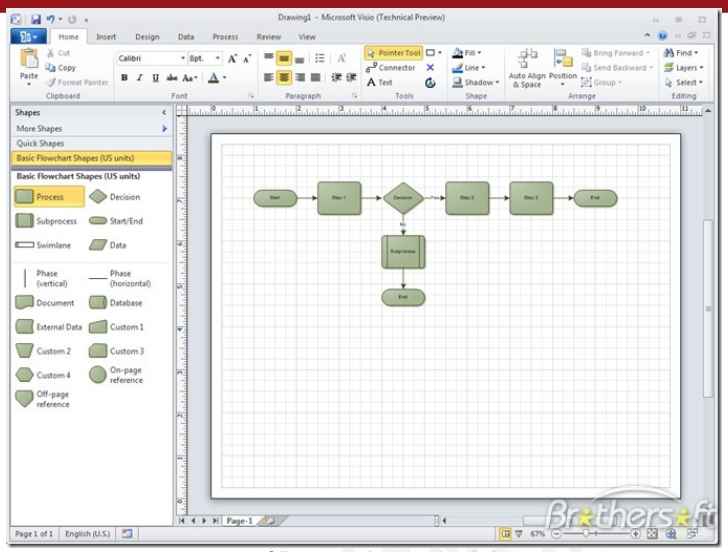


Gráfico 2: Microsoft Visio 2010. Fuente: http://img.brothersoft.com/screenshots/softimage/m/microsoft_visio_viewer_2010-305787-1258685733.jpeg

- **AthTek Code to Flowchart Converter**

Producto que posee una interfaz parecida al Netbeans, que ofrece convertir de C, C++, VC++. Pascal y Delphi a diagrama de flujo y además de exportarlo a MS Visio, Ms Word, XML, SVG. La licencia de este producto es de 119 dólares por usuario, también ofrece una versión de prueba.

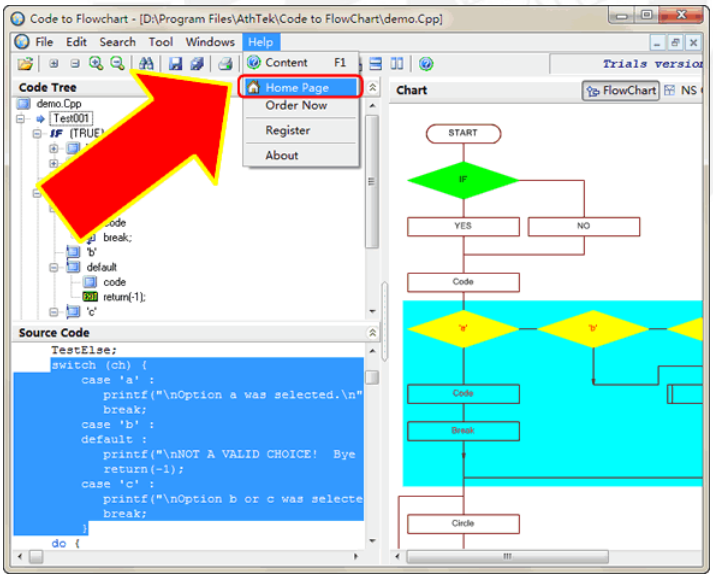


Gráfico 3: Athtek code to flowchart converter. Fuente: http://www.athtek.com/image/acf/version.gif

- **Visustin V6 Flow Chart Generator**

Este producto ofrece un interfaz donde en la mitad del programa se muestra la parte del código de programa y en la otra mitad se observa el diagrama generado a partir del código ingresado. Ofrece la capacidad de convertir de C,

C++, Cobol, Fortran, Java, Javascript, PHP, VBA a diagrama de flujo y poder exportarlo a MS Visio, Excel, Word, PDF, JPEG, PNG, y otras extensiones de imagen. Su costo 249 dólares para la versión estándar.

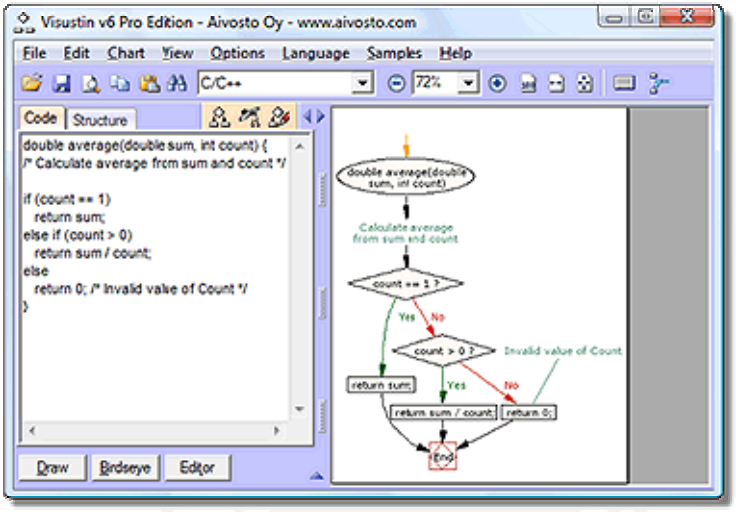


Gráfico 4: Visustin V6. Fuente: http://www.aivosto.com/visustin-fullshot.gif

- **DFD**

Programa que ofrece la posibilidad de crear diagramas de flujo mediante el arrastre de objetos. Ofrece una interfaz gráfica con una barra de herramientas donde se encuentran los elementos del diagrama de flujo. Además permite ejecutar el programa el cual se realiza de manera gráfica.

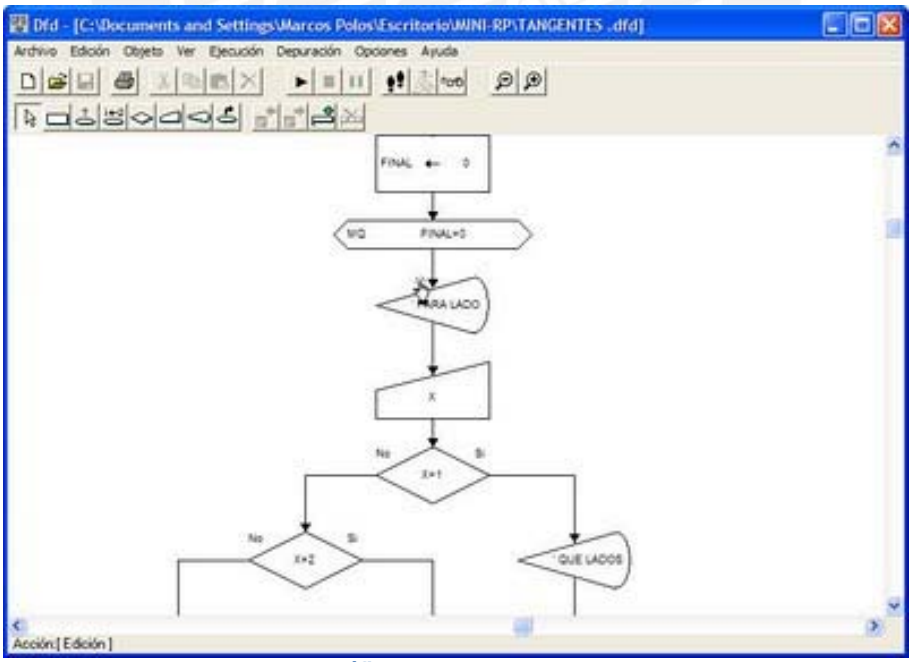


Gráfico 5: DFD. Fuente: http://marcopolojacometoss.files.wordpress.com/2011/01/dfd2.jpg?w=400&h=288

2.2.3 Productos no comerciales (de investigación) para resolver el problema

- **Visual Block: Lenguaje de Programación Visual**

Proyecto de fin de carrera realizado por Diaz en la que se desarrolla un lenguaje de programación visual en el cual se desarrolla programas arrastrando los objetos y se muestran gráficamente, tanto la depuración y ejecución se realiza en el mismo entorno de manera gráfica. Pero esta herramienta es muy básica y no posee todos los elementos de un diagrama de flujo tales como la llamada a procedimientos y funciones.

