PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

ENTORNO DE DESARROLLO PARA LA EJECUCIÓN Y TRADUCCIÓN DE PSEUDOCÓDIGO

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

Juan Carlos Jara Loayza

ASESOR: Dr. Andrés Melgar Sasieta

Lima, diciembre de 2013

Resumen

El presente proyecto de fin de carrera corresponde a la construcción de un entorno de desarrollo que permita la ejecución y ejecución de pseudocódigo como herramienta de apoyo a las etapas de diseño, ejecución y validación de un algoritmo. Se permitirá la traducción a VBA, Java, Ruby, Python y c++.

El documento presenta siete capítulos, en el primer capítulo se describen generalidades del proyecto como la problemática, objetivo general, objetivos específicos, resultados esperados, alcance, justificación, viabilidad y plan de actividades del proyecto.

En el segundo capítulo se presenta el marco conceptual donde se describen los conceptos necesarios para entender el problema que se desea solucionar con el presente proyecto.

En el tercer capítulo se presenta el estado del arte. En lo referente al estado del arte se realizó una búsqueda entre productos comerciales y no comerciales que intentan solucionar algunos aspectos del problema a resolver.

En el cuarto capítulo, se describe cómo se realizó la definición y la validación de la gramática del pseudocódigo mediante el uso de la notación BNF.

En el quinto capítulo se presenta el desarrollo del intérprete que permite la ejecución del pseudocódigo y del traductor que permitirá transformar el pseudocódigo a código en VBA, Java, Ruby, Python y C++.

En el sexto capítulo se realiza la descripción del entorno de desarrollo que permitirá la integración del intérprete y traductor para que puedan ser utilizados por el usuario final.

En el séptimo capítulo se exponen las conclusiones

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:	**Entorno de desarrollo para la ejecución y traducción de pseudocódigo**
ÁREA:	CIENCIAS DE LA COMPUTACIÓN
PROPONENTE:	Dr. Melgar Sasieta, Héctor Andrés
ASESOR:	Dr. Melgar Sasieta, Héctor Andrés
ALUMNO:	Jara Loayza, Juan Carlos
CÓDIGO:	20084715
TEMA N°:	517
FECHA:	San Miguel, 24 de febrero de 2014

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Facultad de Ciencias e Ingeniería
07 MAR. 2014
DANIEL TORREALVA DAVILA
DECANO

DESCRIPCIÓN

Las dificultades al desarrollar programas generalmente están ligadas a las habilidades del programador, a la sintaxis del lenguaje escogido, al diseño del algoritmo, entre otros. Este proceso se torna más complejo debido a que las propuestas de solución son implementadas en lenguajes de programación específicos y muchas veces los codificadores realizan la implementación del programa sin verificar la validez del mismo.

Por otro lado, verificar la correctitud del algoritmo planteado no es una tarea trivial. Pese a que existen diversas formas de representar el algoritmo como: diagramas de flujo, pseudocódigo, diagrama NS, debido a la naturaleza de éstos, no se pueden ejecutar directamente en un computador. Además, caso se haya representado y verificado el algoritmo usando pseudocódigo, se debe codificar en el lenguaje seleccionado. Esta etapa puede presentar algunos problemas pues se requiere conocer completamente la sintaxis del lenguaje y la equivalencia de las estructuras u operaciones.

Despues de haber realizado la codificación, se requiere de un intérprete o compilador, comúnmente incluido en un entorno de desarrollo, para la ejecución final. Existen herramientas que apoyan a estas etapas de la resolución de problemas con ayuda del computador, sin embargo, estas no se encuentran integradas. Es decir, se debe hacer uso de un programa, luego de otro para completar todas las etapas.

Este proyecto de fin de carrera busca construir un entorno de desarrollo que permita al programador principiante o con problemas en la etapa de diseño de algoritmo o codificación, la ejecución del diseño representado mediante pseudocódigo, la traducción a código VBA, Python, Ruby, Java, C++, y finalmente la ejecución del código VBA generado.

Av. Universitaria 1801
San Miguel, Lima – Perú

Apartado Postal 1761
Lima 100 – Perú

Teléfono:
(511) 626 2000 Anexo 4801

FACULTAD DE
CIENCIAS E INGENIERÍA
ESPECIALIDAD DE
INGENIERÍA INFORMÁTICA

PONTIFICIA
UNIVERSIDAD CATÓLICA
DEL PERÚ

OBJETIVO GENERAL

Desarrollar un entorno de desarrollo y un intérprete de pseudocódigo que genere y ejecute código en *Visual Basic for Applications* (VBA), como herramienta de apoyo al diseño, codificación y ejecución de un algoritmo.

OBJETIVOS ESPECÍFICOS

1. Definir la estructura y sintaxis del pseudocódigo para la representación del diseño del algoritmo..
2. Permitir la ejecución de algoritmos que sean diseñados bajo la estructura y sintaxis del pseudocódigo para su verificación.
3. Permitir la codificación de pseudocódigo a código en lenguaje VBA, Python, Ruby, Java y C++, por medio de la traducción de código de un lenguaje al otro.
4. Implementar un entorno de desarrollo que permita invocar las acciones de ejecución de programas en pseudocódigo y código VBA, y que además permita invocar la acción de traducción de pseudocódigo a código VBA, Python, Ruby, Java y C++.

ALCANCE

- Se desea brindar un entorno de desarrollo que sea capaz de ejecutar pseudocódigo y código VBA, y que además posea la funcionalidad de traducir pseudocódigo a código VBA, Python, Ruby, Java y C++.
- El entorno de desarrollo brindará la capacidad, por medio de componentes gráficos, de ejecución o traducción de código. Además poseerá un editor de texto donde se podrá colocar el programa en pseudocódigo para su posterior ejecución o traducción.
- El pseudocódigo permitirá hacer uso de las estructuras selectivas, iterativas, algunas funciones matemáticas, de cadenas, entre otras. El detalle de estas se podrá verificar en el anexo A.

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 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 Andrés Melgar 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.

Índice

Tabla de ilustraciones

CAPÍTULO 1

1 Introducción

Las dificultades al desarrollar programas generalmente están ligadas a las habilidades del programador, a la sintaxis del lenguaje escogido, al diseño del algoritmo, entre otros. Este proceso se torna más complejo debido a que las propuestas de solución son implementadas en lenguajes de programación específicos y muchas veces los codificadores realizan la implementación del programa sin verificar la validez del mismo.

Por otro lado, verificar la correctitud del algoritmo planteado no es una tarea trivial. Pese a que existen diversas formas de representar el algoritmo como: diagramas de flujo, pseudocódigo, diagrama NS, debido a la naturaleza de éstos, no se pueden ejecutar directamente en un computador. Además, caso se haya representado y verificado el algoritmo usando pseudocódigo, se debe codificar en el lenguaje seleccionado. Esta etapa puede presentar algunos problemas pues se requiere conocer completamente la sintaxis del lenguaje y la equivalencia de las estructuras u operaciones.

Después de haber realizado la codificación, se requiere de un intérprete o compilador, comúnmente incluido en un entorno de desarrollo, para la ejecución final. Existen herramientas que apoyan a estas etapas de la resolución de problemas con ayuda del computador, sin embargo, estas no se encuentran integradas. Es decir, se debe hacer uso de un programa, luego de otro para completar todas las etapas. Este proyecto de fin de carrera busca construir un entorno de desarrollo que permita al programador principiante o con problemas en la etapa de diseño de algoritmo o codificación, la ejecución del diseño representado mediante pseudocódigo, la traducción a código VBA, Python, Ruby, Java, C++ y finalmente la ejecución del código VBA generado.

2 Problemática

En estudios realizados en el año 2012, se identificaron problemas en los cursos introductorios de programación asociados al proceso de aprendizaje y enseñanza (MASSA 2012). Estos estudios muestran que el alumno invierte poco tiempo en el diseño del código, le brinda más importancia al lenguaje de programación que al proceso de construir un programa y que el alumno toma un rol pasivo en el proceso de aprendizaje (ídem). El aprendizaje se enriquece al tener una herramienta que permita una interacción más flexible y dinámica con el estudiante (CHUMPITAZ 2005: 20). Además, se necesita de la constante práctica y conocimientos en matemáticas, no obstante, la mayor dificultad es la falta de habilidad para resolver problemas (GOMES 2007). En la mayoría de los casos, esto se debe a que el aprendizaje se enfoca más en la sintaxis del lenguaje de programación que en el diseño del algoritmo que resuelva el problema (ídem).

La resolución de problemas con el uso de computadoras se divide en al menos 4 etapas: análisis del problema, diseño del algoritmo, codificación, ejecución y validación del programa (JOYANES 2008: 46). La parte de análisis comprende la lectura y compresión del problema a resolver. Se debe identificar las entradas, las salidas, posibles restricciones y el método que produce el resultado deseado y se puede representar mediante el uso del diagrama IPO (*Input-Proccess-Output*) (NORTON 2006: 504). En la siguiente etapa, diseño del algoritmo, se pueden usar diversas herramientas para representar el algoritmo, como diagramas de flujo, diagrama de

Nassi-Scheneiderman y pseudocódigo. A veces, este paso es omitido al aprender a programar para acelerar el proceso de registrar el algoritmo en el lenguaje de programación elegido. Lo cual puede generar dificultades en el diseño al seguir la sintaxis del lenguaje escogido (GARNER 2003).

