

# **Documento de requerimientos de software**

***Compilador de un Sublenguaje de C***

***Fecha: [28/03/2019]***

## Versión

Fecha	Versión	Autor	Organización	Descripción
28/03/19	1.0	Alejandro Montecillo Oscar Gutierrez	La Peste Negra (LPN)	Creación de documento

## Información del Proyecto

Empresa / Organización	La Peste Negra (LPN)
Proyecto	Compilador de Sublenguaje de C
Fecha de preparación	29 de Marzo de 2019
Cliente	Ing. Norberto Jesus Ortigoza Marquez
Patrocinador principal	Ing. Norberto Jesus Ortigoza Marquez
Gerente / Líder de Proyecto	Oscar Gutiérrez Castillo
Gerente / Líder de Análisis de negocio y requerimientos	Oscar Gutiérrez Castillo

# Aprobaciones

Nombre y Apellido	Cargo	Departamento u Organización	Fecha	Firma
Ing. Norberto Jesus Ortigoza Marquez	Dueño	Bunsan		
Oscar Gutiérrez Castillo	Project Manager	LPN	29/03/2019	
José Alejandro Montecillo Sandoval	Integrador de Código	LPN	29/03/2019	
Francisco Javier Espinoza Jimenez	Arquitecto	LPN	29/03/2019	

# Indice

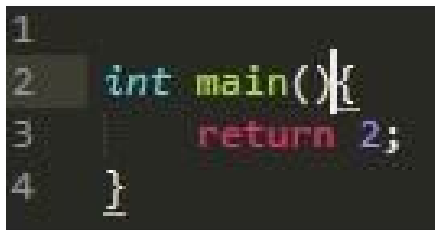
<b>Versión</b>	<b>2</b>
<b>Información del Proyecto</b>	<b>2</b>
<b>Aprobaciones</b>	<b>3</b>
<b>1. Propósito</b>	<b>5</b>
<b>2. Alcance del producto / Software</b>	<b>5</b>
<b>3. Funcionalidades del producto</b>	<b>6</b>
<b>4. Clases y características de usuarios</b>	<b>6</b>
<b>5. Entorno operativo</b>	<b>6</b>
<b>6. Requerimientos funcionales</b>	<b>6</b>
6.1. Generación de código ejecutable	7
6.3. Proyección de árbol AST	8
6.4. Generación de código ensamblador	9
6.5. Banderas estándar Unix para interacción	9
<b>7. Reglas de negocio</b>	<b>10</b>
<b>8. Requerimientos de interfaces externas</b>	<b>10</b>
8.1. Interfaces de usuario	10
8.2. Interfaces de software	10
Ruta absoluta	11
Ruta relativa	11
<b>Anexo 1 Uso de banderas</b>	<b>13</b>
<b>Anexo 2 instalación</b>	<b>14</b>
<b>Anexo 3. Entrevista de levantamiento de requerimientos</b>	<b>15</b>

# 1. Propósito

El documento describe la creación de un compilador escrito en elixir para un subconjunto del lenguaje C, diseñado para la ejecución por terminal en sistemas UNIX/Linux.

## 2. Alcance del producto / Software

En esta primer versión corresponde al reconocimiento, validación y ejecución, con la posibilidad de mostrar las fases intermedias, lista de tokens, árbol ast y el código ensamblador del siguiente programa:

A screenshot of a code editor with a dark background. It shows a C program snippet with four lines of code. Line 1 is empty. Line 2 contains 'int main(){'. Line 3 contains 'return 2;'. Line 4 contains '}'. The code is color-coded: 'int' is blue, 'main' is green, 'return' is red, and '2' is blue. The opening and closing curly braces are white. Line numbers 1, 2, 3, and 4 are visible on the left side of the editor.

```
1  
2 int main(){  
3     return 2;  
4 }
```

Al tratar solo este subconjunto del lenguaje, el software deberá ser capaz de interpretar y detectar errores dentro del código de lenguaje C que sea dado a analizar

El software final deberá ser capaz de procesar finalmente el retorno de varios tipos de datos, uso de variables, estructuras de control (condiciones y bucles) y manejo de funciones con argumentos.

El software en funcionamiento será similar al compilador gcc de linux, el cual permite el uso de banderas (ver anexo 1), pudiendo así mostrar diferentes estados en los que se encuentre el código, además, podrá al momento de generar el código ejecutable, renombrar el nombre del ejecutable que genera por uno más familiar por el usuario.

Finalmente para la implementación de los algoritmos y la implementación del software se usará como referencia el desarrollo de la ingeniera de software Nora Sandler, la documentación de Elixir y el uso de la Single Unix Specification para la

implementación de manejo de banderas

### 3. Funcionalidades del producto

Generación de código

Ejecutable del Programa.