- **Intérprete y entorno de desarrollo para el aprendizaje de lenguajes de programación estructurada**

Proyecto de fin de carrera realizado por Hirsh (HIRSH 2007) donde se desarrolla un entorno de desarrollo integrado en el cual se crean los programas en un lenguaje de pseudocódigo en español, creado por el autor.

Gráfico 6: IC Helper. Fuente : HIRSH 2007

- **Intérprete para un lenguaje de programación orientado a objetos con mecanismos de optimización y modificación dinámica de código**

En este proyecto de fin de carrera implementado por Renzo Gomez se desarrolla un lenguaje con su entorno de desarrollo integrado (IDE) donde se puede crear programas en el lenguaje pas++ (creado por el autor) posee una tabla de símbolos donde se puede modificar el valor de las variables definidas en el programa durante la ejecución del programa.

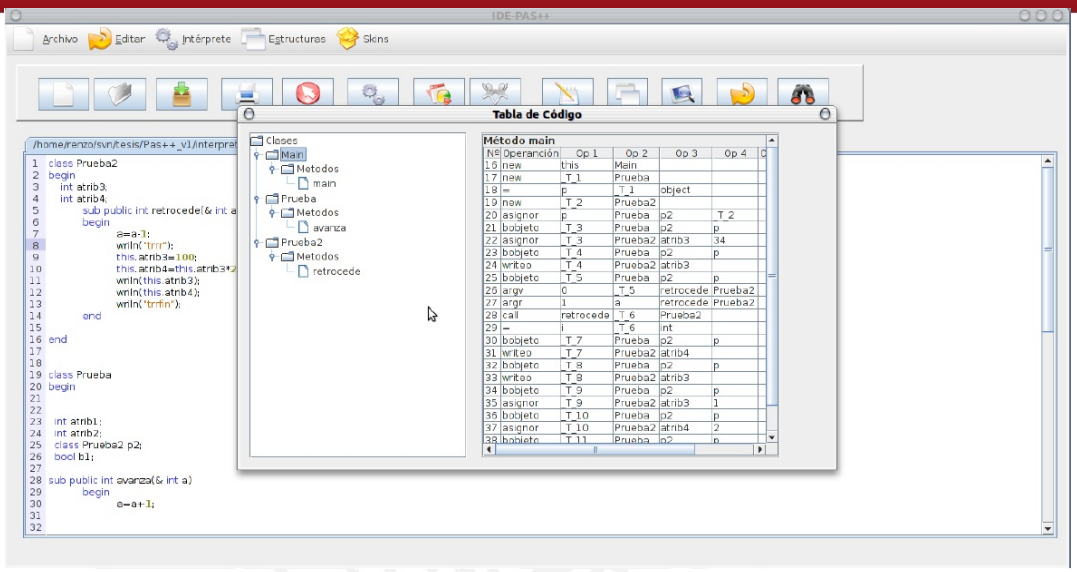


Gráfico 7: Entorno de Pas++. Fuente: GOMEZ 2012

2.2.4 Problemas relacionados

Característica/Producto	Libre	Dibujar diagrama	Diagrama a código	Ejecución de código	Uso de IDE	Idioma
Microsoft Visio	Trial de 30 días	Si	Si(XML)	No	Si	Español y otros
AthTek	Trial de 30 días	Si	Si	No	Si	Ingles
Visustin V6	Trial de 30 días	Si	Si(Java)	No	Si	Ingles
DFD	Si	Si	No	Si	Si	Español
Visual Block	Si	Si	No	Si	Si	Español
Tesis (Hirsh)	Si	No	No	Si	Si	Español
Tesis (Diaz)	Si	Si	No	Si	Si	Español

Tabla 2: Cuadro de comparación de productos.

2.2.5 Conclusiones sobre el estado del arte

Como resultado de la búsqueda de productos, si bien existen programas que permiten la creación de diagrama de flujo y ejecución de manera visual como el caso de Visual Block pero no permite exportar a un lenguaje de programación. Por otro lado tenemos Visustin V6 o Microsoft Visio que permiten exportar el diagrama a un lenguaje de programación pero al no poseer un compilador no se puede validar el código generado. Como en el caso de los proyectos de Hirsh o Díaz que poseen un compilador, y en el caso de Díaz de optimizar el código, pero no ofrecen una creación visual de su programa. Entonces se necesita de algún programa que ofrezca una interfaz sencilla para el usuario donde él pueda crear fácilmente un diagrama de flujo. Este seguirá los estándares de los cursos de programación introductorios, que es el enfoque de

esta tesis, y que será de ayuda para la etapa de diseño de la resolución de problemas.

Capítulo 3: Entorno para dibujar diagramas de flujo adaptando las librerías de Microsoft office.

En este capítulo se presenta las herramientas y las funcionalidades que conforman el entorno para dibujar diagramas de flujo.

3.1 Entorno para dibujar diagramas de flujo

Para el desarrollo del entorno se llevó a cabo una investigación de las herramientas que se ofrecen para la creación de componentes visuales con el soporte de drag and drop. Entre las herramientas encontradas se tuvo a java con su librería Java.Swing, c# con su librería Windows Forms, o utilizar las librerías de Microsoft visio o crear un plug-in. Se optó por utilizar c# debido al mayor conocimiento de los componentes gráficos y el manejo de sus eventos.

Luego se decidió utilizar Microsoft Visio 2010 y no crear todo el apartado gráfico de los elementos del diagrama en c# debido a que Visio ya posee los elementos definidos y la estructura definida y ordenada necesarios. Pero se optó por utilizar las librerías de Visio y no realizar un plug-in debido a que Visio ofrece otras herramientas adicionales a la creación de diagramas como opciones para insertar gráficos, imágenes. Si el usuario tiene acceso a todas estas herramientas puede generar errores en el proceso de traducción del diagrama a pseudocódigo al insertar componentes no reconocibles. Por lo tanto al utilizar las librerías se le limita el acceso a las herramientas ofreciéndole lo necesario para la creación de diagramas.

A partir de ello se desarrolló el entorno conformado de los siguientes componentes, resultado obtenido de este capítulo, y se observan en la gráfico 8: la paleta de dibujo que contiene los componentes del diagrama de flujo (1), la barra de herramientas que contiene las siguientes opciones: nuevo documento, abrir documento, guardar documento, guardar como y generar el cual genera el pseudocódigo del diagrama de flujo (2), hoja de dibujo donde se arrastran los elementos para la creación del diagrama de flujo (3). Estos requerimientos son de elaboración propia y están especificados en el anexo B.

