Compilador

Entrega Final

Integrantes

-Barrera Rangel Guillermo

-Fernández Mora José Enrique

-Guerrero López Enrique

-López Soto Miguel Ángel

-Luna Fierros Leonardo

1.- Introducción

Se sabe que a lo largo del tiempo la tecnología va evolucionando pero en el caso de los compiladores, se ha estado evolucionando e innovando en este rubro, muy muy lento ya que considerando que el primer compilador fue usado en 1952 y en el 2003 fue el año donde el compilador fue llevado al grado de poder optimizarlo y usarlo de manera modular, esto fue un avance de suma importancia para el tema de compiladores, pero una pregunta que debemos hacernos es ¿porque tuvieron que pasar 51 años para poder seguir innovando y avanzando en este tema?.

Algo de historia sobre compiladores

- Corrado Böhm First compiler description 1951
- Alick Glennie First implemented compiler 1952
- Grace Hopper First use of the "Compiler" word 1952
- John W. Backus First commercial compiler 1957
- Chris Lattner First compiler toolkit (LLVM) 2003

Un compilador es un programa que transforma el código fuente escrito por el desarrollador en un lenguaje de programación de alto nivel en su expresión equivalente en lenguaje máquina, que puede ser interpretado por el procesador. El proceso de convertir el código fuente a su representación en código de objeto es conoció como compilación.

Un compilador ejecuta cuatro funciones principales:

Scaning: El escáner lee un elemento a la vez del código fuente y mantiene presente que elementos existen por línea de código.

Análisis Léxico: El compilador convierte la secuencia de elementos o caracteres que aparecen en el código fuente a una serie de cadenas de caracteres conocidos como "tokens" que son asociados a un a regla especifica por el programa llamada analizador léxico. Una tabla de símbolos o estructura de datos análoga es usada por el analizador léxico para guardar las cadenas del código fuente que corresponden con el token generado.

Análisis Sintáctico: En este paso, el análisis de la sintaxis es realizado, esto conlleva reprocesar la información capturada para verificar si los tokens creados durante el análisis léxico están propiamente ordenados para su uso. El orden correcto de un conjunto de palabras clave o "keywords" para obtener el resultado deseado se conoce como sintaxis. El compilador tiene que examinar que el código fuente cumpla con la sintaxis esperada.

Análisis Semántico: Este paso esta compuesto de varios procesos intermedios. Primero la estructura de tokens es verificada, así como el orden de estos de acuerdo con la gramática especifica de cada lenguaje. El significado de la estructura de tokens es interpretado por el

