UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

título del TFG
Documentación Técnica

Presentado por Rubén Marcos González
en Universidad de Burgos — 28 de junio de 2020
Tutor: Carlos Pardo Aguilar

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Se quiere crear un asistente de programación para C y en este anexo se comentará acerca del planificación que se ha seguido y su viabilidad

A.2. Planificación temporal

El proyecto se irá planificando semana a semana, estableciendo tareas y horario. Si una tarea no se ha acabado en una semana se aplazará a la siguiente semana o se aplazará indefinidamente si han surgido muchas contrariedades en la realización de la misma dejando la opción de empezar con otras tareas.

A.3. Estudio de viabilidad

El proyecto en primera instancia pare plausible tanto técnicamente como legalmente pero no se sabe hasta que punto las diferencias que hay entre el lenguaje C y Python permitirán la realización del mismo.

Viabilidad económica

Considerando que ya hay asistentes de programación gratuitos en internet nuestra única opción económica es poner en manos de los usuarios donaciones para el proyecto como ya hace Eclipse.

Viabilidad legal

No hay ninguna oposición legal a este proyecto siempre que se declare la autoría tanto del parser[1] como de los iconos[2] utilizados en el proyecto.

[1]Enlace al repositorio del parser original:https://github.com/eliben/pycparser
[2]Fuente de los iconos:https://www.flaticon.com/packs/essential-set-2

Apéndice B

Especificación de Requisitos

B.1. Introducción

El objetivo general de este programa es la depuración de código en lenguaje C. Para ello se requerirá poder abrir, crear, editar, compilar y depurar archivos con la extensión .c

B.2. Objetivos generales

OBJ-01: Depuración de código C.

B.3. Catalogo de requisitos

RI-01: Abrir archivos .c y .h

RI-02: Crear archivos .c y .h

RI-03: Editar archivos .c y .h

RI-04: Guardar archivos .c y .h

RI-05: Compilación de archivos .c

RI-06: Depuración de archivos .c

B.4. Especificación de requisitos

Tabla B.1: RF-01 Abrir archivos .c y .h

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	_	
Descripción	El programa deberá ser capaz de abrir archivos con la extensión .c y .h	
Precondición	El usuario deberá haber iniciado el asistente	
Secuencia normal	Paso	
	1:	solicita al sistema el explorador de archivos
	2:	una vez seleccionado el archivo se accederá al editor
Excepciones	Paso	
	2:	el usuario abrirá un archivo con extensión no permitida
	2:	el usuario abrirá un archivo corrupto
Postcondicón	El usuario deberá tener cargado un archivo con la extensión .c o .h	

Tabla B.2: RF-02 Crear archivos .c y .h

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	_	
Descripción	El programa deberá ser capaz de crear nuevos archivos con la extensión .c y .h	
Precondición	El usuario deberá haber iniciado el asistente	
Secuencia normal	Paso	
	1:	solicita al programador la ruta en la que quiere trabajar
	2:	se solicita al programador el nombre del archivo
	3:	se generará un archivo con el nombre y la extensión seleccionados
Excepciones	Paso	
	1:	el usuario selecciona una ruta inexistente: se le preguntará sobre si crear las carpetas correspondientes o si quiere seleccionar una nueva ruta
	3:	el usuario escogerá un nombre ya existente y se le preguntara si quiere sobreescribir el archivo
Postcondicón	El sistema deberá haber creado el archivo correspondiente y tenerlo cargado en el asistente	

Tabla B.3: RF-03 Editar archivos .c y .h

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	RF-01 Abrir archivos .c y .h RF-02 Crear archivos .c y .h	
Descripción	El programa deberá ser capaz de editar archivos con la extensión .c y .h	
Precondición	El usuario deberá tener cargado un archivo en el asistente	
Secuencia normal	Paso	
	1:	el usuario modificará lineas de código
Excepciones	Paso	
	1:	el usuario no dispondrá de permisos de escritura por lo cual no se le permitirá modificar lineas de codigo
Postcondicón	El asistente deberá tener en memoria el archivo modificado para guardarlo posteriormente	

Tabla B.4: RF-04 Guardar archivos .c y .h

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	RF-03 Editar archivos .c y .h	
Descripción	El programa deberá ser capaz de guardar archivos con la extensión .c y .h	
Precondición	El usuario deberá tener cargado en el programa un archivo	
Secuencia normal	Paso	
	1:	solicita al sistema el explorador de archivos
	2:	se sobrescribirá el archivo con el nombre y la extensión elegidas
Excepciones	Paso	
	2:	no haya espacio en el disco duro, se vuelve al paso 1
	2:	no existe dicho archivo, el sistema creará uno con el nombre y extensión elegidas
Postcondicón	El sistema deberá haber modificado correctamente el archivo	

Tabla B.5: RF-05 Compilar archivos .c

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	RF-04 Guardar archivos .c y .h	
Descripción	El programa deberá ser capaz de compilar archivos .c y crear ejecutables a partir de ellos	
Precondición	El usuario deberá haber guardado los cambios del archivo	
Secuencia normal	Paso	
	1:	sobrescribirá el archivo .o correspondiente a dicho archivo
	2:	sobrescribirá el ejecutable
Excepciones	Paso	
	1:	las modificaciones del archivo no han sido guardadas: se ejecutará el RF-04: Guardar archivos .c y .h
	1:	el código de un error de compilación y se interrumpirá el proceso
	1:	no existe el archivo .o correspondiente por lo que se creará uno nuevo
	2:	no existe el ejecutable correspondiente por lo que se creará uno nuevo
Postcondicón	Disponer de un ejecutable funcional	

Tabla B.6: RF-06 Ejecutar linea de código (entrando en función)

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	RF-05 Compilar archivos .c	
Descripción	El programa deberá ser capaz de ejecutar una a una las lineas de código de un programa en lenguaje C	
Precondición	El usuario deberá haber compilado el archivo	
Secuencia normal	Paso	
	1:	se ejecutará una linea de código
	2:	Irá mostrando las variables en memoria de ese programa acorde al RF-09 Mostrar variables en memoria
Excepciones	Paso	
	1:	no se dispone de un ejecutable funcional: se vuelve al RF-05 Compilar archivos .c
	2:	encuentra una función por saltará al RF-07 Ejecutar linea de código (entrando en función) sobre esa función
Postcondicón	–	