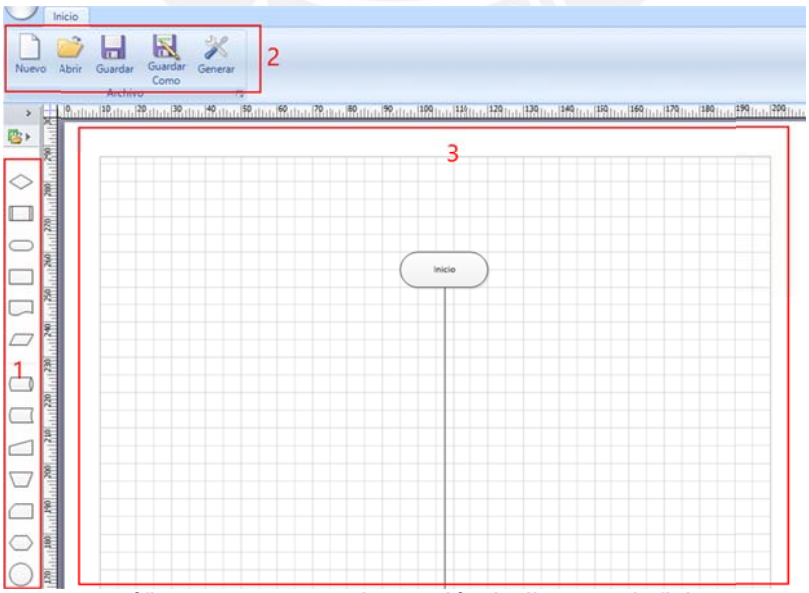


Gráfico 8: Entorno para la creación de diagrama de flujo

A continuación se describe los componentes de la paleta de dibujo:

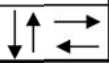
Símbolo	Bloque	Explicación
	Inicio o fin	Representa el inicio o fin del diagrama de flujo
	Datos	Para la lectura de entrada de datos
	Condicional	Utilzado para las estructuras IF/ELSE y WHILE
	Proceso	Dentro de ella se realizan operaciones aritméticas(suma,resta,multiplicación,división) y llamadas a funciones
	Documento	Utilizado para la escritura de datos
	Preparación	Utilizado para la estructura FOR
	Subproceso	Utilizado para los procedimientos o funciones
	Dirección	Para mostrar la dirección del diagrama de flujo

Gráfico 9: Cuadro descriptivo de los componentes que utiliza el diagrama de flujo.

También se detalla las operaciones y datos que soporta o no el diagrama de flujo:

Operaciones	Símbolo	Soporta
Asignación	:=	Si
Funciones	LLAMAR()	Si
Procedimientos	PROCEDIMIENTO	Si
Suma	+	Si
Resta	-	Si
Multiplicación	*	Si
División	/	Si
Datos primitivos	entero, booleano, carácter, reales	No
Datos Compuestos	objetos, punteros, dato abstracto	No
Datos algebraicos	sen, cos, tan, log	Si

Tabla 3: Cuadro descriptivo del soporte de operaciones del diagrama de flujo.

Capítulo 4: Gramática del Intérprete

En el siguiente capítulo se presentan los procesos necesarios para obtener el resultado esperado 2: Gramática definida para el pseudocódigo que será utilizado por el intérprete utilizando la notación Backus-Naur (BNF).

4.1 Desarrollo de la gramática

Para la creación de la gramática del pseudocódigo se utilizó la notación BNF (Backus-Naur Form) la cual permite representar gramáticas de contexto libre que se componen de reglas (COOPER 2004: 87). Una manera de representar estas reglas se ve en el siguiente ejemplo: “V -> w”. Donde "V" es un símbolo no terminal, es decir, permite más derivaciones, y w es una cadena de valores terminales, no permite más derivaciones. La notación BNF nos presenta una forma de representar estas gramáticas de libre contexto de una forma muy similar a la antes descrita, pero con algunos cambios como el uso de “< >” en símbolos no terminales, “” para símbolos terminales y “ ::= ” para la asignación: <V> ::= “w”.

Existen otros símbolos como “|”, el cual representa los caminos alternativos en la regla, y valores que están incluidos en las comillas simples que representan valores no terminales.

```
<term>  ::= <term> '*' <factor>
          | <term> '/' <factor>
          | <term> <mod> <factor>
```

En el ejemplo de arriba se puede observar que <term> se puede convertir en 3 conjuntos. El primer conjunto que representa es una multiplicación (<term> “*” <factor>), el segundo conjunto que representa una división (<term> ‘/’ <factor>) y el último conjunto que representa la operación de residuo (<term> <mod> <factor>).

Para el pseudocódigo se utiliza una gramática tomando como base la estructura de un lenguaje del paradigma imperativo que posee una estructura sencilla de comprender. A continuación se explica cómo representar las reglas de la gramática utilizando el siguiente ejemplo:

```
<funcion> ::= <funcion> <var> '(' <listaparam> ')' <enter> <listainstr> <finfuncion>
```

Como se puede apreciar en el ejemplo anterior, la notación BNF nos ayuda a representar, en este caso, la estructura del pseudocódigo. Usando esta regla se puede representar el siguiente programa bajo la estructura y sintaxis del pseudocódigo:

```
FUNCION HallarSuma(a,b)

FINFUNCION
```

A partir de este ejemplo tenemos que los símbolos <funcion> <var> '(' <listaparam> ')' <enter> <finfuncion>, generan como única regla los símbolos terminales “FUNCION”, “HallarSuma” ,”(”, ”a”, ”,” ,”b” ,”)” ,”\n” y “FINFUNCION”, donde “a,b” forma parte del símbolo no terminal <listaparam>. No olvidar que existe el símbolo terminal <listaInstr> que en este caso toma el valor de su segunda opción el cual es el valor de cambio de línea “\n” o <enter>. Si este proceso de recorrer la gramática se realiza de manera

recursiva se pueden generar sentencias más elaboradas, como las estructuras iterativas, condicionales, selectivas, etc.

Las reglas que se generen para la gramática del pseudocódigo serán posteriormente utilizados en el proceso de análisis sintáctico del intérprete para la definición de la estructura y sintaxis del mismo, esto se explicará a detalle en un capítulo posterior. La gramática en algunos casos puede presentar errores de ambigüedad, esto sucede cuando existen diferentes caminos que parten de un símbolo no terminal, o símbolos inaccesibles (símbolos terminales o no terminales que no son alcanzables por ningún símbolo). Con el correcto desempeño del analizador sintáctico se puede verificar la gramática del pseudocódigo.

A continuación se presenta la gramática definida para el pseudocódigo aplicando el método definido en la parte superior:

<program> ::= <programa> <enter> <inicio> <enter> <listainstr> <fin> <enter>;

<listainstr> ::= <listainstr> <instr>
|
;

<instr> ::= <imprimir>
| <instrif>
| <asignacion>
| <enter>
| <cond>
| <procedimiento>
| <funcion>
| <retornarvalor>
| <instrwhile>
| <instrfor>
| <instrswitch>
;

<asignacion> ::= <var> <asign> <cond> <enter> ;

<imprimir> ::= <outln> <listavars> <enter> ;

<listavars> ::= <listavars> ',' <cond>
| <cond>
;

<instrif> ::= <if> <cond> <then> <enter> <listainstr> <listaelse> <fin> <if> <enter>
;

<listaelse> ::= <else> <enter> <listainstr>
|
;

<instrwhile> ::= <mientras> <cond> <hacer> <enter> <listainstr> <fin> <mientras> <enter>
;