Es recomendable el uso de las herramientas mencionadas para la etapa de diseño debido a que le ofrecen al algoritmo la independencia de la estructura y sintaxis del lenguaje de programación a usar (JOYANES 2008: 48). La siguiente etapa es codificar el algoritmo, es decir, escribir el programa (pseudocódigo) bajo la sintaxis de un lenguaje de programación, lo que es sencillo a partir del pseudocódigo ya que el diseño del algoritmo posee una estructura similar a la mayoría de lenguajes de programación. Finalmente, en la etapa de ejecución y validación, se pone en funcionamiento el código generado y, luego, se verifican los resultados obtenidos. Para ejecutar el código se necesita un compilador o intérprete, que comúnmente se incluye en un entorno de desarrollo (IDE), el cual facilita la labor de ejecución del código, o se invoca por la línea de comandos. En la ejecución se puede conocer la existencia de errores o bugs. Estos errores pueden ser de 3 tipos: de compilación (sintaxis), de ejecución (instrucción no ejecutable) o lógico (diseño) (JOYANES 2008: 52, GARNER 2003: 218-219).

Los errores lógicos son los más difíciles de detectar debido a que el programa aparentemente logra funcionar, y la única forma de detectarlos es evaluando el resultado obtenido. Usualmente estos se generan en la etapa de diseño, en donde se usan herramientas que representan el algoritmo, una de las cuales puede ser el pseudocódigo. El pseudocódigo posee como ventaja el priorizar la lógica y estructura del algoritmo sobre la sintaxis de un lenguaje de programación. Además, se identificó que existe mayor dificultad en el uso de los lenguajes de programación que en la programación en sí (GRANDELL 2006). Sin embargo, el pseudocódigo no puede ser ejecutado por una computadora (JOYANES 2008: 70) ello induce a veces a codificar el algoritmo bajo un lenguaje de programación para su ejecución y verificación porque la correctitud del algoritmo no se puede verificar mediante una computadora (GODSE 2008:14). Otro problema relacionado al uso de pseudocódigo es la falta de un estándar para su estructura y sintaxis, dos personas pueden no compartir la misma lógica (MARRER 2009:56).

Un posible lenguaje para iniciantes podría ser el lenguaje VBA (*Visual Basic for Application*), el cual forma parte de los lenguajes BASIC (*Beginner's All-purpose Symbolic Instruction Code*) los cuales poseen partes de paradigma imperativo, y tienen una sintaxis y estructura sencilla de comprender que fueron diseñadas especialmente para programadores principiantes (DEITEL 2009: 10). Además posee una gran similitud al pseudocódigo en estructura y sintaxis.

Después de lo expuesto, se desea brindar apoyo para solucionar los problemas relacionados con las etapas de la resolución de problemas, específicamente las etapas de: diseño, codificación y ejecución. Además se busca enfatizar la importancia del diseño del algoritmo y su verificación antes de codificarlo, sin las limitaciones de sintaxis de un lenguaje de programación permitiendo la ejecución del diseño, una fácil traducción y ejecución del diseño codificado. Esto se podrá realizar a través de una interfaz (IDE) que permita la traducción pseudocódigo a distintos lenguajes de programación, entre ellos VBA y la ejecución del pseudocódigo.

3 Objetivo general

Desarrollar un entorno de desarrollo y un intérprete de pseudocódigo que genere y ejecute código en *Visual Basic for Applications* (VBA), como herramienta de apoyo al diseño, codificación y ejecución de un algoritmo.

4 Objetivo general

Desarrollar un entorno de desarrollo y un intérprete de pseudocódigo que genere y ejecute código en *Visual Basic for Applications* (VBA), como herramienta de apoyo al diseño, codificación y ejecución de un algoritmo.

5 Objetivos específicos

- Objetivo 1: Definir la estructura y sintaxis del pseudocódigo para la representación del diseño del algoritmo.
- Objetivo 2: Permitir la ejecución de algoritmos que sean diseñados bajo la estructura y sintaxis del pseudocódigo para su verificación.
- Objetivo 3: Permitir la codificación de pseudocódigo a código en lenguaje VBA, Python, Ruby, Java y C++ por medio de la traducción de código de un lenguaje al otro.
- Objetivo 4: Implementar un entorno de desarrollo que permita invocar las acciones de ejecución de programas en pseudocódigo y código VBA, y que además permita invocar la acción de traducción de pseudocódigo a código VBA, Python, Ruby, Java y C++.

6 Resultados esperados

- Resultado 1 para el objetivo 1: Gramática definida para el lenguaje de pseudocódigo que será usada en el intérprete de pseudocódigo, usando la notación BNF.
- Resultado 2 para el objetivo 2: Intérprete que permita la ejecución de pseudocódigo.
- Resultado 3 para el objetivo 3: Programa que permitirá al intérprete de pseudocódigo la traducción a código VBA, Python, Ruby, Java y C++ a partir de pseudocódigo.
- Resultado 4 para el objetivo 4: Entorno de desarrollo que use como flujo de entrada programas en pseudocódigo para su conversión a código VBA, Python, Ruby, Java y C++ y posterior ejecución.
- Resultado 5 para el objetivo 4: Entorno de desarrollo que acepte como flujo de entrada programas en lenguaje de programación VBA para su posterior ejecución.

7 Herramientas, métodos y procedimientos

En la siguiente sección se presentarán las metodologías y procedimientos a emplear para desarrollar los resultados esperados previamente descritos.

7.1 Mapeo

Resultado Esperado	Resultado esperado 1
Métodos, metodología o procedimiento	Se usará la notación llamada Backus Naur Form (BNF), como se podrá ver en el capítulo 3, es una forma para describir la sintaxis de un lenguaje. Permite representar gramáticas de libre contexto por medio de producciones.

	Cada producción consta de reglas, símbolos terminales y no terminales los cuales son identificados previamente para el lenguaje de pseudocódigo (LEE 2008: 23).
Justificación	Se empleará este método debido a que es sencillo de entender y no es ambiguo (ídem: 23).

Resultados Esperados	Resultado esperado 2
Métodos, metodología o procedimiento	Para la construcción del intérprete, se seguirán las etapas de las fases del compilador, front end y back end, definidas en el marco teórico. Para el desarrollo del intérprete se realizarán etapas independientes. Es decir, se tendrá las etapas de análisis léxico, sintáctico, semántico, generación de código intermedio y generación de código por separado. Para dar soporte a la etapa de análisis sintáctico se hará uso de YACC, el cual genera un analizador sintáctico a partir de una gramática escrita en una notación similar a BNF.
Justificación	La estructura del intérprete se divide en 2 fases: análisis (front end) y síntesis (back end). Estas fases de dividen en las etapas definidas anteriormente (AHO 2007: 4).

Resultados Esperados	Resultado esperado 3
Métodos, metodología o procedimiento	Se dividirá la etapa de generación de código en 2 ramas, una de las cuales permitirá la funcionalidad de traducción de pseudocódigo a código VBA, Python, Ruby, Java y C++. Para la traducción se empleará la tabla de símbolos y la tabla de código intermedio generado.
Justificación	Se utilizará las otras etapas desarrolladas para el intérprete con el fin de ofrecer al usuario la posibilidad de traducir o ejecutar el pseudocódigo.

Resultados Esperados	Resultado esperado 4 y 5
Métodos, metodología o procedimiento	Se utilizará el método de cascada iterativo incremental que permitirá presentar funcionalidades por iteraciones. Además se considerarán las etapas de especificación de requisitos (mencionados en el capítulo 5), estos serán basados en las funcionalidades básicas del entorno de desarrollo. Se realizarán las etapas de diseño, implementación, pruebas e integración entre el intérprete y el entorno de desarrollo (WEITZENFELD 2004: 50).
Justificación	Es más fácil de comprender y probar las funcionalidades en menor tiempo debido a que los incrementos son de menor tamaño (WEITZENFELD 2004: 51). Debido a que la complejidad del proyecto se enfoca más en la implementación de los intérpretes se generará un entorno de desarrollo básico para su integración.

Las instrucciones del lenguaje de programación deben ser traducidas para poder ser ejecutadas por la computadora. El compilador permite está traducción, puesto que toma de entrada un programa escrito en algún lenguaje de programación (código fuente) y produce un programa equivalente (lenguaje objeto). El lenguaje objeto es comúnmente código en lenguaje máquina. Sin embargo, en algunos casos se puede

generar código en otro lenguaje de programación, estos compiladores son conocidos como traductores de fuente a fuente. Mientras que un intérprete en lugar de generar código objeto ejecuta el código fuente. La búsqueda de errores en el intérprete es mejor que la de un compilador; sin embargo, su ejecución es más lenta (COOPER 2004: 2-3).

El compilador se divide en 2 fases, que a su vez contienen diversas etapas:

- Análisis – Front End: en esta etapa se establece la estructura gramatical del código, se detecta si el programa tiene errores verificando la sintaxis y gramática. Si se encuentra algún error se informa del mismo, sino se procede a crear código intermedio y la tabla de símbolos. En esta última se almacenan las variables del programa, posee información como nombre, valor, tipo de dato y hasta ámbito de la variable. (AHO 2007: 11). Esta fase presenta las siguientes etapas:

  - Analizador léxico: en esta etapa se lee todo el código fuente y se agrupa y clasifica los caracteres en grupos con significados llamados lexemas, cada uno de esto produce un token el cual cuenta con nombre y valor (ídem: 5-6).

  - Analizador sintáctico: a partir de los tokens se generará un árbol sintáctico, el cual sigue una estructura gramatical. Un ejemplo sería en una operación, en la que los nodos interiores representan operadores y sus nodos hijos representan los argumentos (ídem: 8).

  - Analizador semántico: esta etapa se enfoca a revisar el significado del código. Existen programas en los cuales se puede verificar la semántica antes de su ejecución (semántica estática) como los tipos de datos. Sin embargo, otros necesitan de la etapa de ejecución para verificarse (semántica dinámica), por ejemplo cuando se realizan acciones sobre una variable que se lee (input del usuario) en este caso no se conoce el tipo de la variable hasta el momento de la ejecución. (LOUDEN 2004: 9-10).