Tabla B.7: RF-07 Ejecutar linea de código (saltando función)

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	RF-05 Compilar archivos .c	
Descripción	El programa deberá ser capaz de ejecutar una a una las lineas de código de un programa en lenguaje C	
Precondición	El usuario deberá haber compilado el archivo	
Secuencia normal	Paso	
	1:	se ejecutará una linea de código
	2:	Irá mostrando las variables en memoria de ese programa acorde al RF-09 Mostrar variables en memoria
Excepciones	Paso	
	1:	no se dispone de un ejecutable funcional: se vuelve al RF-05 Compilar archivos .c
Postcondicón	–	

Tabla B.8: RF-08 Ejecutar código completo

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	RF-05 Compilar archivos .c	
Descripción	El programa deberá ser capaz de ejecutar un programa en lenguaje C completo	
Precondición	El usuario deberá haber compilado el archivo	
Secuencia normal	Paso	
	1:	se ejecutará el programa completo
	2:	se mostrá el estado final de todas la variables acorde al RF-09 Mostrar variables en memoria
Excepciones	Paso	
	1:	no se dispone de un ejecutable funcional: se vuelve al RF-05 Compilar archivos .c
Postcondicón	–	

Tabla B.9: RF-09 Mostrar variables en memoria

Versión	Versión 1.0	
Autor	Rubén Marcos Gonzalez	
Tutor	Carlos Pardo Aguilar	
Objetivos asociados	OBJ-01:Depuración de código C.	
Requisitos asociados	RF-06 Ejecutar linea de código (entrando en función) RF-07 Ejecutar linea de código (saltando función) RF-08 Ejecutar código completo	
Descripción	El programa deberá ser capaz de mostrar en una tabla las variables en memoria un un programa C	
Precondición	El usuario deberá empezado a ejecutar un programa C	
Secuencia normal	Paso	
	1:	cada vez que se cree o modifique una variable se mostrará en la tabla
Excepciones	Paso	
	–	–
Postcondicón	–	

Apéndice C

Especificación de diseño

C.1. Introducción

Aquí se especificará las estructuras de datos utilizadas al igual que las trazas de las funciones empleadas.

C.2. Diseño de datos

Esta aplicación no guarda datos en ningún sitio más allá de los archivos C creados por el usuario, esto no quiere decir que no tenga estructuras de datos internas con las que trabaja.
Esta aplicación hay 5 estructuras de datos grandes[1]:

1. funciones: Es un diccionario que guarda las funciones declaradas en el archivo sobre el que estás trabajando, la clave del diccionario es el nombre de la función y el valor es una bicola que cada elemento es una linea de la función

2. iterexec: es una lista de bicolas, las bicolas son las funciones que se van a ejecutar y se guardan en esta lista en el orden que se deben ejecutar, en caso de que se invoque a una función se añade a esta lista y en caso de un return se borra

[1]En el caso de usar bicolas o listas es por optimización, en caso de que las operaciones mas utilizadas sean las de indexación se usan listas, y si en caso contrario las operaciones mas usadas sean las de push(llamada append en Python) y pop se usaran bicolas

3. vardicts: es una lista de diccionarios, en dichos diccionarios se almacenan las variables que se usan en cada función, la clave de estos diccionarios es el nombre de la variable y el valor es una lista compuesta por el tipo de la variable y el valor de la misma. Cada vez que se entra a una función se genera un nuevo diccionario y cada vez que se sale, bien por un return o porque se ha llegado al final de la función, se borra uno

4. estructuras: es un diccionario que como clave tiene las estructuras que son utilizadas en el archivo sobre el que estamos trabajando, la clave es el nombre de la estructura y el valor es un diccionario que actua de forma similar a los usados en vardicts que se componen por el nombre de los campos por claves y su tipo y valor como los valores del diccionario

5. retornos: es una bicola para manejar los retornos del archivo que estamos ejecutando, cada vez que se entra una función se añade un elemento a la bicola y si se llega un retorno se extrae un elemento para ser modificado

C.3. Diseño procedimental

Lo primero de todo para poder hacer funcionar la aplicación debemos trabajar sobre un archivo, para ello tenemos dos opciones crear uno nuevo (ver traza:C.2) y guardarlo o cargar (ver traza:C.3) uno ya creado, una vez guardado (ver traza:C.4) el archivo ya se podría compilar. Si compilamos (ver traza:C.5) el archivo podremos acceder a las opciones de ejecutar (ver traza:C.6) y activar el modo debugger (ver traza:C.7). Si accedemos al modo debugger podemos optar por ejecutar una linea entrando (ver traza:C.8) o no (ver traza:C.9) en las funciones.
Para ver una traza general del programa ver:C.1