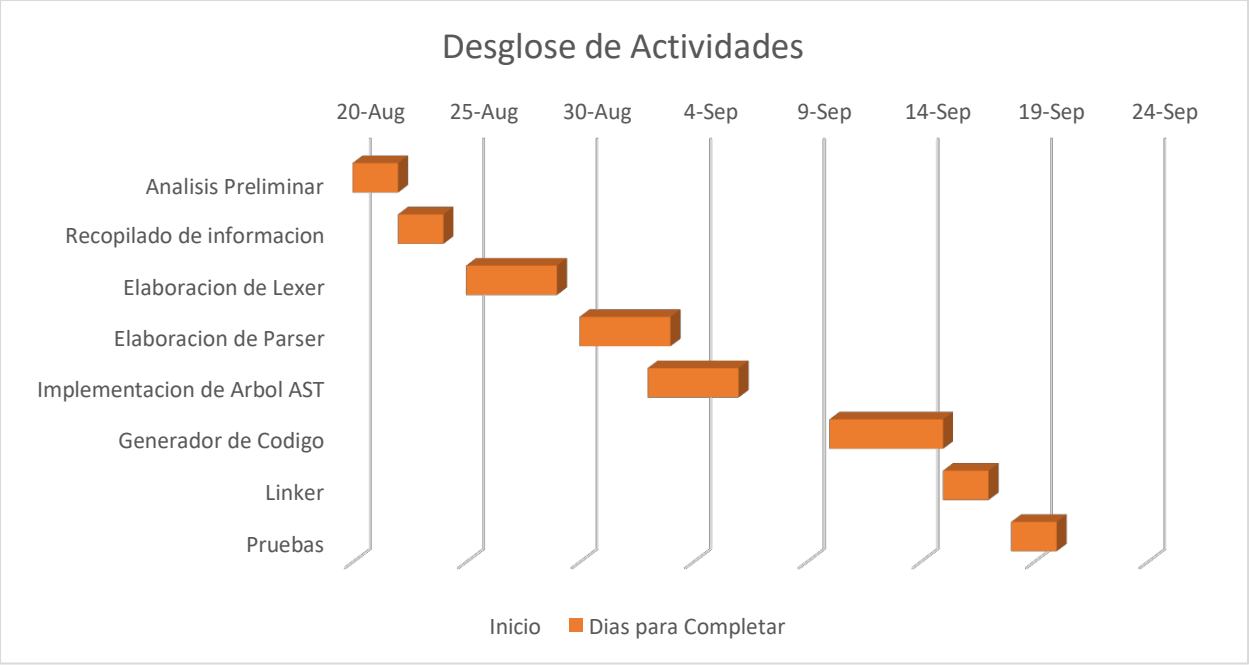
parser y el analizador para finalmente generar código intermedio, llamado también como código de objeto. El código de objeto incluye instrucciones que representan comandos interpretables por el procesador

2.- Plan de proyecto

La elaboración del proyecto fue coordinada de manera exitosa a lo largo de 25 días, plazo necesario para cumplir con todas las tareas necesarias para cumplir con los requerimientos planteados. Se sostuvieron sesiones semanales en las que se discutió el avance de la elaboración de los entregables, además cada integrante de equipo compartió el esquema de actividades que seguiría en los días posteriores.

La fase de análisis preliminar y recopilado de información corresponden a las actividades necesarias para familiarizar a los integrantes con conceptos fundamentales de funcionamiento de compiladores, programación funcional, y sintaxis básica del lenguaje de programación Elixir.

Las secciones posteriores en el diagrama de Gantt obedecen a los pasos y metodologías expresadas en el tutorial de la programadora Nora Sandler.



2.1 High-Level Project Description

El proyecto va a consistir en la implementación de un compilador del lenguaje de programación C, el cual va a a poder compilar el siguiente código:

```
int main(){
   return (3+4 <= 4 || 1&&2 != 3>-6); }
```

2.2 Key Deliverables

Los entregables están asociados a las características del compilador. En cada fecha de entrega se presentará una nueva funcionalidad del compilador.

13 de septiembre de 2019 - Enteros
11 de octubre de 2019 - Operadores unarios
8 de noviembre de 2019 - Operadores binarios
29 de noviembre de 2019 - Aún más operadores binarios

2.3 High-Level Requirements

El compilador será del lenguaje de programación C.
El desarrollo se llevará a cabo en el lenguaje de programación Elixir.
Con la implementación realizada se podrá generar el código en ensamblador, además de un binario ejecutable en un ambiente POSIX-Unix.
Para desarrollar el proyecto, se contempla una arquitectura de 64 bits.

2.4 Strategy and Implementation

Senior Developer - Miguel Ángel López Soto
Tester - Enrique Guerrero López
Project Manager - Guillermo Barrera Rangel
System Integrator - Leonardo Luna Fierros System
Architect - Jose Enrique Fernandez Mora

2.5 General Activities

A continuación se plantean las actividades generales que se llevarán a cabo para el desarrollo del proyecto:

- Se tendrán juntas de trabajo 3 veces por semana donde se discutirá el avance del proyecto de cada uno de los integrantes del equipo, se discutirá el plan de trabajo para la siguiente semana.

- Se verificará que el avance individual de los integrantes esté acorde con lo planteado en el cronograma general para cada entrega.

- Se hará una revisión de todos los cambios realizados al repositorio del equipo.