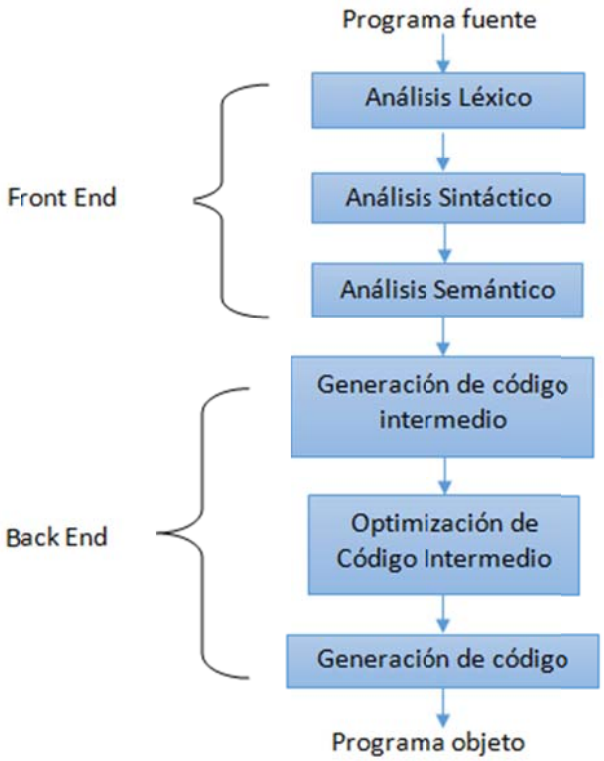


Ilustración 1: Etapas del compilador. Fuente:propia.

- Síntesis- Back End: se interpreta el código intermedio, se establece donde se guardan los comandos ejecutados, así como el orden de ejecución (COOPER 2004: 18-19). Además se construye el programa objeto a partir del código intermedio y la tabla de símbolos (AHO 2007: 4). Esta fase presenta las siguientes etapas:

  - Generación de código intermedio: el código intermedio es una representación del código fuente que debe ser fácil de producir y traducir por el computador. El árbol generado en el análisis del código fuente es un ejemplo de código intermedio (ídem: 9).

  - Fase de optimización: se busca obtener código más rápido, menor código, que consuma menor cantidad de recursos. Esto se realiza eliminando redundancias o sentencias innecesarias, cambiando modos de direccionamiento. Sin embargo, está etapa es opcional puesto que puede retrasar el proceso completo de compilación (ídem: 11, LOUDEN 2004: 12).

  - Generación de código: a partir del código intermedio se traduce en el lenguaje objetivo (AHO 2007: 11).

Para representar la sintaxis de un lenguaje se puede usar BNF (Backus-Naur Form), la cual es una notación que permite representar símbolos terminales, no terminales y las derivaciones para llegar a estos (COOPER 2004: 87).

En la construcción de un compilador existen diversas herramientas para facilitar los resultados de cada etapa. Se presentan algunos a continuación (AHO 2007: 12):

- Generador de scanner: a partir de expresiones regulares de los tokens, produce un analizador léxico.
- Generador de parser: produce analizadores sintácticos a partir de la gramática definida por el lenguaje, usualmente usa BNF.
- Motores de traducción de sintaxis directa: que producen un conjunto de rutinas para recorrer el árbol sintáctico o el código intermedio.
- Generador de generador de código: produce un generador de código a partir de las reglas para traducir el código intermedio a código objetivo.
- Motor de análisis de flujo de información: muestra como los valores son transmitidos entre las partes del programa, esta herramienta es útil para la etapa de optimización de código.
- Herramienta para la construcción del compilador: ofrece rutinas para la construcción de diversas fases del compilador.

Para ofrecer al usuario una manera fácil de usar el producto se necesita un entorno de desarrollo o también llamado IDE (Integrated Development Environment). El cual contiene a las herramientas y componentes como por ejemplo: editor de texto, el intérprete, un depurador, etc. (ESLAVA 2012: 10)

8 Alcance

- Se desea brindar un entorno de desarrollo que sea capaz de ejecutar pseudocódigo y código VBA, y que además posea la funcionalidad de traducir pseudocódigo a código VBA, Python, Ruby, Java y C++.
- El entorno de desarrollo brindará la capacidad, por medio de componentes gráficos, de ejecución o traducción de código. Además poseerá un editor de

texto donde se podrá colocar el programa en pseudocódigo para su posterior ejecución o traducción.
- El pseudocódigo permitirá hacer uso de las estructuras selectivas, iterativas, algunas funciones matemáticas, de cadenas, entre otras. El detalle de estas se encuentra en el anexo A.

8.1 Limitaciones

- No se realizará soporte especializado a la etapa de pruebas de la resolución de problemas, ya que esta requiere de herramientas más sofisticadas que exceden el alcance del proyecto.
- No se implementará la etapa de optimización de código de los intérpretes indicados. Debido a que el intérprete de pseudocódigo soportará algunas estructuras y funciones detalladas en anexo A y se encuentran enfocados a programadores principiantes.

8.2 Riesgos

Los riesgos identificados, impacto y medidas correctivas para el proyecto se presentan a continuación:

Riesgo identificado	Impacto en el proyecto	Medidas correctivas para mitigar
Ausencia de asesor por motivo de viaje.	Medio	Buscar un nuevo asesor para la revisión de los siguientes entregables, sin generar retrasos en los avances del proyecto.
Dificultad para integrar intérprete con entorno de desarrollo	Medio	Buscar tecnologías que permitan otras formas de integración.
Retraso en la entrega de algún avance.	Alto	Invertir más tiempo para poder cumplir con el plan establecido.

9 Justificación y viabilidad

En la siguiente sección se verificará la viabilidad y la justificación para llevar a cabo el proyecto.

9.1 Justificativa del proyecto de tesis

El presente proyecto de fin de carrera brindará apoyo en la etapa de diseño del algoritmo mediante el uso de pseudocódigo. Se seleccionó el pseudocódigo porque es sencillo de usar ya que combina sentencias del lenguaje español, en este caso, y sentencias similares a las usadas por lenguajes de programación sin necesariamente seguir la estructura definida por estos (FARELL 2012:14).

Además se permitirá la ejecución del pseudocódigo, lo cual evitará la necesidad de codificar el algoritmo para poder validar los resultados. De este modo no se necesitará conocer la sintaxis definida por algún lenguaje de programación para poder diseñar una solución verificable.

Para dar apoyo a la etapa de codificación se podrá traducir el programa en pseudocódigo a lenguaje Visual Basic for Applications (VBA), el cual fue seleccionado debido a que posee una sintaxis orientada a programadores principiantes (DEITEL 2009: 10). Además, las estructuras del lenguaje VBA son similares a las estructuras del pseudocódigo que se empleará. Adicionalmente, para poder validar los resultados del programa codificado en VBA se permitirá su ejecución.

Para la ejecución y traducción de los programas generados se hará uso de intérpretes, a los que se tendrá acceso a través de un entorno de desarrollo que permitirá el diseño de estos programas. En este entorno se podrá escribir el programa en pseudocódigo, ingresar el input y a través de la ejecución o traducción obtener el resultado del programa o el programa traducido a VBA respectivamente.

Para concluir, se realizó unas preguntas a un grupo de alumnos que se encuentra cursando el curso de Introducción a la Computación de la PUCP, detalladas en el Anexo E, en la cual se concluyó que el pseudocódigo no es usado porque no se puede verificar y una herramienta que ejecute y traduzca sería de gran ayuda.

9.2 Análisis de viabilidad del proyecto de tesis

Como se puede apreciar en el diagrama de Gantt, se tiene un tiempo estimado para el proyecto de 4 meses y se utilizarán herramientas gratuitas lo cual evitará limitaciones financieras. Además, se estima que una sola persona podrá terminar el proyecto en el tiempo propuesto, y se podrá desarrollar con los conocimientos que se poseen sobre el tema más un análisis y selección de herramientas tecnológicas adecuadas para las etapas previamente definidas. De esta manera se puede verificar la viabilidad del presente proyecto.

10 Plan de actividades

Entregable	Hito
Gramática definida para el pseudocódigo Analizador léxico	02/09/2013
Análisis sintáctico: operaciones aritméticas, operadores de comparación, asignación de variables, lectura y escritura de datos, estructura condicional (IF)	09/09/2013
Análisis sintáctico: estructuras For, While y Case	16/09/2013
Análisis Semántico	23/09/2013
Análisis sintáctico: operaciones trigonométricas y con cadenas Documentación de resultado esperado 1	30/09/2013
Análisis sintáctico: procedimiento y funciones	07/10/2013
Traducción de pseudocódigo a código VBA (operaciones aritméticas, operadores de comparación, asignación de variables, estructuras IF, For , While ,Else)	14/10/2013
Traducción de pseudocódigo a código VBA(procedimientos, funciones)	21/10/2013
Traducción de pseudocódigo a código VBA(lectura y escritura de datos) Documentación de resultado esperado 2	28/10/2013
Ejecución de código traducido (código VBA) Documentación de resultado esperado 3	04/11/2013
Entorno de desarrollo (integración con intérprete de pseudocódigo) Documentación de resultado esperado 4	11/11/2013
Entorno de desarrollo (integración con traductor y ejecución de código VBA)	18/11/2013
Entorno de desarrollo (desarrollo del menú de opciones)	25/11/2013

CAPÍTULO 2: Marco Conceptual

En la siguiente sección se definirán conceptos que permitirán tener un mejor entendimiento del problema presentado anteriormente y de la propuesta de solución.

