Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Carrera Ingeniería Electrónica

Proyecto integrador para la obtención del título de grado de Ingeniero en Computación

---

TÍTULO EN MAYÚSCULA

Alumnos

Apellido, Nombres y Apellido, Nombres

Director

Ing. Apellido, Nombres

Co-Director

Ing. Apellido, Nombres

Córdoba, República Argentina

Marzo / 2022

Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Escuela de Ingeniería Electrónica

El Tribunal Evaluador reunido en este acto y luego de haber aprobado la Solicitud de Aprobación de Tema y efectuado las distintas instancias de correcciones del Informe del Proyecto Integrador para la obtención del Título de Grado "Ingeniero Electrónico" y cumpliendo con el Reglamento correspondiente, declaran el Informe Final de los estudiantes **Apellido, Nombres** y **Apellido, Nombres** como "aceptado sin correcciones" y la defensa oral Aprobada. Por lo tanto, luego de haber tenido en cuenta los aspectos de evaluación que indica el Reglamento, el Proyecto Integrador se considera Aprobado.

Se firma el Acta de Examen correspondiente y se distribuyen los ejemplares impresos.

Firma y aclaración del Tribunal Evaluador:

Fecha:

Dedicatoria

Para nuestras familias...

Agradecimientos

A nuestros padres, madres y hermanos, por su incondicional apoyo a lo largo de toda la carrera.

A nuestros directores, Apellido Nombre y Apellido Nombre, por la excelente predisposición, la confianza y todo el soporte brindado que hizo posible este proyecto.

A nuestros amigos y futuros colegas, quienes hicieron de estos años de estudio una experiencia más placentera.

A -Lugar donde se realizó el PI-, junto a todo su personal, por las oportunidades y enseñanzas compartidas.

A la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de Córdoba, por la oportunidad de realizar esta carrera de grado.

Resumen

En este trabajo...

Áreas Temáticas del Proyecto Integrador:

Asignaturas:

Palabras Claves:

Abstract

In this thesis...

Key Words:

Índice general

Lista de Figuras

Índice de figuras

Lista de Tablas

Índice de tablas

Lista de Códigos

Lista de Símbolos y Convenciones

GCC GNU Compiler Collection

Capítulo 1

Introducción

Párrafo con la descripción del contenido, lo que espera encontrar el lector.

Introducción

Presentación general clara y breve del contenido del PI, no debe incluir resultados ni conclusiones. La introducción es lo primero que se lee por lo tanto se debe tener un especial cuidado en la redacción. Incluir éstos títulos:

- Antecedentes breves del problema.
- Relevancia de trabajo.
- Motivación para la elección del tema.
- Formulación del problema.
- Objetivo General y Objetivos Específicos.
- Metodología utilizada para lograr los objetivos propuestos.
- Orientación al lector de la organización del texto.

Los Capítulos se numeran del 1 (Introducción) al último en forma consecutiva incluyendo Marco Teórico y Marco Metodológico

1.1. Antecedentes breves del problema

BLABLABLA: blablabla. [1]

1.2. Relevancia de trabajo

1.3. Motivación para la elección del tema

1.4. Formulación del problema

1.5. Objetivo General y Objetivos Específicos

1.6. Metodología utilizada para lograr los objetivos propuestos

1.7. Orientación al lector de la organización del texto

Parte I

Marco teórico

Capítulo 2

Capítulo 2

Un párrafo con la descripción del contenido, lo que espera encontrar el lector

Parte II

Marco metodológico

Capítulo 3

Compiladores

3.1. Conceptos básicos sobre compiladores

3.1.1. Concepto de Compilación

Compilación: Conjunto de procesos efectuados para obtener un archivo ejecutable a partir de código fuente. [1]

3.1.2. Tipos de Compiladores

Los siguientes conceptos son fundamentales para introducir la clasificación.

- ***Host***: arquitectura de la máquina en la que va a correr el compilador.
- ***Build***: arquitectura que se usa para generar el compilador.
- ***Target***: arquitectura en la que deberá correr el ejecutable generado por el compilador.

Así, se presenta la clasificación de los distintos tipos de compiladores (Von Hagen, [1]) en la tabla 3.1.