<instrfor> ::= <para> <asignacion> <hasta> <tokenfinales> <hacer> <enter> <listainstr> <fin> <para> <enter>

```
            ;

<instrswitch> ::= <segun> <var> <hacer> <enter> <listacasos> <fin> <segun> <enter>
                ;

<listacasos> ::= <caso> <num> <enter> <listainstr> <break> <enter> <listacasos>
               |
               ;

<funcion> ::= <funcion> <var> '(' <listaparam> ')' <enter> <listainstr> <finfuncion>
<enter>
            ;

<retornarvalor> ::= <retornar> <cond> <enter>
                  ;

<procedimiento> ::= <procedimiento> <var> '(' <listaparam> ')' <enter> <listainstr>
<finprocedimiento> <enter>
                  ;

<listaparam> ::= <listaparam> ',' <var>
               | <var>
               |
               ;


<cond>  ::= <cond> '<' <expr>
          | <cond> '>'<expr>
          | <cond> '<' '=' <expr>
          | <cond> '>' '=' <expr>
          | <cond> '=' '=' <expr>
          | <cond> '<' '>' <expr>
          | <expr>
          | <cond> <and> <expr>
          | <cond> <or> <expr>
          | <coseno> <param>
          | <seno> <param>
          | <tangente> <param>
          | <arcotangente> <param>
          | <absoluto> <param>
          | <raíz> <param>
          | <exponencial> <param>
          | <ln> <param>
          ;


<param> ::= '(' <expr> ')'
          ;

<expr>  ::= <expr> '+' <term>
          | <expr> '-' <term>
          | <term>

<term>  ::= <term> '*' <factor>
          | <term> '/' <factor>
```

```
                | <term> <mod> <factor>
                | <factor>
                ;


<factor> ::= '(' <expr> ')'
                | '-' <factor>
                | <factor> '^' <factor>
                | <tokenfinales>
                ;

<tokenfinales> ::= <num>
                        | <var>
                        | <cad>
                        ;

<programa>          ::= "PROGRAMA" ;
<enter>   ::= '\n' ;
<inicio>   ::= "INICIO" ;
<fin>       ::= "FIN" ;
<asign> ::= ":=" ;
<outln> ::= "ESCRIBIR" ;
<if>        ::= "SI" ;
<then>    ::= "ENTONCES" ;
<else>    ::= "SINO" ;
<mientras>          ::= "MIENTRAS" ;
<hacer>  ::= "HACER" ;
<para>    ::= "PARA" ;
<hasta>  ::= "HASTA" ;
<segun> ::= "SEGUN";
<caso>    ::= "CASO";
<break> ::= "BREAK";
<funcion>             ::= "FUNCION" ;
<finfuncion>          ::= "FINFUNCION" ;
<retornar>            ::= "RETORNAR" ;
<procedimiento>   ::= "PROCEDIMIENTO" ;
<finprocedimiento>              ::= "FINPROCEDIMIENTO" ;
<and>     ::= "Y" ;
<or>       ::= "O" ;
<coseno>              ::= "COSENO" ;
<seno>    ::= "SENO" ;
<tangente>            ::= "TANGENTE" ;
<arcotangente>     ::= "ARCOTANGENTE" ;
<absoluto>            ::= "ABSOLUTO" ;
<raiz>      ::= "RAIZ" ;
<exponencial>       ::= "EXPONENCIAL"
<ln>        ::= "LN" ;
<mod>     ::= "MOD" ;
<num>     ::= ( [0-9]+ )  |  ( [0-9]+ [.] [0-9]+ );
<var>      ::= [a-zA-Z]+ ;
<cad>      ::=  ["][a-zA-Z]*["] ;
```

CAPÍTULO 5: Método de traducción para convertir el diagrama de flujo en pseudocódigo

En este capítulo se explicará el método empleado para convertir el diagrama de flujo en pseudocódigo.

5.1 Método de traducción

El propósito de esta conversión es ayudar al estudiante a interpretar un diagrama de flujo. Por otro lado el computador solo interpreta sentencias estructuradas, por lo que se requiere de un método que convierta diagramas de flujo a pseudocódigo.

Como se mencionó en el capítulo 3, se utiliza las librerías de Visio ya que permite exportar a XML el diagrama de flujo. Este formato esta indentificado con la extensión .vdx que posee la estructura de un archivo XML pero con las notaciones propias de Microsoft, esta estructura se explicará posteriormente en este capítulo. Al obtener el diagrama de flujo guardado en una estructura definida por el lenguaje XML podemos leer su contenido mediante un editor de texto.

Tomemos como ejemplo el siguiente diagrama de flujo

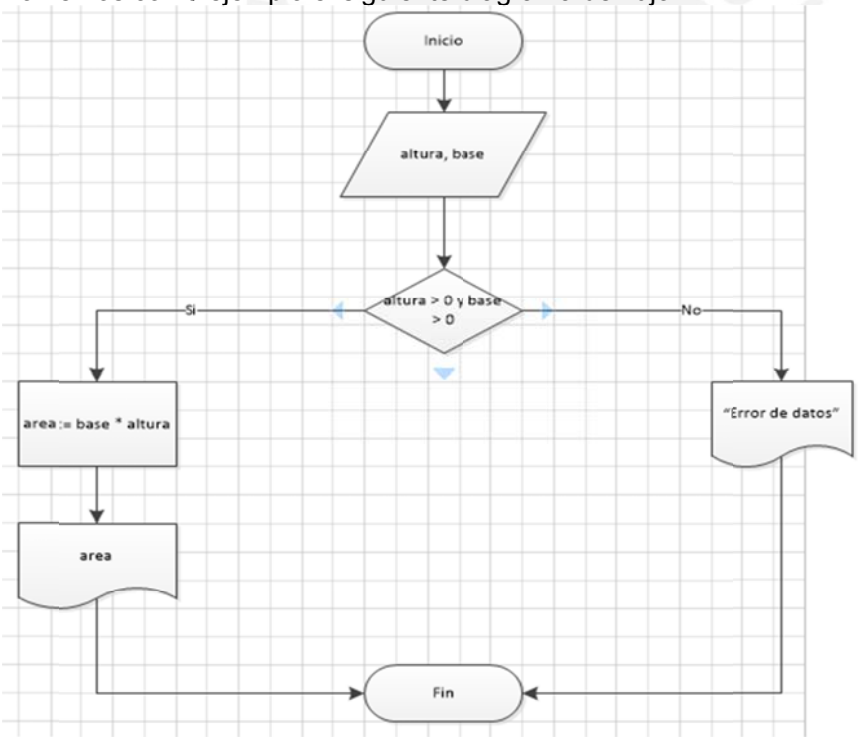


Gráfico 10: Ejemplo de un diagrama de flujo

Para el diagrama de flujo mostrado el archivo generado en XML consta de la siguiente estructura general:

```
<?xml version='1.0' encoding='utf-8' ?>
<VisioDocument key='6C60C88CC4635136B320A2915D7E32AA73F7C1758381C250B6CDB68AC69FD09
start='190' metric='1' DocLangID='3082' buildnum='7011' version='14.0' xml:space='p
xmlns:v14='http://schemas.microsoft.com/office/visio/2010/extension' xmlns:vx='http
<DocumentProperties>
<Creator>Jairo Palomino Masco</Creator><Template>C:\Program Files (x86)\Microsoft O
<BuildNumberCreated>939531107</BuildNumberCreated><BuildNumberEdited>939531107</Bui
.....
</DocumentProperties>
<Masters>
<Master ID='6' NameU='Document' Name='Documento' Prompt='Arrastre esta forma hasta
.....
</Master></Masters>
.....
.....
<Shapes>
.....
<Shape ID='4' NameU='Document' Name='Documento' Type='Shape' Master='6'>
<XForm><PinX>7.726377952755906</PinX><PinY>4.035433070866141</PinY><Width>0.8858267
.....
</Shape><Shape ID='11' Type='Shape' Master='6'>
```

Gráfico 11: Archivo XML generado del ejemplo del gráfico 10

La primera línea identifica al archivo como un archivo XML. La segunda línea es un sello con el cual Visio identifica que es un archivo XML con formato de Visio. Luego sigue el elemento “DocumentProperties” que contiene información como: creador, plantilla que se utilizó, etc.

Posteriormente está el elemento “Masters” que contiene la información de los tipos de bloque usados en el diagrama de flujo. Los tipos de bloque son: Conector, Decisión, Documento, Inicio o Finalización, Datos, Proceso. Estos tipos de bloque están contenidos en elementos hijos llamados “Master” que posee los atributos:

- ID el identificador único.
- nameU el nombre universal del elemento.
- name el nombre del elemento.