Conceptos relacionados al problema

La enseñanza de lenguajes de programación puede implicar dificultades en el aprendizaje del interesado algunas causas provienen de los métodos usados para enseñar, los métodos de estudio de los futuros usuarios, de sus habilidades y aptitudes, de características de la programación, y efectos psicológicos. A continuación de describirán brevemente estas causas (GOMEZ 2007):

- Interpretación del problema: se debe entender completamente el problema antes de intentar resolverlo, caso contrario se implementará una solución incorrecta. Intentar resolver el problema antes de comprenderlo puede ser origen de la ansiedad del programador o mala interpretación del problema.

- Habilidades: como se mencionó antes, se debe enfatizar en el diseño de los algoritmos. Se necesita establecer pasos para resolver un problema, empezando con problemas de poca dificultad e ir incrementando la dificultad gradualmente. Además, se necesita conocimientos matemáticos que permitan mejorar el razonamiento lógico.

- Características de la programación: se necesita mejorar las habilidades de abstracción, pensamiento y generalización para poder proponer soluciones que luego al obtener experiencia permitan resolver problemas más complejos. También la complejidad del lenguaje de programación a usar reta al programador a construir el algoritmo y analizar cómo se codificará posteriormente.

- Efectos psicológicos: la motivación es un factor importante para la programación, así como la perseverancia al analizar y diseñar el algoritmo, y no rendirse al primer intento fallido.

Conociendo los factores que afectan el aprendizaje de los lenguajes de programación, se pueden explicar las fases que permiten la resolución de problemas con el uso de la computadora.

La resolución de problemas en general empieza al enfrentarnos a un problema. Lo primero que se debe hacer es obtener toda la información necesaria. Luego se analiza el problema para verificar si es uno ya antes resuelto o se divide en partes más pequeñas lo que permite llegar a la solución de forma más sencilla. Finalmente se establece los pasos necesarios para resolverlo (algoritmo) (DALE 2007: 23-24).

Todo algoritmo debe cumplir fundamentalmente 3 características:

- Preciso, al indicar el orden de realización de cada paso.

- Definido, puesto que se debe obtener el mismo resultado siempre.

- Finito, debe tener una cantidad limitada de pasos a seguir.

Entre los métodos para resolver problemas uno de los más usados es divide o vencerás[1]. Este método propone subdividir un problema complejo en subproblemas más simples que pueden volver a dividirse hasta poder resolverlos (JOYANES 2008: 64-65). Estas divisiones pueden ser llamadas módulos, los cuales deben diseñarse bajo criterios de acoplamiento y cohesión. Se desea que el acoplamiento sea el mínimo posible, es decir, que exista independencia entre módulos, por consiguiente si se sustituye un módulo, ninguno o pocos módulos se deben ver afectados por ello. Además la cohesión indica que solo debe realizar una función, así al encontrar una falla se puede identificar rápidamente al módulo defectuoso. Una vez definido el algoritmo debe usarse una representación del algoritmo (FOROUZAN 2003: 200-201).

Existen diversas herramientas de aprendizaje que pueden dar soporte a la etapa de diseño de algoritmos. Entre ellas tenemos:

- Pseudocódigo: "es un lenguaje de especificación (descripción) de algoritmos."(JOYANES 2008: 70) .Se utilizó inicialmente para representar estructuras de control en la programación de este tipo. Además este lenguaje facilita el proceso de codificación y la traducción a un lenguaje de programación. El pseudocódigo consta de palabras clave, para representar palabras reservadas en los diversos lenguajes, escritas en el idioma más usado o sencillo de comprender. La principal ventaja del uso de esta herramienta, es que permite al programador preocuparse más de la lógica del programa que del lenguaje de programación. Además permite modificar errores en el diseño del programa sin tener limitaciones de sintaxis o estructuras del lenguaje utilizado (ídem: 70-71).

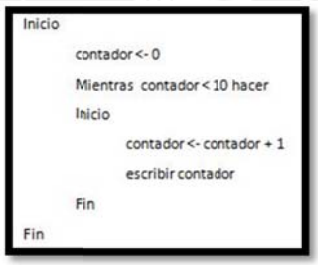


Ilustración 2: Pseudocódigo. Fuente:propia

- Diagrama de flujo: es considerada una herramienta de las más antiguas usadas para representar algoritmos. Se modela mediante el uso de figuras que siguen el estándar ANSI (American National Standars Institute) o ISO (International Standard Organization) que permiten representar las estructuras mediantes elementos gráficos los cuales se unen con otros mediante flechas llamadas líneas de flujo (JOYANES 2008: 71; CAIRO 2003: 4) .

[1] Idea usada en Babilonia 200 AC para ordenar lista de grandes números. Usado también en el algoritmo de Euclides de máximo común divisor y frase usada por el líder romano Julio César.

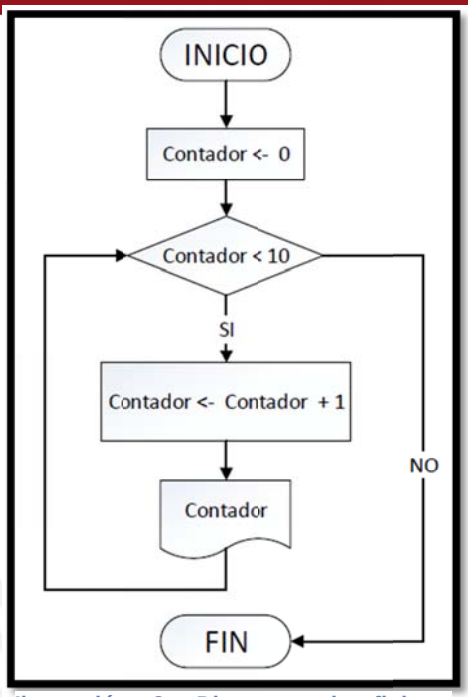


Ilustración 3: Diagrama de flujo . Fuente:propia.

- Diagrama NS: o diagrama de Nassi Schneiderman o de Chapin, su representación es similar a los diagramas de flujo pero las cajas son contiguas y no aparecen las flechas de unión (JOYANES 2008: 80).

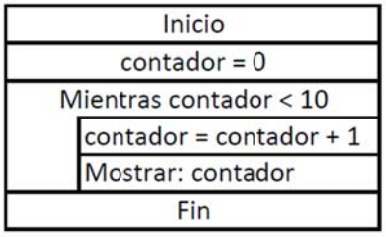


Ilustración 4: Diagrama NS. Fuente:propia

Además un algoritmo es representado haciendo uso de diferentes estructuras:

- Secuenciales: “es aquella en la que una acción (instrucción) sigue a otra en secuencia” (ídem: 128) .De este modo el orden de las tareas indican que la salida de una es la entrada de otra.
- Repetitivas: son instrucciones que se repiten muchas veces, esta secuencia de instrucciones son conocidas como bucles. Estas se repiten una cantidad determinada de veces hasta cumplir la condición de parada (ídem : 158).
- Selectivas: mediantes expresiones lógicas se establecen las condiciones para tomar la decisión de que acción realizar (ídem: 130).

En la etapa de codificación se transforma el algoritmo a código de manera que cumpla la sintaxis y estructura de un lenguaje de programación. Este último es un lenguaje formal diseñado para que los humanos puedan comunicarse con el computador mediante secuencias de operaciones o instrucciones. Los lenguajes de programación

han ido evolucionando a la par con los paradigmas de programación y enfoques para la solución de problemas. A continuación se listan los paradigmas clásicos:

- Paradigma imperativo o procedimental: posee un conjunto de instrucciones las cuales son ejecutadas secuencialmente, excepto si existen instrucciones de salto o control. En este paradigma se definen una secuencia instrucciones donde cada una es una orden para que la computadora realice una tarea indicada. Ejemplos de lenguaje imperativo son C, Pascal, BASIC, etc. (JOYANES 2008: 40)
- Paradigma declarativo: el programador debe describir el problema en lugar de buscar una solución algorítmica. Se hace uso del razonamiento lógico para responder al problema. Algunos lenguajes de programación de este paradigma son Haskell, Lisp, Prolog, etc. (ídem: 41)
- Paradigma orientado a objetos: enlaza los datos con las funciones, permite descomponer el programa en entidades llamadas objetos y construye funciones sobre estos. A continuación algunas características: se centra más en los datos que en los procedimientos, oculta la información dentro del objeto para que no pueda ser accedida por otros objetos, la comunicación entre objetos se da a través de funciones, etc. (BALAGURUSAMY 2008: 6-7)

Existen dificultades relacionadas al aprendizaje de los diversos paradigmas. A continuación se detallan algunas de estas dificultades:

- Paradigma imperativo: no es muy difícil de entender, puesto que es sencillo convertir el algoritmo a código usando este paradigma. Las dificultades que se presentan son el entendimiento de la iteración y recursión. Las más sencillas de entender son asignación y secuencias, ya que la idea de asignar o guardar información es muy usada en el día a día. Sin embargo, usualmente estos conceptos básicos no son completamente entendidos y causan confusiones cuando se trata de incluir iteraciones y recursión. Esta última es considerada la más compleja para entender (VUJOŠEVIĆ 2008: 69- 70).
- Paradigma declarativo: se considera de complejidad alta, debido a que se debe comprender el funcionamiento de la recursividad. Además, programadores principiantes que proponen soluciones sencillas, comúnmente iterativas, encuentran dificultad al representar estas mediante reglas basándose en la recursión (ídem: 75).

- Paradigma orientado a objetos: las dificultades de este paradigma están relacionadas a la identificación de la estructura de datos más adecuada (objetos) y la definición definir el flujo del programa. Además si se empieza con este paradigma se crea una mentalidad orientada a objetos lo cual no desarrolla las habilidades básicas de programación (ídem: 72). También, existe dificultad sobre conceptos de este paradigma, como constructores, destructores, sobrescribir funciones, herencia, etc. (ALA-MUTKA 2012).

