UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Asistente de Programación en lenguaje C

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

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Carlos Pardo Aguilar, profesor del departamento de nombre Ingeniería Informática, área de nombre Lenguajes y Sitemas.

Expone:

Que el alumno D. Rubén Marcos González, con DNI 71313587V, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 28 de junio de 2020

Vº. Bº. del Tutor:

D. nombre tutor

Resumen

Se pretende crear un asistente con interfaz sencilla para programadores novatos para el lenguaje C.

Descriptores

Lenguajes de programación, didáctico, procesadores de texto, depurador.

Abstract

The target of this project is the creation of a simple programming assitant aimed to rookie programmers

Keywords

Programming languages, didactical, text processors, debugger.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La intención de este proyecto es crear un asistente de programación para el lenguaje C que permita ver a los nuevos alumnos como se van modificando las variables linea a linea, teniendo como prioridad la simplicidad para la fácil comprensión del mismo.

Para ello se utilizará un interprete de C que se modificará convenientemente para que vaya mostrando de forma gráfica como las variables se van modificando.

El proyecto se realizará sobre Python por familiaridad con el lenguaje.

Capítulo 2

Objetivos del proyecto

- El principal objetivo del proyecto es la creación de una interfaz limpia y clara que permita monitorizar las variables para nuevos alumnos que nunca hayan estudiado programación. Para ello nos basaremos en interfaces de desarrollo vistos en distintas clases prácticas: MatLab, Spyder y Eclipse.

- Uno de los objetivos es que el interprete de C muestre por pantalla el estado de todas la variables utilizadas en la ejecución, mostrando su valor, su espacio en memoria y su tipo ver 2.1.

- Otro de los objetivos es que el interprete nos permita ejecutar las lineas de código una a una

Nombre	**Tipo**	**valor**
Num	int	1528
Letra	char	?
Vector	int[3]	[23,57,31]
Libro	Book(Struct)	-
->Titulo	char[100]	El viento en los sauces
->Autor	char[50]	Kenneth Grahame
->ISBN	int	1530059984

Tabla 2.1: Ejemplo de visualización de la tabla de variables

- Siguiendo con el objetivo anterior permitir la inclusión de breakpoints para facilitar la depuración de código

Capítulo 3

Conceptos teóricos

Compilador : un compilador es un traductor que se encarga de convertir el código fuente a código máquina para poder ser entendido por una maquina

Interprete : a diferencia de los compiladores que traducen todo el código de un vez, los interpretes, van convirtiendo el código poco a poco, linea a linea, instrucción a instrucción, etc.

Lexer o analizador léxico : el lexer analiza una secuencia de caracteres y la convierte en una secuencia de tokens, por ejemplo: a = 2 + 6. El lexer lo traduciría como:

1. identificador [a]
2. operador de asignación [=]
3. constante numérica [2]
4. operador de suma [+]
5. constante numérica [6]

Parser o analizador sintáctico : agrupa los token resultados del lexer formando frases gramaticales que posteriormente serán analizadas por el compilador. Utilizando el ejemplo visto con el lexer (a = 2 + 6), así quedaría con el parser ver:3.1

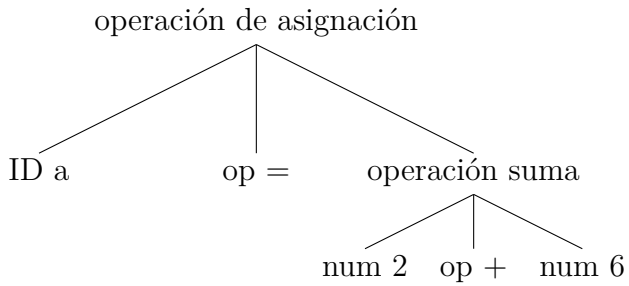

Figura 3.1: Árbol definido por el parser de la sentencia a = 2 + 6

Capítulo 4

Técnicas y herramientas

Para la realización de este proyecto se usará un debugger de C y se adaptará a una nueva interfaz mas clara y limpia para los alumnos nuevos que den sus primeros pasos en programación.