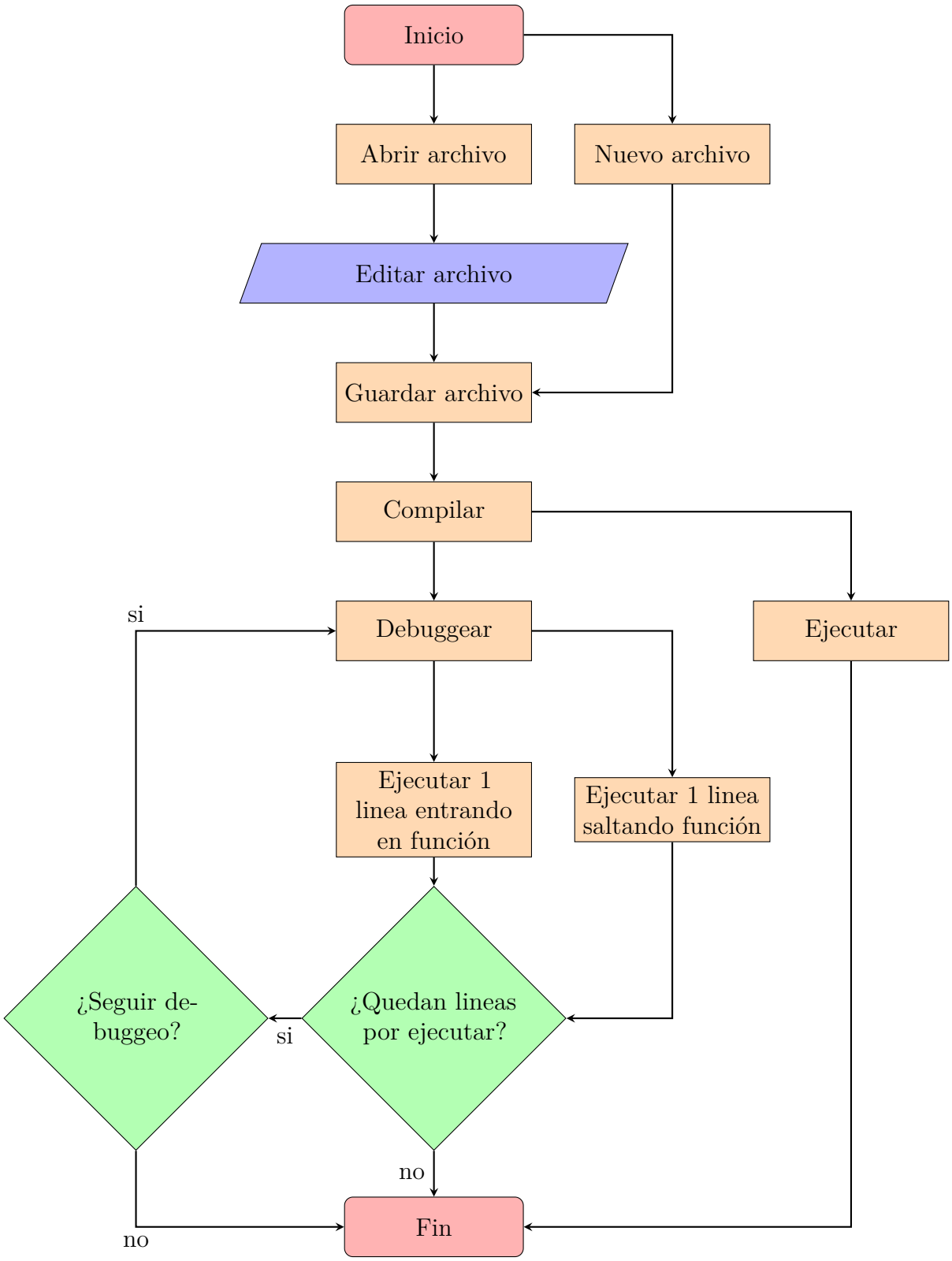

Figura C.1: Ejemplo de una ejecución normal del programa

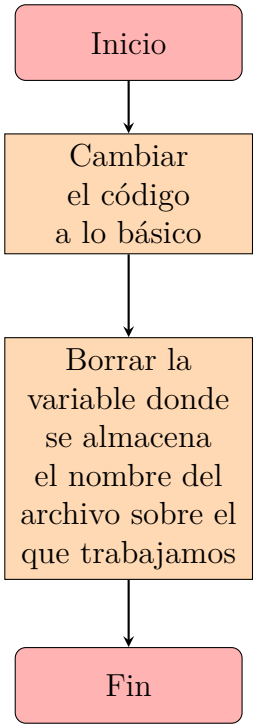

Figura C.2: Traza de la función nuevo

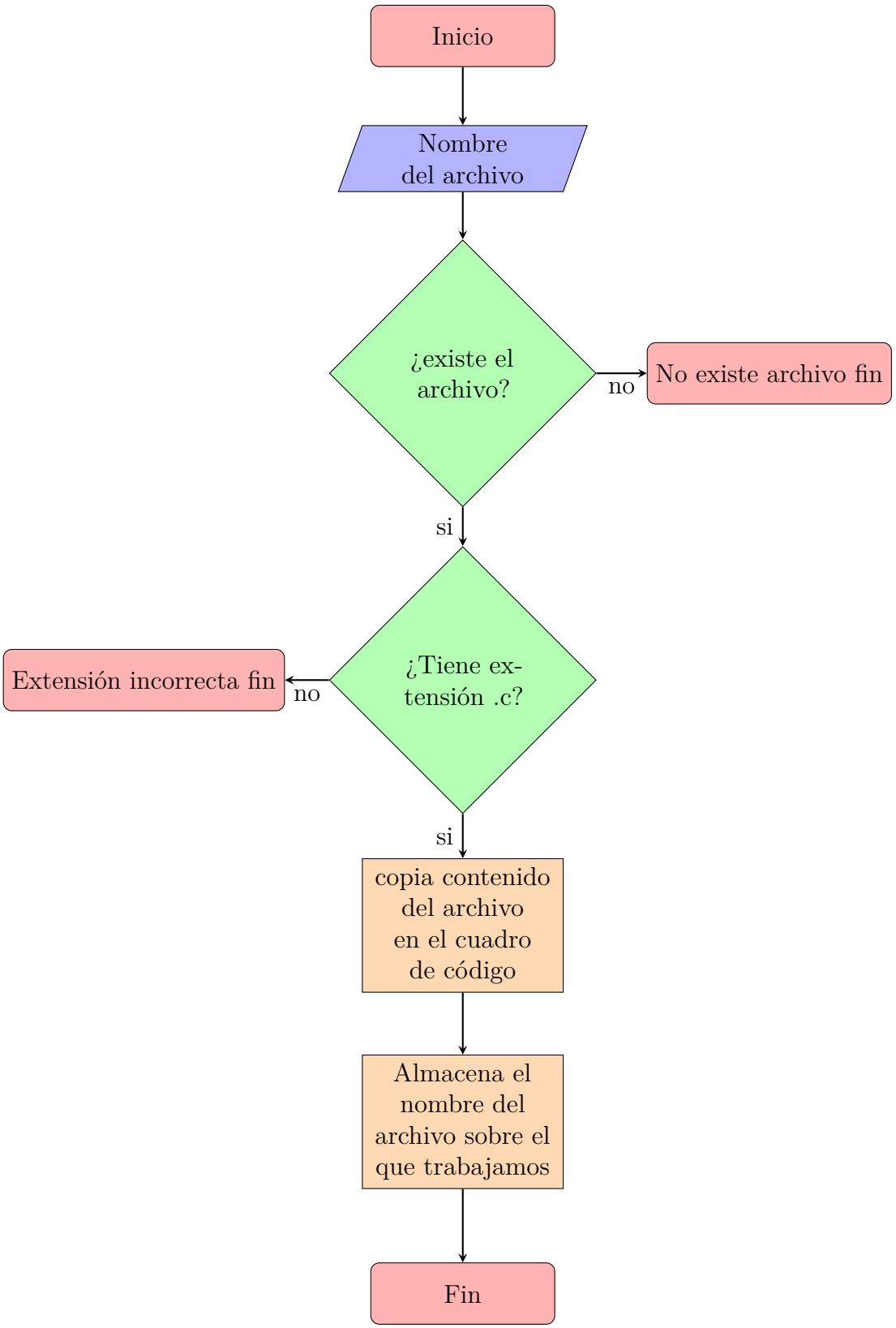

Figura C.3: Traza de la función abrir

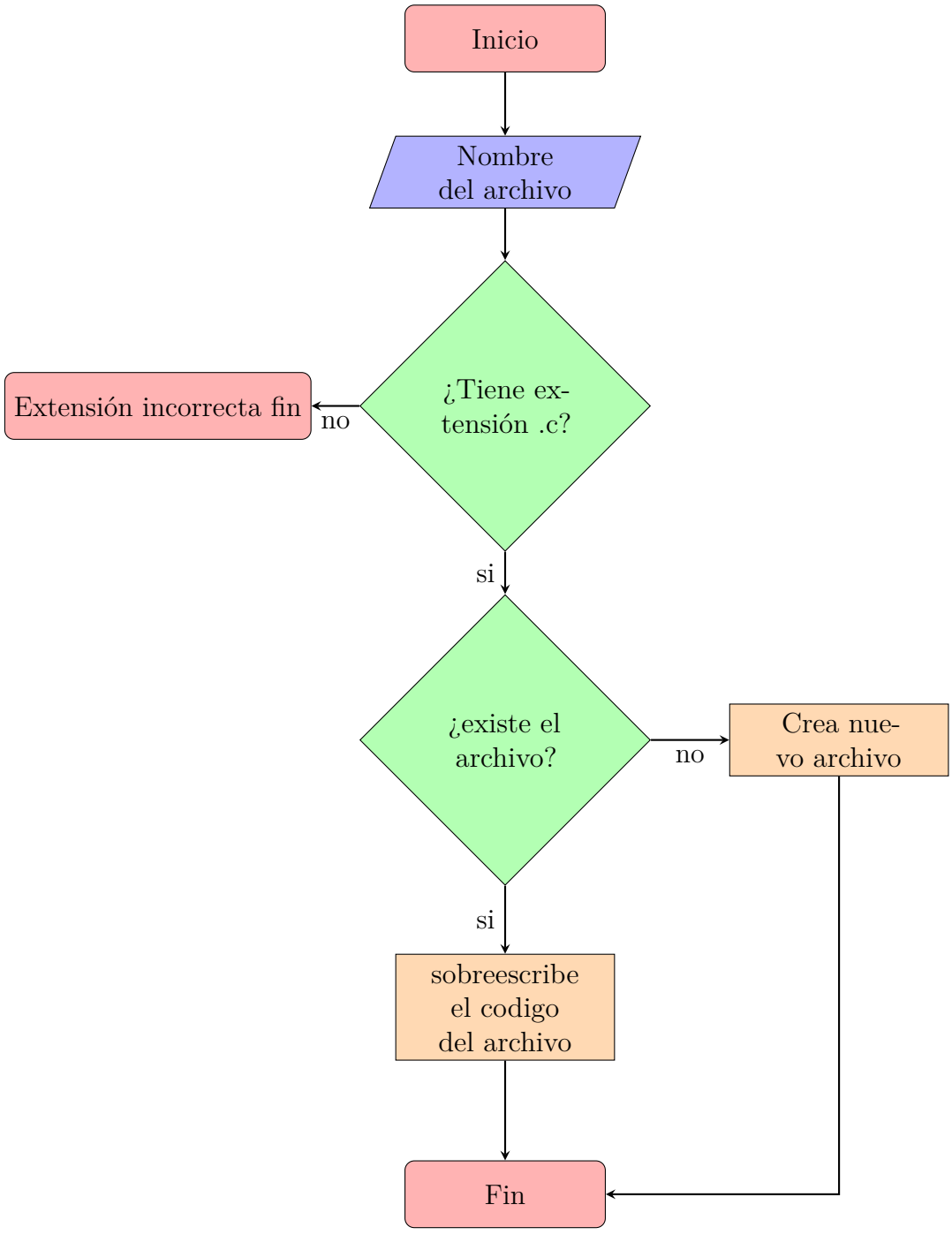

Figura C.4: Traza de la función guardar

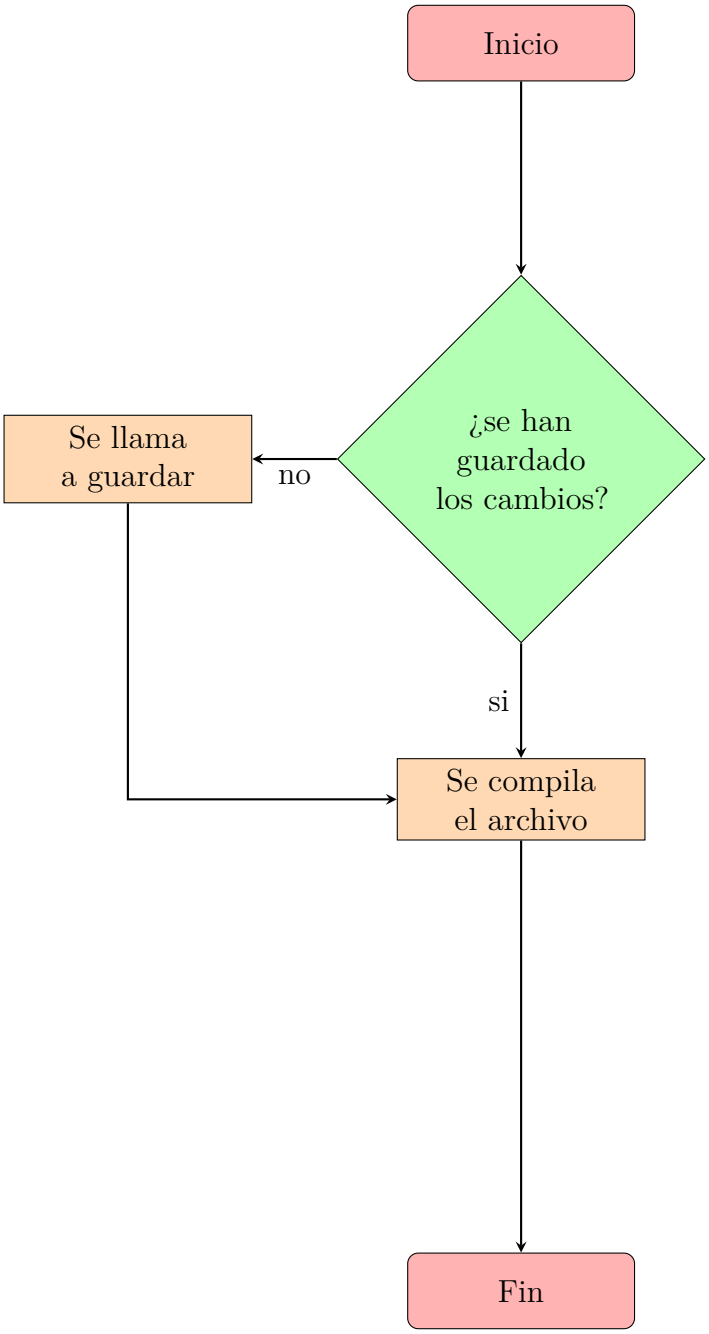

Figura C.5: Traza de la función compilar

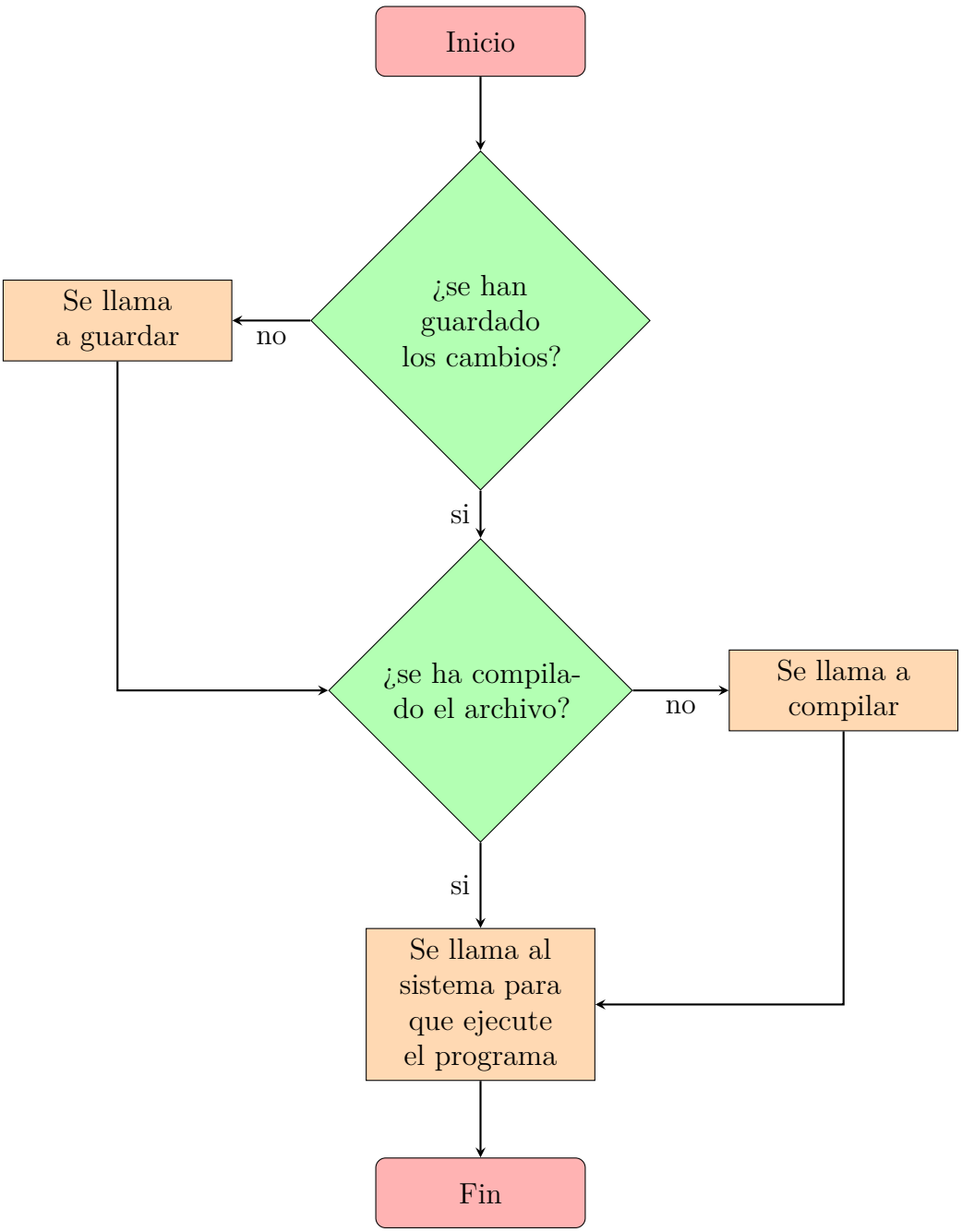

Figura C.6: Traza de la función compilar

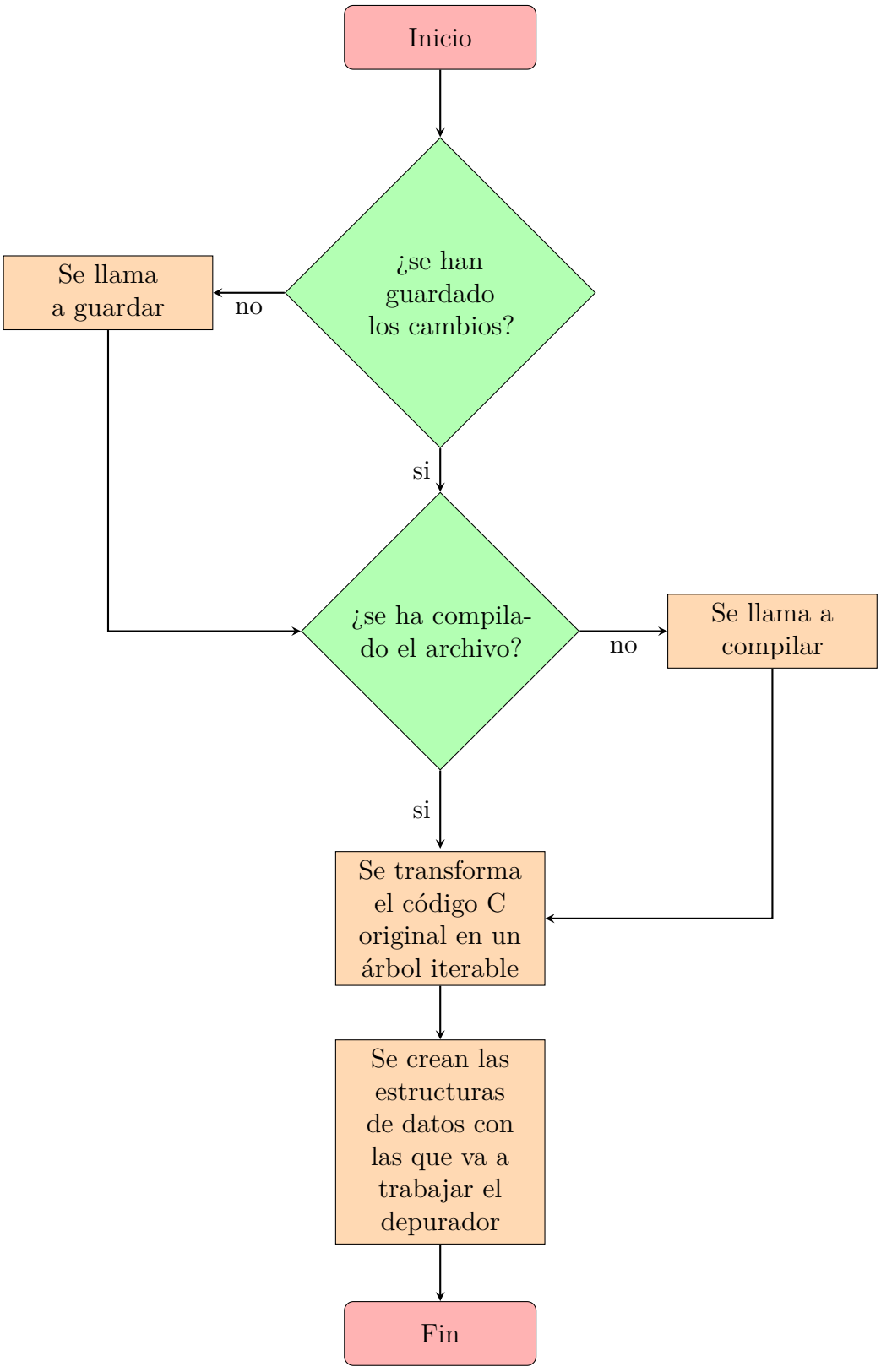