CAPÍTULO 3: Estado del arte

En la siguiente sección, tras realizar una revisión empírica para conocer cómo se ha intentado afrontar el problema, se presentan soluciones aproximadas al problema planteado.

La forma más usada y "confiable" para verificar un algoritmo es el seguimiento del mismo a mano. Para esta forma se recomienda tener una estructura sencilla de entender, usualmente una tabla, donde se puedan apreciar las variables y su cambio a lo largo de ejecución de las sentencias. Este método es conocido como *desk-check* debido a que la verificación se hace sin el uso de un computador (ZAK 2011: 18) .Sin embargo, este método algunas veces es obviado y el programador novato solo observa la solución propuesta y debe confiar en la correctitud algoritmo planteado. Dejando a las siguientes etapas de codificación y pruebas, como las indicadas para encontrar errores en el diseño del programa.

1 Productos comerciales

Existen diversos programas que permiten la ejecución, traducción o conversión entre lenguajes de programación, entre ellos tenemos:

Envision es una herramienta de modelado de pseudocódigo y es un complemento de Eclipse, permite diagramar de manera sencilla utilizando drag and drop el pseudocódigo. Permite generar código en diversos lenguajes de alto nivel como: Java, JavaScript, PHP, Action Script, PHP, Haxe. Se resalta la importancia de la resolución de problemas y para lograrlo se usa pseudocódigo para no tener restricciones de algún lenguaje de programación. Posee una versión gratuita que tiene una menor cantidad de lenguajes que la pagada ($50 mensual), y planea tener una versión cloud que permita acceder y modificar el modelado a través de la web.

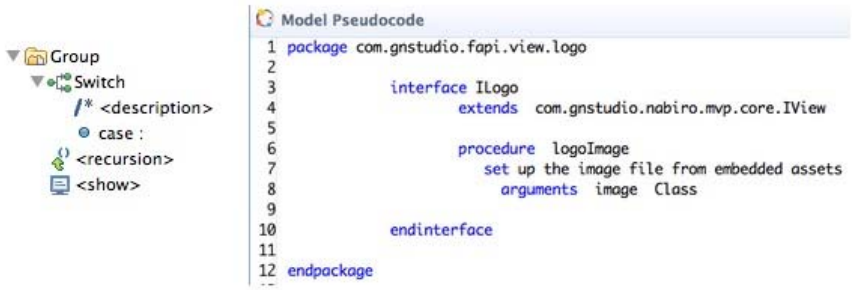


Ilustración 5: Envision. Fuente: http://www.envision-apdt.com/features.

La corporación Tangible Software Solutions posee muchos productos que permiten la conversión de código de diferentes lenguajes de programación en su mayoría perteneciente a la plataforma de .Net, aunque también se incluye Java. Lo cual permite facilitar la tarea del programador al no tener que construir todo el código ya generado en otro lenguaje sino solo traducirlo a uno de su elección.

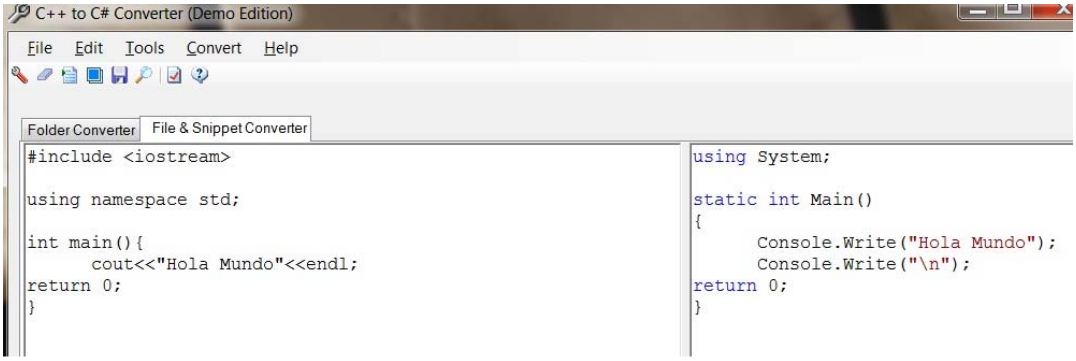


Ilustración 6: C++ to C#.Fuente: http://www.tangiblesoftwaresolutions.com/Demo.htm.

Otro programa que permite traducción de código es HTML to Anycode Convertor, que permite a partir de código escrito en HTML obtener código en JavaScript, ASP, PHP, JSP y Perl, lo cual permitirá ahorrar gran cantidad de tiempo.

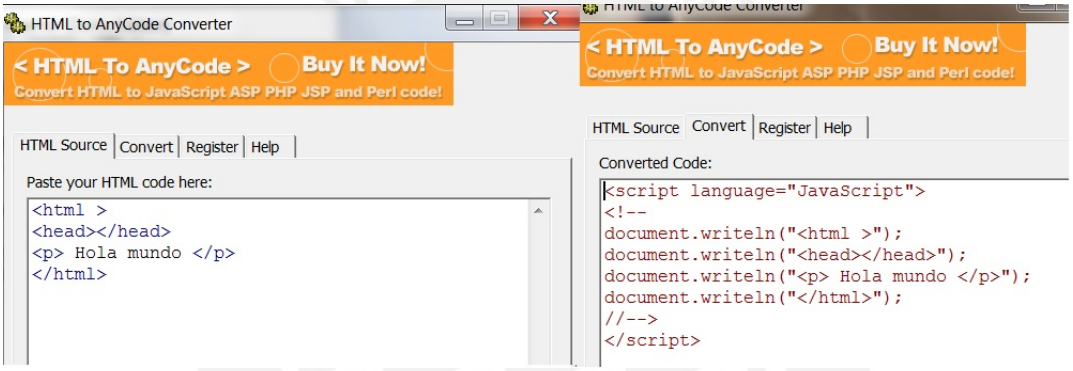


Ilustración 7: HTML to Anycode Convertor.

Fuente http://www.exactcom.com/products/htmltoanycodeconverter/

2 Productos no comerciales

Se puede observar que la enseñanza de lenguajes de programación ha sido fuente para diversas tesis.

Una de ellas es, “Intérprete para un lenguaje de programación orientado a objetos, con mecanismos para optimización y modificación dinámica de código”, en esta tesis se crea un lenguaje llamado Pas++ que hace uso del paradigma orientado a objetos y otorga información sobre las fases del intérprete. También muestra la tabla de símbolos y permite su modificación en tiempo de ejecución, y se implementa la etapa de optimización de código. (GOMEZ 2012)

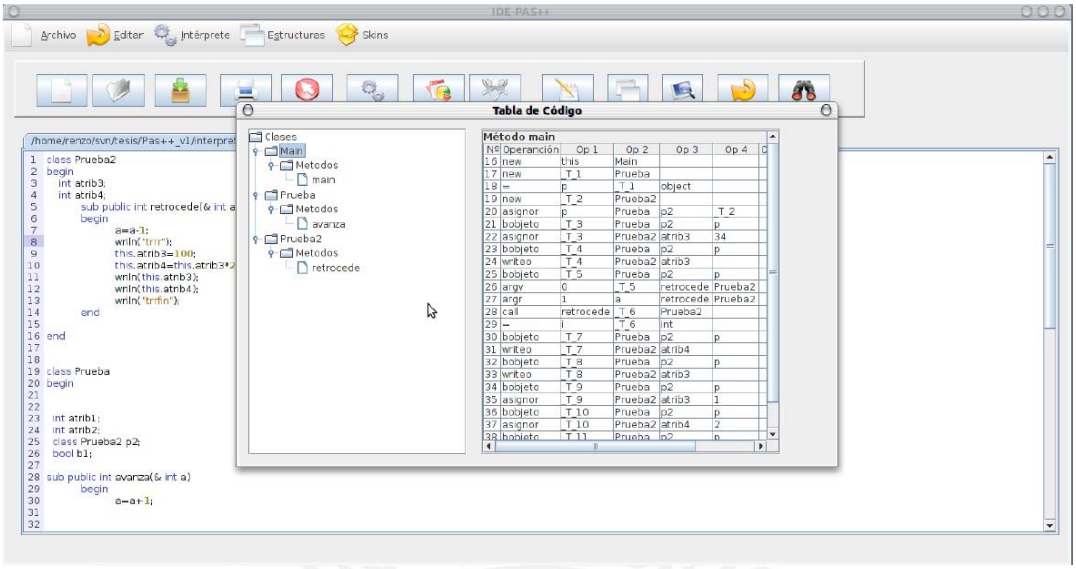


Ilustración 8: Entorno de Pas++ . Fuente: GOMEZ 2012

Otra tesis orientada a nuevos programadores es “Intérprete y entorno de desarrollo para el aprendizaje de lenguajes de programación estructurada”, la cual ofrece una herramienta orientada a estudiantes de pregrado para facilitar el aprendizaje de lenguajes de programación, ofreciendo un lenguaje estructurado en español y un entorno para la ejecución de este (HIRSH 2007).

Ilustración 9: IC Helper. Fuente: HIRSH 2007

Un grupo desarrolló un producto llamado GPC PseudoCode Converter, que convierte pseudocódigo a código en c++, lo cual permite compilar el código obtenido pero después que el usuario realice algunas modificaciones (SCHNEIDERMAN: 2012).