Generación de etapas intermedias del código

Implementación de banderas para hacer que el programa pueda interpretar las solicitudes del usuario

### 4. Clases y características de usuarios

El usuario final de este compilador de sublenguaje C esta orientado a usuarios que tengan conocimientos básicos sobre el uso de una terminal en los sistemas operativos tipo Unix ya sea una distribución GNU/Linux o en el sistema MAC OS

Por el tipo de software y la orientación a la plataforma de implementar es necesario que el usuario que use este software posea los permisos necesarios para poder ejecutar el software.

### 5. Entorno operativo

El software sera diseñado para sistemas UNIX, con uso orientado a la terminal.

Al ser construido en elixir, el usuario deberá poseer el software instalado así como el compilador GCC el cual únicamente se dedicara a ensamblar y linkear el código ensamblador. (Para facilitar esto se agrega un manual de instalación en el anexo 2)

### 6. Requerimientos funcionales

ID de requerimiento	Requerimiento	Prioridad	Riesgo
1	Generación de	Alta	Alta

	código ejecutable		
2	Proyección de lista de tokens	Alta	Baja
3	Proyección de árbol AST	Alta	Media
4	Generación de código ensamblador	Alta	Media
5	Banderas estándar Unix para interacción	Media	Baja

## 6.1. Generación de código ejecutable

Descripción: Generar código ejecutable con nombre por defecto o designado por el usuario.

Prioridad: Alto

Acciones iniciadas y comportamiento esperado:

El usuario pasará por línea de comando en forma de argumento el código C, el cual podrá existir dentro de la misma ruta donde se encuentre el compilador o anexando ruta, relativa o absoluta, y este generará en la ruta donde se encuentre el compilador un ejecutable de nombre 'a' el cual se podrá ejecutar.

```
[ator97@maxwelldaemon Desktop]$ ls
lpncc.beam  prueba.c
[ator97@maxwelldaemon Desktop]$ lpncc.beam prueba.c
[ator97@maxwelldaemon Desktop]$ ls
a lpncc.beam  prueba.c
```

```
[ator97@maxwelldaemon Desktop]$ ls
lpncc.beam  prueba.c
[ator97@maxwelldaemon Desktop]$ ls ..
Desktop Documents Downloads Music Pictures prueba2.c Public Templates Videos
[ator97@maxwelldaemon Desktop]$ lpncc.beam ../prueba2.c
[ator97@maxwelldaemon Desktop]$ ls
a lpncc.beam prueba.c
[ator97@maxwelldaemon Desktop]$
```

Para poder asignar nombre se agregará la bandera -o ó -out posterior al archivo en C y el nombre del archivo

```
[ator97@maxwelldaemon Desktop]$ ls
lpncc.beam  prueba.c
[ator97@maxwelldaemon Desktop]$ lpncc.beam prueba.c -o prueba.out
[ator97@maxwelldaemon Desktop]$ ls
lpncc.beam  prueba.c  prueba.out
[ator97@maxwelldaemon Desktop]$
```

Requerimientos funcionales: Para esto se requieren los requerimientos 2,3,4 del punto 9 de la matriz de requerimientos

## 6.2. Proyección de lista de tokens

Descripción: Mostrar en consola la lista de los tokens obtenidos durante el proceso de análisis lexico

Prioridad: Alta

Acciones iniciadas y comportamiento esperado:

La generación de la lista de tokens es automática ya que es parte del proceso interno del compilador para poder identificar los elementos/símbolos que estén dentro del archivo .c que le demos analizar al compilador; sin embargo como funcionalidad al usuario se le permite con el uso de banderas, ver anexo 1, observar etapas intermedias.

Nota: al momento de usar esta bandera, se detiene el análisis del código.

## 6.3. Proyección de árbol AST

Descripción: Mostrar en consola el árbol ast obtenido durante el proceso de análisis



parseo

Prioridad: Alta

Acciones iniciadas y comportamiento esperado:

La generación del árbol AST es automática ya que es parte del proceso interno del compilador para poder identificar que el orden elementos/símbolos que estén dentro del archivo .c que le demos analizar al compilado sea el corrector; sin embargo como funcionalidad al usuario se le permite con el uso de banderas, ver anexo 1, observar etapas intermedias.

Nota: al momento de usar esta bandera, se detiene el análisis del código.