3.1.3. Estructura de un Compilador

Las fases que componen un compilador [2] pueden verse en la Fig. 3.1.

3.1.3.1. Analizador léxico

El analizador léxico, también llamado *scanner* o *lexer*, es aquel que lee la secuencia de caracteres del código fuente y los agrupa en otras secuencias llamadas **lexemas**. Por cada lexema, el *scanner* genera un *token* de la forma: La Fig. 3.2 muestra las entradas y salidas de la fase en la parte superior.

$$< \textit{nombre-token}, \textit{valor-atributo} > \tag{3.1}$$

Tabla 3.1: Clasificación de compiladores (Fuente: The definitive guide to GCC, [1])

Tipo de compilador	Descripción
Compilador nativo	Compilador que genera ejecutables para el mismo tipo de sistema en el cual está operando. (Ídem *build*, *host* y *target*)
Compilador "cruzado"	Llamado *cross-compiler* en inglés, compilador que corre en una arquitectura específica y genera código para otra distinta. (idem *build* y *host*, distinto *target*)
Crossback compiler	El sistema en donde corre y el de ejecución de binarios son iguales pero el sistema *host* es distinto. (idem *build* y *target*, distinto *host*). Se usan para construir un *cross-compiler* que corra en el sistema en el que el compilador corre)
Crossed native compiler	El sistema *target* y el *host* son los mismos pero el sistema en donde se construye el compilador es distinto. Usa un *cross-compiler* para construir un compilador nativo en un tercer sistema. (idem *target* y *host*, distinto *build*)
Compilador canadiense	El sistema en donde se construye, el *host* y el de destino son todos distintos. Un compilador que construye sobre una arquitectura, corre en otra arquitectura y crea código para una tercera arquitectura. (*build*, *host* y *target* son distintos)

Siendo,

- **nombre-token**: símbolos abstractos usados durante el análisis sintáctico.
- **valor-atributo**: puntero a una entrada en la tablas de símbolos.

Tabla de símbolos El compilador guarda los nombres de variables o nombres de funciones (que aparecen en el código fuente) en la **tabla de símbolos**. También, almacena atributos varios de dicha variable, por ejemplo: tipo, *scope*, argumentos, tipo de pasaje de argumentos, tipo de retorno, etc. Luego, se realiza un mapeo de *tokens* con sus respectivos nombres de la tabla.

3.1.3.2. Analizador sintáctico

El analizador sintáctico, también llamado *parser*, utiliza el primer componente de cada *token* (*nombre-token* en ecuación 3.1) para crear una representación en forma de árbol que muestre la estructura de los *tokens*. Se genera un **árbol sintáctico**. La Fig. 3.2 refleja un esquema de esta fase en la parte superior.

3.1.3.3. Analizador semántico

El analizador semántico emplea el árbol sintáctico y la tabla de símbolos (creada por el *scanner*) para revisar la consistencia semántica del código fuente con respecto a la definición del lenguaje. También se realizan conversiones de un tipo de dato a otro (llamadas coerciones). Ver Fig. 3.2 en la fase correspondiente.