Figura C.7: Traza de la función modo debug

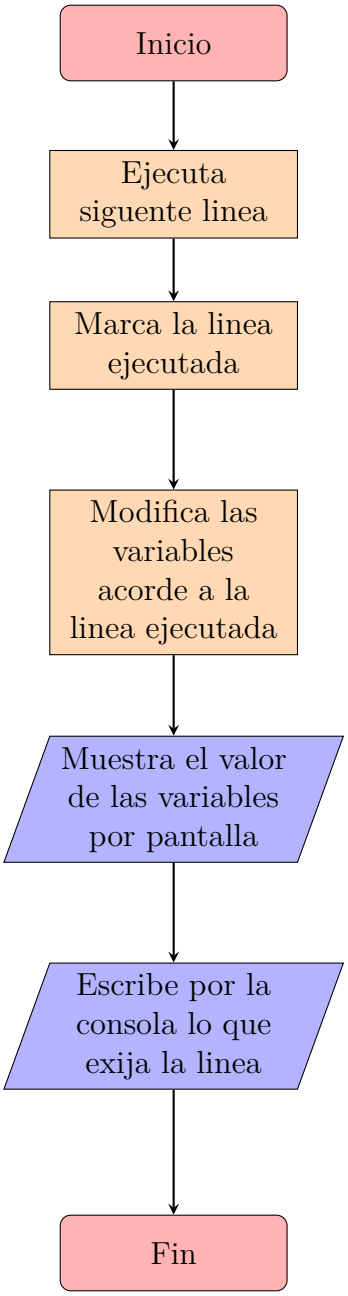

Figura C.8: Traza de la función ejecutar una linea entrando en función

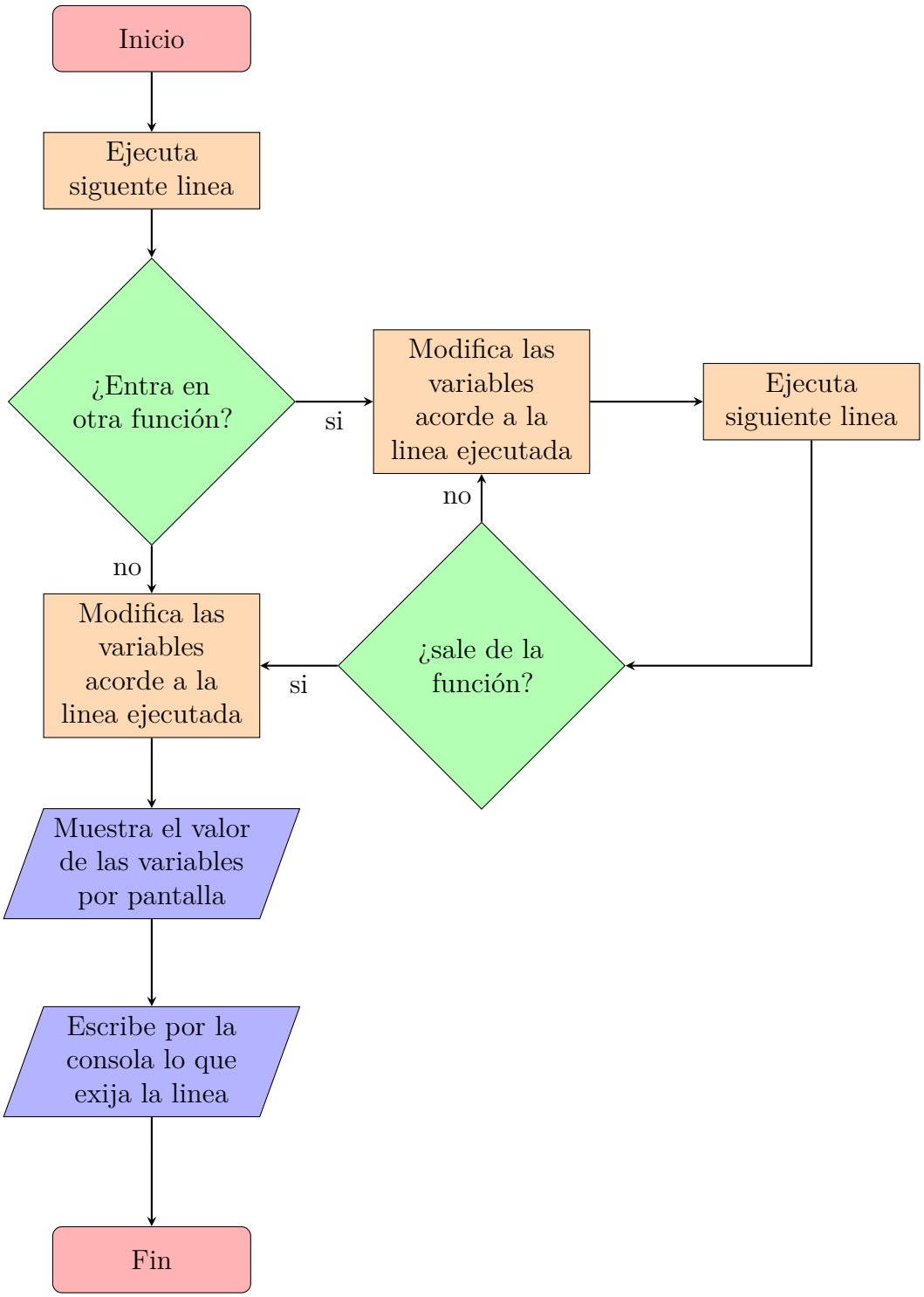

Figura C.9: Traza de la función ejecutar una linea saltando función

C.4. Diseño arquitectónico

En cuanto al numero de usuarios que van a usar la aplicación y el número de funciones a las que van a poder acceder es muy simple. Habrá un solo usuario llamado alumno el cual tendrá acceso a todas las funciones de la aplicación. VerC.10