Para la realización del proyecto me he ayudado de un parser de C para la librería PLY de Python, pycparser [1]

Para programar se ha utilizado Spyder del paquete de Anaconda 3 [2]

Para la documentación en LaTeXse ha utilizado Overleaf[3], he utilizado Overleaf porque al tener el proyecto en nube me es mas cómodo y fácil acceder a él desde cualquier dispositivo además de incluir un previsualizador online del pdf resultante del documento de LaTeX

Utilizo el compilador de gcc, que al no venir de base en Windows se ha utilizado el paquete de herramientas de MinGW [4]

[1] Repositorio de GitHub: https://github.com/eliben/pycparser
[2] Sitio web de anaconda para su descarga: https://www.anaconda.com
[3] Sitio web de Overleaf: https://www.overleaf.com
[4] Sitio web de MinGW: http://mingw.org/

Herramientas	Aplicación	Memoria
Spyder	X	
Pycparser	X	
GitHub Desktop	X	X
Overleaf		X

Tabla 4.1: Herramientas y tecnologías utilizadas en cada parte del proyecto

Capítulo 5

Aspectos relevantes del desarrollo del proyecto

En un principio se contemplo la posibilidad de realizar el trabajo en Google Colaboratory[1] pero por problemas de comunicación entre el notebook de Google Colab y la maquina local se terminó descartando la idea.

Posteriormente se intento realizar el trabajo en un notebook local utilizando Jupyter, incluido en el paquete de Anaconda 3, pero debido a la restricción de permisos que tienen los exploradores de internet se terminó descartando también.

Para el interprete de C se pensó aprovechar la herramienta gdb incluida en Linux de base y en Windows mediante MinGW pero ante la imposibilidad de crear una comunicación continua entre la consola de la maquina y la aplicación se terminó por descartar esta opción.

Se tenía la intención de diseñar el parser y lexer enteros utilizando una gramática[2] y un léxico[3] encontrados en internet.

[1]Sitio web de Googe Colab: https://colab.research.google.com/notebooks/intro.ipynb

[2]Enlace a gramática; https://www.lysator.liu.se/c/ANSI-C-grammar-y.html

[3]Enlace a léxico: https://www.lysator.liu.se/c/ANSI-C-grammar-l.html

Capítulo 6

Trabajos relacionados

Algunos ejemplos de proyectos similares a este son los asistentes que hemos ido usando a lo largo de la carrera como son Eclipse, Visual Studio y Spyder.

Otros ejemplos serían los asistentes online como C tutor[1] y GDBOnline[2]

[1]Sitio web de CTutor: http://www.pythontutor.com/c.html
[2]Sitio web de GDBOnline: https://www.onlinegdb.com/

Capítulo 7

Conclusiones y Líneas de trabajo futuras

7.1. Aprendizaje

Este proyecto me ha servido mucho para profundizar en como funcionan los lenguajes de programación, los compiladores, los debuggers, etc.
En base a esto también he aprendido como funcionan asistentes como Spyder o Eclipse a la hora de marcar con colorines las palabras clave.
De forma similar al anterior los editores de texto como Word o Writer para notificarte faltas de ortografía y de construcción de frases.
En resumidas cuentas, me quedaron pendientes algunos conceptos de la asignatura de procesadores del lenguaje y este TFG ha sido muy útil para reforzarlos.

También me ha servido para establecerme metodologías de trabajo, horarios y fechas, establecer tareas para hacer un día o semana y por extensión también he aprendido a manejar la frustración a la hora de trabajar y la importancia de los descansos.

He aprendido como de distinto funcionan los punteros en C y Python, se profundizará en el tema cuando hable de las limitaciones de la herramienta.