- Se verificará la documentación pertinente al rol de cada integrante de equipo.

- Se discutirán dudas y sugerencias acerca de las tareas asignadas.

- Se discutirá el avance del proyecto con el experto de dominio y cliente.

2.6 Work Environment

- Como lenguaje de desarrollo se utilizará el lenguaje de programación Elixir V1.9 junto con Erlang 20.0

- Para el desarrollo del proyecto se utilizará sistema operativo Ubuntu 18.04 LTS.

- Como editor de código se utilizará Visual Studio Code con extension de vscode-elixir para la facilidad de lectura de la sintaxis.

- Para complementar el desarrollo del proyecto se hará uso del toolkit LLVM el cual será de ayuda para realizar el generador de código.

- Para el control de versiones se hará uso de git junto con su repositorio.

2.7 Project Organization

Por requerimiento cliente la organización de los entregables será sobrellevada usando un repositorio en el sitio web https://github.com / donde se tendrán dos carpetas principales de trabajo, cada una destinada a un módulo del compilador (Front End y Back End).

2.8 Activities Schedule

Requerimiento	Descripción Breve	Fecha de Entrega	Responsables	Horas Programación Estimadas	Horas Prueba Estimadas	Horas Documentación Estimadas
1	Desarrollar un lector de archivos con terminación .c	30/08/2019	System Architect System Integrator Project Manager Developer Tester	4	2	2
2	Desarrollar un lexer que pueda tokenizar la expresión.	30/08/2019	System Architect System Integrator Project Manager Developer Tester	20	8	2
3	Desarrollar un parser que pueda analizar la gramática formal asociada al lenguaje y realizar los árboles de síntesis abstracta.	6/09/2019	System Architect System Integrator Project Manager Developer Tester	25	8	2
4	Desarrollar en generador de código acorde con la arquitectura y SO planteados por el cliente	12/09/2019	System Architect System Integrator Project Manager Developer Tester	20	8	2
5	Implementar las optimizaciones necesarias y suite de pruebas del código para producción.	12/09/2019	System Architect Project Manager System Integrator Developer Tester	20	10	2

Compiler					
Segunda Entrega					
Task id	Task	Details	Assigned	Starts	**Due**
6	Terminar el Generador de Código		Architect	Oct 17, 2019	**Oct 17, 2019**
7	Realizar los test para el Parser y Generado de Código	Agregar las nuevas pruebas	Tester		**Oct 17, 2019**
8	Verificar el funcionamiento.	Verificar lo hecho en la entrega 1 y hacer lo que el compilador debe hacer para la entrega 2	Integrator	Oct 18, 2019	**Oct 18, 2019**

9	Actualizar documentación		Project Manager, Tester	Oct 18, 2019	**Oct 18, 2019**
Entrega 03					
Task id	Task	Details	Assigned	Starts	**Due**
10	Investigar los Requerimientos		Tester, Architect	Oct 21, 2019	**Oct 23, 2019**
11	Manejo de Errores	Tener el manejo de errores por cada fase de compilación e indicar la línea donde se produjo el error	Project Manager, Tester, Integrator	Oct 21, 2019	**Oct 24, 2019**
12	**Modificar Lexer**		**Integrator**	**Oct 27, 2019**	**Oct 29, 2019**
13	**Modificar Parser**		**Project Manager, Architect**	**Oct 30, 2019**	**Nov 01, 2019**
14	Modificar Generador de Código.		Tester, Architect	Nov 03, 2019	**Nov 05, 2019**
15	Actualizar la documentación	Tener una documentación legible y entendible.	Project Manager, Integrator	Nov 03, 2019	**Nov 06, 2019**