## **6.4. Generación de código ensamblador**

Descripción: Generar un archivo con terminación .asm, el cual contendrá el código ensamblador procesado del archivo .c que sea dado al compilador

Prioridad: Media

Acciones iniciadas y comportamiento esperado:

El uso de las banderas será siguiendo el formato unix es decir, entre el nombre de la herramienta y el archivo se incluirán las banderas

## **6.5. Banderas estándar Unix para interacción**

Descripción: Crear una especie de interfaz para que el usuario pueda pedir datos específicos del código que ha pasado al compilador

Prioridad: Alta

Acciones iniciadas y comportamiento esperado:

La generación del código ensamblador es automática ya que es parte del proceso interno del compilador para poder transformar a código máquina el código en lenguaje C con ayuda del linker y assembler de la herramienta gcc, sin embargo como funcionalidad al usuario se le permite con el uso de banderas, ver anexo 1, observar etapas intermedias.

Nota: al momento de usar esta bandera, se detiene el analisis del codigo.

## 7. Reglas de negocio

Las reglas de negocio que se acordaron con el cliente Norberto fueron las siguientes:

- 1.-Ejecutable de un sublenguaje de C que retorne un valor 2
- 2.-Se acordó que la forma de ejecución sea por línea de comandos
- 3.-Compilador
  - Retorno de ejecutable
  - Salida -> automática o dictable por el usuario
  - lista de tokens
  - AST
  - Ensamblador

## 8. Requerimientos de interfaces externas

### 8.1. Interfaces de usuario

- Sistema diseñado para uso por terminal
- Interacción mediante el uso de banderas
- Solo puede procesar un archivo a la vez
- El archivo que se pase tiene que terminar en .c

### 8.2. Interfaces de software

- Compilador GCC
- Compilador de Elixir

## 9. Glosario

Término	Descripción
Compilador	consiste en una aplicación que tiene como premisa fundamental la Traducción del lenguaje específico de

	<p>programación</p> <p>Para nuestro contexto traducimos de Lenguaje C a lenguaje Ensamblador</p>
GCC	<p>El GNU Compiler Collection (colección de compiladores GNU) es un conjunto de compiladores creados por el proyecto GNU. GCC es software libre y lo distribuye la Free Software Foundation (FSF) bajo la licencia general pública GPL.</p>
Elixir	<p>Elixir es un lenguaje de programación funcional, concurrente, de propósito general.</p>
Sistema	<p>Un sistema informático es un sistema que permite almacenar y procesar información.</p>
Banderas	<p>Son opciones que permiten acceder a funciones especiales de un programa</p>
Rutas relativas y absolutas	<p>En informática, una ruta (<i>path</i>, en inglés) es la forma de referenciar un archivo informático o directorio en un sistema de archivos de un sistema operativo determinado.</p> <p>Ruta absoluta</p> <p>Las rutas absolutas señalan la ubicación de un archivo o directorio desde el directorio raíz del sistema de archivos.</p> <p>Por ejemplo, es una ruta absoluta /home/dir1/arc1.fil, que señala la ubicación del archivo arc1.fil desde la raíz del sistema de archivos.</p> <p>Ruta relativa</p> <p>Las rutas relativas señalan la ubicación de un archivo o directorio a partir de la posición actual del sistema operativo en el sistema de archivos.</p> <p>Por ejemplo, es una ruta relativa dir1/arc1.fil que señala al archivo arc1.fil</p>

	<p>dentro del directorio dir1 en la ubicación actual. En sistemas tipo UNIX, la ruta ~/ es una ruta relativa que lleva al directorio personal del usuario que ha insertado la ruta relativa; por ejemplo, si el usuario Fulano tiene una imagen en su directorio personal, esta imagen podría tener dos rutas de acceso, una relativa y una absoluta:</p> <ul style="list-style-type: none"> <li>• La absoluta: /home/fulano/imagen.jpg</li> <li>• La relativa: ~/imagen.jpg</li> </ul> <p>En este caso, la ruta relativa sólo puede ser verdaderamente válida si el sistema está ubicado en el usuario de Fulano. En este mismo caso, ~/ sería el sinónimo relativo de la ruta /home/fulano/.</p> <p>También se puede crear, borrar, copiar, etcétera, directorios y archivos con este tipo de ruta desde una interfaz de línea de comandos.</p> <p>Algunos ejemplos:</p> <ul style="list-style-type: none"> <li>• mkdir ~/Fotos</li> <li>• chmod 777 ~/</li> <li>• rm ~/foto.jpg</li> </ul>
Token	<p>Un token o también llamado componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación</p>
Arbol AST	<p>Es una representación de árbol de la estructura sintáctica simplificada del código fuente escrito en cierto lenguaje de programación. Cada nodo del árbol denota una construcción que ocurre en el código fuente. La sintaxis es abstracta en el sentido que no representa cada detalle que aparezca en la sintaxis verdadera.</p>
Código Ensamblador	<p>es un lenguaje de programación de bajo</p>