En cuanto a como interactúan las estructuras de datos con las funciones tampoco es complejo ya que las funciones externas ajenas a la depuración no modifican dichas estructuras. Las funciones propias de la depuración interactúan de la siguiente forma:

- El primer paso es llamar al modo debug el cual crea las estructuras de: funciones, iterexec, vardicts, estructuras y retornos. Las que primero se crean son funciones y estructuras, posteriormente se añade a iterexec el main y a vardicts un diccionario vacío donde el main almacenará las variables y finalmente se crea retornos que se inicializara como una bicola vacía.
- Cada vez que ejecutar linea llega a la inicialización de una variable esta se añadirá al diccionario correspondiente a la función que actualmente estamos depurando con valor desconocido si no se inicializa con ningún valor o con el valor con el que se inicialice si se da el caso.
- Cada vez que se declara una variable como un struct se busca dicho struct en el diccionario de estructuras y se añade la variable con el tipo struct al diccionario correspondiente de vardicts a la función actual.
- cada vez que se altera una variable se busca en el diccionario correspondiente a la función que se está ejecutando actualmente y se modifica su valor.
- Cada vez que se llama a una función se busca en funciones la misma y se añade a iterexec para ser ejecutada, también se añade a vardicts un diccionario para las variables de la función recién añadida y en caso de que la función no este declarada como void se añadirá a la pila de retornos un retorno.
- Cada vez que se llega a un retorno se devuelve un valor de la pila de retornos y se vacía la pila de la ejecución de la función que se está depurando.