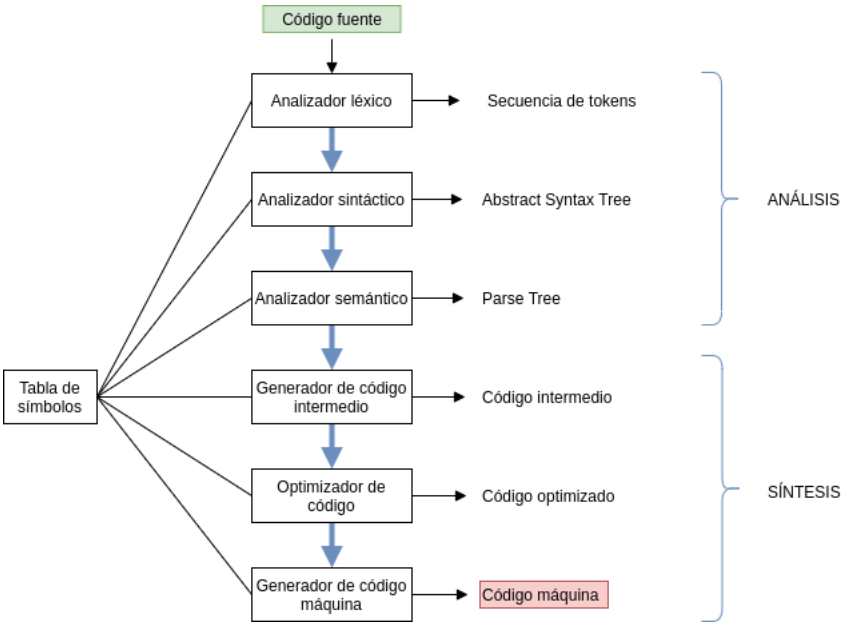

Figura 3.1: Fases de compilación (Fuente: (Modificado) Compilers: principles, techniques, & tools, [2])

3.1.3.4. Generador de código intermedio

El generador intermedio tiene como meta producir código de tipo intermedio para obtener un lenguaje flexible y optimizable para la parte *backend* del compilador. Observar dentro de la Fig. 3.2 para una ilustración de la fase.

El código intermedio puede tener muchas formas distintas (dependiendo de cada compilador).

Propiedades de todo código intermedio Todo código intermedio debe cumplir las propiedades de: *a*) ser fácil de generar; *b*) ser fácil de traducir a lenguaje máquina

Código de tres direcciones Es una forma de código intermedio que consiste en una secuencia de instrucciones cuasi-*assembler* con tres operandos por cada instrucción (cómo máximo). Cada operando puede actuar como un registro.

Algunas características de este tipo de código son:

- Las instrucciones deben tener como máximo un **único operador** (órden de operaciones)
- El compilador debe generar un **nombre temporal** para guardar el valor retornado por la instrucción de tres direcciones (variables temporales tienen nombre).

3.1.3.5. Optimizador de código intermedio

El optimizador de código intermedio realiza mejoras sobre dicho código para obtener un "mejor código" como salida de cada sucesiva optimización. Se presenta también una ilustración en la parte

inferior de la Fig. 3.2.

Mejor código = más rápido, más corto o de menor consumo de potencia.

3.1.3.6. Generador de código final

El generador de código final utiliza el código optimizado para generar el código final deseado. Si el lenguaje final es código máquina, se eligen los registros o lugares de memoria para cada variable usada en el programa. En este trabajo, el código final buscado es código máquina. Por ello, se empleará el término "generador de código máquina". En este caso, se traducen instrucciones inmediatas a secuencias de instrucciones máquina. La parte inferior de la Fig. 3.2 muestra una ilustración sobre esta fase.

3.1.3.7. Instrucciones de compilación

```
1        $ gcc -fdump-tree-original-raw -fdump-tree-all-graph -fdump-tree-gimple -
   ↪ da -S ejemplo.c
2        $ gcc -o ejemplogcc ejemplo.c
```

Listing 3.1: Comando de compilación del archivo `ejemplo.c` [5] para GCC.