	nivel. Consiste en un conjunto de mnemónicos que representan instrucciones básicas para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables
mnemónicos	es una palabra que sustituye a un código de operación (lenguaje de máquina), con lo cual resulta más fácil la programación, es de aquí de donde se aplica el concepto de lenguaje ensamblador.
Linker	Es el encargado de combinar diferentes archivos con código objeto en un único archivo
Asembler	Se refiere a un tipo de programa informático que se encarga de traducir un fichero fuente escrito en un lenguaje ensamblador, a un fichero objeto que contiene código máquina, ejecutable directamente por el microprocesador.
Código ejecutable	corresponde a unidades de programas. Donde la computadora puede realizar las instrucciones compiladas que tendrán enlazadas una o varias bibliotecas.

## Anexo 1 Uso de banderas

Nombre largo	Nombre corto	Descripción	Salida
-o	--out	Renombre el archivo generado	NO
-s	--ensamblador	Genera un archivo con el código de	Archivo, nombre del programa

		ensamblador resultante de la transformación del proceso lexico y sintactico	salida: nombre_programa .asm
-t	--tokens	Muestra la lista de tokens generados	Pantalla
-a	--ast	Muestra el árbol ast generado	Pantalla
-h	--help	Muestra un menú de auxilio	Pantalla

## Anexo 2 instalación

Las instrucciones de instalación para cada sistema operativo pueden ser encontradas en [Elixir-lang.org](https://elixir-lang.org) en la guía [Installing Elixir](#)(en inglés).

Copiar el repositorio :

```
$ git clone https://github.com/hiphoox/compilers2019_2.git
$ cd compilers2019_2
```

Para trabajar con la rama del proyecto :

```
$ git checkout -b lpn
```

Ejecución.

```
$ mix escript.build
```

## Anexo 3. Entrevista de levantamiento de requerimientos

### Puntos previstos

- 1.-Ejecutable de requerimiento que retorne un valor 2
- 2.-Formato de ejecucion, línea de comandos
- 3.-Compilador
  - Retorno de ejecutable
  - Salida -> automática o dictable por el usuario
  - lista de tokens
  - ASM
  - Ensamblador

### Puntos a revisar

- 1-. ¿Qué tanto deberá escalarse el programa posterior a esta entrega?
  - ¿retorno de más tipos de datos?
  - ¿impresiones y recepciones por pantalla?
  - ¿condiciones y bucles?
  - ¿Funciones?
  - Portabilidad de programa (!)
  - Como fregados vamos a hacerlo portable
  - Forma de configuración de SO para windows
  - Forma de configuración para resto de sistemas Unix-like

### 3-. Generación de ejecutable

- Misma ruta
- Mismo nombre
- Uso de banderas
  - o --out        nombre de la salida del archivo
  - s --ensamblador ensamblador archivo por defecto
  - t --tokens    tokens        consola
  - a --ast        ast        consola
  - h --help       instrucciones
- Combinación de banderas no afecta

Se detiene generacion de codigo,

extras

-f -> dirección a archivo

-ss -> dirección a pantalla

b.- Salida -> automática o dictable por el usuario No se a que se refieren con esto Retorno de más tipos de datos De preferencia si. Pero no es obligatorio en esta entrega

" Impresiones por pantalla y obtención de valores ¿Se espera poder hacer esto en alguna parte de las etapas del entregable? En caso de ser afirmativa,¿la parte de obtención de valores será por entrada estándar o por carga de archivo?" No entiendo a que se refieren con esto

"condiciones y bucles Según el texto de Nora indica que sí, sin embargo, ¿Como cliente lo requerirá? ¿Funciones?" En la semana uno esto no es necesario ahorita

" ¿Para que sistemas debemos de considerar?" Unix

Ator97e Yesterday at 8:50 PM

B Si no se pasan banderas, se genera en la misma ruta o si el usuario desea se cambia el nombre y ruta por defecto Impresiones por pantalla y obtención de valores Obtener datos por entrada estándar como scanf o tipo lectura de archivo

hiphoox Yesterday at 8:51 PM

El código fuente a compilar se pasa por archivo

Ator97eYesterday at 9:00 PM

Oka, respecto a todo lo demás, confirma?

hiphooxYesterday at 9:08 PM

Si

Ator97eYesterday at 9:12 PM

Perfecto, muchas gracias

MarianYesterday at 11:12 PM

Profesor con respecto al plan de trabajo requiere que sea para todas las entregas o solo para cada entrega ?

hiphooxToday at 12:04 AM

Todas