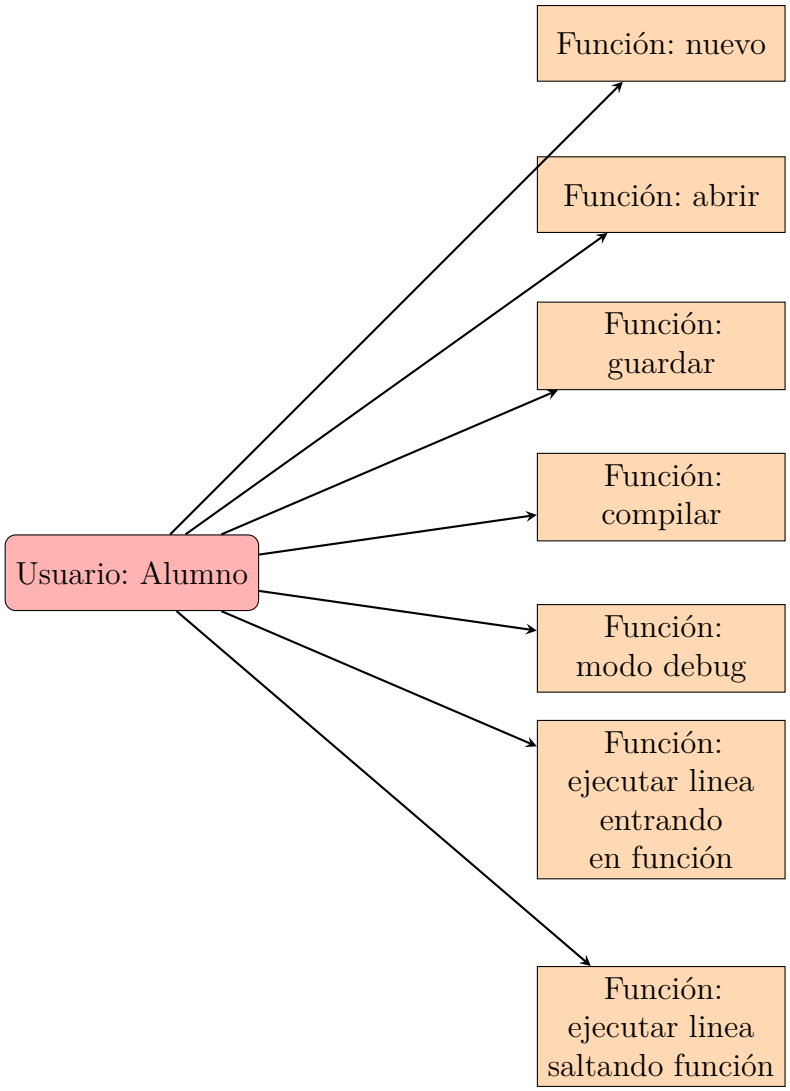

Figura C.10: Diagrama de usuarios

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este anexo se explicará al programador que quiera mejorar el proyecto como puede hacerlo

D.2. Estructura de directorios

Tenemos una estructura de directorios muy simple (ver D.1). Tenemos una carpeta para los iconos de los botones de la aplicación llamada icons, una carpeta llamada pycparser en la que está el parser, una carpeta llamada codes donde se almacenan los códigos de prueba para la aplicación y finalmente en la carpeta raíz tenemos TFG.py que sería el núcleo del trabajo y el que se debería modificar para futuras mejoras del proyecto, tooltip.py que es una clase utilizada para los tooltips de la aplicación y lextab.py y yacctab.py que son dos archivos creados por el parser para apoyarse para hacer el análisis léxico y sintáctico.

D.3. Manual del programador

Para este apartado se ha optado por un comentario exhaustivo del código en vez mostrar capturas del mismo explicándolo.

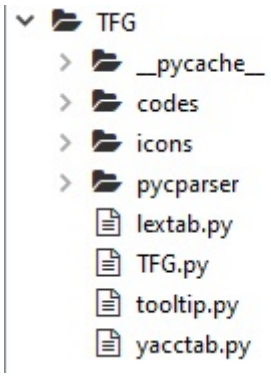

Figura D.1: Directorios del programa

D.4. Compilación, instalación y ejecución del proyecto

Para poder usar esta herramienta se deberá tener instalado el paquete de Anaconda 3 para ejecutar el proyecto y MinGW para compilar los archivos .c que queramos depurar. Video explicativo de como descargar y ejecutar el proyecto: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EV0K9OXoahJDsmgPYIYb6aQB971SI_AAsKBiYx271UNXMg?e=NCeIwd

D.5. Pruebas del sistema

Pruebas en formato video sobre el funcionamiento de la aplicación

1. Marca los errores de compilación en rojo: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/Edqt9ViG2sNHvBCA1C9HLRA
e=n0L42k

2. Los operadores unarios funcionan correctamente: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EcdA6aLBDOVFouB1g8PAl9s
e=vkzMMc

3. Pruebas de aritmética: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EewaAlBQjKJMtqbZ2T4XRDYBErusbBCDNr
e=CFT6gj

4. Recursividad funciona correctamente: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EWCeF2I1IXRDuRUeSp8n8owBw0Sxn
e=sYuJMS

5. Llamadas a funciones en cabecera funciona correctamente: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/ERta1Udad3xHsFfwj98kZQ8B1WsL3LAFof5oyg_MXxXYRQ?e=VO8rGN

6. Funcionan los scan: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EZqjr5tkZO9Eg95EqejMbCkBWbSX5NOXHNXZHnRe
e=sf5Scb

7. Los bucles funcionan: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/Ee5Pp02fZthFrB_YGRNihvUBRQAhyeKK1iu34eAC
e=E7B8BX

8. Los structs se crean correctamente: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EdIiHJFhiZJLg9HqHe280xgByZZa6
e=22dJfc

9. Los arrays se crean correctamente: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EfYnYtvseAxFsx3iZl2kS1oBNvREP
A?e=9YdLoK

Apéndice E

Documentación de usuario

E.1. Introducción

Aquí se explicará al usuario como y que necesita para ejecutar la aplicación.

E.2. Requisitos de usuarios

El usuario deberá tener instalado el compilador gcc incluido en MinGW[1] al igual que Anaconda[2] para poder abrir y ejecutar el proyecto

E.3. Instalación

El usuario deberá descargar el proyecto desde la url del proyecto: https://github.com/RubenMarcosUBU/TFG Y cargarlo desde Spyder. Ver: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EV0K9OXoahJDsmgPYIYb6aQB971SI_AAsKBiYx271UNXMg?e=NCeIwd

E.4. Manual del usuario

Manual de usuario formato vídeo: https://universidaddeburgos-my.sharepoint.com/:v:/g/personal/rmg0094_alu_ubu_es/EdR2bv5rPUpNnZDD8HPlhloBi2qk44oxQy
e=nazsnP

[1] http://mingw.org/download/installer
[2] https://www.anaconda.com/products/individual

Bibliografía