Para nuestro ejemplo el ID 6 corresponde a un bloque del tipo Documento que es usado para la impresión de datos.

Luego tenemos el elemento “Shapes” que contiene los bloques utilizados en el diagrama de flujo. Estos bloques están contenidos en elementos hijo llamados “Shape” el cual posee los siguientes atributos:

- ID identificador único.
- nameU nombre universal del elemento.
- name nombre del elemento.
- master ID del elemento “master” de donde hereda sus datos.

Los datos nameU y name son opcionales por lo que no se muestran cuando un bloque es utilizado varias veces. A continuación se muestran dos casos de estos subelementos. En el primer ejemplo no es necesario el ID del master puesto que ya se conoce el tipo de bloque, pero el segundo ejemplo no posee los atributos nameU y name, sólo el ID del maestro con el cual se obtiene el tipo de bloque que es.

```
<Shape ID='4' NameU='Document' Name='Documento' Type='Shape' Master='6'>
<XForm><PinX>7.726377952755906</PinX><PinY>4.035433070866141</PinY><Width>0.8858267
```

Gráfico 12: Primer ejemplo del subelemento “Shape”

```
<Shape ID='11' Type='Shape' Master='6'>
```

Gráfico 13: Segundo ejemplo del subelemento "Shape"

El elemento "Shape" también posee un elemento hijo llamado "Text" en el cual se guarda el texto interno del bloque el cual ha sido ingresado por el usuario.

Los elementos mencionados anteriormente son almacenados en clases que se han definido como Maestro y Nodo. La clase Maestro posee los atributos: id y nombre, mientras que Nodo los atributos: id, nombre, tipo, hijosIzquierda, hijosDerecha. En el caso que el bloque sea del tipo "Condicion" en el atributo hijosIzquierda se agruparan los nodos del lado que cumplan la condición mientras que en hijosDerecha los nodos que no cumplan la condición. En el caso de los bloque del tipo FOR y WHILE solo usan el atributo hijoDerecha.

Luego se tiene al elemento "Connect" en el cual se guardan los conectores utilizados para unir los bloques. De ellos se obtiene el ID del bloque donde inicia y del bloque al que conecta. A partir de ello podemos obtener el orden en que los bloques han sido creados por lo que no se requiere que el usuario cree el diagrama en un orden establecido siempre y cuando los bloques estén debidamente conectados.

Para almacenar la estructura completa del diagrama de flujo se utilizan las siguientes estructuras: listaMaestro y listaNodos y conectores. Donde listaMaestro almacena todos los elementos de clase Maestro, listaNodos los elementos de clase Nodo y conectores donde se guarda el id del bloque inicio y fin.

En el ejemplo mostrado, se procede a recorrer el archivo XML empezando primero a extraer los tipos de bloque recorriendo los "Master" de la sección "Masters" y almacenándolos en listaMaestro. Luego se sigue con la sección "Shapes" para obtener los bloques y sus datos necesarios y almacenarlo en la listaNodos. Y se continúa con la sección "Connects" para obtener el orden de los bloques.

Despues se procede a verificar si se encuentran los tipos de bloque de las estructuras IF, FOR y WHILE para proceder a llenar los nodos hijo que posean. Para poder diferenciar donde empieza y termina una de estas estructuras los nodos pertenecientes a su interior deben terminar en un bloque común. De esta manera se van agregando los hijos de cada estructura.

A continuación se muestra la estructura final del ejemplo siguiendo el método explicado anteriormente

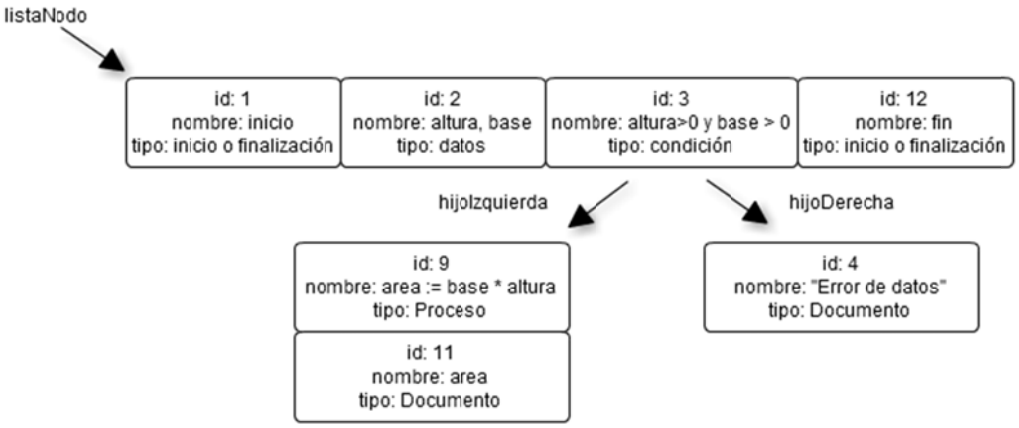


Gráfico 14: Estructura de listaNodos resultante del ejemplo del gráfico 10

Hay que tomar en cuenta que los elementos “Shape” también incluyen a los conectores que unen los bloques y estos poseen su propio ID. Por esta razón se observa que los ID de los Nodos no son consecutivos.

Durante el proceso de formación de la estructuras se debe tener algunas consideraciones de posibles errores gráficos que se pueden obtener entre ellos que algunos bloques no estén unidos mediante un conector. También que los conectores que unen a los bloques deben tener una dirección si no es posible conocer la dirección del diagrama. Estos errores son notificados mediante un cuadro de diálogo.

A partir de esta estructura generamos el archivo en pseudocódigo agregándole las sentencias propias de este lenguaje basándonos en la gramática definida en el capítulo anterior. En el caso del nodo 3 al ser un tipo de bloque “Condición” se debe agregar al principio de la cadena “SI” y al final “ENTONCES”, para el caso de la indentaciones si se encuentran antes o al final de la cadena se eliminan si se encuentra dentro son ignorados por el compilador durante su verificación. Para el nodo 4 se tiene que agregar al principio de la cadena “IMPRIMIR” y así respectivamente con los demás nodos. Obteniendo de esta manera el siguiente código:

```
PROGRAMA
INICIO
    LEER altura,base
    SI altura > 0 y base > 0 ENTONCES
        area := base * altura
        ESCRIBIR area
    SINO
        ESCRIBIR "Error de datos"
    FIN SI
FIN
```

Gráfico 15: Archivo en pseudocódigo generado a partir del ejemplo del gráfico 10

Finalmente para que el usuario pueda visualizar el código obtenido se ha implementado un editor de texto el cual posee las herramientas básicas como: tamaño de letras, tipo de fuente, copiar, pegar, cortar, guardar, buscar, reemplazar, etc. Este será invocado mediante la opción generar del entorno para crear diagrama de flujo del capítulo 1, el cual tiene la apariencia como se muestra en la Gráfico 15.