Además, existe una tesis que ofrece un producto llamado PsiCoder que permite la ejecución de pseudocódigo y la traducción código en c++. (PAEZ: 2008)

3 Conclusiones sobre el estado del arte

Característica /Producto	Libre	Conversión de lenguajes	Ejecución de pseudocódigo	Uso de lenguaje propio	Idioma
Envision	Trial 30 días	Si	No	Si	Inglés
Tangible Software Solution	Demo	Si	No	No	Inglés
HTML to Anycode Convertor	Demo	Si	No	No	Inglés
Tesis (GOMEZ 2012)	Si	No	Si	Si	Español
Tesis (HIRSH 2007)	Si	No	Si	Si	Español
GPC	Si	Si	No	No	Inglés
PsiCoder	Si	Si	Si	Si	Español

En conclusión, se puede apreciar que existen diversos productos que intentan dar soporte a la etapa del diseño del algoritmo a través de la ejecución de pseudocódigo, sin embargo; la sintaxis utilizada no es la misma debido a que no existe un estándar establecido. Además, otros productos permiten la traducción de pseudocódigo a lenguajes de programación pero no permiten la ejecución del programa traducido lo cual plantea la necesidad de otra herramienta para la ejecución y validación. De esta manera, lo que se plantea es un entorno de desarrollo que permite la ejecución de pseudocódigo, cuyas palabras reservadas puedan ser modificadas, y la traducción y ejecución de código VBA. Además se podrá realizar la traducción a Java, Ruby, Python y C++.

CAPÍTULO 4: Gramática del pseudocódigo

1 Introducción

En el siguiente capítulo se presentan los pasos necesarios para alcanzar el resultado esperado 1: Gramática definida para el lenguaje de pseudocódigo que será usada en el intérprete de pseudocódigo, usando la notación BNF.

2 Desarrollo de la gramática

Para definir la gramática del pseudocódigo se usó la notación BNF (*Backus-Naur Form*). Como se mencionó en el capítulo 2 esta notación permite representar gramáticas de libre contexto (COOPER 2004: 87). Esta gramática de libre contexto se compone de reglas. Una regla se puede representar de la siguiente manera: V -> w. Donde "V" es un símbolo no terminal, es decir, permite más derivaciones, y w es una cadena de valores terminales, no permiten 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” .

Además existen otros símbolos como "|" que representa alternativas de tomar otro camino en la regla, y valores incluidos en las comillas que representan valores terminales.

<expr> ::= <num> "+" <num>
| < num> "-" <num>

En el ejemplo anterior, se puede apreciar que <expr> se puede convertir en 2 conjuntos, un conjunto que representa una suma (<num> "+" <num>) y otro la resta (<num> "-" <num>).

Para el pseudocódigo se presenta una gramática con estructura similar a un lenguaje del paradigma imperativo, el cual como se presentó en el capítulo 2, posee una estructura sencilla de comprender. A continuación se muestra una regla de la gramática del AnexoB:

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

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:

PROGRAMA
INICIO

FIN

Donde los símbolos como <program> <enter> <inicio><fin>, generan como única regla los símbolos terminales: "PROGRAMA", "\n", "INICIO" y "FIN", respectivamente. Además existe el símbolo no terminal <listaInstr> que en este caso toma el valor de su segunda alternativa que es el cambio de línea o <enter>. De esta manera, recorriendo la gramática mediante recursión se pueden generar sentencias más elaboradas, como estructuras iterativas, selectivas, condicionales, operaciones aritméticas, etc. Además la mayoría de las palabras y caracteres reservados del pseudocódigo (referido a los

símbolos terminales) podrán ser modificados a través de un archivo con formato xml, lo cual permitirá modificar, sin poder cambiar la estructura del lenguaje, el valor final de los símbolos terminales.

Las reglas de la gramática generada para el pseudocódigo, serán utilizadas en el análisis sintáctico del intérprete para la definición de la estructura y sintaxis del mismo. Las gramáticas pueden poseer errores de ambigüedad, cuando existen muchos caminos iguales a partir de un símbolo no terminal, o símbolos inaccesibles, símbolos terminales o no terminales los cuales no pueden ser alcanzados a partir de ningún otro símbolo. La verificación de la gramática de pseudocódigo se podrá verificar con la correcta ejecución del intérprete, específicamente, el correcto desempeño del analizador sintáctico.

CAPÍTULO 5: Intérprete y traductor de pseudocódigo

1. Introducción

En el siguiente capítulo se presentarán los pasos necesarios para alcanzar los resultados esperados 2 y 3, los cuales se refieren a la creación del intérprete de pseudocódigo y el traductor de pseudocódigo a código VBA, Python, Ruby, Java y C++.

2. Desarrollo del intérprete y traductor

Mediante el siguiente diagrama se puede apreciar la estructura del intérprete a desarrollar:

Para la etapa de análisis léxico se recibe como entrada el programa escrito en pseudocódigo y se genera una lista de tokens. Para obtener la lista de tokens, se creó una función en Java que recorría el programa escrito en pseudocódigo, mientras se recorría carácter por carácter se analizaba cada conjunto leído para identificar si era un grupo con significado, también llamado lexema, y que tipo de token era y se agregaba a la lista de tokens. Como se mencionó en el capítulo anterior, el pseudocódigo posee una gramática la cual define que elementos son los terminales,

además la mayor parte de estos símbolos terminales se encuentran en un archivo con formato xml, por lo cual el valor del símbolo terminal puede cambiar. Esto permite tener un pseudocódigo para el idioma que se desee siempre y cuando se le otorgue un valor a todos los nodos del archivo xml. Por ejemplo:

<IF> Si </IF> (para pseudocódigo en español)
<IF> If </IF> (para pseudocódigo en inglés)

Estos tokens son palabras reservadas para nuestro lenguaje, además existen otros símbolos reservados los cuales, en este caso, no pueden variar como son los relacionados a operaciones aritméticas (+ , - , * , /) , asignaciones (:=) , comparaciones(>, < , == , <> , >= , <=), etc. La lista de tokens se almacena en un ArrayList que contiene una lista de objetos de la clase Token, la cual contiene información del lexema (cadena de token), identificador del token, línea en cual fue encontrado, tipo en caso sea una variable, número o cadena.

Para el análisis sintáctico se recibe como entrada una lista de tokens y se genera tablas de símbolos y de códigos. Los símbolos representan variables, números constantes, nombres de funciones, procedimientos y cadenas, mientras que los códigos son las acciones que se realizan sobre los símbolos como suma, resta, comparaciones, etc. La gramática definida para el pseudocódigo, detallada en el Anexo B, nos muestra las reglas de sintaxis que posee este lenguaje. Algunas de las reglas definidas poseen llamadas a otras reglas y tokens, para identificar que regla se debe seguir de todas las existentes se hace uso de los tokens generados en la etapa de análisis léxico. Mientras se recorre la lista de tokens se extrae cada token para verificar si corresponde a una parte de la regla y así sucesivamente hasta comprobar que sigue la regla totalmente. En este punto, cuando se logrna identificar las reglas utilizadas se transforman los tokens a símbolos. Los símbolos ofrecen una mejor representación de los tokens obtenidos, por ejemplo para la siguiente sentencia:

4 + 3 * 4

Se tienen: 5 tokens o hasta 6 si se toma en cuenta el cambio de línea. Estos tokens serían: 4 , + , 3 , * , 4. Se puede ver que el lexema 4 se repite ; sin embargo, para el caso de símbolos solo existiría un símbolo con el valor de 4. Esta transformación aunque no parezca importante en el caso de lexemas, cuyos valores no cambiarán a lo largo del programa, es decir, el valor de 4 siempre será 4 en todo el programa, es muy importante para el caso de las variables. Supongamos, que tenemos una variable llamada “longitud” en un programa, una acción común sobre esta variable sería la asignación de un valor “ longitud := 4 “, otra acción podría ser una operación en la cual se haga uso de la variable. Para este ejemplo planteado se necesita un solo elemento “longitud” para asignarle un valor y obtener ese valor de la misma variable, es por eso que se realiza una transformación a un símbolo para volver a usar el valor asignado o modificado. Un token no sólo permite generar símbolos, sino también genera códigos, como se mencionó anteriormente en este capítulo y en capítulo anterior, existen palabras reservadas definidas para el pseudocódigo que para este caso forman parte de la lista de tokens. Al igual que con los símbolos, al recorrer la lista de tokens se extraen e identifica si son códigos. Un código, para este caso, es una acción que se ejecuta sobre símbolos, es decir, define acciones a realizarse que están ligadas a la estructura del lenguaje. Por ejemplo, analicemos el siguiente fragmento de programa en pseudocódigo:

```
(1)   Si 4 < 5 entonces
(2)       Escribir “4”
(3)   Sino
```

(4) Escribir “5”
(5) Fin Si

A simple vista nos podemos dar cuenta de algunas acciones, primero una comparación específicamente un “menor que” (1), también 2 acciones de impresión “Escribir” (4)(5) , sin embargo, existen otras acciones que están relacionadas a la estructura condicional , para este caso estas acciones son “saltar” y “saltar en falso”. Para cualquier comparación existen solo 2 respuestas verdadero y falso, entonces si la condición evaluada es verdadera se ejecuta la línea 2, en cambio, si la condición es falsa se ejecuta la línea 4. Además si la condición era verdadera no se debe ejecutar la línea 4, sino se debe saltar al final de la condicional (5). Entonces es necesario crear una acción “saltar” que al ejecutar la línea 2 realice un salto al final de la estructura (5), además en caso que sea falso no se debe ejecutar la línea 2 entonces se debe realizar una acción “saltar en falso” para saltar hacia la línea 4 en caso que la condición no se cumpla. Mediante estas acciones explícitas o requeridas por la naturaleza de las estructuras se va llenando la tabla de código.