16	Hacer las pruebas para la Entrega 03	Implementar al compilador las nuevas pruebas de Nora.	Tester	Nov 05, 2019	**Nov 07, 2019**
17	Verificar y Corregir	Verificar con respecto a las pruebas.	Project Manager, Tester	Nov 06, 2019	**Nov 08, 2019**
18	Checar los Commits	Checar que los commits tengan etiqueta, para poder llevar un mejor control.	Integrator	Nov 04, 2019	**Nov 08, 2019**
Entrega 04					
Task id	Task	Details	Assigned	Starts	**Due**
19	Checar integridad del Proyecto	Afinar detalles para prepáranos para la entrega final	Project Manager, Tester, Integrator, Architect	Nov 11, 2019	**Nov 12, 2019**
20	**Nuevos Requerimientos**	**Tener lo necesario para la entrega final**	**Architect**	**Nov 13, 2019**	**Nov 14, 2019**
		Corregir errores en caso de haber hecho algo mal o hacer mejoras que el profesor indique.	**Project Manager, Tester**		

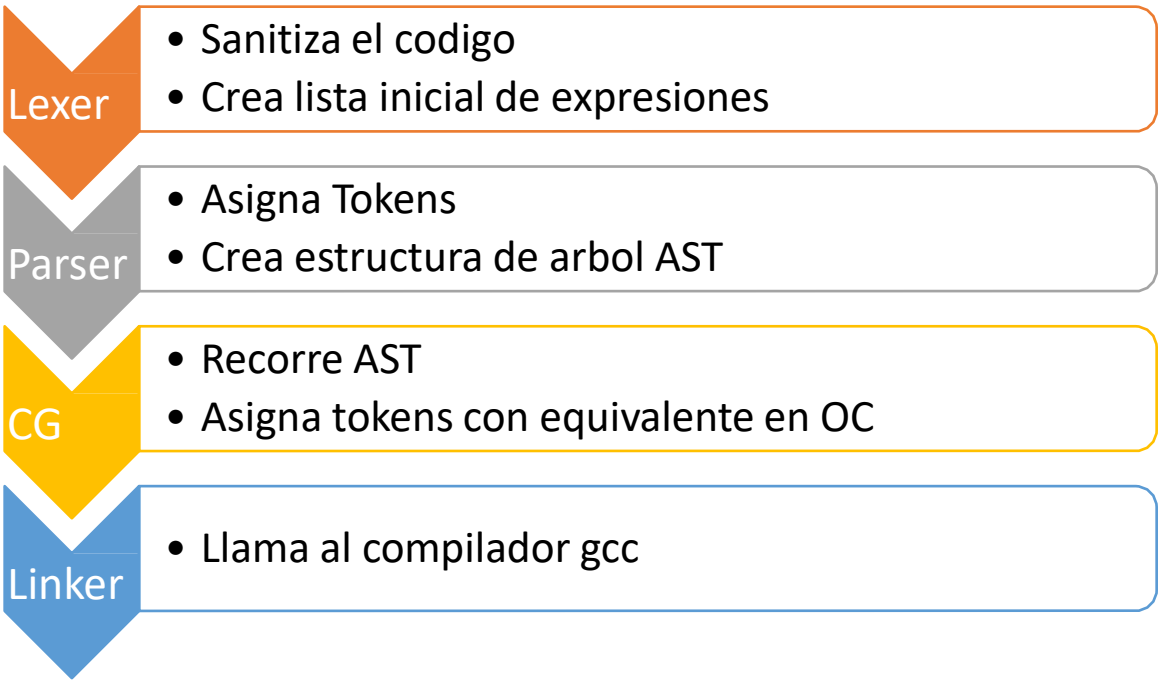
21	**Corregir errores**			**Nov 13, 2019**	**Nov 15, 2019**
22	Modificar Lexer	Hacer los cambios necesarios para el Lexer	Integrator	Nov 17, 2019	**Nov 19, 2019**
23	Modificar Parser	Hacer los cambios necesarios en el parser.	Project Manager, Architect	Nov 18, 2019	**Nov 20, 2019**
24	Modificar Generador de Código	Hacer los cambios necesarios para el Generador de Código	Tester, Integrator	Nov 20, 2019	**Nov 22, 2019**
25	Realizar las Pruebas para Entrega Final	Añadir las pruebas pertinentes para probar en su totalidad el compilador	Tester	Nov 23, 2019	**Nov 25, 2019**
26	Actualizar documentación	Actualizar documentación, verificar que sea clara	Project Manager, Integrator	Nov 20, 2019	**Nov 25, 2019**
			Project Manager, Tester,Integrator,		