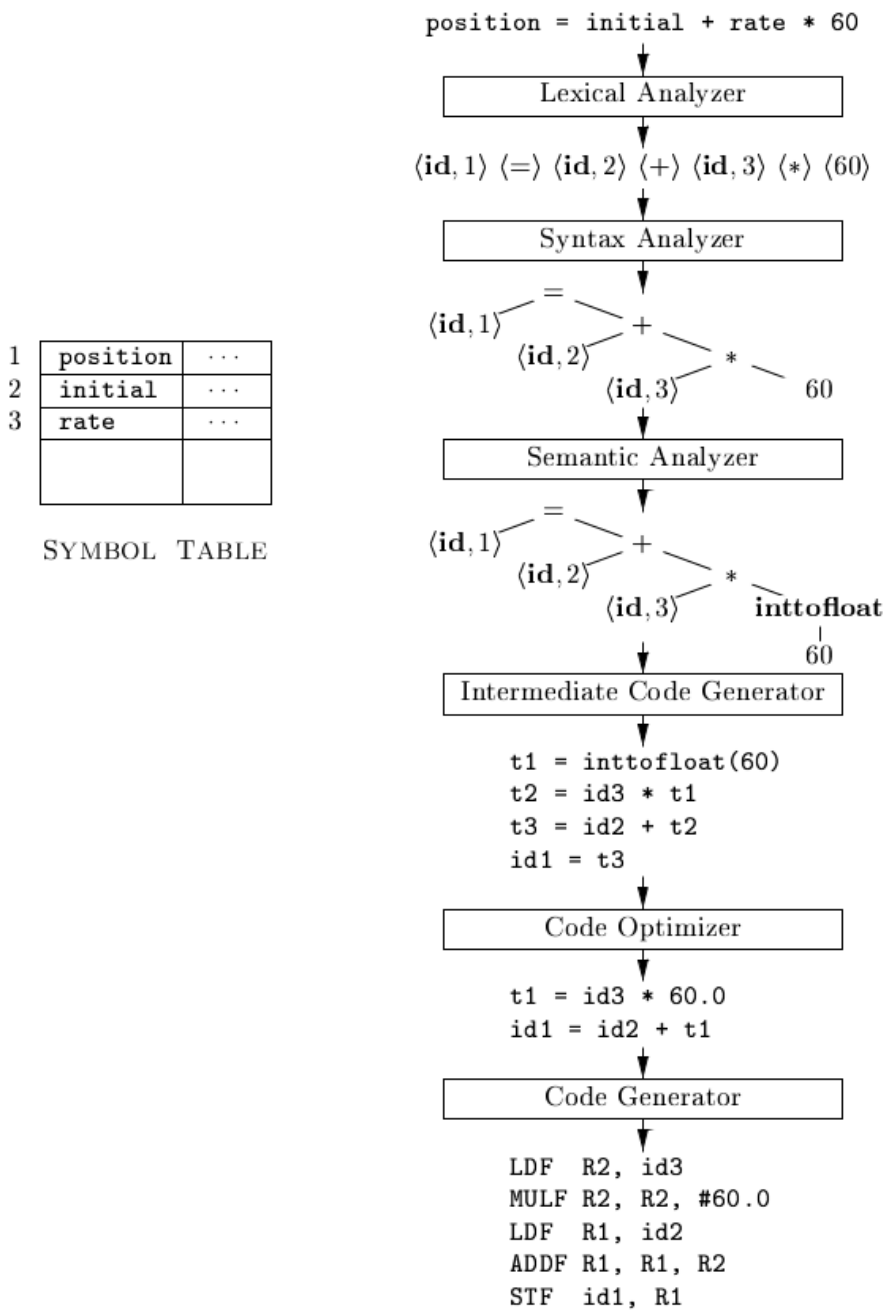

Figura 3.2: Ejemplo completo de fases de compilación (Fuente: Compilers: principles, techniques, & tools, [2])

Capítulo 4

Descripción del modelo experimental

ES UN CAPITULO OPCIONAL

Puede formar parte del último Capítulo o en un Capítulo aparte. Se describe el modelo terminado, su funcionalidad, operación, manual de instrucciones, mediciones, etc.

Capítulo 5

Resultados

Aquí se describen los resultados obtenidos luego de todo el proceso a modo de resumen haciendo referencia a lo desarrollado anteriormente.

Los Resultados no deben incluirse en un Capítulo, es un apartado individual.

Capítulo 6

Conclusiones

Las Conclusiones deben tener una redacción clara, concreta y directa. No son un resumen del trabajo. Deben reflejar los alcances y las limitaciones del estudio, recomendaciones, indicaciones de posible continuidad del trabajo, etc. Se sugieren incluir los aspectos siguientes:

- Resultados obtenidos en relación a la Solicitud de Aprobación de Tema.
- Conclusión General.
- Aporte que hace a la Ingeniería o a un campo de conocimiento.

Las Conclusiones no deben incluirse en un último Capítulo, es un apartado individual

Bibliografía

[1] W. Von Hagen, *The definitive guide to GCC*, 2nd ed. Berkeley, CA: Apress, 2006, ISBN: 9781590595855.

[2] A. V. Aho y A. V. Aho, eds., *Compilers: principles, techniques, & tools*, 2nd ed. Boston: Pearson/Addison Wesley, 2007, ISBN: 9780321486813.