Para el manejo de las funciones y procedimientos debe existir una división entre los símbolos y códigos, porque se puede invocar a un procedimiento muchas veces y los valores deben ser independientes en cada invocación. Para ello se crea una tabla de símbolo y código para cada definición de procedimiento, función y para cada invocación, es decir, cada vez que se invoque a un procedimiento o función se tomará como base su tabla de códigos y símbolos y se creará una copia para crear independencia entre cada llamada.

Al finalizar esta etapa se obtienen lista de códigos y símbolos del programa en pseudocódigo. Al recorrer la tabla de código se puede ejecutar todo el programa, sin embargo, pueden existir errores de tipos los cuales se pueden identificar antes de la ejecución mediante el análisis semántico.

En el análisis semántico se realiza una pseudoejecución del programa, es decir, el programa se ejecuta pero no se actualizan valores de los símbolos sino sólo se actualizan y verifican los tipos. De esta forma, al evaluar las acciones de la tabla de código, por ejemplo, una multiplicación 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 ese análisis se pueden identificar errores de tipo antes de ser necesaria la ejecución; no obstante, algunos tipos no pueden ser totalmente detectados, como son las variables cuyo valor se obtiene a partir de una lectura, cuyo tipo no puede ser identificado sino hasta el momento de la ejecución.

Finalmente se realiza la ejecución del código, para este caso es necesario recorrer la tabla de código y realizar las acciones registradas como operaciones aritméticas, trigonométricas, de cadenas, saltos a otras tabla de código y símbolo (en el caso de llamada a procedimientos o funciones). De esta manera recorriendo la tabla de código y ejecutando cada acción se realiza la ejecución final del programa escrito en pseudocódigo.

Para construir el traductor de pseudocódigo se realiza, de igual forma que con la ejecución, un recorrido a las tablas de código y símbolos, que fueron creadas en las etapas anteriores. Sin embargo, se deben agregar algunas acciones extras para abarcar mayor posibilidad de lenguajes a los cuales se pueda traducir. Por ejemplo, en el lenguaje Java se debe realizar algunas acciones distintas al lenguaje VBA como es el caso de los Imports que son necesarios en Java para realizar algunas operaciones, es por este motivo que se debe agregar a la tabla de código una acción referida a la

importación de librerías. Como este, existen muchos casos especiales referidos a sintaxis o reglas propias de los lenguajes a los cuales se desea traducir el pseudocódigo. Por ejemplo, en VBA para retornar una función se debe asignar el valor de retorno al nombre de la función, mientras que en Java sólo se necesita usar la palabra reservada *return* para retornar un valor. Igualmente existen diferencias con lenguajes como Python y Ruby donde no existe una función principal y tampoco tipos de datos o en C++ donde cada función, si se desea declarar luego del *main* se debe declarar su prototipo, se necesitan includes, entre otros. Luego de agregar todas las acciones se recorre las tablas y se busca equivalencia de cada acción en el archivo XML, que contiene la sintaxis del lenguaje al cual se traducirá el pseudocódigo. Debido a las diferencias entre lenguajes se diseñó un archivo XML que tenga un formato el cual al modificar los valores de los nodos permita la traducción al nuevo lenguaje ingresado. Para el presente proyecto se tiene archivos XML para la traducción a lenguajes VBA, Python, Ruby, Java y C++. En el Anexo C se muestra la estructura del XML para el lenguaje VBA.

3. Pruebas

Se realizaron pruebas del intérprete y traductor resolviendo una pregunta del la tercera práctica y una del primer examén del curso Introducción a la Computación del ciclo 2013-2. Estas se encuentran en detalle en el anexo D.

CAPÍTULO 6: Entorno de Desarrollo

1. Introducción

En el siguiente capítulo se presenta información necesaria para alcanzar los resultados esperados 4 y 5 referidos al entorno de desarrollo que permita ejecutar y traducir programas en pseudocódigo.

2. Desarrollo

Para la implementación del intérprete y del IDE se utilizó la metodología de cascada incrementar iterativa. Esto se puede evidenciar en el plan de actividades en las presentaciones de diferentes funcionalidades mientras estas se iban incrementando.

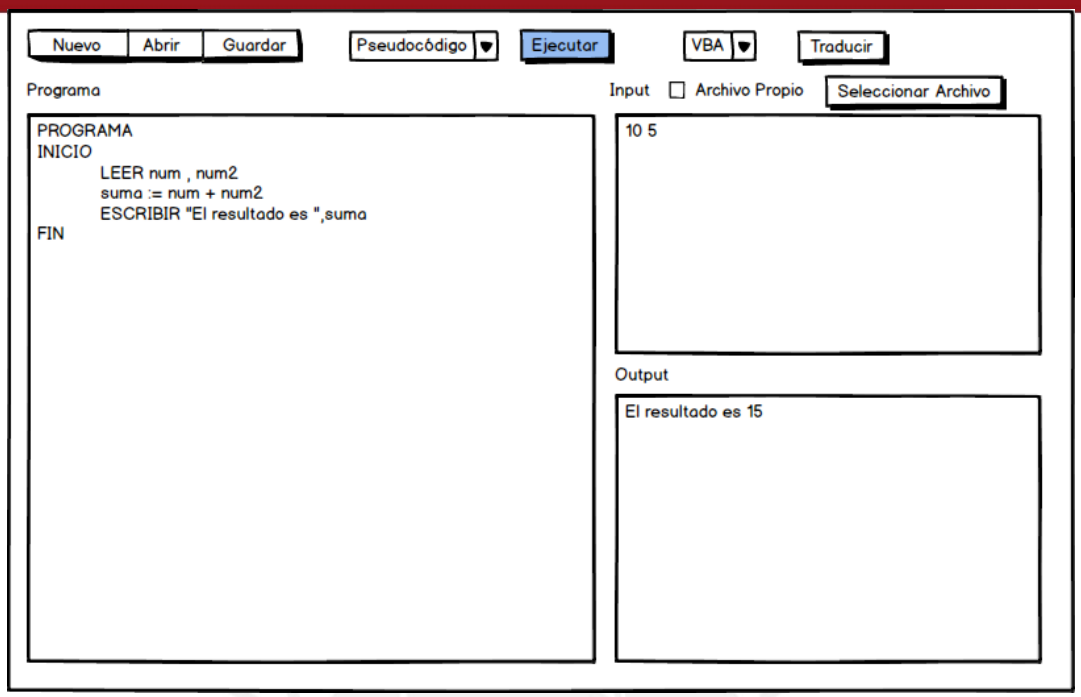
Para el entorno de desarrollo se definieron ciertas funcionalidades que se listan a continuación:

- El entorno debe permitir ejecutar programas en pseudocódigo.
- El entorno debe permitir ingresar datos en caso que el programa requiera una lectura de estos.
- El entorno debe mostrar los resultados de la ejecución de los programas.
- El entorno debe permitir ejecutar programas en VBA.
- El entorno debe permitir traducir programas en pseudocódigo a código VBA.
- El entorno debe permitir guardar el programa creado.
- El entorno debe permitir abrir un programa anterior.
- El entorno debe permitir crear un nuevo programa.
- El entorno debe permitir traducir programas en pseudocódigo a código en lenguaje Java.

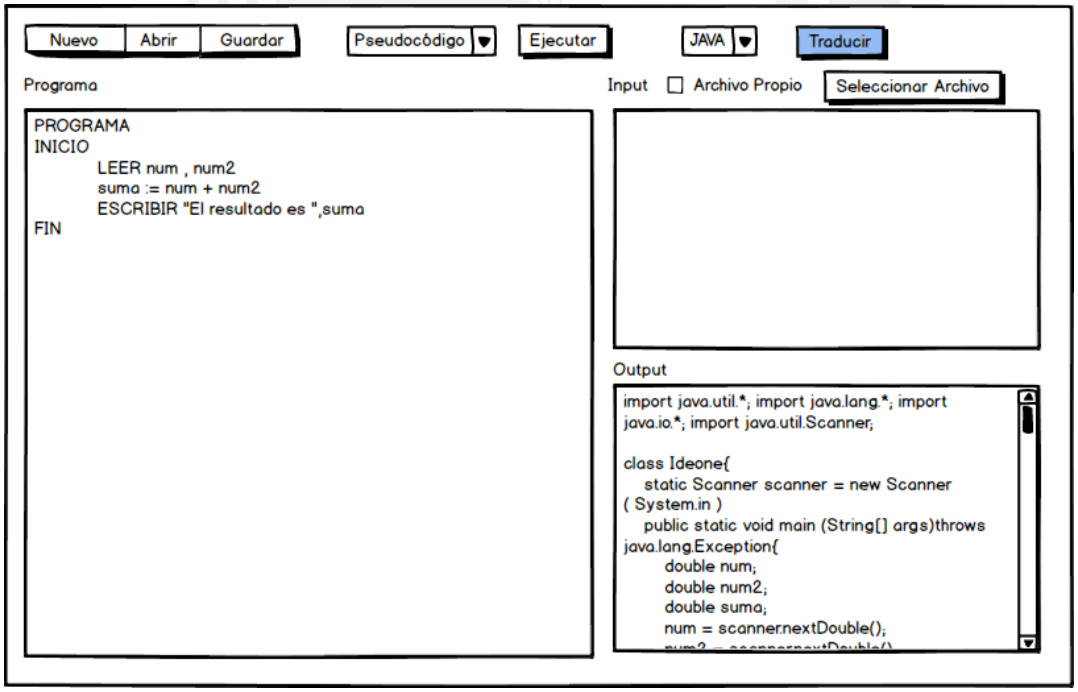
Se decidió usar tecnología .NET para realizar el entorno de desarrollo por la facilidad ingresar los programas ejecutados en VBA a EXCEL para su ejecución. En el caso de pseudocódigo se identificó una forma de integración con el entorno por medio de un archivo .bat que invocaría al intérprete a través de la línea de comandos. Además este archivo .bat recibirá parámetros que indican que acción se debe realizar con el pseudocódigo, las cuales son ejecución o traducción. Para el caso de traducción se le envía un parámetro más que es el nombre del lenguaje al cual se traducirá el programa.