		Corregir errores o agregar detalles para que el compilador sea lo más eficiente posible	**Architect**		
27	**Corregir Detalles**			**Nov 25, 2019**	**Nov 28, 2019**

Arquitectura

La estructura general del compilador está fundamentada en las recomendaciones del tutorial de Nora Sandler. Por lo anterior el compilador cuenta con cuatro módulos fundamentales, estos son: Lexer, Parser, Code Generator, Linker. En esta fase el compilador no realiza tareas adicionales como optimización de código.

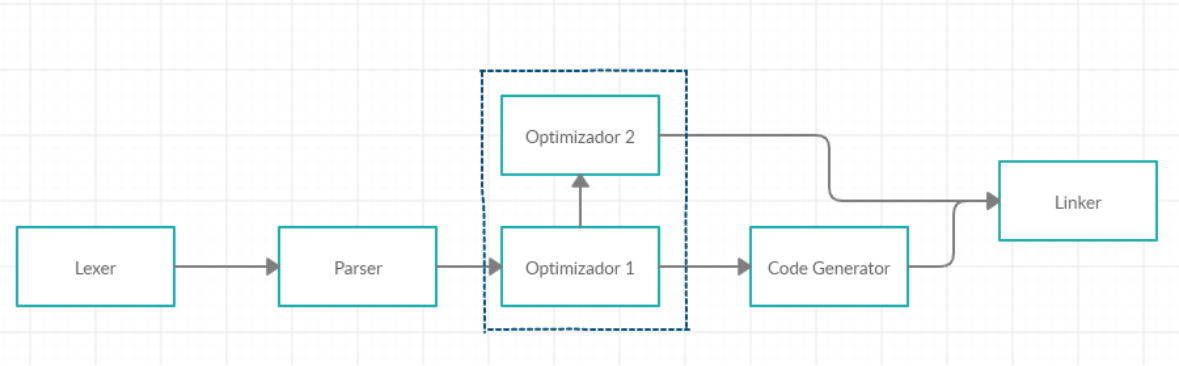
Los módulos anteriormente mencionados procesan el código de manera secuencial como se ve en el siguiente diagrama.



La estructura del proyecto tiene tres secciones principales, con la estructura que se presenta a continuación:

- _build
  - .mix
  - Consolidated
  - ebin
- lib
  - CodeGenerator
  - Nodo
  - Token
  - Compiler
  - Lexer
  - Linker
  - parser
- test
  - compiler_test.exs
  - test_helper.exs

La estructura siguiente es la general del compilador, consta de varias fases y optimizadores que complementan el proceso de compilación de un archivo en lenguaje C.



Lexer:

Este módulo convierte el texto en una lista de tokens con información suficiente para indicar la línea y el carácter compuesto el cual es necesario para lanzar los errores mediante un raise.

Parser:

Para el parser utilizamos la siguiente gramática:

<program> ::= <function>

<function> ::= "int" <id> "(" ")" "{" <statement> "}"

<statement> ::= "return" <exp> ";"

<exp> ::= <logical-and-exp> { "||" <logical-and-exp> }

<logical-and-exp> ::= <equality-exp> { "&&" <equality-exp> }

<equality-exp> ::= <relational-exp> { ("!=" | "==") <relational-exp> }

<relational-exp> ::= <additive-exp> { ("<" | ">" | "<=" | ">=") <additive-exp> }

<additive-exp> ::= <term> { ("+" | "-") <term> }

<term> ::= <factor> { ("*" | "/") <factor> }

<factor> ::= "(" <exp> ")" | <unary_op> <factor> | <int>

<unary_op> ::= "!" | "~" | "-"

Además de que utilizamos el algoritmo recursivo descendente para que, por cada derivación, haya una función asociada y por lo tanto se pueda validar y tener una mejor estructura al momento de recorrer la lista de tokens. Al igual que en el lexer, el manejo de errores se hace mediante excepciones. Genera un árbol ast el cual contiene la estructura para que el árbol se pueda recorrer y mantenga la semántica del programa.