[3] P. H. Dave y H. B. Dave, *Compilers: Principles and Practice*. English. Pearson India, 2012, ISBN: 9788131776117.

[4] B. C. Lopes y R. Auler, *Getting Started with LLVM Core Libraries: Get to Grips with LLVM Essentials and Use the Core Libraries to Build Advanced Tools*. English. Packt Publishing, 2014.

Referencias

[5] J. Cancinos y J. Gonzalez, *Codigo fuente - ejemplo.c*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/GCC/ejemplo.c (visitado 17-11-2021).

[6] *Three address code in Compiler*, en-us, mayo de 2018. [En línea]. Disponible en: https://www.geeksforgeeks.org/three-address-code-compiler/ (visitado 17-11-2021).

[7] *GCC Frontend HOWTO: Compiler Tools*, publisher: The Linux Documentation Project. [En línea]. Disponible en: https://tldp.org/HOWTO/GCC-Frontend-HOWTO-3.html (visitado 17-11-2021).

[8] Department of Computer Science and Engineering - Indian Institute of Technology, *Introduction to RTL*, Bombay, 2010. [En línea]. Disponible en: https://www.cse.iitb.ac.in/~uday/courses/cs715-09/gcc-rtl.pdf (visitado 17-11-2021).

[9] Department of Computer Science and Engineering ,Indian Institute of Technology y U. Khedker, *GCC Translation Sequence and Gimple IR*, Bombay, 2010. [En línea]. Disponible en: https://www.cse.iitb.ac.in/~uday/courses/cs715-09/gcc-gimple.pdf (visitado 17-11-2021).

[10] LLVM, *Passes in LLVM*. [En línea]. Disponible en: https://llvm.org/devmtg/2014-04/PDFs/Talks/Passes.pdf (visitado 17-11-2021).

[11] LLVM Project, *LLVM Language Reference Manual — LLVM 12 documentation*. [En línea]. Disponible en: https://llvm.org/docs/LangRef.html (visitado 17-11-2021).

[12] *Toolchain build procedures*. [En línea]. Disponible en: https://hugh712.gitbooks.io/embeddedsystem/content/toolchain.html (visitado 17-11-2021).

[13] LinuxBaya, *Cross-compiler example*. [En línea]. Disponible en: https://linuxbaya.blogspot.com/2020/09/cross-platform-development-toolchain.html (visitado 17-11-2021).

[14] C. Hock-Chuan, *GCC and Make - A Tutorial on how to compile, link and build C/C++ applications*. [En línea]. Disponible en: https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html (visitado 17-11-2021).

[15] J. Cancinos y J. Gonzalez, *Repositorio de trabajo*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/tree/main/PPS/GCCvsCLANGLLVM (visitado 17-11-2021).

[16] J. Cancinos y J. Gonzalez, *Código resultante del preprocesador en GNU GCC - ejemplo.i*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/GCC/ejemplo.i (visitado 17-11-2021).

[17] J. Cancinos y J. Gonzalez, *Código resultante del preprocesador en CLANG/LLVM - ejemplo.i*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/CLANGLLVM/ejemplo.i (visitado 17-11-2021).

[18] J. Cancinos y J. Gonzalez, *CLANG/LLVM - tokens.txt*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/CLANGLLVM/tokens.txt (visitado 17-11-2021).

[19] J. Cancinos y J. Gonzalez, *GCC - ejemplo.c.004t.original*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/GCC/ejemplo.c.004t.original (visitado 17-11-2021).

[20] J. Cancinos y J. Gonzalez, *CLANG/LLVM - ejemplo.ast*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/CLANGLLVM/ejemplo.ast (visitado 17-11-2021).

[21] J. Cancinos y J. Gonzalez, *GCC - ejemplo.c.005t.gimple*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/GCC/ejemplo.c.005t.gimple (visitado 17-11-2021).

[22] J. Cancinos y J. Gonzalez, *CLANG/LLVM - Readable LLVM IR - ejemplo.ll*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/CLANGLLVM/ejemplo.ll (visitado 17-11-2021).

[23] J. Cancinos y J. Gonzalez, *GCC - ejemplo.c.317r.final*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/GCC/ejemplo.c.317r.final (visitado 17-11-2021).