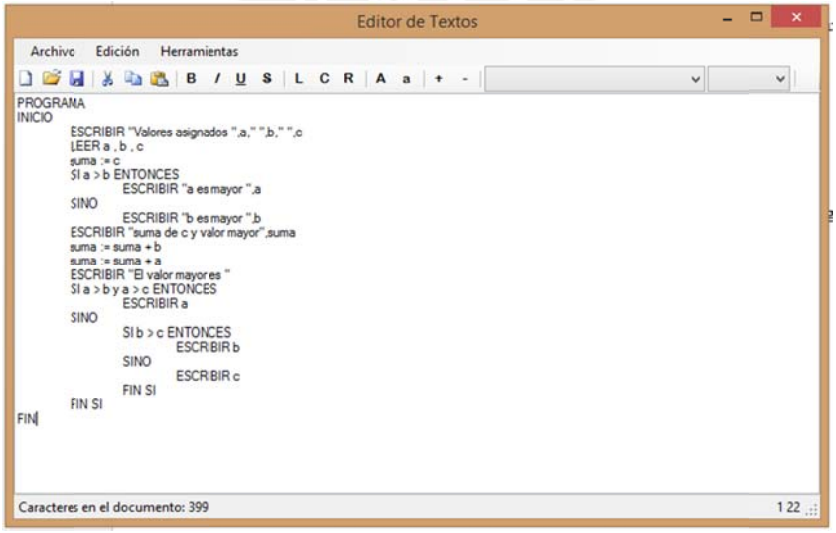


Gráfico 16: Editor de texto donde se visualiza el código generado del diagrama de flujo.

Capítulo 6: Intérprete para verificar la lógica, sintaxis, semántica del pseudocódigo aplicando las fases del intérprete (análisis léxico, sintáctico y semántico)

En este capítulo se describen los pasos necesarios para el desarrollo del Objetivo Específico 4: Intérprete para realizar la validación léxica, sintáctico y semántica del pseudocódigo.

6.1 Desarrollo del intérprete

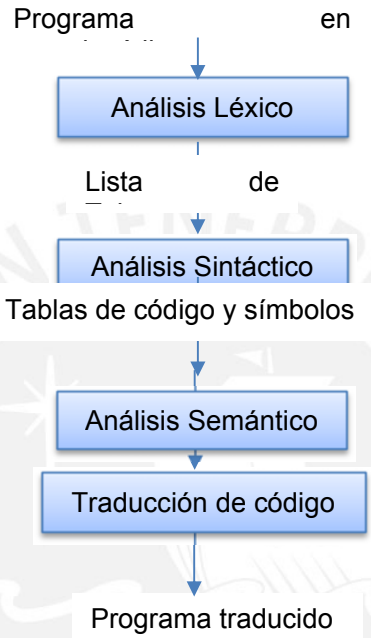


Gráfico 17: Cuadro resumen de las fases del intérprete

Como se puede observar en el gráfico para poder verificar la correctitud de un programa generado en pseudocódigo se realizan una serie de análisis del código estas son: análisis léxico, sintáctico y semántico. Estos análisis son realizados por el intérprete siguiendo el orden mencionado.

En el análisis léxico tenemos como datos de entrada al programa escrito en pseudocódigo a partir del cual se genera un arreglo de Tokens. Para la creación de este arreglo se desarrolló una función en JAVA que toma como dato de entrada el programa en pseudocódigo y comienza a recorrer el programa caracter por caracter. Durante el recorrido se analiza cada conjunto leído para identificar el tipo de token que es y si es un grupo con significado, conocido como lexema.

Estos tokens son palabras reservadas para nuestro lenguaje utilizado, pero además existen otros símbolos reservados que simbolizan operaciones aritméticas (+, *, -, /), asignaciones (:=), comparaciones (>, <, ==, <>, >=, <=), etc. Estos símbolos son fijos y no se pueden utilizar para implementar otro tipo de operación.

El arreglo de tokens se almacena en un ArrayList el cual contiene una lista de objetos de la clase Token. Esta clase contiene los siguientes atributos: el identificador del token, información del lexema, línea en el que fue encontrado, el tipo de dato solo en caso de una variable (número o cadena).

En el análisis sintáctico se toma como datos de entrada el arreglo de tokens resultado del análisis léxico con lo cual se genera una tabla de símbolos y códigos. Los símbolos son la representación de las variables, números constantes, nombre de funciones, procedimientos y cadenas. Y los códigos son las acciones u operaciones que se realizan sobre los símbolos tales como suma, resta, multiplicación, división, etc.

Para la obtención de la tabla de símbolos y códigos se hace uso de la gramática definida para el pseudocódigo (ver Capítulo 5) que contiene las reglas de sintaxis que posee. Estas reglas pueden tener llamadas a otras reglas y tokens. Para identificar qué regla se debe seguir de todas las existentes se hace uso de los tokens generados en la etapa anterior. Para ello se recorre la lista de tokens extrayendo uno a uno para verificar si corresponde a una parte de la regla y así sucesivamente hasta corroborar que se sigue completamente la regla. Es en este momento donde se identifica la regla utilizada y se procede a transformar los tokens en símbolos. Estos símbolos ofrecen una mejor representación de los tokens obtenidos.

Tomemos como ejemplo la siguiente sentencia, para una mayor explicación de la obtención de la tabla de símbolos y códigos:

5 + 9 / 5

Para este ejemplo se tienen 5 tokens, 6 en caso se tome en cuenta el cambio de línea. Los tokens son 5, +, 9, /, 5 como se puede observar el lexema 5 se repite pero para el caso de los símbolos solo existe un símbolo con valor 5. Esta transformación no es muy trascendental en el caso d los lexemas con valores que nunca varían a lo largo del programa, como el caso del valor 5 que seguirá siendo 5 durante todo el programa. Pero si es importante en el caso de una variable.

Por ejemplo tenemos la variable llamada “altura” en un programa y durante el transcurso del programa se puede realizar una acción sobre la variable como la asignación de un valor “altura := 6”, después se realiza otra acción “área := base * altura” donde se hace uso de la misma variable. En el ejemplo solo se necesita de un elemento “altura” para asignarle un valor y obtener el valor de la misma variable, por eso se realiza la transformación a un símbolo.

Un token no solo permite generar símbolos sino también códigos. Los códigos son las acciones que se ejecutan sobre los símbolos. Estos se obtienen durante el recorrido de los tokens donde se extraen y verifican si son códigos. Por ejemplo analicemos el siguiente fragmento de pseudocódigo de un programa en pseudocódigo:

```
(1)   Si 33 < 45 entonces
(2)      Escribir “33”
(3)   Sino
(4)      Escribir “45”
(5)   Fin si
```

Con una simple inspección podemos encontrar algunas acciones, en la primera línea tenemos una operación de comparación (menor que) y en las líneas 2 y 4 tenemos acciones de impresión (Escribir “33” y Escribir “45”).

Pero existen otras acciones que están relacionadas a la estructura condicional las cuales son “saltar” y “saltar en falso”. Para las operaciones de comparación solo existen 2 respuestas verdadero y falso. En el ejemplo si la condición es verdadera se ejecuta la instrucción en la línea 2 y no se debe ejecutar la instrucción de la línea 4, si la condición es falsa se ejecuta la instrucción de la línea 4 y se salta a la instrucción de la línea 5. Por lo tanto es necesario crear una acción “saltar” que al ejecutar la

instrucción de la línea 2 realice un salto a la línea 5, también para el caso en que la condición sea falsa se necesita de una acción “saltar en falso” para que realice un salto desde la línea 1 hasta la línea 4. Con estas acciones explícitas o requeridas por la naturaleza de las estructuras es como se va llenando la tabla de código.

En el caso de la ejecución de funciones y procedimientos debe existir una división entre los símbolos y códigos, porque durante un programa se puede realizar varias invocaciones y los valores que se utilicen en cada invocación deben ser independientes. Para ello se crea una tabla de símbolo y código por cada definición de procedimiento, función y por cada invocación. Entonces cada vez que se realice una llamada a un procedimiento o función se toma como base esta tabla y se crea una copia de ella para tener independencia en cada llamada.

Al finalizar este proceso se obtiene la lista de códigos y símbolos del programa en pseudocódigo. Recorriendo la tabla de código se puede ejecutar el programa, sin embargo, pueden existir errores de tipos de variable los cuales se identifican antes de la ejecución y se realizan durante el análisis semántico.