Optimizador 1:

Esta fase del optimizador lo que hace es utilizar el algoritmo de pos orden para recorrer el árbol generado por el parser, en el nodo que se evalúa se pasan diferentes fases de optimización, en este caso se hizo para optimizar 3 tipos de operadores unarios:

"-" : Si encuentra dos ocurrencias en los hijos izquierdos el nodo que se está analizando se recupera el nodo hijo después de esas dos ocurrencias. Ejemplo:

- - - -2. Aquí se recorrerá de la siguiente forma: 2- - - por lo tanto, cuando este en el segundo menos | negación tomará el valor de 2 y reemplazará el hijo izquierdo del nodo actual por ese nodo. Si fuera el caso de un número par de negaciones hará lo mismo y solo regresara un 2 en este ejemplo
- ~ ~ ~ 2. Aquí es la misma lógica que en la negación, si hay dos negaciones, éstas se pueden reemplazar por el valor sin aplicar la operación.
- ! ! ! 2. Para la negación lógica optamos por tomar un camino distinto, sabemos que después de la tercera operación si se repite el operador, se repite la secuencia de 0,1,0. Por lo tanto en este ejemplo tenemos que el resultado de la ejecución seria 0,1,0 por lo que la primera aparición al no saber exactamente si la expresión es igual o diferente de cero optamos por dejarla ya que al final de cuentas va a dar el mismo resultado que es 0. Por lo que la expresión final quedará !2. Para los casos pares como !!!!2 pasa lo mismo, el resultado de la operación es 0,1,0,1 por lo que en todos los casos si hay 3 apariciones o más, se eliminan dos operadores.

Esto genera un árbol AST mejorado.

Optimizador 2:

Este optimizador es más agresivo ya que toma el árbol ast y calcula el valor de toda la expresión para así solo retornar una constante, en este caso retorna el código ensamblador ya que se utiliza la misma estructura del generador de código para recorrer el árbol y calcular las expresiones. Pasa por el optimizador 1 para recortar el árbol y así sea más fácil el realizar las

operaciones. Dado que en esta versión solo se soportan valores constantes podemos hacer esto.

Generador de Código:

Aquí se utiliza el algoritmo de pos orden para recorrer el árbol y recoger primero las operaciones y expresiones de menor precedencia y así ir acomodando todo el código conforme debe ser procesado. Para las etiquetas en los saltos con operaciones binarias lógicas se utilizó Agentes para ir controlando las apariciones de una operación or o and y así que no haya conflicto.

Linker:

Manda a llamar al linker de gcc para compilar el archivo con extensión .s generado en el Generador de código o el optimizador 2 y así generar el ejecutable.

Plan de pruebas y test suite

Para la realización de las pruebas se utilizó el código provisto por Nora Sandler para el testeo del proyecto, dichas pruebas se realizaron de maneara automática y se verifico que el compilador se comportara de la forma esperada para casos en donde la sintaxis y gramática del programa fueran correctas y casos en donde esta condición no se satisficiere.

Conclusiones

La elaboración del compilador probo ser un reto mas grande de lo que todos los miembros del equipo anticipábamos, no solo fue necesario familiarizarnos por primera vez con paradigmas como la programación funcional sino también adaptarnos al uso de un nuevo lenguaje de programación, así como retomar conceptos relacionados con ciencia de la computación que en ciertos casos no aplicábamos desde hace varios meses.