[24] J. Cancinos y J. Gonzalez, *CLANG/LLVM - machineinstruct.txt*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/CLANGLLVM/machineinstruct.txt (visitado 17-11-2021).

[25] J. Cancinos y J. Gonzalez, *GCC- ejemplo.s*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/GCC/ejemplo.s (visitado 17-11-2021).

[26] J. Cancinos y J. Gonzalez, *CLANG/LLVM - ejemplo.s*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/GCCvsCLANGLLVM/CLANGLLVM/ejemplo.s (visitado 17-11-2021).

[27] *Drysdale, David. "How programs get run: ELF binaries." 2015*, en-us. [En línea]. Disponible en: https://lwn.net/Articles/631631/ (visitado 17-11-2021).

[28] *Understanding the ELF File Format, linuxhint*, en-us. [En línea]. Disponible en: https://linuxhint.com/understanding_elf_file_format/ (visitado 17-11-2021).

[29] *What is the significance of ".comment"section in ELF?, quora*, en-us. [En línea]. Disponible en: https://www.quora.com/What-is-the-significance-of-comment-section-in-ELF (visitado 17-11-2021).

[30] *Declaring Attributes of Functions, GNU*, en-us. [En línea]. Disponible en: https://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Function-Attributes.html (visitado 17-11-2021).

[31] *Assembler warning with gcc warning when placing data in .text, stackoverflow*, en-us. [En línea]. Disponible en: https://stackoverflow.com/questions/58455300/assembler-warning-with-gcc-warning-when-placing-data-in-text (visitado 17-11-2021).

[32] *Add ASCII comments to a C binary?, stackoverflow*, en-us. [En línea]. Disponible en: https://stackoverflow.com/questions/32740668/add-ascii-comments-to-a-c-binary (visitado 17-11-2021).

[33] *How to define a string literal in gcc command line?, stackoverflow*, en-us. [En línea]. Disponible en: https://stackoverflow.com/questions/2410976/how-to-define-a-string-literal-in-gcc-command-line (visitado 17-11-2021).

[34] Cancinos, Jose and Gonzalez, Julian, *Repositorio de trabajo*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/tree/main/PPS/ELF/GCC (visitado 17-11-2021).

[35] Cancinos, Jose and Gonzalez, Julian, *Insercion de codigo usando Inline ASM*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/ELF/GCC/InlineASM/simpleEjInline.c (visitado 17-11-2021).

[36] Cancinos, Jose and Gonzalez, Julian, *Insercion de codigo usando comando objcopy*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/ELF/GCC/AddComment_OBJCOPY/Makefile (visitado 17-11-2021).

[37] Cancinos, Jose and Gonzalez, Julian, *Insercion de codigo usando atributos de C*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/ELF/GCC/AddComment_AtributeC/simpleEjATR.c (visitado 17-11-2021).

[38] Cancinos, Jose and Gonzalez, Julian, *Insercion de codigo en seccion read only data*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/ELF/GCC/String_rodata/cod_rodata.c (visitado 17-11-2021).

[39] Cancinos, Jose and Gonzalez, Julian, *Insercion de codigo utilizando macro y atributos de C en una nueva seccion en ejecutable*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/ELF/GCC/MacrosComment/code_macros.c (visitado 17-11-2021).

[40] Cancinos, Jose and Gonzalez, Julian, *Programa de lectura de metadata insertada en ejecutable ELF*. [En línea]. Disponible en: https://github.com/josemacan/PSCompiladores/blob/main/PPS/LecturaELF/lecturaELF.c (visitado 17-11-2021).

Anexo I

Hello

Puede haber varios Anexos, numerar e iniciar cada uno en hoja nueva

Todo material complementario se ordena en los anexos, según el tipo de material pueden utilizarse varios anexos. Ejemplos: Hojas de datos. Desarrollos complejos que se llevan al anexo para simplificar la lectura del cuerpo del trabajo. Procedimientos complementarios que son se incidencia directa en el trabajo. Líneas de código que no se incluyen en el texto para no complicar la lectura (si el código es extenso debe incluirse en un archivo en el DVD

Anexo II

Hola