Finalmente en el análisis semántico se realiza una pseudoejecución del programa, es decir, el programa se ejecuta pero no se actualizan los valores de los símbolos, sólo se actualizan y verifican los tipos. Por ello al evaluar las acciones de la tabla de código como en el caso de la suma se realiza la verificación de los símbolos asociados que para este caso deben ser de tipo numérico. En esta fase también se asignan y verifican los tipos de los parámetros de los procedimientos y funciones. Mediante este análisis se pueden identificar errores de tipo, antes de ser necesaria la ejecución.

Para invocar al intérprete se realizar mediante una línea de comandos, la cual se activa al seleccionar la opción de compilar en el editor de texto como se observa en el gráfico 18.

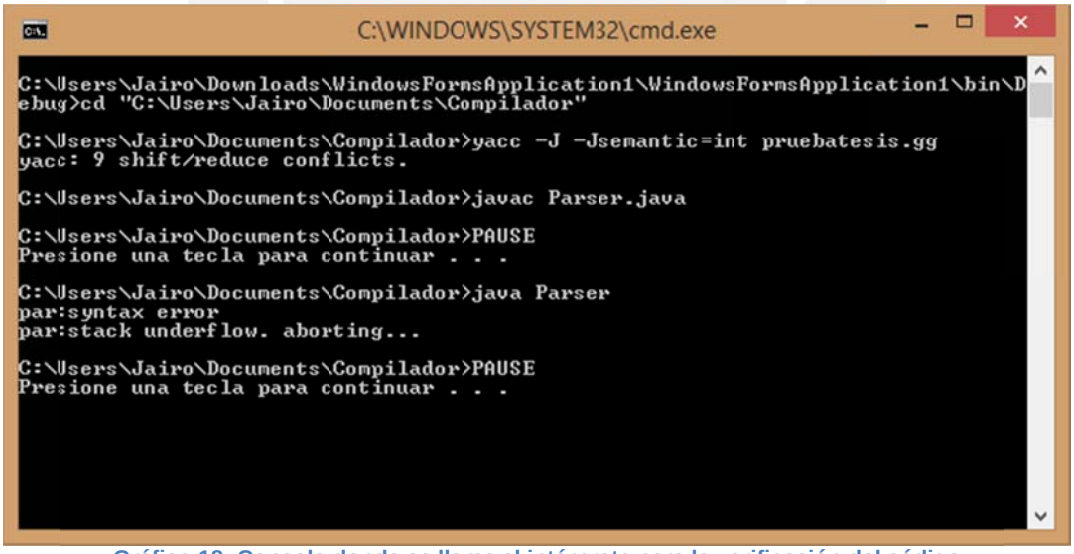


Gráfico 18: Consola donde se llama al intérprete para la verificación del código

Como resultado de la ejecución, el intérprete arroja los resultados en un archivo de texto con los errores detectados. En caso sea correcto el programa mostrará un mensaje “Compilación correcta” tal como se observa en la gráfico 18 el cual tiene 2 tipos de errores uno de token, el cual indica el carácter y línea de la incidencia, y otro de sintaxis. El error de sintaxis consiste en no cumplir con la sintaxis definida de

nuestra gramática con lo cual se muestra de error pero sin detalles, esto debido a que el análisis sintáctico lo realiza la herramienta Byacc el cual utiliza una máquina de estados para recorrer el código y verificar las reglas de sintaxis. Byacc realiza el manejo de errores en un procedimiento independiente al del análisis con lo cual no se tiene acceso a la información necesaria, además de desconocer la estructura de ella.

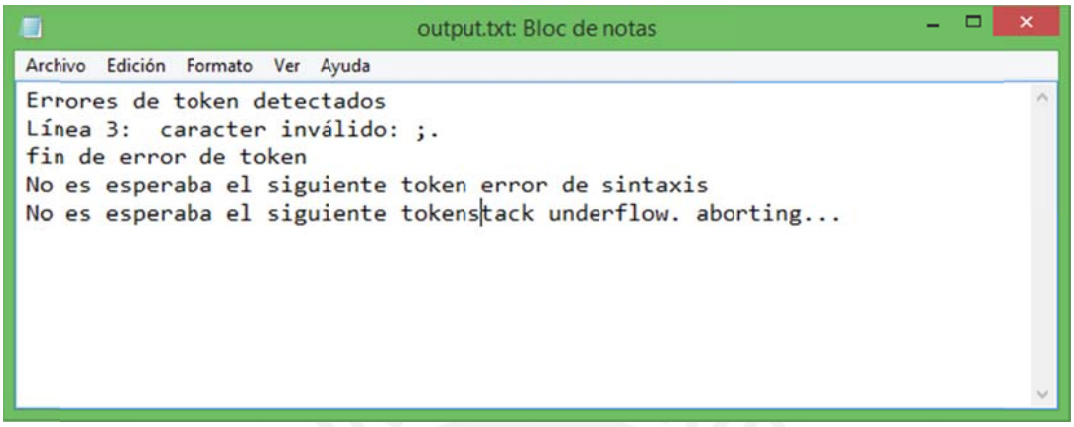


Gráfico 19: Archivo de texto con el resultado del intérprete

Capítulo 7: Conclusiones y Trabajos futuros

En esta sección se planteará las conclusiones del proyecto de fin de carrera así como los trabajos futuros que pueden desarrollarse.

7.1 Conclusiones

El desarrollo de este proyecto ha implicado en conocer los errores y dificultades que poseen los alumnos de cursos introductorios de programación, tales como el aprendizaje de la sintaxis y estructura de un lenguaje de programación. Debido a este hecho se le ofrece al alumno el uso de herramientas de diseño que le permiten expresar su solución utilizando el lenguaje natural. Existen herramientas que permiten generar este diseño y exportarlo en un lenguaje de programación, pero lamentablemente el código generado no indica si es sintácticamente correcto por lo cual se requiere de un compilador que lo verifique.

Por los motivos anteriormente expuestos, este proyecto de fin de carrera ofrece un entorno donde el alumno puede dibujar diagramas de flujo y como salida obtiene el pseudocódigo que se encuentra validado léxicamente, sintácticamente y semánticamente por el intérprete. De esta manera el alumno no necesita conocer de un lenguaje de programación para implementar su solución y puede enfocarse en buscar una solución al problema que le plantean.

Primero se implementó la interfaz de creación de diagramas de flujo en el cual se utilizó las librerías que ofrece Microsoft Visio. Se optó por utilizar las librerías en vez de crear un plug-in debido a que Visio ofrece otras herramientas adicionales a la creación de diagramas como opciones para insertar gráficos, imágenes. Si el usuario tiene acceso a todas estas herramientas puede generar errores en el proceso de traducción del diagrama a pseudocódigo al insertar componentes no reconocibles.

Luego, se definió la gramática del pseudocódigo que contiene su sintaxis utilizando la notación Backus-Naur Form (BNF). Posteriormente, se creó un método de traducción para convertir el diagrama de flujo en pseudocódigo. Durante este proceso se tuvo consideración de las instrucciones *if, while* y *for* que poseen 2 caminos, teniendo un mayor control del bloque de decisión que puede representar a la instrucción *if* o *while*, así como en el texto interno puede contener llamadas a funciones. Por lo tanto se definieron estructuras de datos que soporten los casos especiales mencionados. También se tuvo que verificar algunos errores visuales que se pueden detectar como la falta de conexión de los bloques o el uso de conectores sin dirección, los cuales son notificados al usuario para su corrección.