7.2. Limitaciones de la herramienta

Por como están hechos los punteros en python y c, a pesar de que Python este implementado en C, no se puede adaptar el depurador a ellos ver 7.1 y 7.2. En Python los tipos primitivos cambian de posición en memoria cada

vez que son editados.

```
1  i=9
2  print(hex(id(i)))
3  i+=15
4  print(hex(id(i)))
5  i+=73
6  print(hex(id(i)))
```

```
0x749a6d40
0x749a6f20
0x749a7840
```

Figura 7.1: Ejemplo de un puntero en Python

```
PS D:\Downloads> Get-Content .\punteros.c
#include <stdio.h>

int main(){
        int a;
        printf("direccion de a %p\n",&a);
        a=5;
        printf("direccion de a %p\n",&a);
        a++;
        printf("direccion de a %p\n",&a);
        a=30;
        printf("direccion de a %p\n",&a);
        return 0;
}
PS D:\Downloads> gcc .\punteros.c -g -o punteros
PS D:\Downloads> .\punteros.exe
direccion de a 0061FF1C
direccion de a 0061FF1C
direccion de a 0061FF1C
direccion de a 0061FF1C
```

Figura 7.2: Ejemplo de un puntero en C

De igual forma se aplica esta problemática a las uniones de C, al no ser un lenguaje fuertemente tipado como puede ser C o Java permite cierta flexibilidad en las variables ver 7.3

Diferencias entre lenguajes aparte, otra limitación viene dada por el propio parser, ya que no puede manejar declaraciones tipo #define ver 7.4

```
1 frase="hola mundo"
2 print(frase)
3 frase=7
4 print(frase)
5 frase=[1,2,3,4,5,6]
6 print(frase)
```

```
hola mundo
7
[1, 2, 3, 4, 5, 6]
```

Figura 7.3: Ejemplo de la flexibilidad de Python en cuanto a tipos

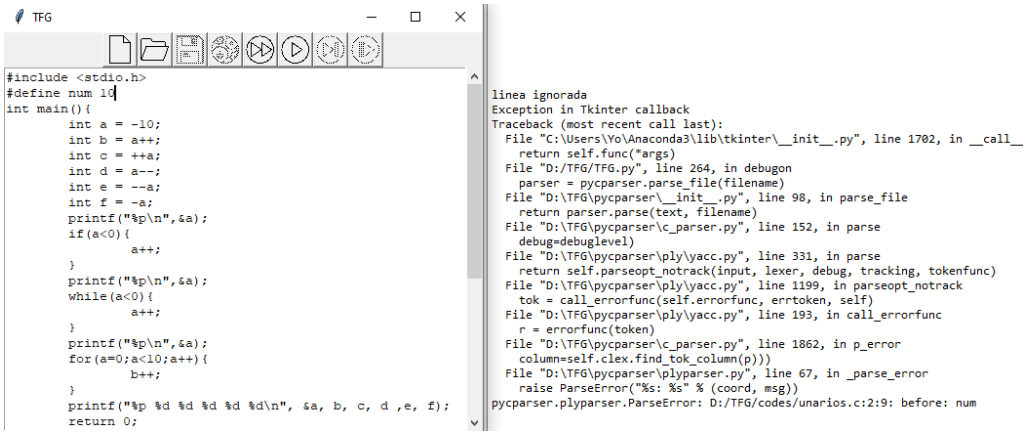

Figura 7.4: Error al procesar los define

7.3. Lineas de trabajo futuras

- Una posibilidad sería añadir colorines de forma similar a lo que hacen el resto de asistenes o programas mas simples como sublime text o notepad++.
- Otra posibilidad es añadir más funciones propias de C, de momento el asistente solo incluye printf, scanf, strcpy, pow y sqrt.
- Crear algún tipo de clase que permita la incorporación de las uniones y punteros.
- Corrección de errores si se encontrasen.

Bibliografía