Otro reto importante que encontramos a lo largo del desarrollo del proyecto fue el de alcanzar la forma más eficiente de coordinar y organizar nuestros esfuerzos, en más de una ocasión la falta de comunicación oportuna provoco retrasos importantes para alcanzar los hitos planeados originalmente. La falta de comunicación y una adecuada planeación generaron también en algunas ocasiones un ambiente de tensión dentro de la dinámica del grupo.

Es por lo anterior que para promover un ambiente mas eficiente y de armonía hemos planteado las siguientes medidas que serán puestas en marcha para la siguiente iteración del proyecto:

- Plantear plazos realistas para elaboración de actividades
- Llevar una mejor coordinación y comunicación de avances
- Promover el trabajo en parejas
- Establecer como fecha limite de entrega una semana anterior a la oficial

Bibliografía

Ruiz Catalán, J. (2010). Compiladores . San Fernando de Henares, Madrid: RC Libros.

Apéndice

https://github.com/hiphoox/c201-whiletrue

WBS

No. Tarea	Nombre de la tarea	Duración	A	P	Holgura
0	Plan de Estudios Ing. Comp.	348 hrs.	N/A	N/A	40 hrs.
1	Administracion del Proyecto	100 hrs.	0	2	40 hrs.
2	Levantamiento de Requerimientos	40 hrs.	1	2.1	24 hrs.
2.1	Creacion del Documento de Req.	20 hrs.	2	3	8 hrs.
3	Análisis	84 hrs.	2.1	3.1, 3.2	40 hrs.
3.1	Análisis del Compilador	60 hrs.	3	3.4	24 hrs.
3.2	Análisis de Interfaz Grafica	56 hrs.	3	3.3	24 hrs.
3.3	Definicion Estandares Inter. Graf.	16 hrs.	3.2	3.4	8 hrs.
3.4	Gnrción del Documt. de Análisis	16 hrs.	3.1, 3.3	3.5	8 hrs.
3.5	Entrega del Documt. de Análisis	8 hrs.	3.5	4	8 hrs.
4	Diseño	88 hrs.	3.5	4.1, 4.3	32 hrs.
4.1	Diseño del Compilador	32 hrs.	4	4.2	16 hrs.
4.2	Modelado del Compiador	24 hrs.	4.1	4.4	12 hrs.
4.3	Diseño de la Interfaz Grafica	32 hrs	4	4.4	18 hrs.
4.4	Generación de Prototipo	24 hrs.	4.2, 4.3	4.5	12 hrs.
4.5	Entrega del Documt. de Diseño	8 hrs.	4.4	5	8 hrs.
5	Desarrollo	48 hrs.	4.5	5.1, 5.3	24 hrs.
5.1	Generación del Compilador.	32 hrs.	5	5.2	20 hrs.
5.2	Desarrollo de Conec. Compilador	16 hrs.	5.1	6	8 hrs.
5.3	Desarrollo de la Interfaz Grafica	32 hrs.	5	6	20 hrs.
6	Pruebas	32 hrs.	5.2, 5.3	6.1, 6.2	16 hrs.
6.1	Pruebas Unitarias Compilador	8 hrs.	6	6.3	8 hrs.
6.2	Pruebas Unitarias I.G.	8 hrs.	6	6.3	8 hrs.
6.3	Pruebas Integrales	8 hrs.	6.1, 6.2	6.4	8 hrs.
6.4	Gnrción. de Evidencia de Pruebas	8 hrs.	6.3	6.5	8 hrs.
6.5	Entrega de Evidencia de Pruebas	8 hrs.	6.4	7	8 hrs.
7	Implementación	40 hrs.	6.5	7.1	8 hrs.
7.1	Pase de código a Implementacíon	16 hrs.	7	7.2	8 hrs.
7.2	Puesta en Producción	8 hrs.	7.1	7.3	8 hrs.
7.3	Manten. de App en Producc.	16 hrs.	7.2	8	8 hrs.
8	Deplay de Aplicación	16hrs.	7.3	8.1	8 hrs.
8.1	Entrega y Cierre de Proyecto	12 hrs.	8	N/A	8 hrs.

Mapa conceptual