Después, se implementó el intérprete que, utilizando la gramática, procede a realizar la validación léxica, sintáctica y semántica del pseudocódigo generado, el cual al detectar alguna omisión o error, muestra un mensaje notificando de las fallas encontradas mostrando el detalle de la línea y carácter u operación infractor para el análisis léxico y semántico respectivamente. Mientras que para el análisis sintáctico solo se indica que existe un error de sintaxis debido a que este proceso lo realiza la herramienta Byacc que maneja la impresión de errores en un procedimiento independiente del proceso de análisis, además de no conocer la estructura y funcionamiento de la máquina de estados.

7.2 Trabajos Futuros

Como extensión directa de este trabajo se pretende adicionar el soporte de tipos de datos al diagrama de flujo debido a las limitaciones de la sintaxis del mismo. El manejo de tipos de datos implicará redefinir la sintaxis del diagrama de flujo para el soporte de tipos de datos simples y compuestos como: caracter, entero, cadena, arreglo, puntero, etc.

También se pretende implementar un intérprete que realice la verificación directamente en el diagrama de flujo de esta manera si se detecta un error este se visualizará en el bloque afectado, aumentando la experiencia de usuario al ofrecer un ambiente más interactivo.

Otro de los temas que se pretende abordar en los trabajos futuros es facilitar el acceso de esta herramienta a los alumnos mediante la conversión de esta aplicación de escritorio a una herramienta online accesible para todos. De esta manera la herramienta desde cualquier ordenador con conexión a internet sin la necesidad de instalar programas adicionales.

Referencias bibliográficas

AHO, Alfred V.
2007 Compilers Principles, Techniques and Tools.
Estados Unidos: Pearson Education.

ALONSO, Fernando
2005 Introducción a la ingeniería de software.
España: Delta Publicaciones Universitarias. Primera Edición.

BIKAS Chaudhuri, Anil
2005 The art of programming through flowcharts and algorithms
Estados Unidos: Firewall Media

BUTTLER ,Mathew y Morgan Michael
2007 "Learning challenges faced by novice programming students studying high level and low feedback concepts". Singapoure: Information and Communication Technologies (ICT). pp: 99-97

BYACC s/f
Consultado el 15 de octubre de 2013. http://byaccj.sourceforge.net/

CAIRO Battitutti, Osvaldo
2003 Metodología de Programación
Mexico D.F: Alfaomega.

CAO, Jill; Felming, Scott D.; Burnett, Margaret
2011 "An Exploration of Design Opportunities for Gardening End-User Programmers' Ideas". Estados Unidos: IEEE Symposium on Visual Languages and Human-Centric Computing. pp: 35-42

COOPER, Keith D.
2011 "Engineering a Compiler".
Estados Unidos: Elsevier. Pp: 475-490

DÍAZ Díaz, José Amadeo Martín
2004 *Visual Block: lenguaje de programación visual / Karin Johana Moya Saavedra*
Tesis de Licenciatura en Ciencias e Ingeniería con mención en Ingeniería Informática. Lima: Pontificia Universidad Católica del Perú, Facultad de Ciencias e Ingeniería.

DEITEL, Harvey
2009 Cómo programar C++: introducción a la programación de juegos y las bibliotecas boost. Naucalpan de Juaréz: Pearson.

DRAZEN Lucanin, Ivan Fabek
2011 "A visual programming language for drawing and executing flowcharts"
Istra: Proceedings of the 34th International Convention MIPRO.

FLORES Cueto, Juan José
2011 Método para la resolución de problemas utilizando la programación orientada a objetos. Perú: Universidad San Martin de Porres (USMP).

GOMEZ Díaz, Renzo y Juan Jesús SALAMANCA

2012 *Intérprete para un lenguaje de programación orientado a objetos, con mecanismos de optimización y modificación dinámica de código*. Tesis de licenciatura en Ciencias e Ingeniería con mención en Ingeniería Informática. Lima: Pontificia Universidad Católica del Perú, Facultad de Ciencias e Ingeniería.

GUÉRIN, Brice-Arnaud
2005 Lenguaje C++.
España: Ediciones ENI.

HÉRNANDEZ Valdelamar, Eugenio Jacobo
2002 El Paradigma de la programación visual.
Jalisco: Presentado en el CNCIIC del ANEII.

HIRSH Martínez, Layla
2007 *Intérprete y entorno de desarrollo para el aprendizaje de lenguajes de programación estructurada*. Tesis de licenciatura en Ciencias e Ingeniería con mención en Ingeniería Informática. Lima: Pontificia Universidad Católica del Perú, Facultad de Ciencias e Ingeniería.

JOYANES, Luis
2008 Fundamentos de programación: Algoritmos, estructuras de datos y objetos.
Cuarta edición. España: McGraw-Hill.

LAHTINEN Essi; Ala-Mutka, Kirsti; Jarvinen, Hannu-Matti
2005 "A study of the difficulties of novice programmers". Estados Unidos: ACM SIGCSE Bulletin. Volumen 37. Número 3. pp: 14-18.

LI, Xu
2007 "Journal of Computing Sciences in Colleges". *RobotStudio: A Universal IDE for Teaching*. Estados Unidos 2007, Volumen 22, pp. 65-72.

LÓPEZ Gaona, Amparo
2007 Introducción al desarrollo de programas con java.
México: México.

MAC Gul, Marcia
2012 Resolución de problemas computaciones: análisis del proceso de aprendizaje
Argentina: Universidad Nacional de la Plata

MARAT, Boshernistan
2004 Visual Programming Languages: A Survey.
California: University of California.

MARRER, Gary
2009 Fundamentals of Programming: With Object Oriented Programming
Estados Unidos: Creative Commons Attribution-Noncomercial-Share Alike 3.0

MICROSOFT VISIO s/f
Consultado el 15 de octubre de 2013. http://office.microsoft.com/es-es/visio/

MONTERO, Eloísa; RUIZ Dávila, María; DÍAZ, Beatriz
2010 Aprendiendo con videojuegos: Jugar es pensar 2 veces.
España: Narcea S.A de Ediciones.

NELL B, Dale
2007 Programación y resolución de problemas con C++.
México D.F: McGraw-Hill.

Ó CONCHÜIR, Deasún
2011 Overview of the PMBOK Guide: Shot Cuts for PMP Certification
Alemania:Springer. 2° Edición.

PROYECT Managment Institute (PMI)
2008 Guía de los fundamentos para la dirección de proyectos (Guía del PMBOK)
Proyect Managment Institute. 4° Edición.

ROGERSON, Christine y Scott, Elsje
2010 "The Fear Factor: How It Affects Students Learning to Program in a Tertiary Environment". Estados Unidos: Journal of Information Technology Education. Volumen 9. pp: 147-171.

SALAS Campos, Heana
2007 Una Propuesta Dinámica para la Programación con Micromundos.
Costa Rica: Editorial Universidad Estatal a Distancia.

TAN, Phit-Huan; TING, Choo-Yee; LING, Siew-Woei
2009 "Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception". Malasya: International Conference on Computer Technology and Development, 13-15 Noviembre, pp: 42-46.

VISUAL STUDIO s/f
Consultado 15 de octubre de 2013. http://www.microsoft.com/visualstudio/eng/visual-studio-2013

VUJOŠEVIĆ-JANIČIĆ Milena y Tošić DUŠAN
2008 The role of programming paradigms in the first programming courses. *La enseñanza de matemáticas.* Belgrade, 2008, volume XI , pp. 63-83.

WEITZENFELD, Alfredo
2005 Ingeniería de Software Orientada a Obtejos con Uml, Java e Internet.
Mexico: Cengage Learning Editores.

ZHIXION, Chen y Marx, Delia
2005 "Journal of Computing Sciences in Colleges". *Experience with eclipse IDE in programming*. Estados Unidos 2005, Volumen 21, pp. 104-112.