Como se mencionó en el capítulo anterior, el lenguaje al cual se traducirá posee un archivo XML que posee un grupo de nodos, donde cada nodo contiene la equivalencia de sintaxis respecto al lenguaje escogido, de esta forma se puede traducir , para este caso , el pseudocódigo a Java o VBA.

A continuación se muestran los prototipos del entorno:



En este primer prototipo se puede apreciar el programa en pseudocódigo en la izquierda, así como la entrada y salida de datos en la parte derecha. Para ejecutar el programa se debe seleccionar del combo anterior el lenguaje actual y luego se debe seleccionar la opción de Ejecutar.



Cuando se desee traducir el pseudocódigo se debe seleccionar el lenguaje de programación a traducir, en este ejemplo JAVA, y luego seleccionar la opción traducir que permitirá traducir el programa y mostrarlo en el panel de salida.

3. Pruebas

Las pruebas realizadas sobre el entorno de desarrollo fueron las mismas que se realizar con el traductor e intérprete que se encuentran detallados en el Anexo D.

CAPÍTULO 7: Conclusiones, recomendaciones y trabajos futuros

En este último capítulo se dan a conocer conclusiones y trabajos futuros que surgieron al desarrollar el presente proyecto:

1. Conclusiones:

Se obtuvieron las siguientes conclusiones:

- Para manejar los tipos de datos del intérprete es bueno tener tipos de datos genéricos como *Double* o *String*, porque permite flexibilidad en las operaciones cuando aún no se conoce el tipo de dato. Además, al ejecutar se puede obtener el valor del tipo requerido dependiendo del tipo establecido en el análisis semántico.

- Es importante tener una buena representación de código intermedio (tabla de código y símbolos) esto permite un sencillo recorrido para la ejecución, y además permite, dependiendo de los tokens añadidos, la traducción a diferentes lenguajes de programación del pseudocódigo.

- Es necesario inicializar algunas variables cuando se requiera un tipo de dato específico y no se pueda inferir con el análisis semántico, por ejemplo, a partir de una lectura de un dato al cual no se le aplica ninguna operación y se requiere que tenga un tipo específico es necesario inicializarlo con un valor del tipo requerido.

- Si se mantiene independencia entre los elementos que se desarrollan se pueden integran diferentes tecnologías sin aparentemente mucha dificultad, como es el caso del intérprete desarrollado en Java que es invocado por un entorno de desarrollo en C#.

- Realizar entregas continuas es muy importante, en referencia al plan de actividades y al método cascada incremental, ya que permite realizar pruebas para cada nueva funcionalidad desarrollada. Esto ayudó mucho en el presente proyecto ya que el código del intérprete crecía rápidamente y de haber postergado las pruebas al final, se hubiera invertido mucho tiempo corrigiendo algunos errores que se identificaron y corrigieron antes de cada entrega.

- Para el manejo de variables y el flujo de procedimientos y funciones se utilizó una tabla de código y símbolos diferentes. Esto no permite el paso de parámetros por referencia, lo cual no fue implementado para mantener una sintaxis sencilla en el pseudocódigo y para tener la capacidad de traducir a lenguajes como Java donde los datos de tipo primitivo no pueden pasar por referencia.

- No es indispensable, pero es bueno manejar los lexemas o palabras reservadas en un archivo aparte para permitir que los valores cambien sin necesidad de tener que modificar el código del intérprete directamente. De igual forma con las equivalencias para la traducción de código.

2. Recomendaciones y trabajos futuros

A continuación se plantean funcionalidades que no pudieron ser implementadas en el presente proyecto debido al limitado tiempo para el desarrollo:

- El lenguaje desarrollado para el pseudocódigo sólo maneja datos básicos pero estos no necesitan ser declarados. Debido a esto un posible trabajo futuro sería incluir la capacidad de definir estructuras de dato, lo cual aumentaría la dificultad al momento de la traducción y sería necesario definir estos nuevos tipos de datos para evitar dificultades al detectar e identificar los tipos de datos.

- El lenguaje actualmente soporta la llamada a procedimientos y funciones; sin embargo, no soporta llamadas recursivas. Sería interesante agregar esta funcionalidad, pero se tendría que tomar en cuenta la capacidad de memoria disponible para realizar estas llamadas.

- El entorno de desarrollo actual permite ejecutar pseudocódigo y código VBA, sería interesante en un posible trabajo futuro aumentar la funcionalidad de asociar compiladores con el entorno para que se permita la ejecución de otros lenguajes de programación.

- Se podría aprovechar la capacidad de traducción de código, debido a que se manejan tabulaciones al traducir para una mejor visualización, para a partir del pseudocódigo poder diseñar el diagrama de flujo. Esto vendría a ser una traducción de acciones a figuras y de tabulaciones a espaciado en el diagrama.

- Sería posible extender la traducción de la herramienta al lenguaje C# teniendo en cuenta las limitaciones en la lectura de datos, debido a que en C# no se puede leer más de 2 valores en una misma línea con una sola sentencia. En este caso se recomendaría leer por cada línea un solo valor.

- Finalmente, un posible trabajo futuro podría ser implementar la etapa de optimización de código para acelerar la ejecución del código de programas en pseudocódigo.

Referencias bibliográficas

AHO, Alfred V.
2007 Compilers: principles, techniques, & tools. Segunda edición. California: Addison Wesley.

ALA-MUTKA, Kirsti
2012 Problems in learning and teaching programming. *Codewitz Needs Analysis.*

BALAGURUSAMY, E.
2008 Object Oriented Programming With C++. Cuarta edición. New Delhi: McGraw-Hill.

CAIRO, Osvaldo
2003 Metodología de la programación. Segunda edición. México, D.F.: Alfaomega.

CHUMPITAZ, Lucrecia
2005 Informática aplicada a los procesos de enseñanza-aprendizaje. Primera edición. Lima: CISE.

COOPER, Keith D. y Linda TORCZON
2004 Engineering a compiler. Segunda edición. Estados Unidos: Elsevier.

DALE, Nell B.
2007 Programación y resolución de problemas con C++ / Nell Dale, Chip Weems. México, D.F.: McGraw-Hill.

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

Diccionario de la Real Academia Española
2012 "Algoritmo". Madrid. Consulta: 21 de septiembre de 2012.
< http://lema.rae.es/drae/?val=algoritmo>

DICTIONARIST
2012 "Bug". Consulta: 21 de septiembre de 2012.
< http://definicion.dictionarist.com/bug>

ESLAVA Muñoz, Vicente Javier
2012 Aprendiendo a programar paso a paso con C. España: Bubok. USA: Cengage Learning.

FARREL Joyce
2012 Programming Logic and Design, Comprehensive. Séptima edición.

FOROUZAN, Behrouz A.
2003 Introducción a la ciencia de la computación: de la manipulación de datos a la teoría de la computación. México, D.F.:Thompsom

GARNER, Stuart
2003 "Learning Resources and Tools to Aid Novices Learn Programming". *Informando ciencia InSITE- "Where Parallels Intersect".*Australia, 2003, pp. 213-222.

GODSE, D.A. y A.P. GODSE

2008 Fundamentals of Programming. Primera edición. Pune: Technical Publications.

GOMES, Anabela, A. J. MENDEZ
2007 "Learning to program – difficulties and solutions". Conferencia Internacional sobre la Enseñanza de la Ingeniería. Portugal.

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.

GRANDELL, Linda y otros
2006 "Why Complicate Things? Introducing Programming in High School Using Python". *ACE '06 Actas de la 8ª Conferencia Australiana sobre la Educación Informática*. Australia, 2006, volumen 52, pp. 71-78.

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.

LEE, Kent D.
2008 Programming Languages: An Active Learning Approach. Primera edición. Iowa: Springer.

LOUDEN, Kenneth C.
2004 Construcción de compiladores: Principios y práctica. México: Thomson.

MARRER, Gary
2009 Fundamentals of Programming: With Object Orientated Programming.

MASSA Stella, Carlos RICO y Raquel HUAYPAYA
2012 "Generación de requerimientos de un objeto de aprendizaje a partir de escenarios: un caso de estudio para un curso de programación inicial". Ponencia presentada en el XVIII Congreso Argentino de Ciencias de la Computación. Consulta: 27 de agosto de 2013.
< http://sedici.unlp.edu.ar/handle/10915/23650>

NORTON, Peter
2006 Introducción a la computación. Sexta Edición. México, DF: McGraw-Hill

PAEZ Pérez, Liliam y Jhon Henry VASQUEZ
2008 PsiCoder: Software Educativo para Facilitar el Proceso de Enseñanza – Aprendizaje en un Curso Básico de Programación. Tesis de grado para optar por el título de Ingeniero de Sistemas. Bógota: Pontificia Universidad Javeriana.

PROJECT MANAGEMENT INSTITUTE (PMI)

2008 Project Management Body Of Knowledge (PMBOK). Cuarta Edición. Pensilvania.

SCHNEIDERMAN , JOHN
2012 " PseudoCode Converter ". Consulta: 23 septiembre 2012.
< http://www.ohloh.net/p/pseudocode >

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
2004 Ingeniería de software orientada a objetos con UML, Java e Internet. Primera edición. México: Thomson.

ZAK , Diane
2011 Clearly Visual Basic: Programming with Microsoft Visual Basic 2010. Segunda edición. Boston: Cengage Learning.